

(CS104)Object Oriented Programming Concepts through Java

Name of the Instructor(s) K Sudheer Kumar

Learning Resources

Chalk & Talk, Course notes PDF's, PPTs, Tutorial Sites and Videos.

Required Resources

TEXT BOOKS:

1. Java 7 Programming - Black Book, By Kogent Learning Solutions Inc., Freamtech Publications
2. Head First Java 2nd Edition by Kathy Sierra, Oreilly Publication
3. T.Budd "Understanding OOP with Java" updated Edition, pearson education, ISBN:10:0201612739
4. Herbert schildt "Java the complete reference", 7th Edition, TMH, ISBN:0072263857

REFERENCE BOOKS:

1. Y. Daniel Liang "Introduction to Java programming" 6th Edition, pearson ducation, ISBN:10:0132221586
2. R.A. Johnson-An introduction to Java programming and object oriented application development, Thomson, ISBN:-10:0619217464

WEB LINKS:

1. www.tatamcgrawhill.com/html/9780070636774.html
2. <http://nptel.iitm.ac.in>
3. <https://www.cl.cam.ac.uk/teaching/0910/OOProg/OOP.pdf>
4. www.java2s.com

Assessment

The course will be evaluated for a total of 100 marks, with 30 marks for Continuous Internal Assessment (CIA) and 70 marks for Semester End Examination (SEE). Out of 30 marks allotted for CIA during the semester, marks are awarded by taking average of two CIA examinations or the marks scored in the make-up examination.

Continuous Internal Assessment (CIA):

CIA is conducted for a total of 30 marks (Table 1), with 20 marks for Continuous Internal Examination (CIE), 05 marks for Assignment and 05 marks for Attendance.

Table 1: Assessment pattern for CIA

Component	Theory			Total Mark s
Type of Assessment	CIE Exam	Assignment/Quiz	Attendance	
CIA Marks	25	5	5	30

Continuous Internal Examination (CIE):

Two CIE exams shall be conducted at the end of the 8th and 16th week of the semester respectively. The CIE exam is conducted for 20 marks of 1 1/2 hours duration consisting of two parts. Part–A shall have ten compulsory questions of half mark each. In part–B, three out of five questions have to be answered where, each question carries 5 marks. Marks are awarded by taking average of marks scored in two CIE exams.

Assignment/Quiz:

Two Quiz exams shall be online examination consisting of 25 multiple choice questions and are to be answered by choosing the correct answer from a given set of choices (commonly four). Marks shall be awarded considering the average of two quizzes for every course. This may include seminars, assignments, quizzes or case study.

Semester End Examination (SEE): The SEE is conducted for 70 marks of 3 hours duration. The syllabus for the theory courses is divided into five units and each unit carries equal weightage in terms of marks distribution.

Quizzes /Assignments Schedule

S.No	Unit	Quiz/Assignment Scheduled Date
1	I	At the end of 4 th Week of instruction
2	II	At the end of 7 th Week of instruction
3	III	At the end of 10 th Week of instruction
4	IV	At the end of 13 th Week of instruction
5	V	At the end of 16 th Week of instruction

Course Activity

✓ Topic 1:

Introduction to Object Oriented Programming, Classes, Objects, Methods, Constructors and Encapsulation

Think Pair Share

✓ Topic 2:

Inheritance, Polymorphism, Abstraction Concepts

Think Pair Share and Group Writing Assignments

✓ Topic 3:

Exception Handling and Multi Threading Concepts

Group Writing Assignments and Case Study

✓ Topic 4:

Introduction to GUI Programming, Applets, Event Handling and AWT Controls

Case Study and Team Based Learning

✓ Topic 5:

MVC Architecture and Swings

Group Writing Assignments and Team Based Learning

Grades will be shared immediately using Edmodo Quizzes in online.

Late Assignments (Define Ground Rules) : Marks will be reduced after deadline .

Note: The knowledge and abilities tested (Blooms Level) along with tentative date/week and time of exam shall be mentioned

How to Contact Instructor:

In-person office hours: 9:30AM-4:00PM Room No 1309

Online office hours: 9:30AM-4:00PM

E-Mail: sudheerkomuravelly@gmail.com

Phone numbers: 9908291292 and 9666967096

WhatsApp: 9908291292

Optional: 4:00PM-5:00PM

Pre-requisite

C Language/Any Introductory Programming Language
Programming Knowledge and Analytical, Problem Solving Skills needed to succeed in this course

Technology Requirements: (optional)

Laptops :needed for class work, lab work and to practice at home
Software : JDK 1.8 and above

Overview of Course:**What is the course about:**

This course explains the fundamental ideas behind the object oriented approach to programming. Knowledge of java helps to create the latest innovations in programming. Like the successful computer languages that came before, java is the blend of the best elements of its rich heritage combined with the innovative concepts required by its unique environment. This course involves OOP concepts, java basics, inheritance, polymorphism, interfaces, packages, Exception handling, multithreading, files, and GUI components.

What are the general topics or focus?

Object Oriented Principles and their Implementation

How does it fit with other courses in the department or on campus?

It can be a pre requisite course for various Computer Science Programming Subjects in the forth coming semesters as well as for other advanced technologies like android and web programming. It helps the students of other branches to perform well in Placement Drives where most of the placements are being held for Software Companies.

Why would students want to take this course and learn this material?

Students want to take this course to gain knowledge of Object Oriented Programming and it is the only language which provides and has more no of jobs in the industry.

Methods of instruction

Lecture
Discussion

Workload

Estimated amount of time to spend on course readings:4 Theory Hours and 3 Laboratory Hours per Week

Estimate amount of time to spend on course assignments and projects: 3 Hours per week

Key concepts

Class, objects, methods, constructors, various object oriented principles, their implementation, Exception handling, multi threading concepts and gui programming

Difficult Topics

multi threading

Optional: Pre Assessment Test -Review of the student's standard: C Basics Test

Lesson Plan

Course Outcomes (COs):

At the end of the course the student should be able to:

1. *Identify* classes, objects, members of a class and relationships among them needed for a specific problem
2. *Demonstrate* the concepts of polymorphism and inheritance and reusability
3. *Illustrate* Java programs to implement error handling techniques using exception handling
4. *Compare* Multithreaded programming with ordinary programming models
5. *Build* GUI based applications.

Course Articulation Matrix: Mapping of Course Outcomes (COs) with Program Outcomes (POs)

Course Outcomes (COs) / Program Outcomes (POs)	1	2	3	4	5	6	7	8	9	10	11	12	PSO1	PSO2
<i>Identify</i> classes, objects, members of a class and relationships among them needed for a specific problem	3	3											3	1
<i>Demonstrate</i> the concepts of polymorphism and inheritance and reusability	3	3	1		1								3	1
<i>Illustrate</i> Java programs to implement error handling techniques using exception handling	3	3	2		2								3	2
<i>Compare</i> Multithreaded programming with ordinary programming models	3	3											2	2
<i>Create</i> GUI based applications.	3	3	2		3				1				3	3

UNIT I

UNIT WISE PLAN

UNIT-I: Object Oriented Thinking		Planned Hours: 09	
S. No.	Topic Learning Outcomes	Cos	Blooms Levels
1	Explain the concept of class and objects with access control to represent real world entities	CO1	L2
2	Demonstrate the behavior of programs involving the basic programming constructs like control structures, constructors, string handling and garbage collection	CO1	L2
3	Use object oriented programming concepts to solve real world problems	CO2	L3
4	Use overloading methodology on methods and constructors to develop application programs.	CO2,CO3	L3

Object Oriented Thinking

Need for oop paradigm:

Two Paradigms of Programming:

As you know, all computer programs consist of two elements: code and data. Furthermore, a program can be conceptually organized around its code or around its data. That is, some programs are written around —what is happening‖ and others are written around —who is being affected.‖ These are the two paradigms that govern how a program is constructed.

The first way is called the process-oriented model. This approach characterizes a program as a series of linear steps (that is, code). The process-oriented model can be thought of as code acting on data. Procedural languages such as C employ this model to considerable success. Problems with this approach appear as programs grow larger and more complex. To manage increasing complexity, the second approach, called object-oriented programming, was conceived.

Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code. As you will see, by switching the controlling entity to data, you can achieve several organizational benefits.

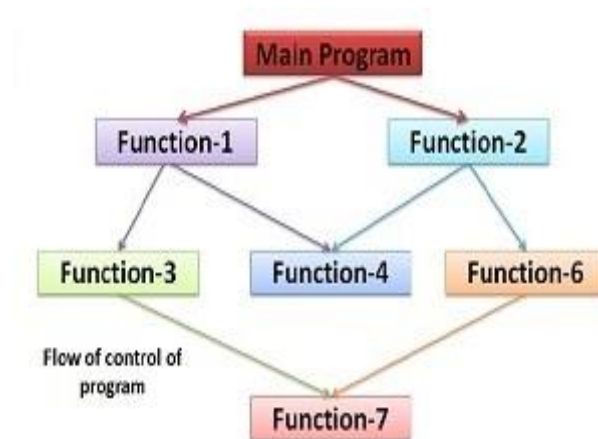
Procedure oriented Programming:

In this approach, the problem is always considered as a sequence of tasks to be done. A number of functions are written to accomplish these tasks. Here primary focus on —Functions‖ and little attention on data.

There are many high level languages like COBOL, FORTRAN, PASCAL, C used for conventional programming commonly known as POP.

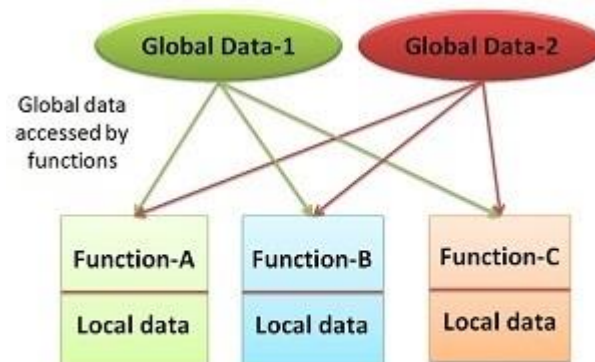
POP basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions.

A typical POP structure is shown in below: Normally a flowchart is used to organize these actions and represent the flow of control logically sequential flow from one to another. In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data. Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we should also revise all the functions that access the data. This provides an opportunity for bugs to creep in.



Structure of procedure oriented program

Drawback: It does not model real world problems very well, because functions are action oriented and do not really corresponding to the elements of the problem.



Characteristics of Procedure oriented Programming:

- ❖ Emphasis is on doing actions.
- ❖ Large programs are divided into smaller programs known as functions.
- ❖ Most of the functions shared global data.
- ❖ Data move openly around the program from function to function.
- ❖ Functions transform data from one form to another.
- ❖ Employs top-down approach in program design.

Object Oriented Programming:

OOP allows us to decompose a problem into a number of entities called objects and then builds data and methods around these entities.

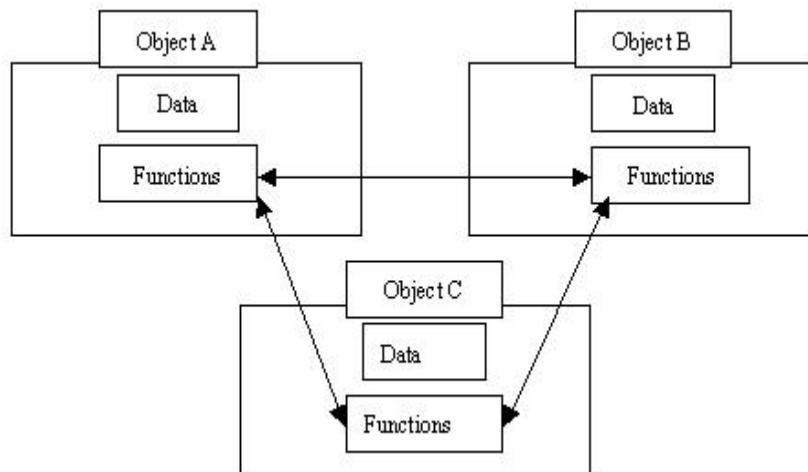
OOP is an approach that provides a way of modularizing programs by creating portioned memory area for both data and methods that can be used as templates for creating copies of such modules on demand.

That is, an object is considered to be a partitioned area of computer memory that stores data and set of operations that can access that data. Since the memory partitions are independent, the objects can be used in a variety of different programs without modifications.

Characteristics of Object Oriented Programming:

- ❖ Emphasis on data .
- ❖ Programs are divided into what are known as methods.
- ❖ Data structures are designed such that they characterize the objects.
- ❖ Methods that operate on the data of an object are tied together .
- ❖ Data is hidden.
- ❖ Objects can communicate with each other through methods.
- ❖ Reusability.
- ❖ Follows bottom-up approach in program design.

Organization of Object Oriented Programming:



Differences between Procedure oriented Programming and Object Oriented Programming:

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called functions .	In OOP, program is divided into parts called objects .
Importance	In POP, Importance is not given to data but to functions as well as sequence of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a real world .
Approach	POP follows Top Down approach .	OOP follows Bottom Up approach .
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is less secure .	OOP provides Data Hiding so provides more security .
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Java Programming: Introduction Born

This language was developed at SUN Microsystems in the year 1995 under the guidance of James Gosling and there team.

Overview of Java

Java is one of the programming language or technology used for developing web applications. Java language developed at SUN Micro Systems in the year 1995 under the guidance of James Gosling and there team. Originally SUN Micro Systems is one of the Academic University (Stanford University Network)

Whatever the software developed in the year 1990, SUN Micro Systems has released on the name of oak, which is original name of java (scientifically oak is one of the tree name). The OAK has taken 18 months to develop.

The oak is unable to fulfill all requirements of the industry. So James Gosling again reviews this oak and released with the name of java in the year 1995. Scientifically java is one of the coffee seed name.

Java divided into three categories, they are

- J2SE (Java 2 Standard Edition)
- J2EE (Java 2 Enterprise Edition)

- J2ME (Java 2 Micro or Mobile Edition)

J2SE

J2SE is used for developing client side applications.

J2EE

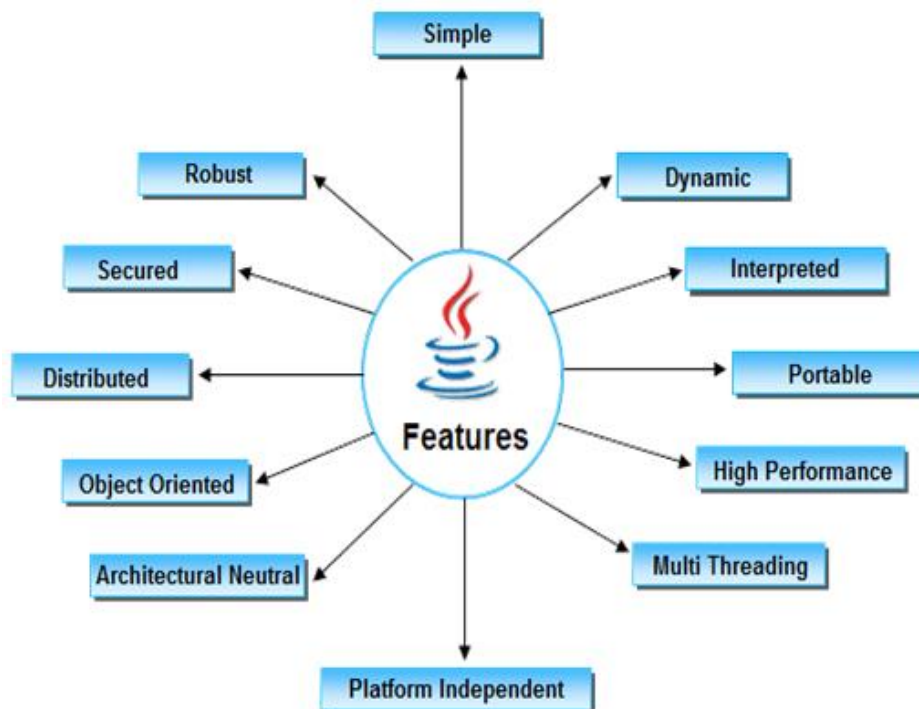
J2EE is used for developing server side applications.

J2ME

J2ME is used for developing mobile or wireless application by making use of a predefined protocol called WAP (wireless Access / Application protocol).

Features(Buzzwords) of Java

Features of a language are nothing but the set of services or facilities provided by the language vendors to the industry programmers. Some important features of java are;



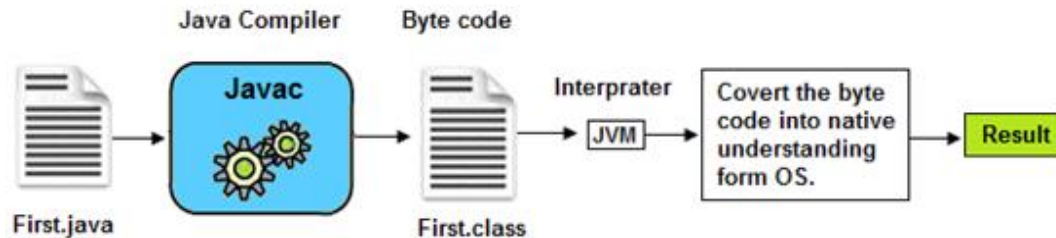
Important Features of Java

- Simple
- Platform Independent
- Architectural Neutral
- Portable
- Multi Threading
- Distributed
- Networked
- Robust
- Dynamic
- Secured
- High Performance
- Interpreted
- Object Oriented

1. Simple

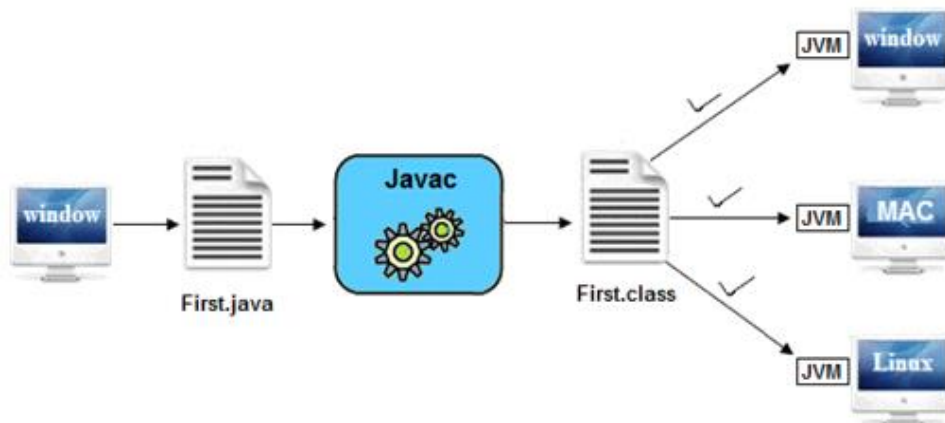
It is simple because of the following factors:

- It is free from pointer due to this execution time of application is improve. [Whenever we write a Java program without pointers then internally it is converted into the equivalent pointer program].
- It has Rich set of API (application protocol interface).
- It has Garbage Collector which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.
- It contains user friendly syntax for developing any applications.



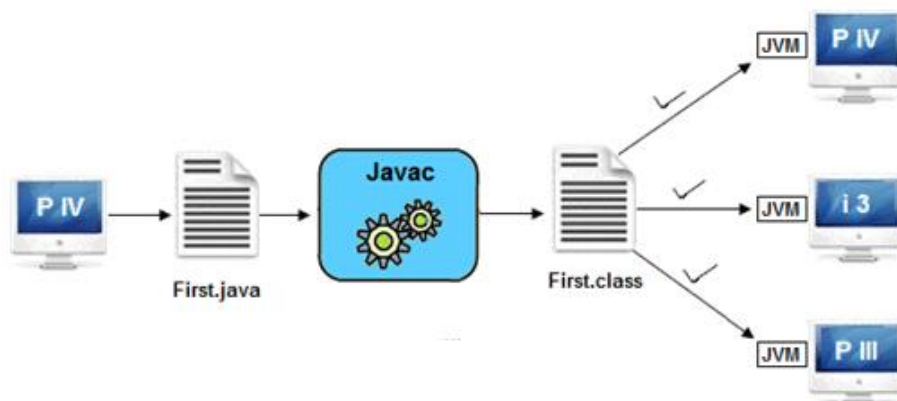
2. Platform Independent

A program or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents O.S).



3. Architectural Neutral

Architecture represents processor. A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering there architecture and vendor (providers) irrespective to its development and compilation.



The languages like C, CPP are treated as architectural dependent.

4. Portable

If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, and Pascal are treated as non-portable language. It is a portable language.

According to SUN Microsystems.

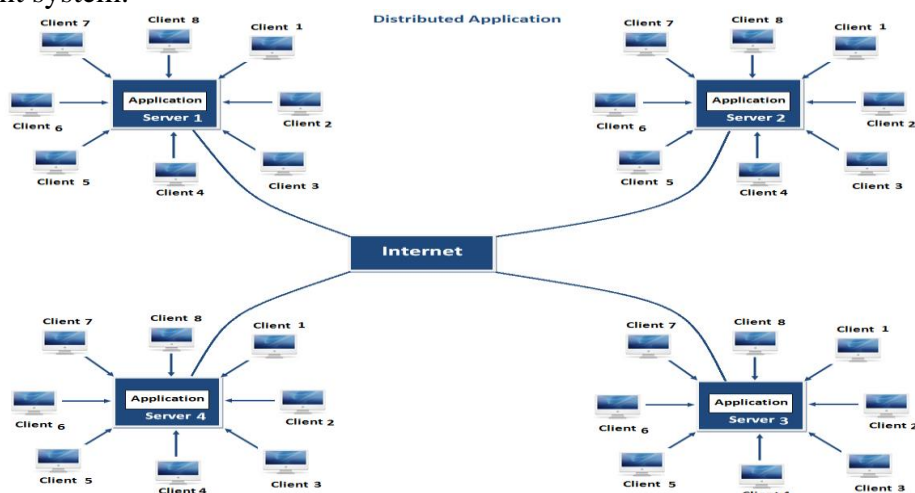
Portability = platform independent + architecture

5. Multithreaded

A flow of control is known as thread. When any Language executes multiple threads at a time that language is known as multithreaded Language. It is multithreaded Language.

6. Distributed

Using this language we can create distributed application. RMI and EJB are used for creating distributed applications. In distributed application multiple client system are depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.



Note: In this architecture same application is distributed in multiple server system.

7. Networked

It is mainly design for web based applications; J2EE is used for developing network based applications.

8. Robust

Simply means of Robust is strong. It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, lack of pointer concept, Exception Handling. All these points make it robust Language.

9. Dynamic

It supports Dynamic memory allocation, due to this memory wastage is reduced and performance of application is improved. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation. To allocate memory space dynamically we use an operator called 'new'. 'new' operator is known as dynamic memory allocation operator.

10. Secure

It is more secured language compare to other language; in this language all code is converted into byte code after compilation which is not readable by human.

11. High performance

It has high performance because of following reasons;

- This language uses Byte code which is faster than ordinary pointer code so Performance of this language is high.
- Garbage collector, collect the unused memory space and improve the performance of application.
- It has no pointers so that using this language we can develop an application very easily.
- It support multithreading, because of this time consuming process can be reduced to execute the program.

12. Interpreted

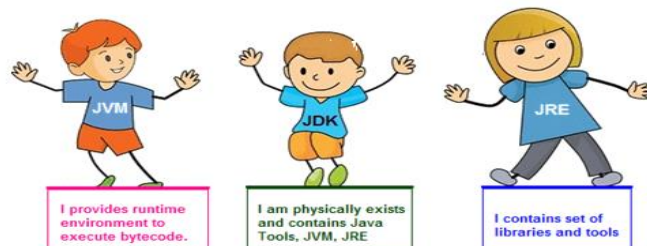
It is one of the highly interpreted programming languages.

13. Object Oriented

It supports OOP's concepts because of this it is most secure language.

Difference between JDK, JVM and JRE

Jvm, Jre, Jdk these all the backbone of java language. Each component has separate works. Jdk and Jre physically exists but Jvm is abstract machine it means it not physically exists.



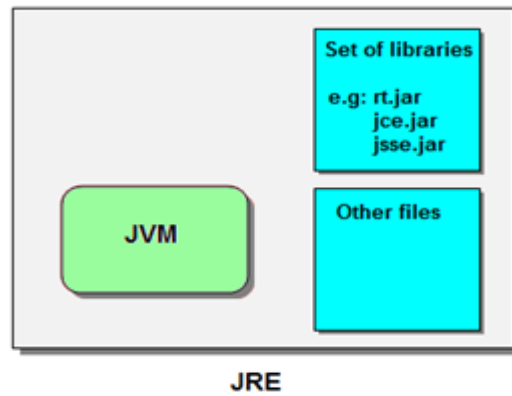
JVM

JVM (Java Virtual Machine) is a software. It is a specification that provides runtime environment in which java byte code can be executed. It not physically exists.

JVMs are not same for all hardware and software, for example for window os JVM is different and for Linux VJM is different. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

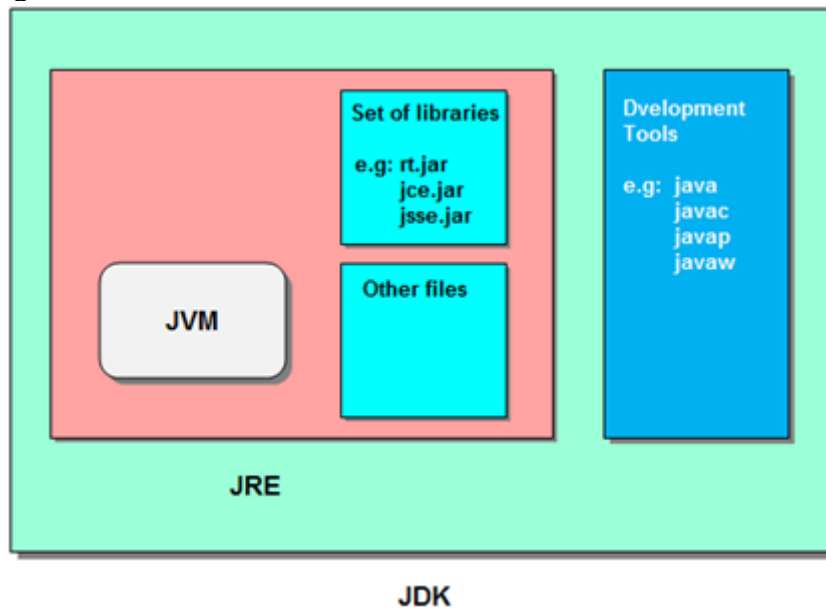
JRE

The Java Runtime Environment (JRE) is part of the Java Development Kit (JDK). It contains set of libraries and tools for developing java application. The Java Runtime Environment provides the minimum requirements for executing a Java application. It physically exists. It contains set of libraries + other files that JVM uses at runtime.



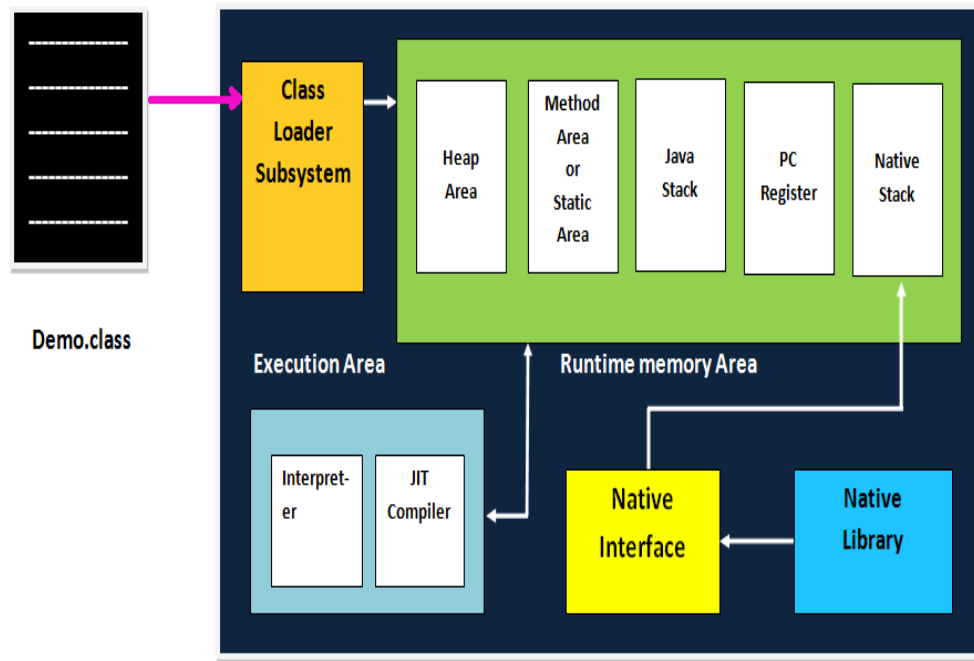
JDK

The Java Development Kit (JDK) is primary component. It physically exists. It is collection of programming tools and JRE, JVM.



JVM Architecture in Java

JVM (Java Virtual Machine) is software. It is a specification that provides Runtime environment in which java byte code can be executed.



Operation of JVM

JVM mainly performs following operations.

- Allocating sufficient memory space for the class properties.
- Provides runtime environment in which java byte code can be executed
- Converting byte code instruction into machine level instruction.

JVM is separately available for every Operating System while installing java software so that JVM is platform dependent.

Note: Java is platform Independent but JVM is platform dependent because every Operating system has different-different JVM which is install along with JDK Software.

Class loader subsystem:

Class loader subsystem will load the .class file into java stack and later sufficient memory will be allocated for all the properties of the java program into following five memory locations.

- Heap area
- Method area
- Java stack
- PC register
- Native stack

Heap area:

In which object references will be stored.

Method area

In which static variables non-static and static method will be stored.

Java Stack

In which all the non-static variable of class will be stored and whose address referred by object reference.

Pc Register

Which holds the address of next executable instruction that means that use the priority for the method in the execution process?

Native Stack

Native stack holds the instruction of native code (other than java code) native stack depends on native library. Native interface will access interface between native stack and native library.

Execution Engine

Which contains Interpreter and JIT compiler whenever any java program is executing at the first time interpreter will come into picture and it converts one by one byte code instruction into machine level instruction. JIT compiler (just in time compiler) will come into picture from the second time onward if the same java program is executing and it gives the machine level instruction to the process which are available in the buffer memory.

Note: The main aim of JIT compiler is to speed up the execution of java program.

What is JIT and Why use JIT

JIT is the set of programs developed by SUN Micro System and added as a part of JVM, to speed up the interpretation phase.

In the older version of java compilation phase is so faster than interpretation phase. Industry has complained to the SUN Micro System saying that compilation phase is very faster and interpretation phase is very slow.

To solve this issue, SUN Micro System has developed a program called JIT (just in time compiler) and added as a part of JVM to speed up the interpretation phase. In the current version of java interpretation phase is so faster than compilation phase. Hence java is one of the highly interpreted programming languages.

How to set path in Java

The path is required to be set for using tools such as javac, java etc.

If you are saving the java source file inside the jdk/bin directory, path is not required to be set because all the tools will be available in the current directory.

But If you are having your java file outside the jdk/bin folder, it is necessary to set path of JDK.

There are 2 ways to set java path:

1. temporary
2. permanent

1) How to set Temporary Path of JDK in Windows

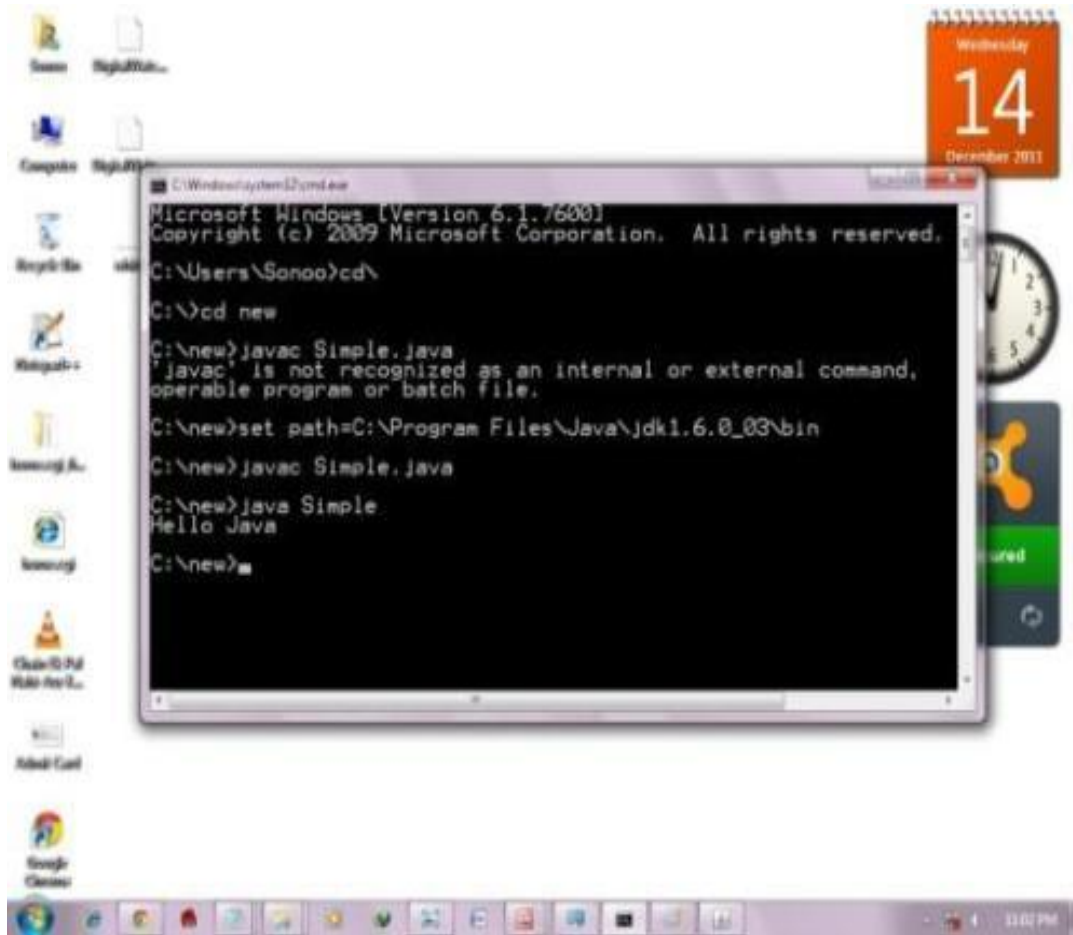
To set the temporary path of JDK, you need to follow following steps:

- Open command prompt
- copy the path of jdk/bin directory
- write in command prompt: set path=copied_path

For Example:

```
set path=C:\Program Files\Java\jdk1.6.0_23\bin
```

Let's see it in the figure given below:



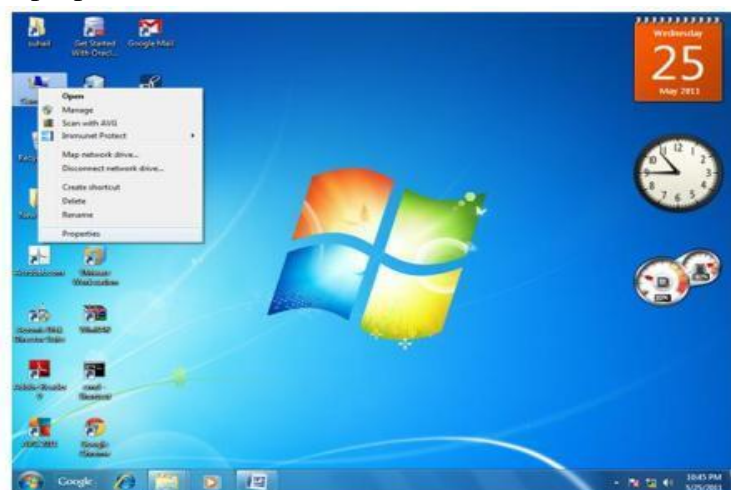
2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

- Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

For Example:

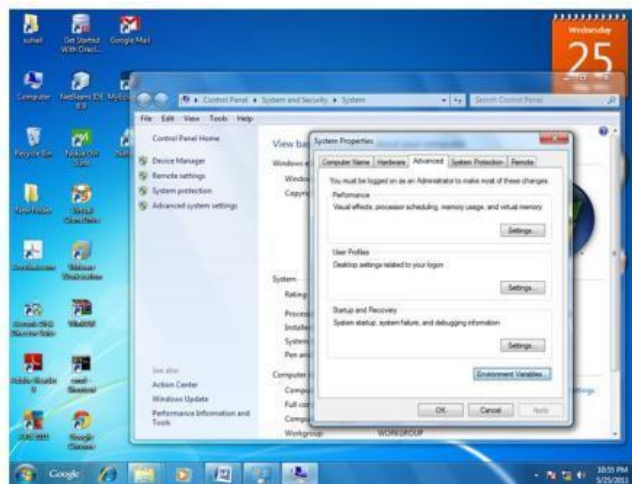
1)Go to MyComputer properties



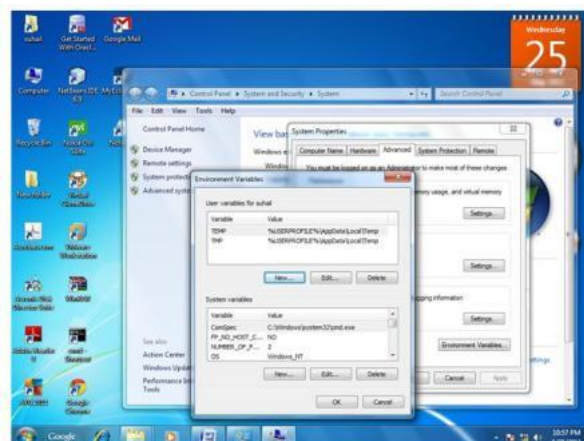
2)click on advanced tab



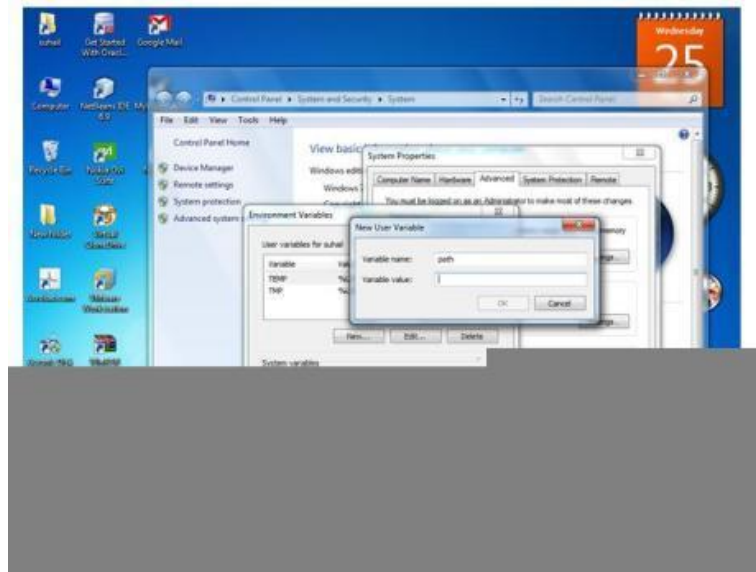
3)click on environment variables



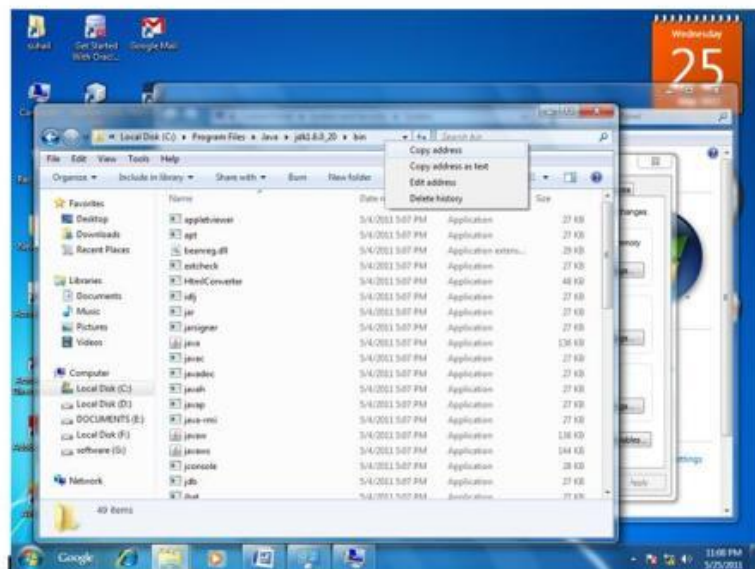
4)click on new tab of user variables



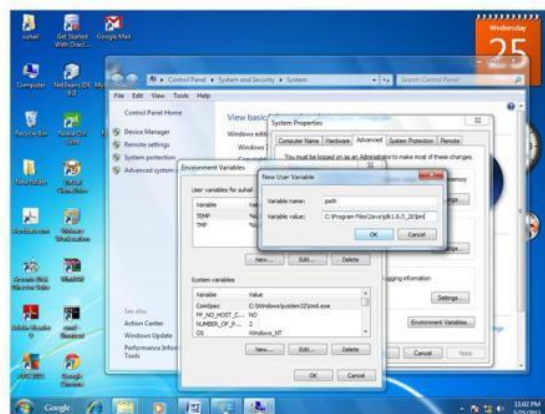
5)write path in variable name



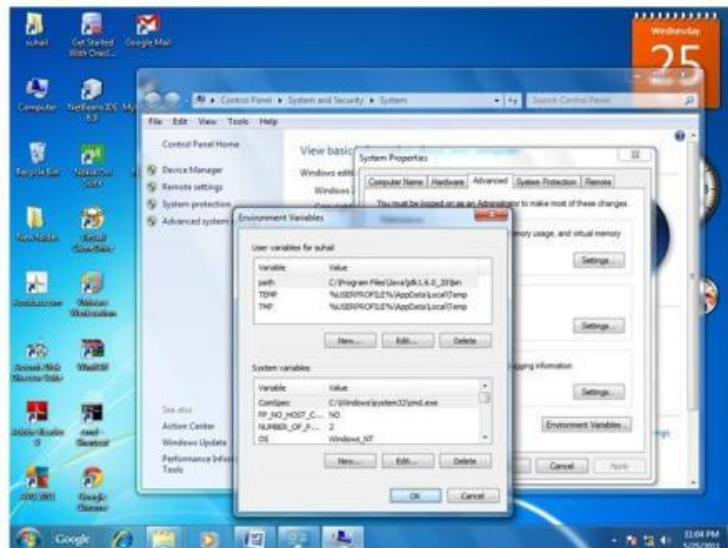
6)Copy the path of bin folder



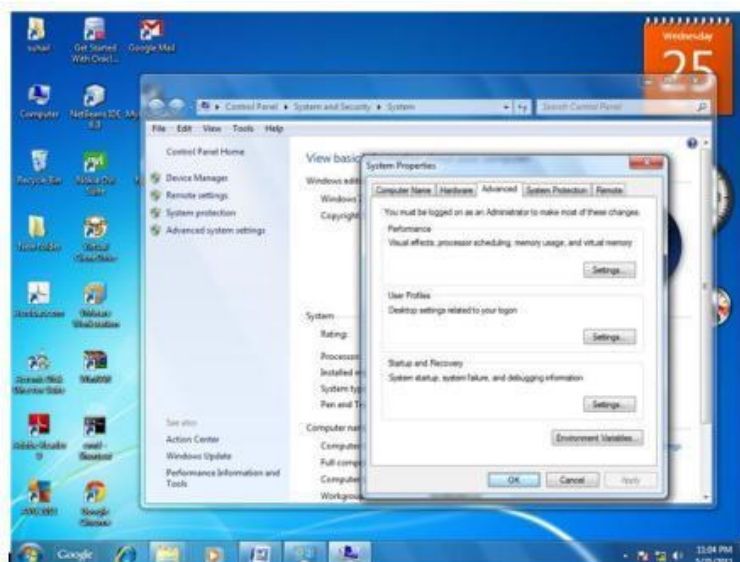
7) paste path of bin folder in variable value



8)click on ok button



9)click on ok button



Now your permanent path is set. You can now execute any program of java from any drive.

Difference between path and classpath in Java

Path

Path variable is set for providing path for all Java tools like java, javac, javap, javah, jar, appletviewer. In Java to run any program we use java tool and for compile Java code use javac tool. All these tools are available in bin folder so we set path upto bin folder.

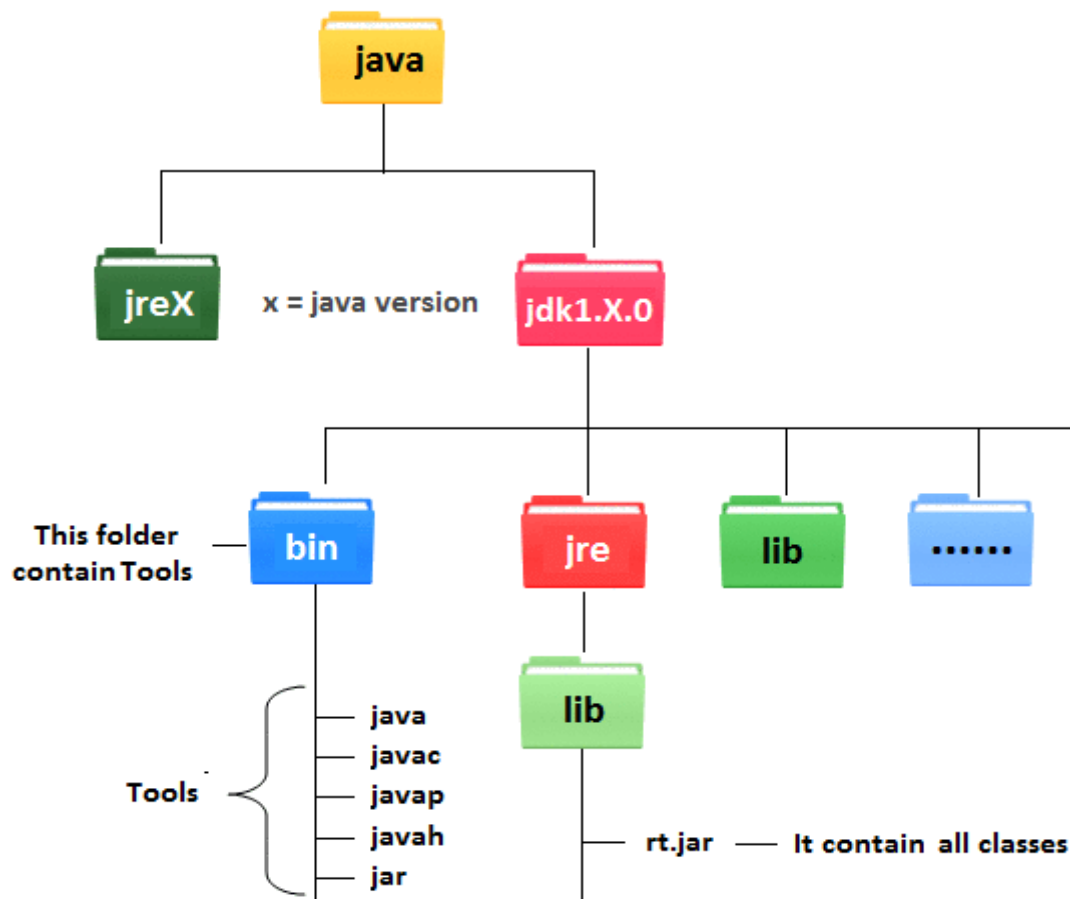
classpath

classpath variable is set for providing path of all Java classes which is used in our application. All classes are available in lib/rt.jar so we set classpath upto lib/rt.jar.

Difference between path and classPath

path	classpath
path variable is set for providing path for all java tools like java, javac, javap, javah, jar, appletviewer	classpath variable is set for provide path of all java classes which is used in our application.

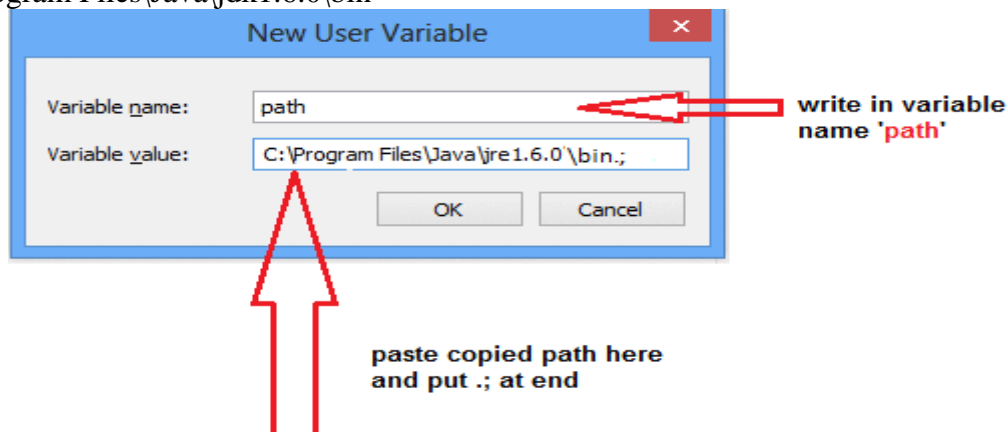
JDK Folder Hierarchy



Path variable is set for use all the tools like java, javac, javap, javah, jar, appletviewer etc.

Example

"C:\Program Files\Java\jdk1.6.0\bin"



C:\Program Files\Java\jre1.6.0\bin.;

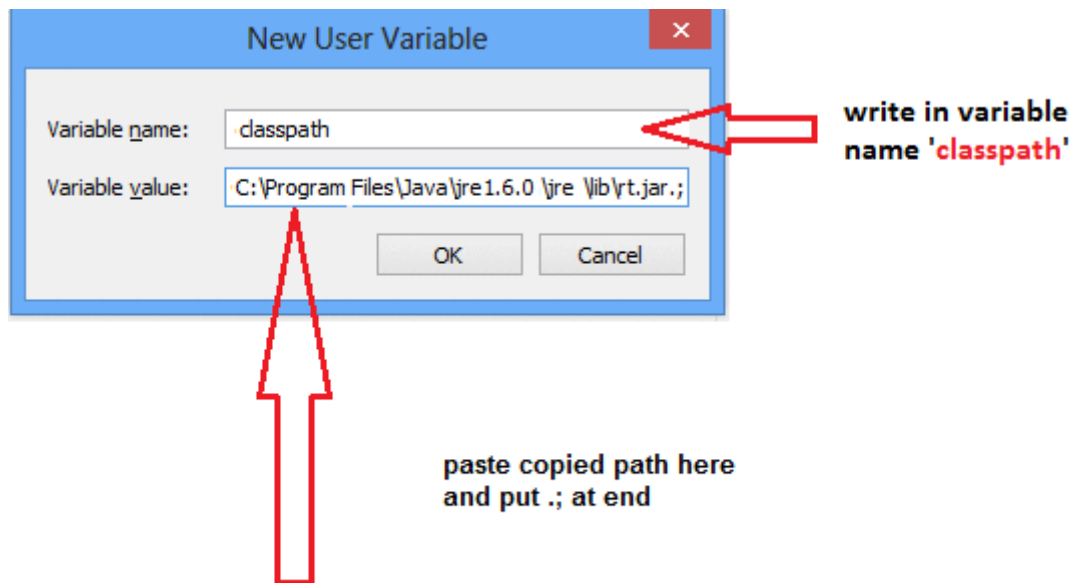
All the tools are present in bin folder so we set path upto bin folder.

Classpath variable is used to set the path for all classes which is used in our program so we set classpath upto rt.jar. In rt.jar file all the .class files are present. When we decompressed rt.jar file we get all .class files.

Example

"C:\Program Files\Java\jre1.6.0\jre\lib\rt.jar"

In above rt.jar is a jar file where all the .class files are present so we set the classpath upto rt.jar.



C:\Program Files\Java\jre1.6.0\jre\lib\rt.jar;

First Java Program

Requirements for java Program

For executing any java program we need given things.

- Install the JDK any version if you don't have installed it.
- Set path of the jdk/bin directory.
- Create the java program
- Compile and run the java program

Steps For compiling and executing the java program

Java is very simple programming language first we write a java program and save it with program class name.

In below program we create a java program with "First" name so we save this program with "First.java" file name. We can save our java program anywhere in our system or computer.

Create First program

Example

Class First

```
{
public static void main(String[] args)
{
System.out.println("Hello Java");
System.out.println("My First Java Program");
}
}
```

Compile and Execute Java Code

To compile: javac First.java

To execute: java First

Output

Hello Java

My First Java Program

Save Java Program

Syntax:

Filename.java

Example:

First.java

Compile Java Program

Syntax:

Javac Filename.java

Example:

Javac First.java

Note: Here Javac and Java are called tools or application programs or exe files developed by sun micro system and supply as a part of jdk 1.5/1.6/1.7 in bin folder. Javac is used for compile the java program and java is used for run the java program.

During the program execution internally following steps will be occurs.

- Class loader subsystem loads or transfer the specified class into main memory(RAM) from secondary memory(hard disk)
- JVM takes the loaded class
- JVM looks for main method because each and every java program start executing from main() method.
- Since main() method of java is static in nature, JVM call the main() method with respect to loaded class (Example: First as First.main(--))

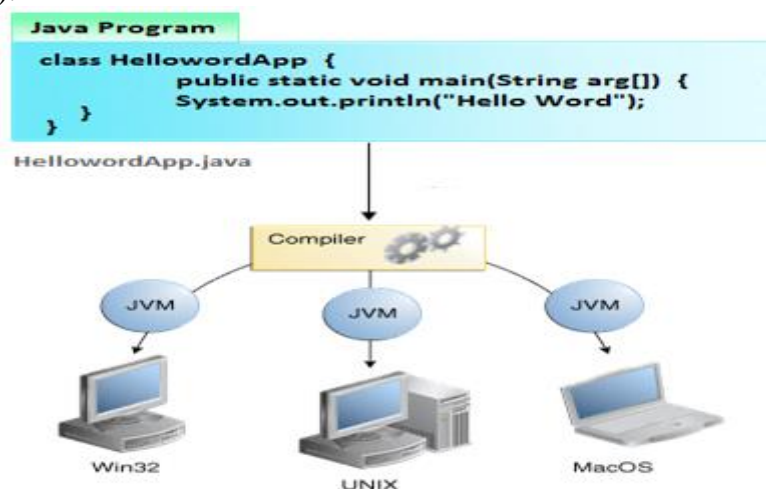
Note: A java program can contain any number of main method but JVM start execution from that main() method which is taking array of object of String class.

Compile and Run Java Program

In the Java programming language, all source code is first written in plain text files and save with the .java extension. After compilation, .class files are generated by javac compiler. A .class file does not contain code that is native to your processor; it instead contains bytecode (it is machine language of the Java Virtual Machine (JVM)).



The java launcher tool then runs your application with an instance of the Java Virtual Machine (JVM).



Steps for compile Java Program

- First Save Java program with same as class name with .java extension.
Example: Sum.java
- Compile: javac Filename.java
Example, javac Sum.java

Note: Here javac is tools or application programs or exe files which is used for Compile the Java program.

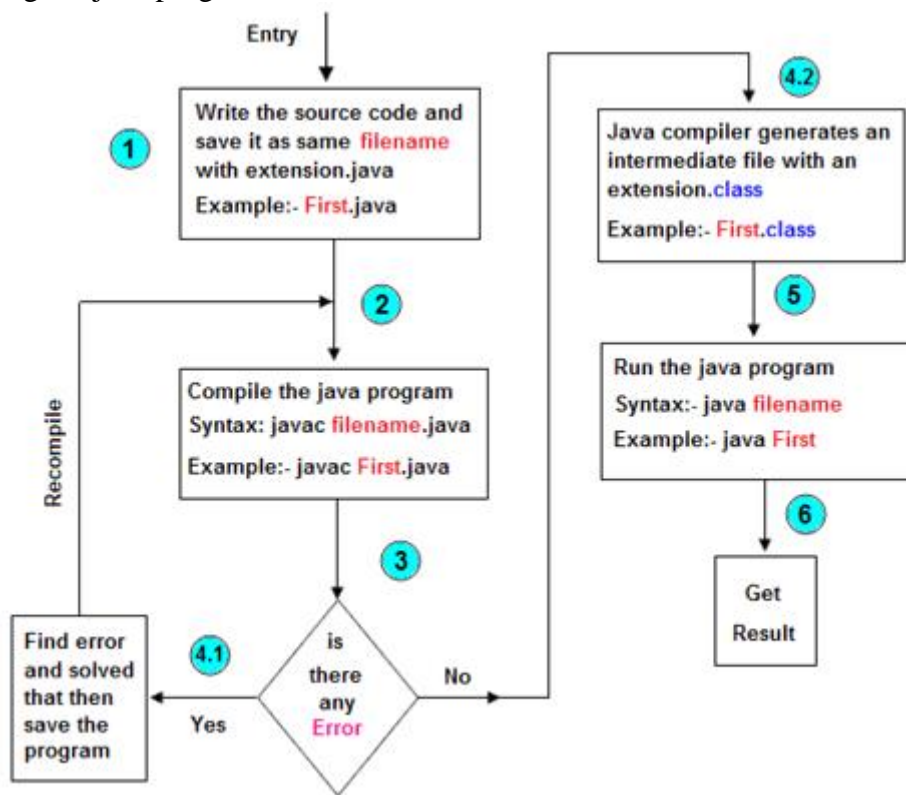
Steps for Run Java Program

- For run java program use java tool.
- Run by: java Filename
Example: java sum

Note: Here java is tools or application programs or exe files which is used for run the Java program.

Steps For compiling and executing the java program

The following sequence of steps represented in the diagram use compiling the java program and executing the java programs.

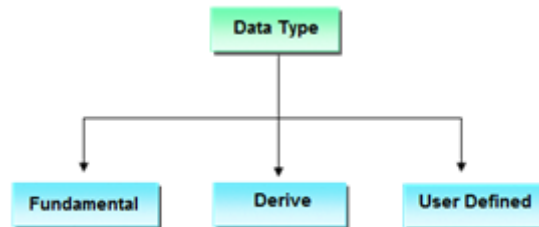


In the above diagram javac and java are called tools or application programs or exe files developed by sun micro system and supply as a part of jdk 1.5/1.6/1.7 in bin folder (starting directory of java is called java home directory).

Data Types in Java

Datatype is a special keyword used to allocate sufficient memory space for the data, in other words Data type is used for representing the data in main memory (RAM) of the computer. In general every programming language is containing three categories of data types. They are

- Fundamental or primitive data types
- Derived data types
- User defined data types.



Primitive data types

Primitive data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.

Example

```
int a; // valid
```

```
a=10; // valid
```

```
a=10, 20, 30; // invalid
```

Here "a" store only one value at a time because it is primitive type variable.

Derived data types

Derived data types are those whose variables allow us to store multiple values of same type. But they never allow storing multiple values of different types. These are the data type whose variable can hold more than one value of similar type. In general derived data type can be achieving using array.

Example

```
int a[] = {10,20,30}; // valid
```

```
int b[] = {100, 'A', "ABC"}; // invalid
```

Here derived data type store only same type of data at a time not store integer, character and string at same time.

User defined data types

User defined data types are those which are developed by programmers by making use of appropriate features of the language.

User defined data types related variables allows us to store multiple values either of same type or different type or both. This is a data type whose variable can hold more than one value of dissimilar type, in java it is achieved using class concept.

Note: In java both derived and user defined data type combined name as reference data type. In C language, user defined data types can be developed by using struct, union, enum etc. In java programming user defined data type can be developed by using the features of classes and interfaces.

Example

```
Student s = new Student();
```

In java we have eight data type which are organized in four groups. They are

- Integer category data types
- Character category data types
- Float category data types
- Boolean category data types

Integer category data types

These category data types are used for storing integer data in the main memory of computer by allocating sufficient amount of memory space.

Integer category data types are divided into four types which are given in following table

	Data Type	Size	Range
1	Byte	1	+ 127 to -128
2	Short	2	+ 32767 to -32768
3	Int	4	+ x to - (x+1)
4	Long	8	+ y to - (y+1)

Character category data types

A character is an identifier which is enclosed within single quotes. In java to represent character data, we use a data type called char. This data type takes two byte since it follows Unicode character set.

Data Type	Size(Byte)	Range
Char	2	+32767 to -32768

Why Java take 2 byte of memory for store character?

Java support more than 18 international languages so java take 2 byte for characters, because for 18 international language 1 byte of memory is not sufficient for storing all characters and symbols present in 18 languages. Java supports Unicode but c support ASCII code. In ASCII code only English language are present, so for storing all English latter and symbols 1 byte is sufficient. Unicode character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters

Float category data types

Float category data type are used for representing float values. This category contains two data types; they are in the given table

Data Type	Size	Range	Number of decimal places
Float	4	+2147483647 to -2147483648	8
Double	8	+ 9.223*1018	16

Boolean category data types

Boolean category data type is used for representing or storing logical values is true or false. In java programming to represent Boolean values or logical values, we use a data type called Boolean.

Why Boolean data types take zero byte of memory?

Boolean data type takes zero bytes of main memory space because Boolean data type of java implemented by Sun Micro System with a concept of flip - flop. A flip - flop is a general purpose register which stores one bit of information (one true and zero false).

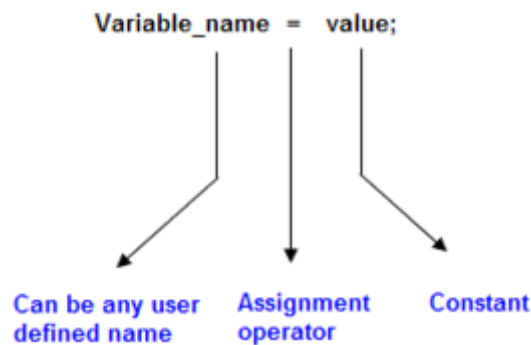
Note: In C, C++ (Turbo) Boolean data type is not available for representing true false values but a true value can be treated as non-zero value and false values can be represented by zero

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte

byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Variable Declaration Rules in Java

Variable is an identifier which holds data or another one variable. Variable is an identifier whose value can be changed at the execution time of program. Variable is an identifier which can be used to identify input data in a program.



Syntax

Variable_name = value;

Rules to declare a Variable

- Every variable name should start with either alphabets or underscore (_) or dollar (\$) symbol.
- No space is allowed in the variable declarations.
- Except underscore (_) no special symbol are allowed in the middle of variable declaration
- Variable name always should exist in the left hand side of assignment operators.
- Maximum length of variable is 64 characters.
- No keywords should access variable name.

Note: Actually a variable also can start with ¥, ¢, or any other currency sign.

Example of Variable Declaration

```
class Sum
{
    public static void main(String[] args)
    {
        int _a, ¢b, ¥c, $d, result;
        _a=10;
        ¢b=20;
        ¥c=30;
        $d=40;
        result=_a+¢b+¥c+$d;
        System.out.println("Sum is :"+result);
    }
}
```

Output

Sum is : 100

Variable declarations

In which sufficient memory will be allocated and holds default values.

Syntax

Datatype variable_name;

byte b1;



Variable initialization

It is the process of storing user defined values at the time of allocation of memory space.

byte b1 = 30;



Variable assignment

Value is assigned to a variable if that is already declared or initialized.

Syntax

Variable_Name = value

int a = 100;



Syntax

int a= 100;

int b;

b = 25; // -----> direct assigned variable

b = a; // -----> assigned value in term of variable

b = a+15; // -----> assigned value as term of expression

Operators in Java

Operator is a special symbol that tells the compiler to perform specific mathematical or logical Operation. Java supports following lists of operators.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators

- Assignment Operators
- Ternary or Conditional Operators

	Operator	Type
unary operator →	++, --	Unary operator
Binary operator {	+, -, *, /, %	Arithmetic perator
	<, <=, >, >=, ==, !=	Relational operator
	&&, , !	Logical operator
	&, , <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator →	?:	Ternary or conditional operator

Arithmetic Operators

Given table shows the entire Arithmetic operator supported by Java Language. Let's suppose variable A hold 8 and B hold 3.

Operator	Example (int A=8, B=3)	Result
+	A+B	11
-	A-B	5
*	A*B	24
/	A/B	2
%	A%4	0

Relational Operators

Which can be used to check the Condition, it always returns true or false. Let's suppose variable A hold 8 and B hold 3.

Operators	Example (int A=8, B=3)	Result
<	A<B	False
<=	A<=10	True
>	A>B	True
>=	A<=B	False
==	A== B	False
!=	A!=(-4)	True

Logical Operators

Which can be used to combine more than one Condition? Suppose you want to combined two conditions A<B and B>C, then you need to use Logical Operator like (A<B) && (B>C). Here && is Logical Operator.

Operator	Example (int A=8, B=3, C=-10)	Result
&&	(A<B) && (B>C)	False
	(B!=-C) (A==B)	True

!	!(B<=-A)	True
---	----------	------

Truth table of Logical Operator

C1	C2	C1 && C2	C1 C2	!C1	!C2
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Assignment operators

Which can be used to assign a value to a variable? Let's suppose variable A hold 8 and B hold 3.

Operator	Example (int A=8, B=3)	Result
+=	A+=B or A=A+B	11
-=	A-=3 or A=A-3	5
=	A=7 or A=A*7	56
/=	A/=B or A=A/B	2
%=	A%=5 or A=A%5	3
=a=b	Value of b will be assigned to a	

Ternary operator

If any operator is used on three operands or variable is known as ternary operator. It can be represented with "?:"

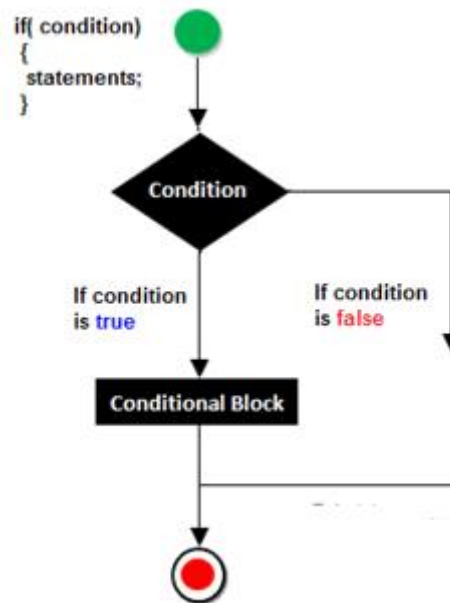
Decision Making/Selection Statements

Decision making statement statements is also called selection statement. That is depending on the condition block need to be executed or not which is decided by condition. If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed. In java there are three types of decision making statement.

- if
- if-else
- switch

if-then Statement

if-then is most basic statement of Decision making statement. It tells to program to execute a certain part of code only if particular condition is true.



Syntax

```

if(condition)
{
    Statement(s)
}

```

Example if statement

```

class Hello
{
    int a=10;
    public static void main(String[] args)
    {
        if(a<15)
        {
            System.out.println("Hello good morning!");
        }
    }
}

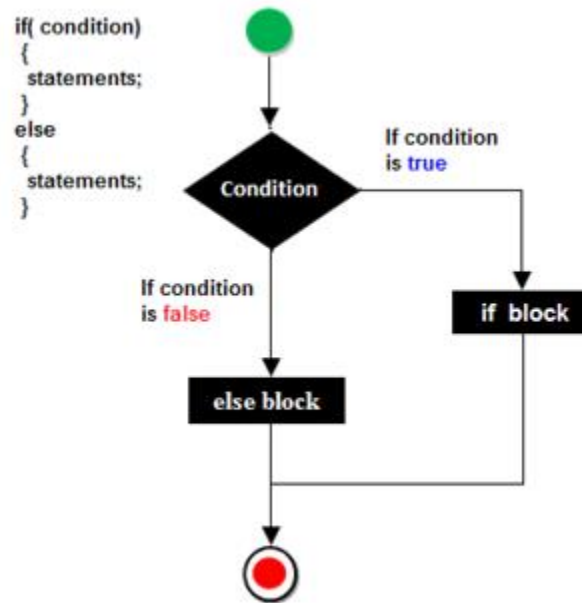
```

Output

Hello good morning

if-else statement

In general it can be used to execute one block of statement among two blocks, in java language if and else are the keyword in java.



Syntax

```

if(condition)
{
  Statement(s)
}
else
{
  Statement(s)
}
.....

```

In the above syntax whenever condition is true all the if block statement are executed, remaining statement of the program by neglecting. If the condition is false else block statement executed and neglecting if block statements.

Example if else

```

import java.util.Scanner;
class Oddeven
{
  public static void main(String[] args)
  {
    int no;
    Scanner s=new Scanner(System.in);
    System.out.println("Enter any number :");
    no=s.nextInt();
    if(no%2==0)
    {
      System.out.println("Even number");
    }
    else
    {
      System.out.println("Odd number");
    }
  }
}

```

}

Output

Enter any number :

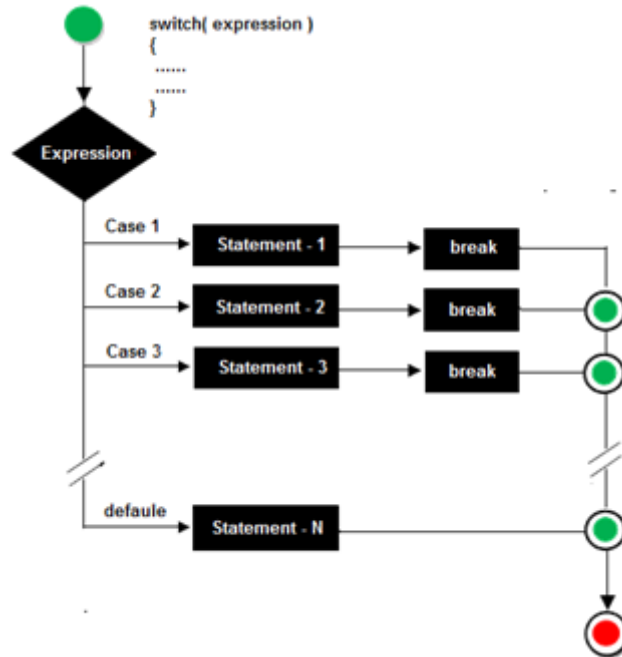
10

Even number

Switch Statement

The switch statement in java language is used to execute the code from multiple conditions or case. It is same like if else-if ladder statement.

A switch statement work with byte, short, char and int primitive data type, it also works with enumerated types and string.



Syntax

```
switch(expression/variable)
```

```
{  
  case value:  
    //statements  
    // any number of case statements  
    break; //optional  
    default: //optional  
    //statements  
}
```

Rules for apply switch statement

With switch statement use only byte, short, int, char data type (float data type is not allowed).

You can use any number of case statements within a switch. Value for a case must be same as the variable in switch.

Limitations of switch statement

Logical operators cannot be used with switch statement. For instance

Example

```
case k>=20: // not allowed
```

Example of switch case

```
import java.util.*;
```



```

class switchCase
{
public static void main(String arg[])
{
int ch;
System.out.println("Enter any number (1 to 7) :");
Scanner s=new Scanner(System.in);
ch=s.nextInt();
switch(ch)
{
case 1:
System.out.println("Today is Monday");
break;
case 2:
System.out.println("Today is Tuesday");
break;
case 3:
System.out.println("Today is Wednesday");
break;
case 4:
System.out.println("Today is Thursday");
break;
case 5:
System.out.println("Today is Friday");
break;
case 6:
System.out.println("Today is Saturday");
break;
case 7:
System.out.println("Today is Sunday");
default:
System.out.println("Only enter value 1 to 7");
}
}
}

```

Output

Enter any number (1 to 7) :

5

Today is Friday

Looping/Iteration statement

Looping statements are the statements execute one or more statement repeatedly several number of times. In java programming language there are three types of loops:while, for and do-while.

Why use loop?

When you need to execute a block of code several number of times then you need to use looping concept in Java language.

Advantage with looping statement

- Reduce length of Code
- Take less memory space.

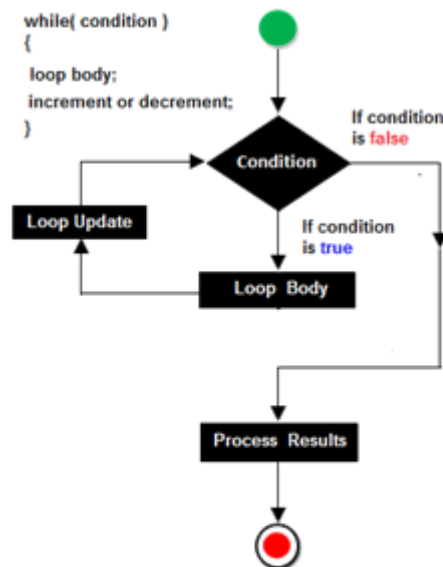
- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.

Difference between conditional and looping statement

Conditional statement executes only once in the program where as looping statements executes repeatedly several number of time.

While loop

In while loop first check the condition if condition is true then control goes inside the loop body otherwise goes outside of the body. while loop will be repeats in clock wise direction.



Syntax

```
while(condition)
```

```
{
Statement(s)
Increment / decrements (++ or --);
}
```

Example while loop

```
class whileDemo
{
public static void main(String args[])
{
int i=0;
while(i<5)
{
System.out.println(+i);
i++;
}
}
```

Output

```
1
2
3
4
5
```

for loop

for loop is a statement which allows code to be repeatedly executed. For loop contains 3 parts Initialization, Condition and Increment or Decrements

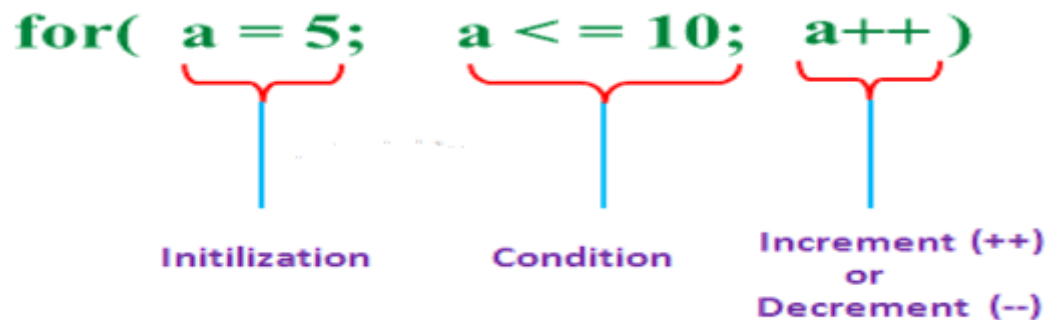
Syntax

```
for ( initialization; condition; increment )
```

```
{
```

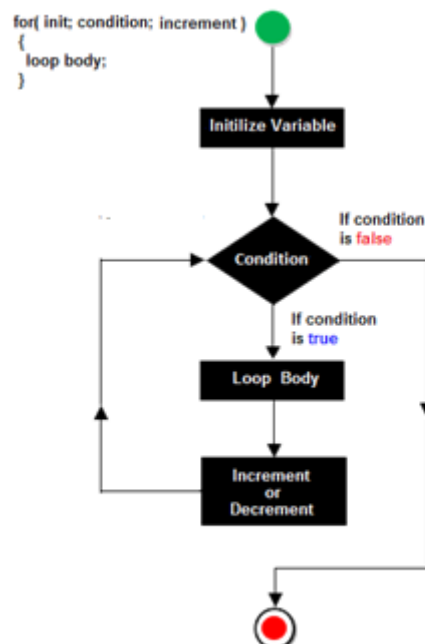
```
  statement(s);
```

```
}
```



- Initialization: This step is executed first and this is executed only once when we are entering into the loop first time. This step is allowed to declare and initialize any loop control variables.
- Condition: This is next step after initialization step, if it is true, the body of the loop is executed, if it is false then the body of the loop does not execute and flow of control goes outside of the for loop.
- Increment or Decrements: After completion of Initialization and Condition steps loop body code is executed and then Increment or Decrements steps is execute. This statement allows updating any loop control variables.

Flow Diagram



Control flow of for loop

```

      ①      ②      ④
for ( a = 1;    a <= 5;    a ++ )
{
    System.out.println("Hello World") ③
}

```

- First initialize the variable
- In second step check condition
- In third step control goes inside loop body and execute.
- At last increase the value of variable
- Same process is repeated until condition not false.

Display any message exactly 5 times.

Example of for loop

```

class Hello
{
public static void main(String args[])
{
int i;
for (i=0; i<5; i++)
{
System.out.println("Hello Friends !");
}
}
}

```

Output

```

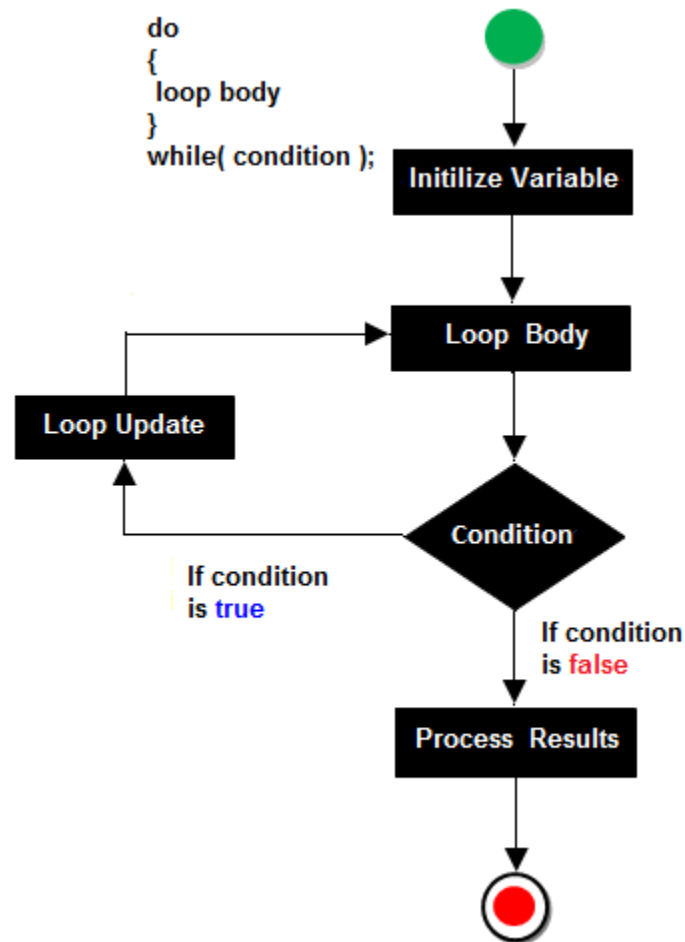
Hello Friends !
Hello Friends !
Hello Friends !
Hello Friends !
Hello Friends !

```

do-while

A do-while loop is similar to a while loop, except that a do-while loop is execute at least one time.

A do while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).



When use do..While loop

When we need to repeat the statement block at least one time then use do-while loop. In do-while loop post-checking process will be occur, that is after execution of the statement block condition part will be executed.

Syntax

```

do
{
Statement(s)

```

increment/decrement (++ or --)

```

}while();

```

In below example you can see in this program i=20 and we chech condition i is less than 10, that means conditon is false but do..while loop execute onec and print Hello world ! at one time.

Example do..while loop

```

class dowhileDemo
{
public static void main(String args[])
{
int i=20;
do
{
System.out.println("Hello world !");
i++;

```

```

}
while(i<10);
}
}
Output
Hello world !
Example do..while loop
class dowhileDemo
{
public static void main(String args[])
{
int i=0;
do
{
System.out.println(+i);
i++;
}
while(i<5);
}
}

```

Output

```

1
2
3
4
5

```

Type Conversion

1.7

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly. For example, assigning an int value to a long variable.

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other

Byte → Short → Int → Long → Float → Double

Widening or Automatic Conversion

Example:

```
class Test
```

```

{
    public static void main(String[] args)
    {
        int i = 100;

        //automatic type conversion
        long l = i;

        //automatic type conversion
        float f = l;
        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }
}

```

Output:

```

Int value 100
Long value 100
Float value 100.0

```

Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.

Double → Float → Long → Int → Short → Byte

Narrowing or Explicit Conversion

Char and number are not compatible with each other. Let's see when we try to convert one into other.

```

//Java program to illustrate incompatible data
// type for explicit type conversion
public class Test
{
    public static void main(String[] argv)
    {
        char ch = 'c';
        int num = 88;
        ch = num;
    }
}

```

Error:

```

7: error: incompatible types: possible lossy conversion from int to char
    ch = num;

```

^
1 error

How to do Explicit Conversion?

Example:

```
//Java program to illustrate explicit type conversion
class Test
{
    public static void main(String[] args)
    {
        double d = 100.04;

        //explicit type casting
        long l = (long)d;

        //explicit type casting
        int i = (int)l;
        System.out.println("Double value "+d);

        //fractional part lost
        System.out.println("Long value "+l);

        //fractional part lost
        System.out.println("Int value "+i);
    }
}
```

Output:

```
Double value 100.04
Long value 100
Int value 100
```

While assigning value to byte type the fractional part is lost and is reduced to modulo 256(range of byte).

Example:

```
//Java program to illustrate Conversion of int and double to byte
class Test
{
    public static void main(String args[])
    {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("Conversion of int to byte.");

        //i%256
        b = (byte) i;
        System.out.println("i = b " + i + " b = " + b);
    }
}
```



```

System.out.println("\nConversion of double to byte.");

//d%256
b = (byte) d;
System.out.println("d = " + d + " b= " + b);
}
}

```

Output:

Conversion of int to byte.
i = 257 b = 1

Conversion of double to byte.
d = 323.142 b = 67

Type promotion in Expressions

While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted. Some conditions for type promotion are:

1. Java automatically promotes each byte, short, or char operand to int when evaluating an expression.
2. If one operand is a long, float or double the whole expression is promoted to long, float or double respectively.

Example:

```

//Java program to illustrate Type promotion in Expressions
class Test
{
    public static void main(String args[])
    {
        byte b = 42;
        char c = 'a';
        short s = 1024;
        int i = 50000;
        float f = 5.67f;
        double d = .1234;

        // The Expression
        double result = (f * b) + (i / c) - (d * s);

        //Result after all the promotions are done
        System.out.println("result = " + result);
    }
}

```

Output:

Result = 626.7784146484375

Explicit type casting in Expressions

While evaluating expressions, the result is automatically updated to larger data type of the operand. But if we store that result in any smaller data type it generates compile time error, due to which we need to type cast the result.

Example:

```
//Java program to illustrate type casting int to byte
class Test
{
    public static void main(String args[])
    {
        byte b = 50;

        //type casting int to byte
        b = (byte)(b * 2);
        System.out.println(b);
    }
}
```

Output

100

NOTE- In case of single operands the result gets converted to int and then it is type casted accordingly.

Example:

```
//Java program to illustrate type casting int to byte
class Test
{
    public static void main(String args[])
    {
        byte b = 50;

        //type casting int to byte
        b = (byte)(b * 2);
        System.out.println(b);
    }
}
```

Output

100

Arrays

An array is a container that holds data (values) of one single type. For example, you can create an array that can hold 100 values of int type.

Array is a fundamental construct in Java that allows you to store and access large number of values conveniently.

How to declare an array?

Here's how you can declare an array in Java:

dataType[] arrayName;

- *dataType* can be a primitive data type like: int, char, Double, byte etc. or an object (will be discussed in later chapters).

- *arrayName* is an identifier.

Let's take the above example again.

```
Double[] data;
```

Here, *data* is an array that can hold values of type *Double*.

But, how many elements can array this hold?

Good question! We haven't defined it yet. The next step is to allocate memory for array elements.

```
data = new Double[10];
```

The length of *data* array is 10. Meaning, it can hold 10 elements (10 *Double* values in this case).

Note, once the length of the array is defined, it cannot be changed in the program.

Let's take another example:

```
int[] age;
```

```
age = new int[5];
```

Here, *age* array can hold 5 values of type *int*.

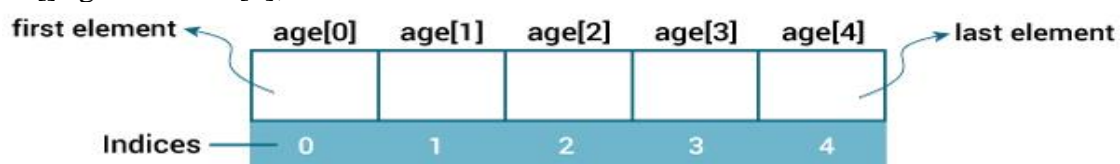
It's possible to declare and allocate memory of an array in one statement. You can replace two statements above with a single statement.

```
int[] age = new int[5];
```

Java Array Index

You can access elements of an array by using indices. Let's consider previous example.

```
int[] age = new int[5];
```



Array age of length 5

The first element of array is *age[0]*, second is *age[1]* and so on.

If the length of an array is *n*, the last element will be *arrayName[n-1]*. Since the length of *age* array is 5, the last element of the array is *age[4]* in the above example.

The default initial value of elements of an array is 0 for numeric types and false for boolean.

We can demonstrate this:

```
class ArrayExample {  
    public static void main(String[] args) {  
  
        int[] age = new int[5];  
  
        System.out.println(age[0]);  
        System.out.println(age[1]);  
        System.out.println(age[2]);  
        System.out.println(age[3]);  
        System.out.println(age[4]);  
    }  
}
```

When you run the program, the output will be:

```
0  
0  
0
```

0
0

There is a better way to access elements of an array by using looping construct (generally for loop is used).

```
class ArrayExample {  
    public static void main(String[] args) {  
  
        int[] age = new int[5];  
  
        for (int i = 0; i < 5; ++i) {  
            System.out.println(age[i]);  
        }  
    }  
}
```

How to initialize arrays in Java?

In Java, you can initialize arrays during declaration or you can initialize (or change values) later in the program as per your requirement.

Initialize an Array during Declaration

Here's how you can initialize an array during declaration.

```
int[] age = { 12, 4, 5, 2, 5};
```

This statement creates an array and initializes it during declaration.

The length of the array is determined by the number of values provided which is separated by commas. In our example, the length of age array is 5.

age[0]	age[1]	age[2]	age[3]	age[4]
12	4	5	2	5

Let's write a simple program to print elements of this array.

```
class ArrayExample {  
    public static void main(String[] args) {  
  
        int[] age = { 12, 4, 5, 2, 5};  
  
        for (int i = 0; i < 5; ++i) {  
            System.out.println("Element at index " + i + ": " + age[i]);  
        }  
    }  
}
```

When you run the program, the output will be:

Element at index 0: 12

Element at index 1: 4

Element at index 2: 5

Element at index 3: 2

Element at index 4: 5

How to access array elements?

You can easily access and alter array elements by using its numeric index. Let's take an example.

```

class ArrayExample {
    public static void main(String[] args) {

        int[] age = new int[5];

        // insert 14 to third element
        age[2] = 14;

        // insert 34 to first element
        age[0] = 34;

        for (int i = 0; i < 5; ++i) {
            System.out.println("Element at index " + i + ": " + age[i]);
        }
    }
}

```

When you run the program, the output will be:

```

Element at index 0: 34
Element at index 1: 0
Element at index 2: 14
Element at index 3: 0
Element at index 4: 0

```

Example: Java arrays

The program below computes sum and average of values stored in an array of type int.

```

class SumAverage {
    public static void main(String[] args) {

        int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
        int sum = 0;
        Double average;

        for (int number: numbers) {
            sum += number;
        }

        int arrayLength = numbers.length;

        // Change sum and arrayLength to double as average is in double
        average = ((double)sum / (double)arrayLength);

        System.out.println("Sum = " + sum);
        System.out.println("Average = " + average);
    }
}

```

When you run the program, the output will be:

```
Sum = 36
```

```
Average = 3.6
```

Couple of things here.

- The for..each loop is used to access each elements of the array.

- To calculate the average, `int` values `sum` and `arrayLength` are converted into `double` since `average` is `double`. This is type casting. Learn more on *Java Type casting*.
- To get length of an array, `length` attribute is used. Here, `numbers.length` returns the length of `numbers` array.

Multidimensional Arrays

It's also possible to create an array of arrays known as multidimensional array. For example, `int[][] a = new int[3][4];`

Here, `a` is a two-dimensional (2d) array. The array can hold maximum of 12 elements of type `int`.

	Column 1	Column 2	Column 3	Column 4
Row 1	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 2	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 3	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Remember, Java uses zero-based indexing, that is, indexing of arrays in Java starts with 0 and not 1.

Similarly, you can declare a three-dimensional (3d) array. For example,

`String[][][] personalInfo = new String[3][4][2];`

Here, `personalInfo` is a 3d array that can hold maximum of 24 ($3*4*2$) elements of type `String`.

In Java, components of a multidimensional array are also arrays.

If you know C/C++, you may feel like, multidimensional arrays in Java and C/C++ works in similar way. Well, it doesn't. In Java, rows can vary in length.

You will see the difference during initialization.

How to initialize a 2d array in Java?

Here's an example to initialize a 2d array in Java.

`int[][] a = { {1, 2, 3}, {4, 5, 6, 9}, {7}, };`

As mentioned, each component of array `a` is an array in itself, and length of each rows is also different.

	Column 1	Column 2	Column 3	Column 4
Row 1	1 <code>a[0][0]</code>	2 <code>a[0][1]</code>	3 <code>a[0][2]</code>	
Row 2	4 <code>a[1][0]</code>	5 <code>a[1][1]</code>	6 <code>a[1][2]</code>	9 <code>a[1][3]</code>
Row 3	7 <code>a[2][0]</code>			

Let's write a program to prove it.

```
class MultidimensionalArray {
    public static void main(String[] args) {

        int[][] a = {           {1, 2, 3},           {4, 5, 6, 9},           {7},           };

        System.out.println("Length of row 1: " + a[0].length);
        System.out.println("Length of row 2: " + a[1].length);
        System.out.println("Length of row 3: " + a[2].length);
    }
}
```

When you run the program, the output will be:

Length of row 1: 3

Length of row 2: 4

Length of row 3: 1

Since each component of a multidimensional array is also an array (a[0], a[1] and a[2] are also arrays), you can use length attribute to find the length of each rows.

Example: Print all elements of 2d array Using Loop

```
class MultidimensionalArray {
    public static void main(String[] args) {

        int[][] a = {           {1, -2, 3},           {-4, -5, 6, 9},           {7},           };
        for (int i = 0; i < a.length; ++i) {
            for(int j = 0; j < a[i].length; ++j) {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

It's better to use for..each loop to iterate through arrays whenever possible. You can perform the same task using for..each loop as:

```
class MultidimensionalArray {
    public static void main(String[] args) {

        int[][] a = {           {1, -2, 3},           {-4, -5, 6, 9},           {7},           };

        for (int[] innerArray: a) {
            for(int data: innerArray) {
                System.out.println(data);
            }
        }
    }
}
```

When you run the program, the output will be:

1
-2
3
-4
-5

6
9
7

How to initialize a 3d array in Java?

You can initialize 3d array in similar way like a 2d array. Here's an example:

// test is a 3d array

```
int[][][] test = { { {1, -2, 3}, {2, 3, 4} }, {  
{-4, -5, 6, 9}, {1}, {2, 3} } };
```

Basically, 3d array is an array of 2d arrays.

Similar like 2d arrays, rows of 3d arrays can vary in length.

Example: Program to print elements of 3d array using loop

```
class ThreeArray {
```

```
    public static void main(String[] args) {
```

```
        // test is a 3d array
```

```
        int[][][] test = { { {1, -2, 3}, {2, 3, 4} }, {  
{-4, -5, 6, 9}, {1}, {2, 3} } };
```

```
        // for..each loop to iterate through elements of 3d array
```

```
        for (int[][] array2D: test) {
```

```
            for (int[] array1D: array2D) {
```

```
                for(int item: array1D) {
```

```
                    System.out.println(item);
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

When you run the program, the output will be:

```
1  
-2  
3  
2  
3  
4  
-4  
-5  
6  
9  
1  
2  
3
```

Object and class in Java

Class: Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties.

A class in java contains:

- Data Members

- Methods
- Constructor
- Blocks
- Class and Interface

Object: Object is an instance of class, object has state and behavior.

An Object in java has three characteristics:

- State
- Behavior
- Identity

State: Represents data (value) of an object.

Behavior: Represents the behavior (functionality) of an object such as deposit, withdraw etc.

Identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

Class is also can be used to achieve user defined data types.

Object is the physical as well as logical entity where as class is the only logical entity.

Real life example of object and class

In real world many examples of object and class like dog, cat, and cow are belong to animal's class. Each object has state and behaviors. For example a dog has state:- color, name, height, age as well as behaviors:- barking, eating, and sleeping.



Vehicle class

Car, bike, truck these all are belongs to vehicle class. These Objects have also different different states and behaviors. For Example car has state - color, name, model, speed, Mileage. as we;; as behaviors - distance travel



Difference between Class and Object

	Class	Object
1	Class is a container which collection of variables and methods.	object is an instance of class
2	No memory is allocated at the time of declaration	Sufficient memory space will be allocated for all the variables of class at the time of declaration.
3	One class definition should exist only once in the program.	For one class multiple objects can be created.

Syntax to declare a Class

```
class Class_Name
{
    data member;
    method;
}
```

Simple Example of Object and Class

In this example, we have created a Employee class that have two data members eid and ename. We are creating the object of the Employee class by new keyword and printing the objects value.

Example

```
class Employee
{
    int eid; // data member (or instance variable)
    String ename; // data member (or instance variable)
    eid=101;
    ename="Hitesh";
    public static void main(String args[])
    {
        Employee e=new Employee(); // Creating an object of class Employee
        System.out.println("Employee ID: "+e.eid);
        System.out.println("Name: "+e.ename);
    }
}
```

Output

Employee ID: 101

Name: Hitesh

Note: “new” keyword is used to allocate memory at runtime, new keyword is used for create an object of class.

Access Modifiers in Java

Access modifiers are those which are applied before data members or methods of a class. These are used to suggest “where to access and where not to access the data members or methods”.

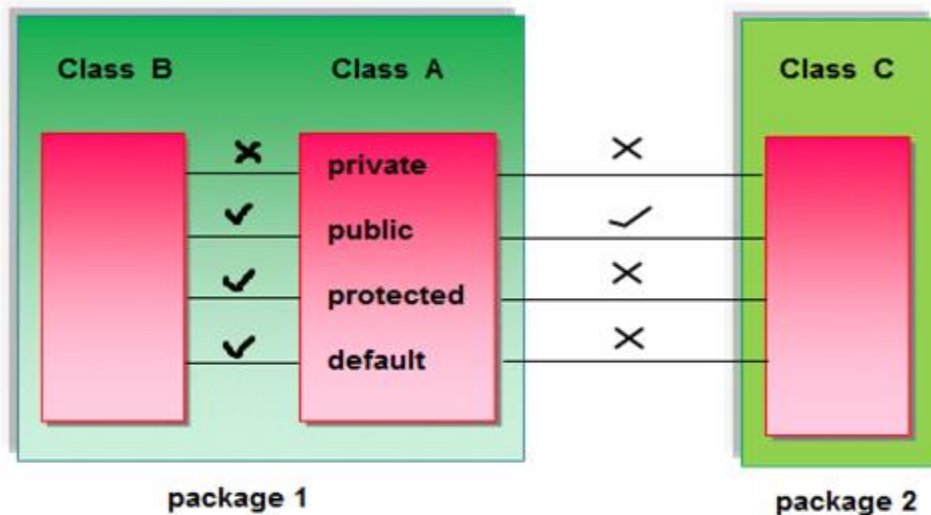
In java programming these are classified into four types:

- Private
- Default (not a keyword)
- Protected
- Public

Note: Default is not a keyword (where as public, private, protected are keywords)

If we are not using private,protected and public keywords then JVM is by default takes default access mode.

Access modifiers are always used for how to reuse the features within the package and access the package between class to class, interface to interface and interface to class. Access modifiers provide features accessing and controlling mechanism among the classes and interfaces.



Note: A protected member of class is accessible within the same class and other class of same package and also accessible in inherited class of other package.

Rules for access modifiers:

The following diagram gives rules for Access modifiers.

Modifiers	Within Same Class	Within other class of Same package	Within derived class of other package	Within external Class of other package
Private (Class level A.S)	Yes	No	No	No
Default (Package level A.S)	Yes	Yes	No	No
Protected (Derived level A.S)	Yes	Yes	Yes	No
Public (Universal A.S)	Yes	Yes	Yes	Yes
A.S --> Access Specifier				

Private: private members of a class are only accessible within the class, but not accessible anywhere in the program. Private is also called class level access modifier.

Example

```
class Hello
{
private int a=20;
private void show()
```

```
{
System.out.println("Hello java");
}
}
```

```
public class Demo
{
    public static void main(String args[])
    {
        Hello obj=new Hello();
        System.out.println(obj.a); //Compile Time Error, you can't access private data
        obj.show(); //Compile Time Error, you can't access private methods
    }
}
```

public: public members of any class are accessible anywhere in the program: inside the same class and outside of class, within the same package and outside of the package. public is also called as universal access modifier.

Example

```
class Hello
{
    public int a=20;
    public void show()
    {
        System.out.println("Hello java");
    }
}
```

```
public class Demo
{
    public static void main(String args[])
    {
        Hello obj=new Hello();
        System.out.println(obj.a);
        obj.show();
    }
}
```

Output

20

Hello Java

Protected: protected members of a class are accessible within the same class and other class of the same package and also accessible in inherited class of the other package. Protected is also called derived level access modifier.

In below example we have created two packages pack1 and pack2. In pack1, class A is public so we can access this class outside of pack1 but method show is declared as a protected so it is only access outside of package pack1 only through inheritance.

Example

```
// save A.java
package pack1;
public class A
{
```

```

protected void show()
{
System.out.println("Hello Java");
}
}
//save B.java
package pack2;
import pack1.*;

class B extends A
{
    public static void main(String args[]){
        B obj = new B();
        obj.show();
    }
}

```

Output

Hello Java

default: default members of a class are accessible only within the same class and the other class of the same package. default is also called package level access modifier.

Example

```

//save by A.java
package pack;
class A
{
    void show()
    {
System.out.println("Hello Java");
    }
}
//save by B.java
package pack2;
import pack1.*;
class B
{
    public static void main(String args[])
    {
        A obj = new A(); //Compile Time Error, can't access outside the package
        obj.show(); //Compile Time Error, can't access outside the package
    }
}

```

Output

Hello Java

Note: private access modifiers is also known as native access modifiers, default access modifiers is also known as package access modifiers, protected access modifiers is also known as inherited access modifiers, public access modifiers is also known as universal access modifiers.

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration –

```
data type variable [ = value][, variable [ = value] ...] ;
```

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java –

Example

```
int a, b, c;      // Declares three ints, a, b, and c.
int a = 10, b = 10; // Example of initialization
byte B = 22;      // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';     // the char variable a is initialized with value 'a'
```

There are three kinds of variables in Java –

- Local variables
- Instance variables
- Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Example

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to only this method.

```
public class Test {
    public void pupAge() {
        int age = 0;
        age = age + 7;
        System.out.println("Puppy age is : " + age);
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.pupAge();
    }
}
```

```
}
```

This will produce the following result –
Puppy age is: 7

Example

Following example uses *age* without initializing it, so it would give an error at the time of compilation.

```
public class Test {  
    public void pupAge() {  
        int age;  
        age = age + 7;  
        System.out.println("Puppy age is : " + age);  
    }  
  
    public static void main(String args[]) {  
        Test test = new Test();  
        test.pupAge();  
    }  
}
```

This will produce the following error while compiling it –

```
Test.java:4:variable number might not have been initialized  
age = age + 7;  
    ^  
1 error
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given

accessibility), they should be called using the fully qualified name.
ObjectReference.VariableName.

Example

```
public class Employee {  
  
    // this instance variable is visible for any child class.  
    public String name;  
  
    // salary variable is visible in Employee class only.  
    private double salary;  
  
    // The name variable is assigned in the constructor.  
    public Employee (String empName) {  
        name = empName;  
    }  
  
    // The salary variable is assigned a value.  
    public void setSalary(double empSal) {  
        salary = empSal;  
    }  
  
    // This method prints the employee details.  
    public void printEmp() {  
        System.out.println("name : " + name );  
        System.out.println("salary :" + salary);  
    }  
  
    public static void main(String args[]) {  
        Employee empOne = new Employee("Ransika");  
        empOne.setSalary(1000);  
        empOne.printEmp();  
    }  
}
```

This will produce the following result –

```
name : Ransika  
salary :1000.0
```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.

- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Example

```
public class Employee {

    // salary variable is a private static variable
    private static double salary;

    // DEPARTMENT is a constant
    public static final String DEPARTMENT = "Development ";

    public static void main(String args[]) {
        salary = 1000;
        System.out.println(DEPARTMENT + "average salary:" + salary);
    }
}
```

```
public class Employee {

    // salary variable is a private static variable
    private static double salary;

    // DEPARTMENT is a constant
    public static final String DEPARTMENT = "Development ";

    public static void main(String args[]) {
        salary = 1000;
        System.out.println(DEPARTMENT + "average salary:" + salary);
    }
}
```

This will produce the following result –

Development average salary:1000

Note – If the variables are accessed from an outside class, the constant should be accessed as Employee.DEPARTMENT

When to use static variables and methods?

Use the static variable for the property that is common to all objects. For example, in class Student, all students share the same college name. Use static methods for changing static variables.

Consider the following java program that illustrates the use of *static* keyword with variables and methods.

```
// A java program to demonstrate use of
// static keyword with methods and variables

// Student class
class Student
{
    String name;
    int rollNo;

    // static variable
    static String cllgName;

    // static counter to set unique roll no
    static int counter = 0;

    public Student(String name)
    {
        this.name = name;

        this.rollNo = setRollNo();
    }

    // getting unique rollNo
    // through static variable(counter)
    static int setRollNo()
    {
        counter++;
        return counter;
    }

    // static method
    static void setCllg(String name){
        cllgName = name ;
    }

    // instance method
    void getStudentInfo(){
        System.out.println("name : " + this.name);
        System.out.println("rollNo : " + this.rollNo);
    }
}
```

```

        // accessing static variable
        System.out.println("cllgName : " + cllgName);
    }
}

//Driver class
public class StaticDemo
{
    public static void main(String[] args)
    {
        // calling static method
        // without instantiating Student class
        Student.setCllg("XYZ");

        Student s1 = new Student("Alice");
        Student s2 = new Student("Bob");

        s1.getStudentInfo();
        s2.getStudentInfo();

    }
}

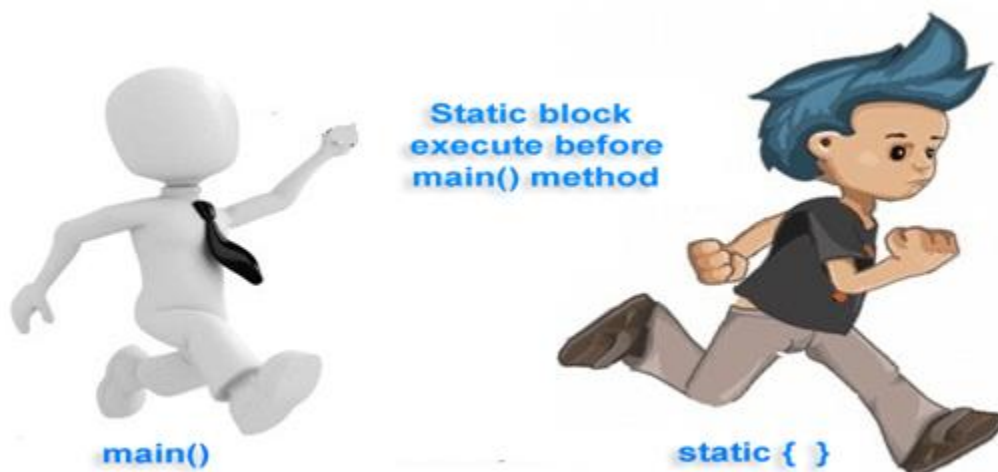
```

Static Block in Java

Static block is a set of statements, which will be executed by the JVM before execution of main method.

At the time of class loading if we want to perform any activity we have to define that activity inside static block because this block execute at the time of class loading.

In a class we can take any number of static block but all these blocks will be execute from top to bottom.



Syntax

static

{

.....

//Set of Statements

.....

}

Note: In real time applications, static block can be used whenever we want to execute any instructions or statements before execution of main method.

Example of Static Block

class StaticDemo

{

static

{

System.out.println("Hello how are u ?");

}

public static void main(String args[])

{

System.out.println("This is main()");

}

}

Output

Hello how are u ?

This is main()

Run java program without main method

class StaticDemo

{

static

{

System.out.println("Hello how are u ?");

}

}

Output

Output:

Hello how are u ?

Exception is thread "main" java.lang.no-suchmethodError:Main

Note: "Exception is thread "main" java.lang.no-suchmethodError:Main" warning is given in java 1.7 and its above versions

More than one static block in a program

class StaticDemo

{

static

{

System.out.println("First static block");

}

static

{

System.out.println("Second Static block");

}

public static void main(String args[])

{

System.out.println("This is main()");

}

}

Output

Output:

First static block

Second static block

This is main()

Note: "Here static block run according to their order (sequence by) from top to bottom.

Why a static block executes before the main method?

A class has to be loaded in main memory before we start using it. Static block is executed during class loading. This is the reason why a static block executes before the main method.

Constructor

Constructor in Java is a special member method which will be called implicitly (automatically) by the JVM whenever an object is created.

Constructors are mainly created for initializing the object. Initialization is a process of assigning user defined values at the time of allocation of memory space.

Syntax

```
className()
```

```
{
```

```
.....
```

```
.....
```

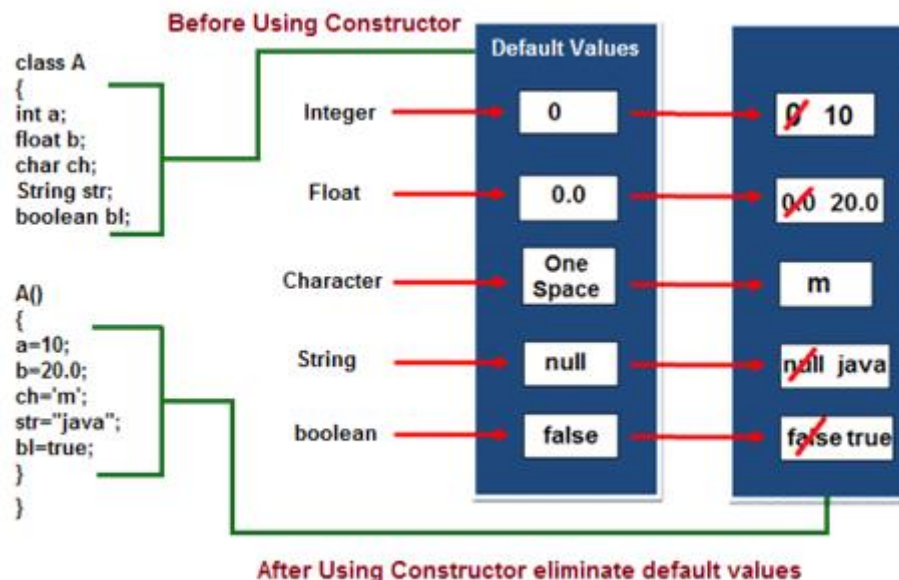
```
}
```

Advantages of constructors in Java

- A constructor eliminates placing the default values.
- A constructor eliminates calling the normal or ordinary method implicitly.

How Constructor eliminate default values?

Constructor are mainly used for eliminate default values by user defined values, whenever we create an object of any class then its allocate memory for all the data members and initialize there default values. To eliminate these default values by user defined values we use constructor.



Example

```
class Sum
```

```
{
```

```
int a,b;
```

```
Sum()
```

```

{
a=10;
b=20;
c=a+b;
System.out.println("Sum: "+c);
}
public static void main(String s[])
{
Sum s=new Sum();
}
}

```

Output

Sum: 30

In above example when we create an object of "Sum" class then constructor of this class call and initialize user defined value in a=10 and b=20. If we cannot create constructor of Sum class then it print " Sum: 0 " because default values of integer is zero.

Rules or properties of a constructor

- Constructor will be called automatically when the object is created.
 - Constructor name must be similar to name of the class.
 - Constructor should not return any value even void also. If we write the return type for the constructor then that constructor will be treated as ordinary method.
 - Constructor definitions should not be static. Because constructors will be called each and every time, whenever an object is created.
 - Constructor should not be private provided an object of one class is created in another class (Constructor can be private provided an object of one class created in the same class).
 - Constructors will not be inherited from one class to another class (Because every class constructor is created for initializing its own data members).
 - The access specifier of the constructor may or may not be private.
1. If the access specifier of the constructor is private then an object of corresponding class can be created in the context of the same class but not in the context of some other classes.
 2. If the access specifier of the constructor is not private then an object of corresponding class can be created both in the same class context and in other class context.

Difference between Method and Constructor

	Method	Constructor
1	Method can be any user defined name	Constructor must be class name
2	Method should have return type	It should not have any return type (even void)
3	Method should be called explicitly either with object reference or class reference	It will be called automatically whenever object is created
4	Method is not provided by compiler in any case.	The java compiler provides a default constructor if we do not have any constructor.

Types of constructors

Based on creating objects in Java constructor are classified in two types. They are

- Default or no argument Constructor
- Parameterized constructor.

Default Constructor

A constructor is said to be default constructor if and only if it never take any parameters. If any class does not contain at least one user defined constructor than the system will create a default constructor at the time of compilation it is known as system defined default constructor.

Syntax

```
class className
{
.....    // Call default constructor
clsname ()
{
Block of statements; // Initialization
}
.....
}
```

Note: System defined default constructor is created by java compiler and does not have any statement in the body part. This constructor will be executed every time whenever an object is created if that class does not contain any user defined constructor.

Example of default constructor.

In below example, we are creating the no argument constructor in the Test class. It will be invoked at the time of object creation.

Example

```
//TestDemo.java
class Test
{
int a, b;
Test ()
{
System.out.println("I am from default Constructor...");
a=10;
b=20;
System.out.println("Value of a: "+a);
System.out.println("Value of b: "+b);
}
}
class TestDemo
{
public static void main(String [] args)
{
Test t1=new Test ();
}
}
```

Output

I am from default Constructor...

Value of a: 10

Value of b: 20

Rule-1:

Whenever we create an object only with default constructor, defining the default constructor is optional. If we are not defining default constructor of a class, then JVM will call automatically system defined default constructor. If we define, JVM will call user defined default constructor.

Purpose of default constructor?

Default constructor provides the default values to the object like 0, 0.0, null etc. depending on their type (for integer 0, for string null).

Example of default constructor that displays the default values

```
class Student
{
int roll;
float marks;
String name;
void show()
{
System.out.println("Roll: "+roll);
System.out.println("Marks: "+marks);
System.out.println("Name: "+name);
}
}
class TestDemo
{
public static void main(String [] args)
{
Student s1=new Student();
s1.show();
}
}
```

Output

Roll: 0

Marks: 0.0

Name: null

Explanation: In the above class, we are not creating any constructor so compiler provides a default constructor. Here 0, 0.0 and null values are provided by default constructor.

Parameterized constructor

If any constructor contain list of variable in its signature is known as parameterized constructor. A parameterized constructor is one which takes some parameters.

Syntax

```
class ClassName
{
.....
ClassName(list of parameters) //parameterized constructor
{
.....
}
.....
}
```

Syntax to call parameterized constructor

```
ClassName objref=new ClassName(value1, value2,.....);
```

OR

```
new ClassName(value1, value2,.....);
```

Example of Parameterized Constructor

```
class Test
{
```



```

int a, b;
Test(int n1, int n2)
{
System.out.println("I am from Parameterized Constructor...");
a=n1;
b=n2;
System.out.println("Value of a = "+a);
System.out.println("Value of b = "+b);
}
}
class TestDemo1
{
public static void main(String k [])
{
Test t1=new Test(10, 20);
}
}

```

Important points Related to Parameterized Constructor

- Whenever we create an object using parameterized constructor, it must define parameterized constructor otherwise we will get compile time error. Whenever we define the objects with respect to both parameterized constructor and default constructor, it must define both the constructors.
- In any class maximum one default constructor but 'n' number of parameterized constructors.

Difference between Static and non-Static Variable in Java

The variable of any class are classified into two types;

- Static or class variable
- Non-static or instance variable

Static variable in Java

Memory for static variable is created only one in the program at the time of loading of class. These variables are preceded by static keyword. Static variable can access with class reference.

Non-static variable in Java

Memory for non-static variable is created at the time of creation an object of class. These variables should not be preceded by any static keyword. These variables can access with object reference.

Difference between non-static and static variable

	Non-static variable	Static variable
1	These variable should not be preceded by any static keyword Example: class A { int a; }	These variables are preceded by static keyword. Example class A { static int b; }
2	Memory is allocated for these variable whenever an object is created	Memory is allocated for these variables at the time of loading of the class.
3	Memory is allocated multiple times	Memory is allocated for these variable only once

	whenever a new object is created.	in the program.
4	Non-static variable also known as instance variable while because memory is allocated whenever instance is created.	Memory is allocated at the time of loading of class so that these are also known as class variable.
5	Non-static variable are specific to an object	Static variable are common for every object that means there memory location can be sharable by every object reference or same class.
6	Non-static variable can access with object reference. Syntax obj_ref.variable_name	Static variable can access with class reference. Syntax class_name.variable_name

Note: static variable not only can be access with class reference but also some time it can be accessed with object reference.

Example of static and non-static variable.

Example

```
class Student
```

```
{
```

```
int roll_no;
```

```
float marks;
```

```
String name;
```

```
static String College_Name="ITM";
```

```
}
```

```
class StaticDemo
```

```
{
```

```
public static void main(String args[])
```

```
{
```

```
Student s1=new Student();
```

```
s1.roll_no=100;
```

```
s1.marks=65.8f;
```

```
s1.name="abcd";
```

```
System.out.println(s1.roll_no);
```

```
System.out.println(s1.marks);
```

```
System.out.println(s1.name);
```

```
System.out.println(Student.College_Name);
```

```
//or System.out.println(s1. College_Name); but first is use in real time.
```

```
Student s2=new Student();
```

```
s2.roll_no=200;
```

```
s2.marks=75.8f;
```

```
s2.name="zyx";
```

```
System.out.println(s2.roll_no);
```

```
System.out.println(s2.marks);
```

```
System.out.println(s2.name);
```

```
System.out.println(Student.College_Name);
```

```
}
```

```
}
```

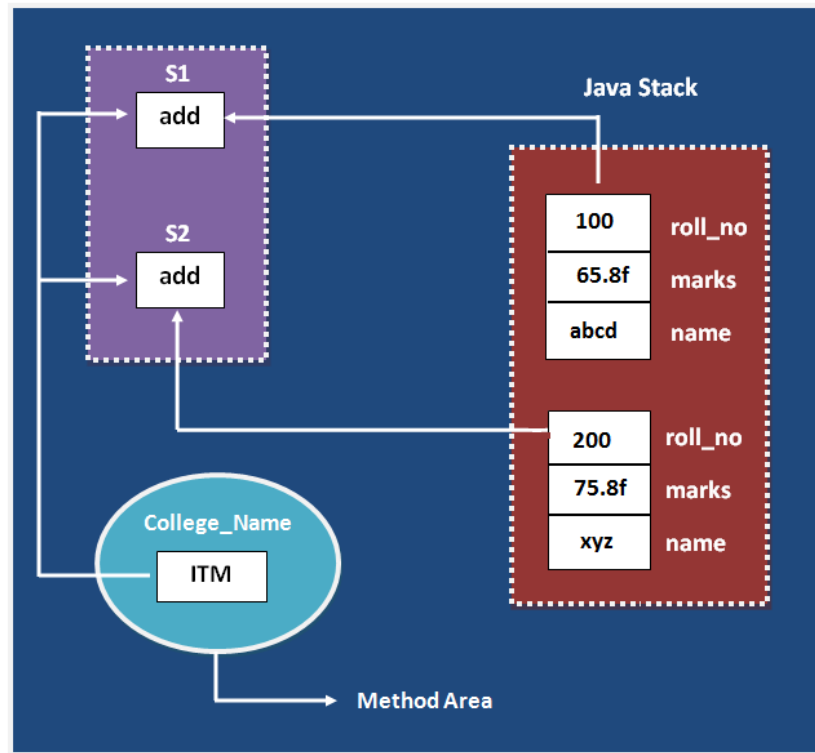
Output

```
100
```

```
65.8
```

```
abcd
```

ITM
200
75.8
zyx
ITM



Note: In the above example College_Name variable is commonly sharable by both S1 and S2 objects.

Understand static and non-static variable using counter

Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

Example

class Counter

```
{
int count=0;//will get memory when instance is created
Counter()
{
count++;
System.out.println(count);
}
public static void main(String args[])
{
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Output

1
1
1

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

Example

```
class Counter
```

```
{
static int count=0; //will get memory only once
Counter()
{
count++;
System.out.println(count);
}
public static void main(String args[])
{
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Output

1
2
3

Difference between Static and non-static method in Java

In case of non-static method memory is allocated multiple times whenever method is calling. But memory for static method is allocated only once at the time of class loading. Method of a class can be declared in two different ways

- Non-static methods
- Static methods

Difference between non-static and static Method

	Non-Static method	Static method
1	These method never be preceded by static keyword Example: void fun1() { }	These method always preceded by static keyword Example: static void fun2() { }
2	Memory is allocated multiple time whenever method is calling.	Memory is allocated only once at the time of class loading.
3	It is specific to an object so that these are also known as instance method.	These are common to every object so that it is also known as member method or class method.
4	These methods always access with object reference	These property always access with class reference

	Syntax: Objref.methodname();	Syntax: className.methodname();
5	If any method wants to be execute multiple time that can be declare as non static.	If any method wants to be execute only once in the program that can be declare as static .

Note: In some cases static methods not only can access with class reference but also can access with object reference.

Example of Static and non-Static Method

Example

```
class A
{
void fun1()
{
System.out.println("Hello I am Non-Static");
}
static void fun2()
{
System.out.println("Hello I am Static");
}
}
class Person
{
public static void main(String args[])
{
A oa=new A();
oa.fun1(); // non static method
A.fun2(); // static method
}
}
```

Output

Hello I am Non-Static

Hello I am Static

Following table represent how the static and non-static properties are accessed in the different static or non-static method of same class or other class.

	within non-static method of same class	within static method of same class	within non-static method of other class	within static method of other class
Non-Static Method or Variable	Access directly	Access with object reference	Access with object reference	Access with object reference
Static Method or Variable	Access directly	Access directly	Access with object reference	Access with object reference

Program to accessing static and non-static properties.

Example

```
class A
{
```

```

int y;
void f2()
{
System.out.println("Hello f2()");
}
}
class B
{
int z;
void f3()
{
System.out.println("Hello f3()");
A a1=new A();
a1.f2();
}
}
class Sdemo
{
static int x;
static void f1()
{
System.out.println("Hello f1()");
}
public static void main(String[] args)
{
x=10;
System.out.println("x="+x);
f1();
System.out.println("Hello main");
B b1=new B();
b1.f3();
}
}

```

Methods

What is a method?

In mathematics, you might have studied about functions. For example, $f(x) = x^2$ is a function that returns squared value of x .

If $x = 2$, then $f(2) = 4$

If $x = 3$, $f(3) = 9$

and so on.

Similarly, in programming, a function is a block of code that performs a specific task.

In object-oriented programming, method is a jargon used for function. Methods are bound to a class and they define the behavior of a class.

Types of Java methods

Depending on whether a method is defined by the user, or available in standard library, there are two types of methods:

- Standard Library Methods

- User-defined Methods

Standard Library Methods

The standard library methods are built-in methods in Java that are readily available for use. These standard libraries come along with the Java Class Library (JCL) in a Java archive (*.jar) file with JVM and JRE.

For example,

- `print()` is a method of `java.io.PrintStream`. The `print("...")` prints the string inside quotation marks.
- `sqrt()` is a method of `Math` class. It returns square root of a number.

Here's an working example:

```
public class Numbers {  
    public static void main(String[] args) {  
        System.out.print("Square root of 4 is: " + Math.sqrt(4));  
    }  
}
```

When you run the program, the output will be:

Square root of 4 is: 2.0

User-defined Method

You can also define methods inside a class as per your wish. Such methods are called user-defined methods.

How to create a user-defined method?

Before you can use (call a method), you need to define it.

Here is how you define methods in Java.

```
public static void myMethod() {  
    System.out.println("My Function called");  
}
```

Here, a method named `myMethod()` is defined.

The complete syntax for defining a Java method is:

```
modifier static returnType nameOfMethod (Parameter List) {  
    // method body  
}
```

Here,

- `modifier` - defines access type whether the method is public, private and so on.
- `static` - If you use `static` keyword in a method then it becomes a static method. Static methods can be called without creating an instance of a class.

For example, the `sqrt()` method of standard `Math` class is static. Hence, we can directly call `Math.sqrt()` without creating an instance of `Math` class.

- `returnType` - A method can return a value.

It can return native data types (int, float, double etc.), native objects (String, Map, List

etc.), or any other built-in and user defined objects.

If the method does not return a value, its return type is void.

- `nameOfMethod` - The name of the method is an identifier.

You can give any name to a method. However, it is more conventional to name it after the tasks it performs. For example, `calculateInterest`, `calculateArea`, and so on.

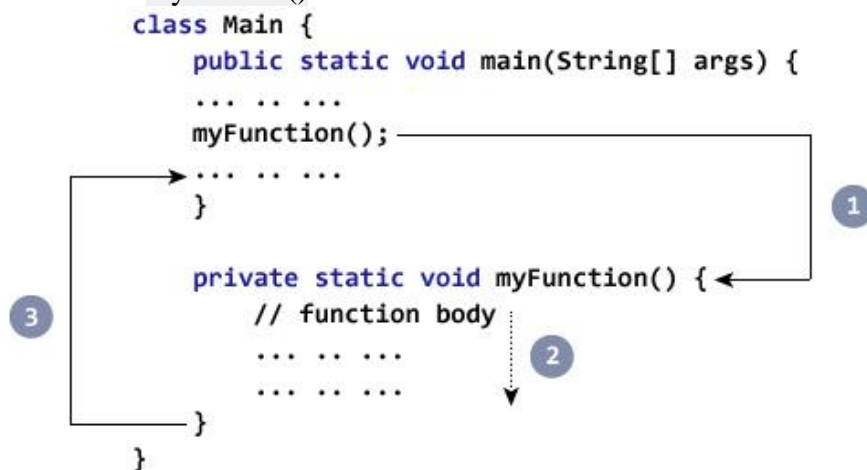
- **Parameters (arguments)** - Parameters are the values passed to a method. You can pass any number of arguments to a method.
- **Method body** - It defines what the method actually does, how the parameters are manipulated with programming statements and what values are returned. The codes inside curly braces `{ }` is the body of the method.

How to call a Java Method?

Now you defined a method, you need to use it. For that, you have to call the method. Here's how:

```
myMethod();
```

This statement calls the `myMethod()` method that was declared earlier.



1. While Java is executing the program code, it encounters `myMethod();` in the code.
2. The execution then branches to the `myFunction()` method, and executes code inside the body of the method.
3. After the codes execution inside the method body is completed, the program returns to the original state and executes the next statement.

Example: Complete Program of Java Method

Let's see a Java method in action by defining a Java class.

```
class Main {  
  
    public static void main(String[] args) {  
        System.out.println("About to encounter a method.");  
  
        // method call  
        myMethod();  
    }  
}
```



```

    System.out.println("Method was executed successfully!");
}

// method definition
private static void myMethod(){
    System.out.println("Printing from inside myMethod()!");
}
}

```

When you run the program, the output will be:

```

About to encounter a method.
Printing from inside myMethod().
Method was executed successfully!

```

The method `myMethod()` in the above program doesn't accept any arguments. Also, the method doesn't return any value (return type is `void`).

Note that, we called the method without creating object of the class. It was possible because `myMethod()` is static.

Here's another example. In this example, our method is non-static and is inside another class.

```

class Main {

    public static void main(String[] args) {

        Output obj = new Output();
        System.out.println("About to encounter a method.");

        // calling myMethod() of Output class
        obj.myMethod();

        System.out.println("Method was executed successfully!");
    }
}

class Output {

    // public: this method can be called from outside the class
    public void myMethod() {
        System.out.println("Printing from inside myMethod().");
    }
}

```

When you run the program, the output will be:

```

About to encounter a method.
Printing from inside myMethod().
Method was executed successfully!

```

Note that, we first created instance of *Output* class, then the method was called using *obj* object. This is because `myMethod()` is a non-static method.

Java Methods with Arguments and Return Value

A Java method can have zero or more parameters. And, they may return a value.

Example: Return Value from Method

Let's take an example of method returning a value.

```
class SquareMain {  
    public static void main(String[] args) {  
        int result;  
        result = square();  
        System.out.println("Squared value of 10 is: " + result);  
    }  
  
    public static int square() {  
        // return statement  
        return 10 * 10;  
    }  
}
```

When you run the program, the output will be:

Squared value of 10 is: 100

In the above code snippet, the method `square()` does not accept any arguments and always returns the value of 10 squared.

Notice, the return type of `square()` method is `int`. Meaning, the method returns an integer value.

```
class SquareMain {  
    public static void main(String[] args) {  
        ... ..  
        100 result = square();  
        ... ..  
    }  
  
    private static int square() {  
        // return statement  
        return 10*10;  
    }  
}
```

The diagram shows a call to `square()` in the `main` method. A solid arrow points from the `square()` call to the `square()` method definition. A dashed arrow points from the `return 10*10;` statement in the `square()` method to the value `100` in the `result = square();` line in the `main` method. The value `100` is highlighted in a blue box in both locations.

As you can see, the scope of this method is limited as it always returns the same value.

Now, let's modify the above code snippet so that instead of always returning the squared value of 10, it returns the squared value of any integer passed to the method.

Example: Method Accepting Arguments and Returning Value

```
public class SquareMain {  
  
    public static void main(String[] args) {  
        int result, n;  
  
        n = 3  
        result = square(n);  
        System.out.println("Square of 3 is: " + result);  
    }  
}
```

```

    n = 4
    result = square(n);
    System.out.println("Square of 4 is: " + result);
}

    static int square(int i) {
    return i * i;
    }
}

```

When you run the program, the output will be:

```

Squared value of 3 is: 9
Squared value of 4 is: 16

```

Now, the square() method returns the squared value of whatever integer value passed to it.

```

class SquareMain {
    public static void main(String[] args) {
        ... ..
        n = 3;
        9 result = square(n);
        ... ..
    }

    private static int square(int i) {
        // return statement
        return i*i;
    }
}

```

Java is a strongly-typed language. If you pass any other data type except `int` (in the above example), compiler will throw an error.

The argument passed `n` to the `getSquare()` method during the method call is called actual argument.

```
result = getSquare(n);
```

The parameter `i` accepts the passed arguments in the method definition `getSquare(int i)`. This is called formal argument (parameter). The type of the formal argument must be explicitly typed.

You can pass more than one argument to the Java method by using commas. For example,

```

public class ArithmeticMain {

    public static int getIntegerSum (int i, int j) {
        return i + j;
    }

    public static int multiplyInteger (int x, int y) {
        return x * y;
    }
}

```

```
public static void main(String[] args) {  
    System.out.println("10 + 20 = " + getIntegerSum(10, 20));  
    System.out.println("20 x 40 = " + multiplyInteger(20, 40));  
}  
}
```

When you run the program, the output will be:

```
10 + 20 = 30  
20 x 40 = 800
```

The data type of actual and formal arguments should match, i.e., the data type of first actual argument should match the type of first formal argument. Similarly, the type of second actual argument must match the type of second formal argument and so on.

Example: Get Squared Value of Numbers from 1 to 5

```
public class JMethods {  
  
    // method defined  
    private static int getSquare(int x){  
        return x * x;  
    }  
  
    public static void main(String[] args) {  
        for (int i = 1; i <= 5; i++) {  
  
            // method call  
            result = getSquare(i)  
            System.out.println("Square of " + i + " is : " + result); }  
        }  
    }  
}
```

When you run the program, the output will be:

```
Square of 1 is : 1  
Square of 2 is : 4  
Square of 3 is : 9  
Square of 4 is : 16  
Square of 5 is : 25
```

In above code snippet, the method `getSquare()` takes `int` as a parameter. Based on the argument passed, the method returns the squared value of it.

Here, argument `i` of type `int` is passed to the `getSquare()` method during method call.

```
result = getSquare(i);
```

The parameter `x` accepts the passed argument [in the function definition `getSquare(int x)`]. `return i * i;` is the return statement. The code returns a value to the calling method and terminates the function.

Did you notice, we reused the `getSquare` method 5 times?

What are the advantages of using methods?

1. The main advantage is code reusability. You can write a method once, and use it multiple times. You do not have to rewrite the entire code each time. Think of it as, "write once, reuse multiple times."
- Methods make code more readable and easier to debug. For example, `getSalaryInformation()` method is so readable, that we can know what this method will be doing than actually reading the lines of code that make this method.

Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking the number of parameters, and their type.

In other words whenever same constructor is existing multiple times in the same class with different number of parameters or order of parameters or type of parameters is known as Constructor overloading.

In general constructor overloading can be used to initialize same or different objects with different values.

Syntax

```
class ClassName
{
    ClassName()
    {
        .....
        .....
    }
    ClassName(datatype1 value1)
    {.....}
    ClassName(datatype1 value1, datatype2 value2)
    {.....}
    ClassName(datatype2 variable2)
    {.....}
    ClassName(datatype2 value2, datatype1 value1)
    {.....}
    .....
}
```

Why overriding is not possible at constructor level.

The scope of constructor is within the class so that it is not possible to achieved overriding at constructor level.

Example of default constructor, parameterized constructor and overloaded constructor

Example

```
class Test
{
    int a, b;
    Test ()
    {
```

```

System.out.println("I am from default Constructor...");
a=1;
b=2;
System.out.println("Value of a =" +a);
System.out.println("Value of b =" +b);
}
Test (int x, int y)
{
System.out.println("I am from double Parameterized Constructor");
a=x;
b=y;
System.out.println("Value of a =" +a);
System.out.println("Value of b =" +b);
}
Test (int x)
{
System.out.println("I am from single Parameterized Constructor");
a=x;
b=x;
System.out.println("Value of a =" +a);
System.out.println("Value of b =" +b);
}
Test (Test T)
{
System.out.println("I am from Object Parameterized Constructor...");
a=T.a;
b=T.b;
System.out.println("Value of a =" +a);
System.out.println("Value of b =" +b);
}
}
class TestDemo2
{
public static void main (String k [])
{
Test t1=new Test ();
Test t2=new Test (10, 20);
Test t3=new Test (1000);
Test t4=new Test (t1);
}
}

```

Note By default the parameter passing mechanism is call by reference.

Method Overloading

In Java, two or more methods can have same name if they differ in parameters (different number of parameters, different types of parameters, or both). These methods are called overloaded methods and this feature is called method overloading. For example:

```
void func() { ... }
```

```
void func(int a) { ... }  
float func(double a) { ... }  
float func(int a, float b) { ... }
```

Here, func() method is overloaded. These methods have same name but accept different arguments.

Notice that, the return type of these methods is not same. Overloaded methods may or may not have different return type, but they must differ in parameters they accept.

Why method overloading?

Suppose, you have to perform addition of the given numbers but there can be any number of arguments (let's say either 2 or 3 arguments for simplicity).

In order to accomplish the task, you can create two methods `sum2num(int, int)` and `sum3num(int, int, int)` for two and three parameters respectively. However, other programmers as well as you in future may get confused as the behavior of both methods is same but they differ by name.

The better way to accomplish this task is by overloading methods. And, depending upon the argument passed, one of the overloaded methods is called. This helps to increase readability of the program.

How to perform method overloading in Java?

Here are different ways to perform method overloading:

1. Overloading by changing number of arguments

```
class MethodOverloading {  
    private static void display(int a){  
        System.out.println("Arguments: " + a);  
    }  
  
    private static void display(int a, int b){  
        System.out.println("Arguments: " + a + " and " + b);  
    }  
  
    public static void main(String[] args) {  
        display(1);  
        display(1, 4);  
    }  
}
```

When you run the program, the output will be:

```
Arguments: 1  
Arguments: 1 and 4
```

2. By changing the datatype of parameters

```
class MethodOverloading {  
  
    // this method accepts int  
    private static void display(int a){  
        System.out.println("Got Integer data.");  
    }  
}
```

```

    }

    // this method accepts String object
    private static void display(String a){
        System.out.println("Got String object.");
    }

    public static void main(String[] args) {
        display(1);
        display("Hello");
    }
}

```

When you run the program, the output will be:

```

Got Integer data.
Got String object.

```

Here, both overloaded methods accept one argument. However, one accepts argument of type `int` whereas other accepts `String` object.

Let's look at a real world example:

```

class HelperService {

    private String formatNumber(int value) {
        return String.format("%d", value);
    }

    private String formatNumber(double value) {
        return String.format("%.3f", value);
    }

    private String formatNumber(String value) {
        return String.format("%.2f", Double.parseDouble(value));
    }

    public static void main(String[] args) {
        HelperService hs = new HelperService();
        System.out.println(hs.formatNumber(500));
        System.out.println(hs.formatNumber(89.9934));
        System.out.println(hs.formatNumber("550"));
    }
}

```

When you run the program, the output will be:

```

500
89.993
550.00

```

Important Points

- Two or more methods can have same name inside the same class if they accept different arguments. This feature is known as method overloading.

- Method overloading is achieved by either:
 - changing the number of arguments.
 - or changing the datatype of arguments.
- Method overloading is not possible by changing the return type of methods.

This keyword

this is a reference variable that refers to the current object. It is a keyword in java language represents current class object

Usage of this keyword

- It can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- It can be used to invoke current class method (implicitly)
- It can be passed as an argument in the method call.
- It can be passed as argument in the constructor call.
- It can also be used to return the current class instance.

Why use this keyword in java ?

The main purpose of using this keyword is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

"this" keyword can be use in two ways.

- this . (this dot)
- this() (this off)

this . (this dot)

which can be used to differentiate variable of class and formal parameters of method or constructor.

"this" keyword are used for two purpose, they are

- It always points to current class object.
- Whenever the formal parameter and data member of the class are similar and JVM gets an ambiguity (no clarity between formal parameter and data members of the class).

To differentiate between formal parameter and data member of the class, the data members of the class must be preceded by "this".

Syntax

this.data member of current class.

Note: If any variable is preceded by "this" JVM treated that variable as class variable.

Example without using this keyword

class Employee

```
{
    int id;
    String name;

    Employee(int id,String name)
    {
        id = id;
        name = name;
    }
    void show()
```

```

{
    System.out.println(id+" "+name);
}
public static void main(String args[])
{
    Employee e1 = new Employee(111,"Harry");
    Employee e2 = new Employee(112,"Jacy");
    e1.show();
    e2.show();
}
}

```

Output

0 null

0 null

In the above example, parameter (formal arguments) and instance variables are same that is why we are using "this" keyword to distinguish between local variable and instance variable.

Example of this keyword in java

class Employee

```

{
    int id;
    String name;

    Employee(int id,String name)
    {
        this.id = id;
        this.name = name;
    }
    void show()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[])
    {
        Employee e1 = new Employee(111,"Harry");
        Employee e2 = new Employee(112,"Jacy");
        e1.show();
        e2.show();
    }
}

```

Output

111 Harry

112 Jacy

Note 1: The scope of "this" keyword is within the class.

Note 2: The main purpose of using "this" keyword in real life application is to differentiate variable of class or formal parameters of methods or constructor (it is highly recommended to use the same variable name either in a class or method and constructor while working with similar objects).

this ()

which can be used to call one constructor within the another constructor without creation of objects multiple time for the same class.

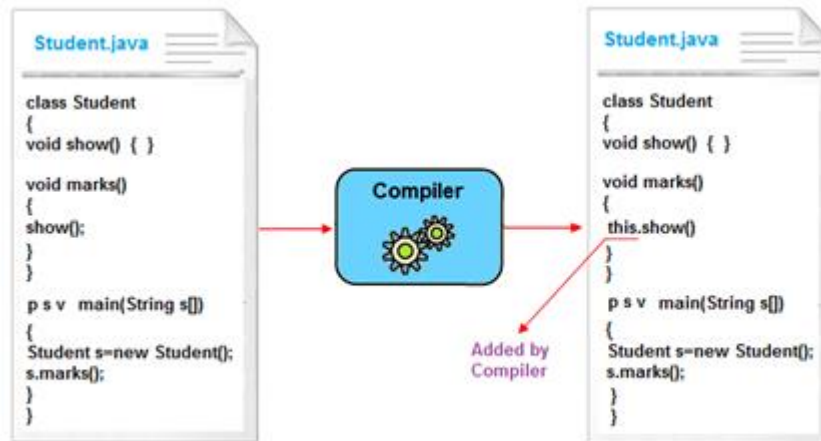
Syntax

this(); // call no parametrized or default constructor

this(value1,value2,.....) //call parametrize constructor

this keyword used to invoke current class method (implicitly)

By using this keyword you can invoke the method of the current class. If you do not use the this keyword, compiler automatically adds this keyword at time of invoking of the method.



Example of this keyword

class Student

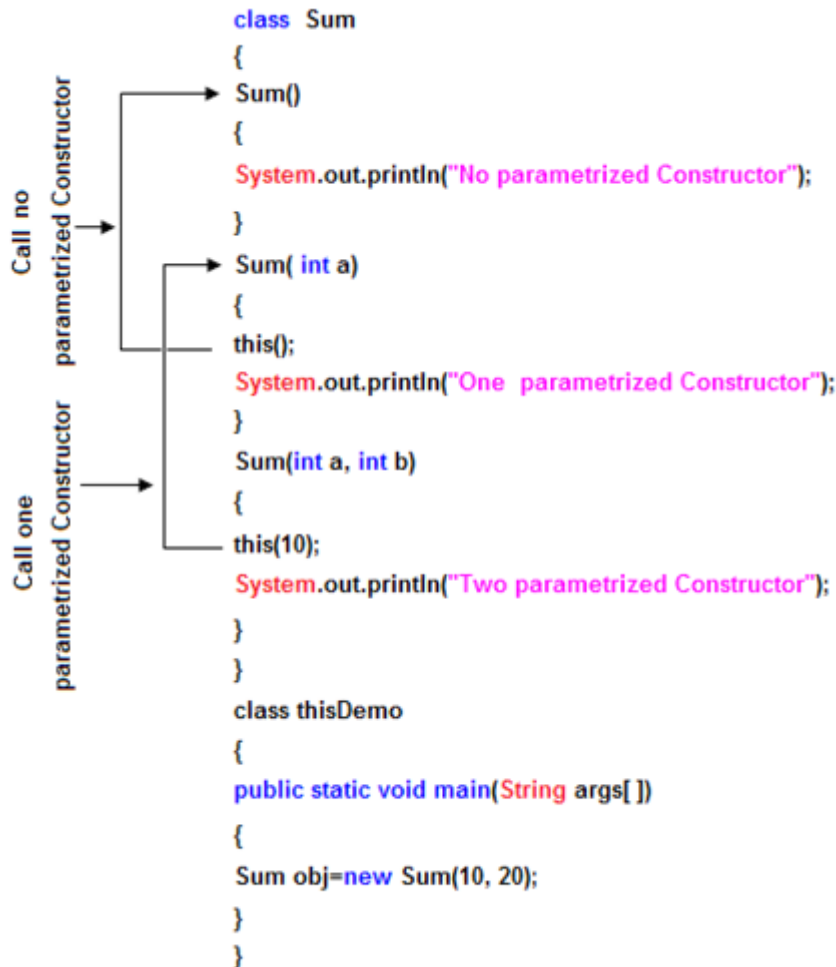
```
{
    void show()
    {
        System.out.println("You got A+");
    }
    void marks()
    {
        this.show(); //no need to use this here because compiler does it.
    }
    void display()
    {
        show(); //compiler act marks() as this.marks()
    }
    public static void main(String args[])
    {
        Student s = new Student();
        s.display();
    }
}
```

Syntax

You got A+

Rules to use this()

this() always should be the first statement of the constructor. One constructor can call only other single constructor at a time by using this().



Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using `free ()` function in C language and `delete ()` in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector (a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

- 1) By nulling a reference:
Employee e=new Employee ();
e=null;
- 2) By assigning a reference to another:
Employee e1=new Employee ();
Employee e2=new Employee ();
e1=e2; //now the first object referred by e1 is available for garbage collection
- 3) By anonymous object:
new Employee ();

finalize () method

The finalize () method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:
protected void finalize(){ }

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.
public static void gc(){ }

Note: Garbage collection is performed by a daemon thread called Garbage Collector (GC). This thread calls the finalize () method before object is garbage collected.

Simple Example of garbage collection in java

```
public class TestGarbage1{
    public void finalize(){System.out.println("object is garbage collected");}
    public static void main(String args[]){
        TestGarbage1 s1=new TestGarbage1();
        TestGarbage1 s2=new TestGarbage1();
        s1=null;
        s2=null;
        System.gc();
    }
}
```

object is garbage collected
object is garbage collected

Note: Neither finalization nor garbage collection is guaranteed.

String Class

What is String in java?

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters.

The java.lang.String class is used to create string object.

How to create String object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes.

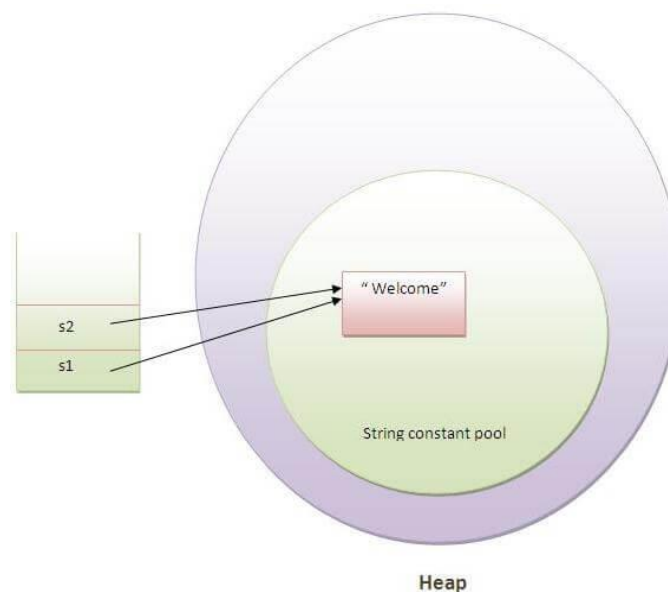
For Example:

```
String s="welcome";
```

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned.

If string doesn't exist in the pool, a new string instance is created and placed in the pool.
For example:

```
String s1="Welcome";  
String s2="Welcome";//will not create new instance
```



In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as string constant pool.

Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

2) By new keyword

```
String s=new String("Welcome");
```

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

String class is immutable that means whose content cannot be changed at the time of execution of program.

String class object is immutable that means when we create an object of String class it never changes in the existing object.

Example:

```
class StringHandling
{
public static void main(String arg[])
{
String s=new String("java");
s.concat("software");
System.out.println(s);
}
}
```

Output
java

Explanation: Here we cannot change the object of String class so output is only java not java software.

Methods of String class

length()

length(): This method is used to get the number of characters of any string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
int l;
String s=new String("Java");
l=s.length();
System.out.println("Length: "+l);
}
}
```

Output

Length: 4

charAt(index)

charAt(): This method is used to get the character at a given index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
char c;
String s=new String("Java");
c=s.charAt(2);
System.out.println("Character: "+c);
}
}
```

Output

Character: v

toUpperCase()

toUpperCase(): This method is use to convert lower case string into upper case.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java";
System.out.println("String: "+s.toUpperCase());
}
}
```

Output

String: JAVA

toLowerCase()

toLowerCase(): This method is used to convert upper case string into lower case.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="JAVA";
System.out.println("String: "+s.toLowerCase());
}
}
```

Output

String: java

concat()

concat(): This method is used to combine two strings.

Example

```
class StringHandling
```



```

{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Reddy";
System.out.println("Combined String: "+s1.concat(s2));
}
}

```

Output

Combined String: HiteshReddy

startsWith()

startsWith(): This method returns true if string starts with given another string, otherwise it returns false.

Example

```

class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.startsWith("Java"));
}
}

```

Output

true

endsWith()

endsWith(): This method returns true if string ends with given another string, otherwise it returns false.

Example

```

class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.endsWith("language"));
}
}

```

Output

true

substring()

substring(): This method is used to get the part of given string.

Example

```

class StringHandling
{
public static void main(String arg[])
{

```

```
String s="Java is programming language";
System.out.println(s.substring(8)); // 8 is starting index
}
}
Output
programming language
```

Example2

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.substring(8, 12));
}
}
Output
prog
```

indexOf()

indexOf(): This method is used find the index value of given string. It always gives starting index value of first occurrence of string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.indexOf("programming"));
}
}
Output
8
```

lastIndexOf()

lastIndexOf(): This method used to return the starting index value of last occurrence of the given string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Java is programming language";
String s2="Java is programming language programming language";
System.out.println(s1.indexOf("programming"));
System.out.println(s2.lastIndexOf("programming"));
}
}
```

Output

8

29

trim()

trim(): This method remove space which are available before starting of string and after ending of string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="  Java is programming language  ";
System.out.println(s.trim());
}
}
```

Output

Java is programming language

split()

split(): This method is used to divide the given string into number of parts based on delimiter (special symbols like @ space ,).

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="contact@srecwarangal.ac.in";
String[] s1=s.split("@"); // divide string based on @
for(String c:s1) // foreach loop
{
System.out.println(c);
}
}
}
```

Output

contact

@srecwarangal.ac.in

replace()

replace(): This method is used to return a duplicate string by replacing old character with new character.

Note: In this method data of original string will never be modify.

Example

```
class StringHandling
{
public static void main(String arg[])
```

```
{  
String s1="java";  
String s2=s1.replace('j', 'k');  
System.out.println(s2);  
}  
}
```

Output
kava

String Comparison

String comparison can be done in 3 ways.

1. Using equals() method
2. Using == operator
3. By CompareTo() method

Using equals() method

equals() method compares two strings for equality. Its general syntax is,

boolean equals (Object str)

It compares the content of the strings. It will return true if string matches, else returns false.

```
String s = "Hell";  
String s1 = "Hello";  
String s2 = "Hello";  
s1.equals(s2); //true  
s.equals(s1) ; //false
```

Using == operator

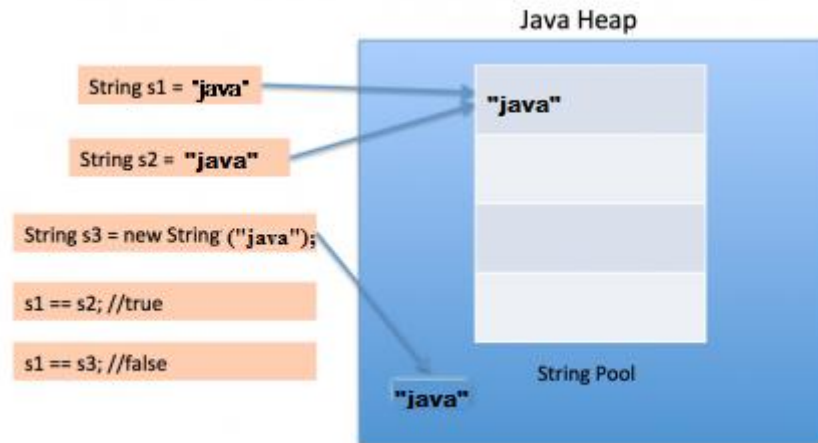
== operator compares two object references to check whether they refer to same instance. This also, will return true on successful match.

```
String s1 = "Java";  
String s2 = "Java";  
String s3 = new String ("Java");  
test(s1 == s2) //true  
test(s1 == s3) //false
```

Reason:

Its because we are creating a new object using new operator, and thus it gets created in a non-pool memory area of the heap. s1 is pointing to the String in string pool while s3 is pointing to the String in heap and hence, when we compare s1 and s3, the answer is false.

The following image will explain it more clearly.



By compareTo() method

compareTo() method compares values and returns an int which tells if the string compared is less than, equal to or greater than the other string. It compares the String based on natural ordering i.e alphabetically. Its general syntax is,

```
int compareTo(String str)
```

```
String s1 = "Abhi";
String s2 = "Viraaaj";
String s3 = "Abhi";
s1.compareTo(s2); //return -1 because s1 < s2
s1.compareTo(s3); //return 0 because s1 == s3
s2.compareTo(s1); //return 1 because s2 > s1
```

equalsIgnoreCase()

equalsIgnoreCase(): This method is case insensitive method, It return true if the contents of both strings are same otherwise false.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="HITESH";
String s3="Raddy";
System.out.println("Compare String: "+s1.equalsIgnoreCase(s2));
System.out.println("Compare String: "+s1.equalsIgnoreCase(s3));
}
}
```

Output

Compare String: true

Compare String: false

compareToIgnoreCase()

compareToIgnoreCase(): This method is case insensitive method, which is used to compare two strings similar to compareTo().

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="HITESH";
int i;
i=s1.compareToIgnoreCase(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

Strings are same

StringBuffer class

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Important Constructors of StringBuffer class

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

Important methods of StringBuffer class

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public synchronized StringBuffer	insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int,

		boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	is used to return the character at the specified position.
public int	length()	is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

```
class StringBufferExample{
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer("Hello ");
        sb.append("Java");//now original string is changed
        System.out.println(sb);//prints Hello Java
    }
}
```

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

```
class StringBufferExample2{
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer("Hello ");
        sb.insert(1,"Java");//now original string is changed
        System.out.println(sb);//prints HJavaello
    }
}
```

```
}  
}
```

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBufferExample3{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello");  
        sb.replace(1,3,"Java");  
        System.out.println(sb);//prints HJavallo  
    }  
}
```

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class StringBufferExample4{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello");  
        sb.delete(1,3);  
        System.out.println(sb);//prints Hlo  
    }  
}
```

5) StringBuffer reverse() method

The reverse() method of StringBuiler class reverses the current string.

```
class StringBufferExample5{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello");  
        sb.reverse();  
        System.out.println(sb);//prints olleH  
    }  
}
```

6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by $(oldcapacity * 2) + 2$. For example if your current capacity is 16, it will be $(16 * 2) + 2 = 34$.

```
class StringBufferExample6{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer();  
        System.out.println(sb.capacity());//default 16  
        sb.append("Hello");  
        System.out.println(sb.capacity());//now 16  
        sb.append("java is my favourite language");  
    }  
}
```



```

System.out.println(sb.capacity()); //now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}

```

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```

class StringBufferExample7{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity()); //default 16
sb.append("Hello");
System.out.println(sb.capacity()); //now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity()); //now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10); //now no change
System.out.println(sb.capacity()); //now 34
sb.ensureCapacity(50); //now (34*2)+2
System.out.println(sb.capacity()); //now 70
}
}

```

StringBuilder class

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Important Constructors of StringBuilder class

Constructor	Description
StringBuilder()	creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str)	creates a string Builder with the specified string.
StringBuilder(int length)	creates an empty string Builder with the specified capacity as length.

Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

<code>public StringBuilder replace(int startIndex, int endIndex, String str)</code>	is used to replace the string from specified startIndex and endIndex.
<code>public StringBuilder delete(int startIndex, int endIndex)</code>	is used to delete the string from specified startIndex and endIndex.
<code>public StringBuilder reverse()</code>	is used to reverse the string.
<code>public int capacity()</code>	is used to return the current capacity.
<code>public void ensureCapacity(int minimumCapacity)</code>	is used to ensure the capacity at least equal to the given minimum.
<code>public char charAt(int index)</code>	is used to return the character at the specified position.
<code>public int length()</code>	is used to return the length of the string i.e. total number of characters.
<code>public String substring(int beginIndex)</code>	is used to return the substring from the specified beginIndex.
<code>public String substring(int beginIndex, int endIndex)</code>	is used to return the substring from the specified beginIndex and endIndex.

Java StringBuilder Examples

Let's see the examples of different methods of StringBuilder class.

1) StringBuilder append() method

The StringBuilder append() method concatenates the given argument with this string.

```
class StringBuilderExample{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello ");
        sb.append("Java");//now original string is changed
        System.out.println(sb);//prints Hello Java
    }
}
```

2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

```
class StringBuilderExample2{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello ");
        sb.insert(1,"Java");//now original string is changed
        System.out.println(sb);//prints HJavaello
    }
}
```

3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBuilderExample3{
```

```

public static void main(String args[]){
    StringBuilder sb=new StringBuilder("Hello");
    sb.replace(1,3,"Java");
    System.out.println(sb);//prints HJavallo
}
}

```

4) StringBuilder delete() method

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

```

class StringBuilderExample4{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello");
        sb.delete(1,3);
        System.out.println(sb);//prints Hlo
    }
}

```

5) StringBuilder reverse() method

The reverse() method of StringBuilder class reverses the current string.

```

class StringBuilderExample5{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello");
        sb.reverse();
        System.out.println(sb);//prints olleH
    }
}

```

6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by $(oldcapacity * 2) + 2$. For example if your current capacity is 16, it will be $(16 * 2) + 2 = 34$.

```

class StringBuilderExample6{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder();
        System.out.println(sb.capacity());//default 16
        sb.append("Hello");
        System.out.println(sb.capacity());//now 16
        sb.append("java is my favourite language");
        System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
    }
}

```

7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the

capacity by $(oldcapacity * 2) + 2$. For example if your current capacity is 16, it will be $(16 * 2) + 2 = 34$.

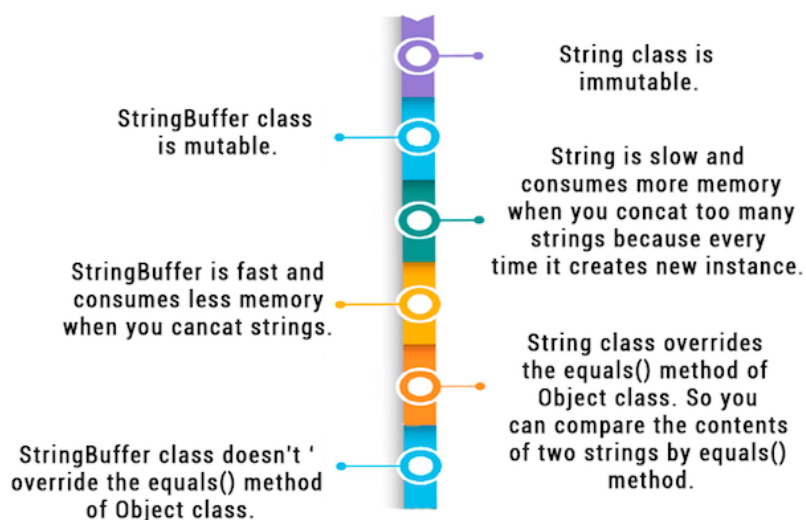
```
class StringBuilderExample7{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder();
        System.out.println(sb.capacity());//default 16
        sb.append("Hello");
        System.out.println(sb.capacity());//now 16
        sb.append("java is my favourite language");
        System.out.println(sb.capacity());//now  $(16 * 2) + 2 = 34$  i.e  $(oldcapacity * 2) + 2$ 
        sb.ensureCapacity(10);//now no change
        System.out.println(sb.capacity());//now 34
        sb.ensureCapacity(50);//now  $(34 * 2) + 2$ 
        System.out.println(sb.capacity());//now 70
    }
}
```

Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you concat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

StringBuffer vs String

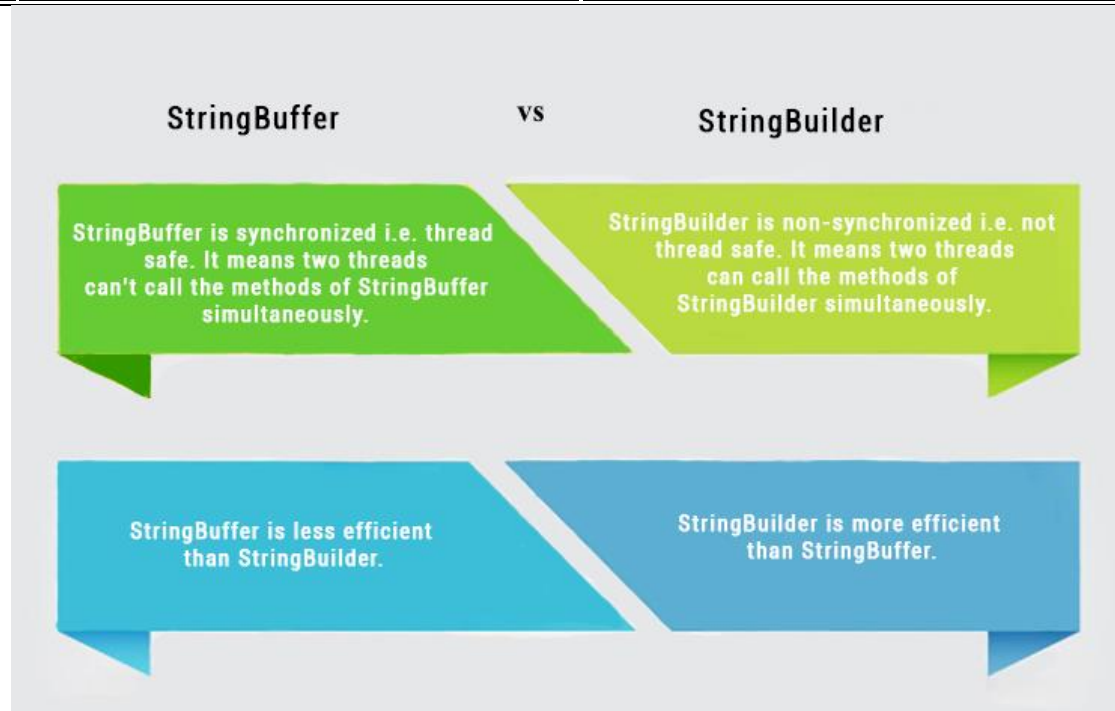


Difference between StringBuffer and StringBuilder

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.

A list of differences between StringBuffer and StringBuilder are given below:

No.	StringBuffer	StringBuilder
1)	StringBuffer is <i>synchronized</i> i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is <i>non-synchronized</i> i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is <i>less efficient</i> than StringBuilder.	StringBuilder is <i>more efficient</i> than StringBuffer.



Fill in the Blanks

1. JVM is an interpreter for_____.
2. Java was conceived by_____,_____,_____.
3. _____ is the mechanism that binds together code & data.
4. One entity behaving differently in different situations is known as_____.
5. Java defines ____ number of types for Commenting.
6. Java's Char Data Type follows_____ Code.
7. Class is a_____ of an Object.
8. Object is a _____ of a Class
9. Method Signature does not include _____
10. Constructor can be called _____ number of times in its Lifetime.
11. Method signature includes _____&_____.
12. Constructor will be called at the time of _____
13. A final variable must be _____ when it is declared.
14. _____is the process of defining something in terms of itself.
15. _____ keyword must be preceded by finalize() method
16. A_____ initializes an object immediately upon creation.

17. The _____ operator dynamically allocates memory for an object and returns a reference to it.
18. _____ statement terminates a statement sequence in a switch statement.
19. The _____ statement is used to explicitly return from a method.
20. The _____ statement is Java's multiway branch statement.

Short Answer Questions		
S. No	Questions	Blooms Taxonomy Level
1	State importance of Object Oriented Programming.	Understand
2	Distinguish between procedural language and OOPs.	Remember
3	Define Encapsulation.	Understand
4	Describe Inheritance.	Remember
5	Define Polymorphism.	Understand
6	List advantages of Object Oriented Programming.	Remember
7	List disadvantages of Object Oriented Programming.	Remember
8	Describe history of java.	Remember
9	List different data types used in java.	Remember
10	Define object with example.	Understand
11	Describe scope and life time of variables.	Remember
12	List and describe different types of operators.	Remember
13	Illustrate different access modifiers in java.	Understand
14	State the need of type casting.	Remember
15	Define enumerated types.	Understand
16	Describe class with real time entities as example.	Remember
17	State the use of this reference.	Remember
18	Describe the constructor.	Understand
19	Define recursion.	Understand
20	State the use of garbage collector.	Remember
Long Answer Questions		
1	Discuss the various characteristics of object oriented programming concepts.	Understand
2	Explain briefly about the features (buzzwords) of Java.	Understand
3	Discuss various Differences between Java and C++.	Understand
4	Describe java is a pure object oriented programming language	Remember
5	Distinguish between applications and applets in Java?	Understand
6	Explain the importance of this keyword with an example.	Understand
7	Interpret method overloading with an example.	Understand
8	Discuss about the constructor overloading with an example.	Understand
9	Explain the concept of arrays with an example.	Understand
10	Explain briefly about String class and discuss various methods in string class with an example.	Understand
11	Illustrate about the java inbuilt functions to accept console input and output.	Remember
12	Discuss about various conditional statements in java with suitable	Understand

	examples	
13	Explain about different loop structures in java with an example.	Understand
14	Describe the use of break and continue statements in java program	Understand
15	Discuss about the operator hierarchy with an example.	Understand
16	Illustrate the use of the operators in java and explain with an example.	Remember
17	Describe about static variable with an example.	Understand
18	Describe static method with an example.	Understand
19	Interpret type conversion and casting with an example.	Understand
20	Explain about for each loop with an example	Understand
Problem Solving and Critical Thinking Questions		
1	Predict the output of the code? Student john12 = new Student(1001, "John", 12); Student john13 = new Student(1002, "John", 13); System.out.println("comparing John, 12 and John, 13 with compareTo : " + john12.compareTo(john13));	Understand
2	Interpret the output of the program. class Lifetime { public static void main(String args[]) { int x; for (x=0; x<3; x++) { int y=-1; System.out.println(" y is : " + y); y=100; System.out.println(" y is now : " + y); } } }	Understand
3	Predict output of the program. public class If2 { static boolean b1, b2; public static void main(String [] args) { int x = 0; if (!b1) { if (!b2) { b1 = true; x++; if (5 > 6) x++; if (!b1) x = x + 10; } } else if (b2 = true) x	Understand

	<pre> = x + 100; else if (b1 b2) x = x + 1000; } } System.out.println(x); } } </pre>	
4	<p>Explain the following code is valid or not.</p> <pre> public String getDescription(Object obj) { return obj.toString; } public String getDescription(String obj) { return obj; } public void getDescription(String obj) { return obj; } </pre>	Understand
5	<p>Predict the output of following program?</p> <pre> public class Test { public int aMethod() { static int i = 0; i++; return i; } public static void main(String args[]) { Test test = new Test(); test.aMethod(); int j = test.aMethod(); System.out.println(j); } } </pre>	Understand
6	<p>Identify output of the program?</p> <pre> public class Test { public static void main(String args[]) { int i =1,j = 0; switch(i) { case 2: j += 6; case 4: j += 1; </pre>	Understand

	<pre> default: j += 2; case 0: j += 4; } System.out.println("j = " + j); } </pre>	
7	<p>Analyze the following program output.</p> <p>Class Test</p> <pre> { public static void main(String args[]) { int x, y; y=20; for(x=0; x<10: x++) { System.out.println("this is x:" + x); System.out.println("this is y:" + y); y= y-2; } } } </pre>	Understand
8	<p>Identify output of the program?</p> <pre> class BitShift { public static void main(String [] args) { int x = 0x80000000; System.out.print(x + " and "); x = x >>> 31; System.out.println(x); } } </pre>	Understand
9	<p>Analyze the program and find out the output.</p> <pre> class Equals { public static void main(String [] args) { int x = 100; double y = 100.1; boolean b = (x = y); System.out.println(b); } } </pre>	Remember

Review Questions

1. What is the most important feature of Java?
2. What do you mean by platform independence?
3. What is a JVM?
4. Are JVM's platform independent?
5. What is the difference between a JDK and a JVM?

6. What is a pointer and does Java support pointers?
7. Is Java a pure object oriented language?
8. Are arrays primitive data types?
9. What is difference between Path and Classpath?
10. What are local variables?
11. What are instance variables?
12. Should a main() method be compulsorily declared in all java classes?
13. What is the return type of the main() method?
14. Why is the main() method declared static?
15. What is the argument of main() method?
16. Can a main() method be overloaded?
17. Does the order of public and static declaration matter in main() method?
18. Can a source file contain more than one class declaration?
19. Can a class be declared as static?
20. When will you define a method as static?
21. What are the restriction imposed on a static method or a static block of code?
22. I want to print "Hello" even before main() is executed. How will you achieve that?
23. What is the importance of static variable?
24. Can we declare a static variable inside a method?

UNIT II

UNIT WISE PLAN

UNIT-II: Inheritance, Packages and Interfaces		Planned Hours: 09	
S. No.	Topic Learning Outcomes	Cos	Blooms Levels
1	Demonstrate the implementation of inheritance (multilevel, hierarchical and multiple) by using extend and implement keywords.	CO3	L2
2	Describe the concept of interface and abstract classes to define generic classes.	CO2,CO3	L2
3	Use dynamic and static polymorphism to process objects depending on their class.	CO3	L3
4	Illustrate different techniques on creating and accessing packages (fully qualified name and import statements).	CO3	L3

Inheritance in Java

The process of obtaining the data members and methods from one class to another class is known as inheritance. It is one of the fundamental features of object-oriented programming.

Important points

- In the inheritance the class which gives the data members and the methods is known as base or super or parent class.
- The class which is taking the data members and the methods is known as sub or derived or child class.
- The data members and methods of a class are known as features.
- The concept of inheritance is also known as re-usability or extendable classes or subclassing or derivation.

Why use Inheritance?

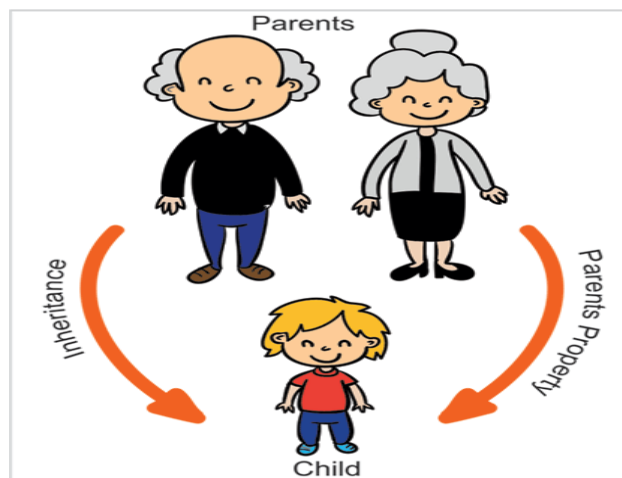
- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
- For code Re-usability

Syntax of Inheritance

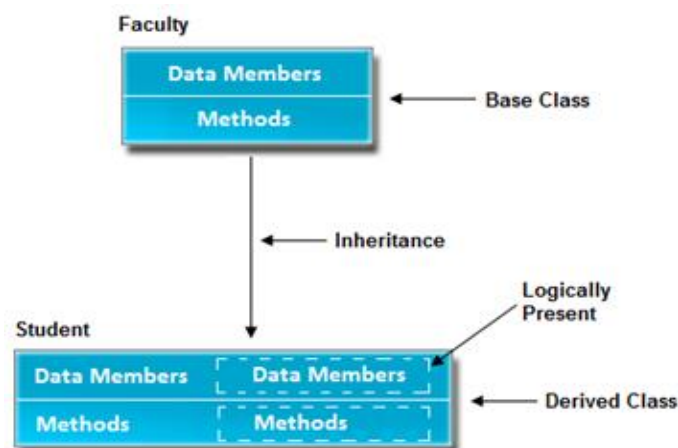
```
class Subclass-Name extends Superclass-Name
{
    //methods and fields
}
```

Real life example of inheritance

The real life example of inheritance is child and parents, all the properties of father are inherited by his son.



The following diagram use view about inheritance.



In the above diagram data members and methods are represented in broken line are inherited from faculty class and they are visible in student class logically.

Advantage of inheritance

If we develop any application using the concept of Inheritance then that application will have the following advantages,

- Application development time is less.
- Application takes less memory.
- Application execution time is less.

- Application performance is enhanced (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

Note: In Inheritance the scope of access modifier increasing is allowed but decreasing is not allowed. Suppose in parent class methods access modifier is default then it's present in child class with default or public or protected access modifier but not private (it decreases the scope).

Types of Inheritance

Based on number of ways inheriting the feature of base class into derived class we have five types of inheritance; they are:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

Single inheritance

In single inheritance there exists single base class and single derived class.

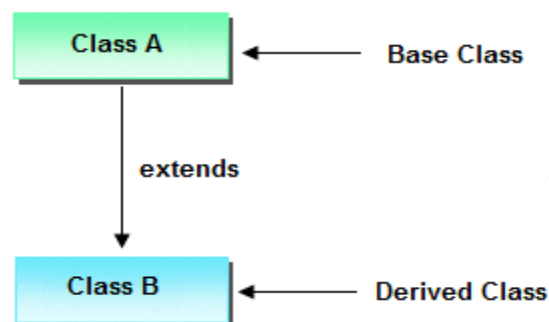
Example of Single Inheritance

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonus=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary is:"+obj.salary);
System.out.println("Bonus is:"+obj.bonus);
}
}
```

Output

Salary is: 30000.0

Bonus is: 2000.0



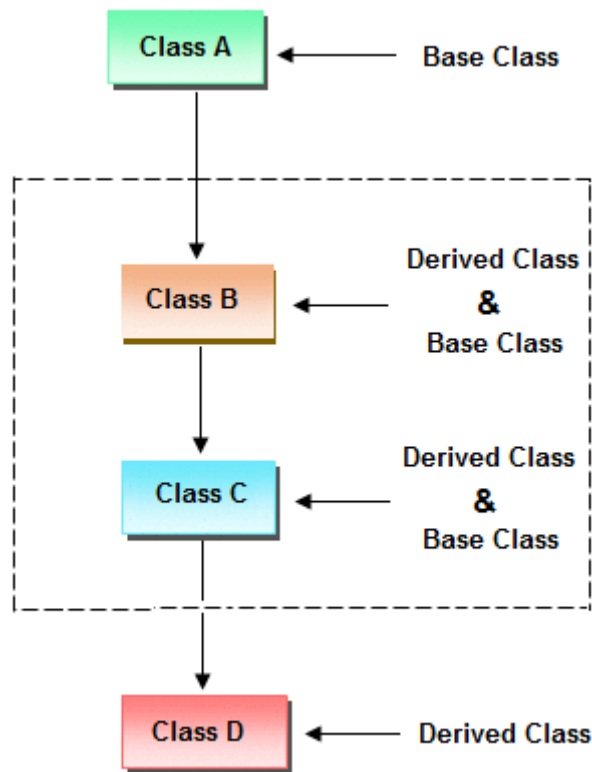
Multilevel inheritances in Java

In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.

Single base class + single derived class + multiple intermediate base classes.

Intermediate base classes

An intermediate base class is one in one context with access derived class and in another context same class access base class.



Hence all the above three inheritance types are supported by both classes and interfaces.

Example of Multilevel Inheritance

```
class Faculty
```

```
{  
float total_sal=0, salary=30000;  
}
```

```
class HRA extends Faculty
```

```
{  
float hra=3000;  
}
```

```
class DA extends HRA
```

```
{  
float da=2000;  
}
```

```
class Science extends DA
```

```
{  
float bonus=2000;  
public static void main(String args[])
```

```

{
Science obj=new Science();
obj.total_sal=obj.salary+obj.hra+obj.da+obj.bnous;
System.out.println("Total Salary is:"+obj.total_sal);
}
}

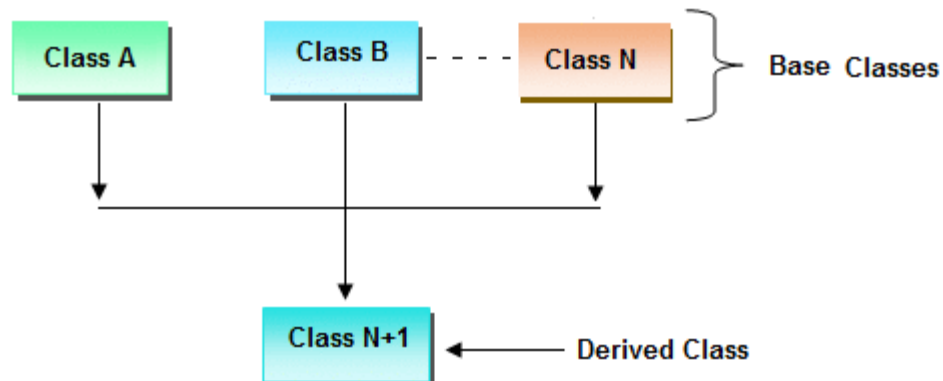
```

Output

Total Salary is: 37000.0

Multiple inheritance

In multiple inheritance there exist multiple classes and single derived class.

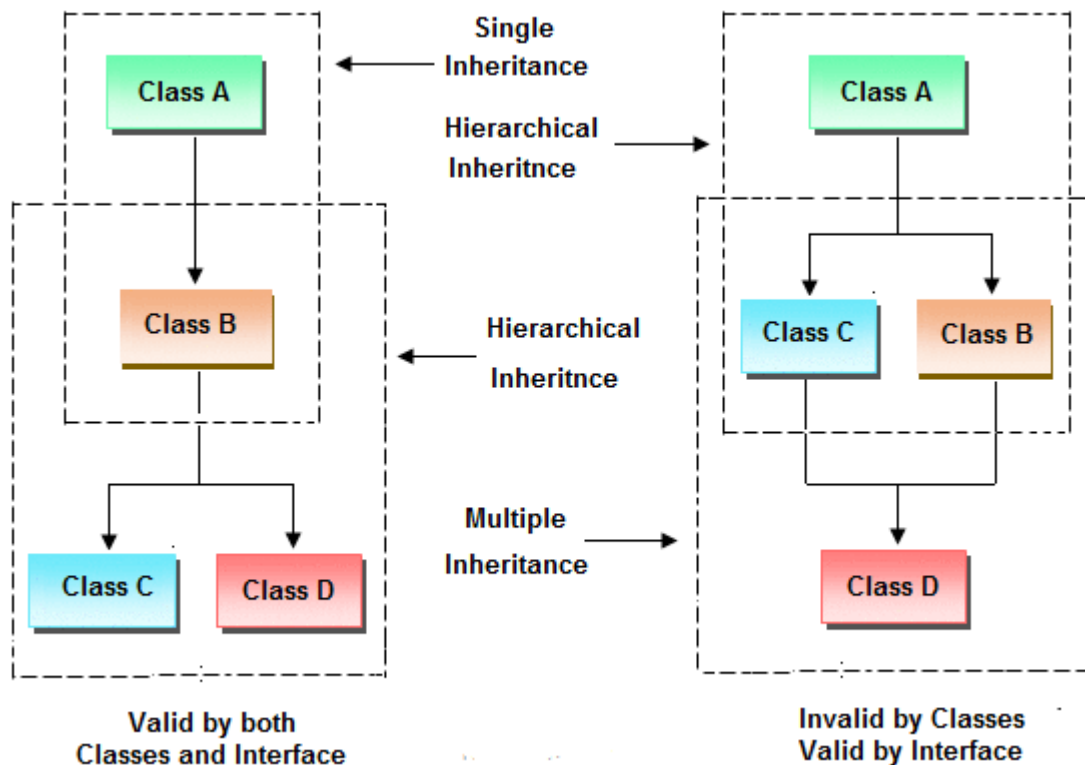


The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.

Hybrid inheritance

Combination of any inheritance type

In the combination if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface.

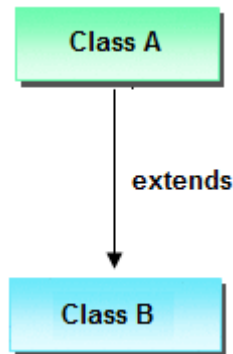


Inheriting the feature from base class to derived class

In order to inherit the feature of base class into derived class we use the following syntax

Syntax

```
class ClassName-2 extends ClasssName-1
{
variable declaration;
Method declaration;
}
```



Explanation

1. ClassName-1 and ClassName-2 represents name of the base and derived classes respectively.
2. extends is one of the keyword used for inheriting the features of base class into derived class it improves the functionality of derived class.

Important Points for Inheritance:

- In java programming one derived class can extends only one base class because java programming does not support multiple inheritance through the concept of classes, but it can be supported through the concept of Interface.
- Whenever we develop any inheritance applications first create an object of bottom most derived class but not for top most base class.
- When we create an object of bottom most derived class, first we get the memory space for the data members of top most base class, and then we get the memory space for data member of other bottom most derived class.
- Bottom most derived class contains logical appearance for the data members of all top most base classes.
- If we do not want to give the features of base class to the derived class then the definition of the base class must be preceded by final hence final base classes are not reusable or not inheritable.
- If we are do not want to give some of the features of base class to derived class than such features of base class must be as private hence private features of base class are not inheritable or accessible in derived class.
- Data members and methods of a base class can be inherited into the derived class but constructors of base class cannot be inherited because every constructor of a class is made for initializing its own data members but not made for initializing the data members of other classes.
- An object of base class can contain details about features of same class but an object of base class never contains the details about special features of its derived class (this concept is known as scope of base class object).
- For each and every class in java there exists an implicit predefined super class called java.lang.Object. Because it provides garbage collection facility to its sub classes for collecting un-used memory space and improves the performance of java application.

Example of Inheritance

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonus=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary is:"+obj.salary);
System.out.println("Bonus is:"+obj.bonus);
}
}
```

Output

Salary is: 30000.0

Bonus is: 2000.0

Why multiple inheritance is not supported in java?

Due to ambiguity problem java does not support multiple inheritance at class level.

Example

```
class A
{
void disp()
{
System.out.println("Hello");
}
}
class B
{
void disp()
System.out.println("How are you ?");
}
}
class C extends A,B //suppose if it were
{
Public Static void main(String args[])
{
C obj=new C();
obj.disp();//Now which disp() method would be invoked?
}
}
```

In above code we call both class A and class B disp() method then it confusion which class method is call. So due to this ambiguity problem in java do not use multiple inheritance at class level, but it support at interface level.

Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

//Java Program to illustrate the use of Java Method Overriding

//Creating a parent class.

```
class Vehicle
```

```
{
```

```
    //defining a method
```

```
    void run()
```

```
{
```

```
System.out.println("Vehicle is running");
```

```
}
```

```
}
```

//Creating a child class

```
class Bike2 extends Vehicle
```

```
{
```

```
    //defining the same method as in the parent class
```

```
    void run()
```

```
{
```

```
System.out.println("Bike is running safely");
```

```
}
```

```
    public static void main(String args[])
```

```
{
```

```
    Bike2 obj = new Bike2();//creating object
```

```
    obj.run();//calling method
```

```
}
```

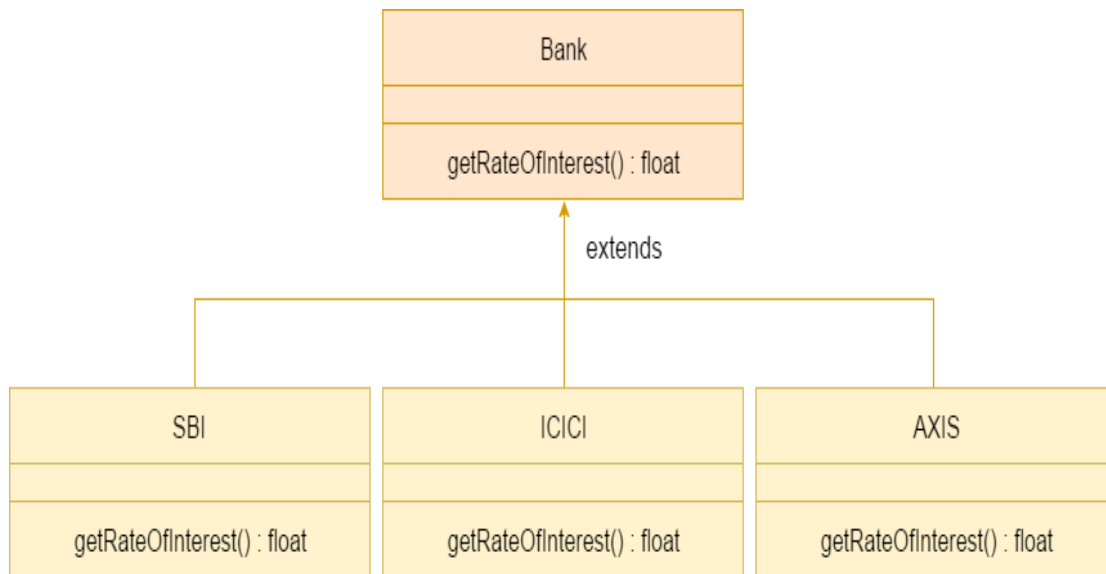
```
}
```

Output:

Bike is running safely

A real example of Java Method Overriding

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



//Java Program to demonstrate the real scenario of Java Method Overriding

//where three classes are overriding the method of a parent class.

//Creating a parent class.

```
class Bank
{
    int getRateOfInterest()
    {
        return 0;
    }
}
```

//Creating child classes.

```
class SBI extends Bank
```

```
{
    int getRateOfInterest()
    {
        return 8;
    }
}
```

```
class ICICI extends Bank
```

```
{
    int getRateOfInterest()
    {
        return 7;
    }
}
```

```
class AXIS extends Bank
```

```
{
    int getRateOfInterest()
    {
        return 9;
    }
}
```

//Test class to create objects and call the methods

```
class Test2
```

```

{
public static void main(String args[])
{
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}

```

Output:

SBI Rate of Interest: 8

ICICI Rate of Interest: 7

AXIS Rate of Interest: 9

Dynamic Method Dispatch or Runtime Polymorphism

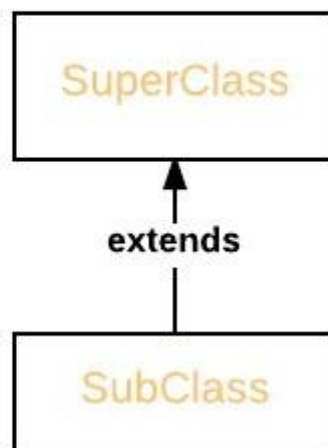
Method overriding is one of the ways in which Java supports Runtime Polymorphism.

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

Upcasting

SuperClass obj = new SubClass



Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different

versions of the method are executed. Here is an example that illustrates dynamic method dispatch:

// A Java program to illustrate Dynamic Method Dispatch using hierarchical inheritance

class A

```
{
    void m1()
    {
        System.out.println("Inside A's m1 method");
    }
}
```

class B extends A

```
{
    // overriding m1()
    void m1()
    {
        System.out.println("Inside B's m1 method");
    }
}
```

class C extends A

```
{
    // overriding m1()
    void m1()
    {
        System.out.println("Inside C's m1 method");
    }
}
```

// Driver class

class Dispatch

```
{
    public static void main(String args[])
    {
        // object of type A
        A a = new A();

        // object of type B
        B b = new B();

        // object of type C
        C c = new C();

        // obtain a reference of type A
        A ref;

        // ref refers to an A object
        ref = a;

        // calling A's version of m1()
    }
}
```

```

ref.m1();

// now ref refers to a B object
ref = b;

// calling B's version of m1()
ref.m1();

// now ref refers to a C object
ref = c;

// calling C's version of m1()
ref.m1();
}
}

```

Output:

Inside A's m1 method

Inside B's m1 method

Inside C's m1 method

Explanation :

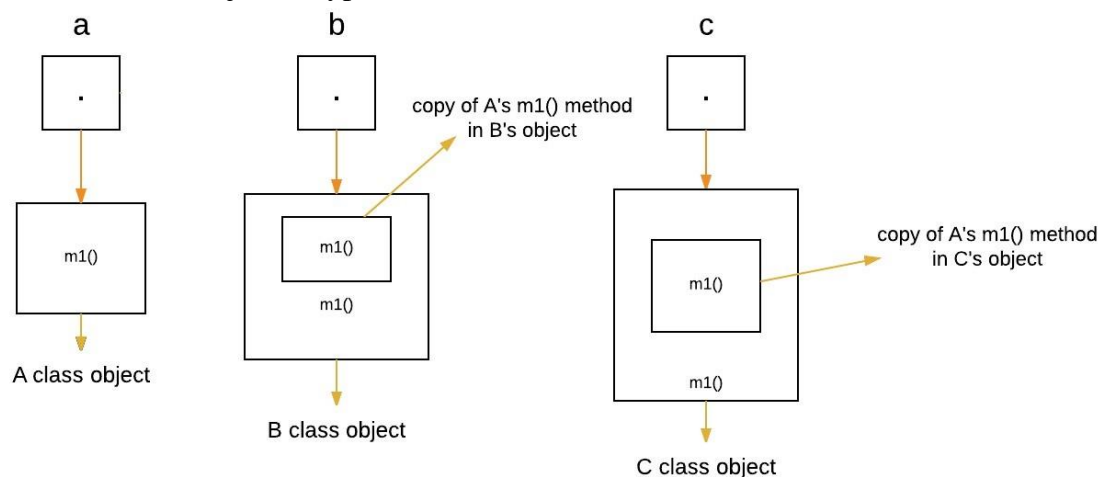
The above program creates one superclass called A and it's two subclasses B and C. These subclasses overrides m1() method.

1. Inside the main() method in Dispatch class, initially objects of type A, B, and C are declared.

A a = new A(); // object of type A

B b = new B(); // object of type B

C c = new C(); // object of type C



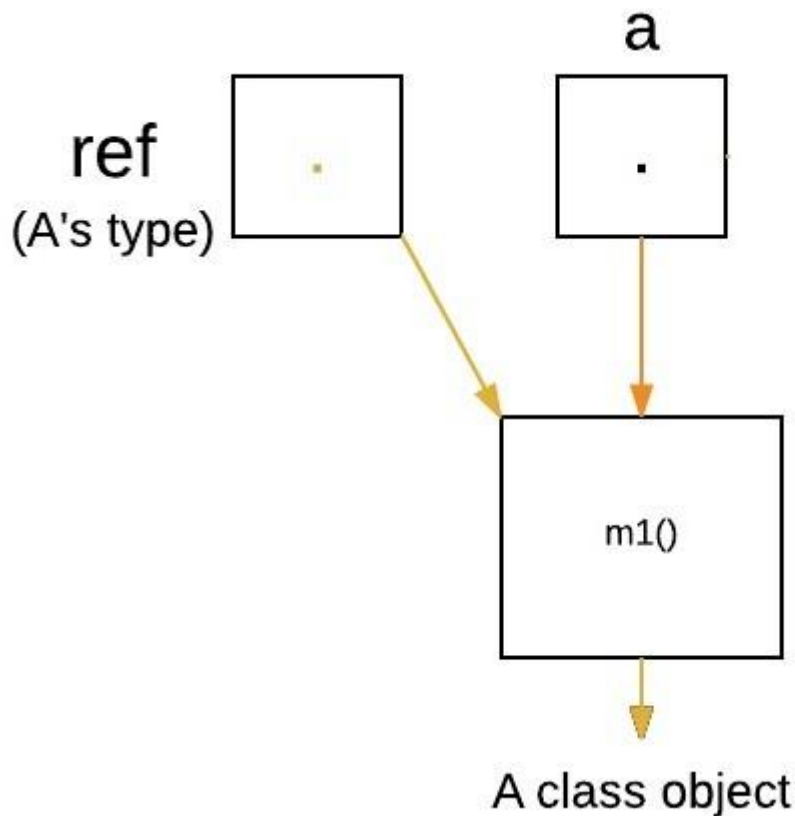
2. Now a reference of type A, called ref, is also declared, initially it will point to null.

A ref; // obtain a reference of type A

ref
(A's type) null

3. Now we are assigning a reference to each type of object (either A's or B's or C's) to *ref*, one-by-one, and uses that reference to invoke *m1()*. As the output shows, the version of *m1()* executed is determined by the type of object being referred to at the time of the call.

```
ref = a; // r refers to an A object  
ref.m1(); // calling A's version of m1()
```



Super keyword

Super keyword in java is a reference variable that is used to refer parent class object. Super is an implicit keyword created by JVM and supply each and every java program for performing important role in three places

- At variable level
- At method level
- At constructor level

Need of super keyword:

Whenever the derived class inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

Syntax

super.baseclass features.

Super at variable level:

Whenever the derived class inherits base class data members there is a possibility that base class data member are similar to derived class data member and JVM gets an ambiguity.

In order to differentiate between the data member of base class and derived class, in the context of derived class the base class data members must be preceded by super keyword.
Syntax

super.baseclass datamember name

if we are not writing super keyword before the base class data member name than it will be referred as current class data member name and base class data member are hidden in the context of derived class.

Program without using super keyword

Example

```
class Employee
{
float salary=10000;
}
class HR extends Employee
{
float salary=20000;
void display()
{
System.out.println("Salary: "+salary);//print current class salary
}
}
class Supervariable
{
public static void main(String[] args)
{
HR obj=new HR();
obj.display();
}
}
```

Output

Salary: 20000.0

In the above program in Employee and HR class salary is common properties of both class the instance of current or derived class is referred by instance by default but here we want to refer base class instance variable that is why we use super keyword to distinguish between parent or base class instance variable and current or derived class instance variable.

Program using super keyword at variable level

Example

```
class Employee
{
float salary=10000;
}
class HR extends Employee
{
float salary=20000;
void display()
{
System.out.println("Salary: "+super.salary);//print base class salary
}
}
```

```

class Supervariable
{
public static void main(String[] args)
{
HR obj=new HR();
obj.display();
}
}

```

Output

Salary: 10000.0

Super at method level

The super keyword can also be used to invoke or call parent class method. It should be use in case of method overriding. In other word super keyword use when base class method name and derived class method name have same name.

Example of super keyword at method level

Example

```

class Student
{
void message()
{
System.out.println("Good Morning Sir");
}
}

```

```

class Faculty extends Student

```

```

{
void message()
{
System.out.println("Good Morning Students");
}
}

```

```

void display()
{
message();//will invoke or call current class message() method
super.message();//will invoke or call parent class message() method
}

```

```

public static void main(String args[])
{
Student s=new Student ();
s.display();
}
}

```

Output

Good Morning Students

Good Morning Sir

In the above example Student and Faculty both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority of local is high.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.

Program where super is not required

Example

```
class Student
```

```
{
void message()
{
System.out.println("Good Morning Sir");
}
}
```

```
class Faculty extends Student
```

```
{

void display()
{
message();//will invoke or call parent class message() method
}
```

```
public static void main(String args[])
```

```
{
Student s=new Student();
s.display();
}
}
```

Output

Good Morning Sir

Super at constructor level

The super keyword can also be used to invoke or call the parent class constructor.

Constructor are calling from bottom to top and executing from top to bottom.

To establish the connection between base class constructor and derived class constructors

JVM provides two implicit methods they are:

- Super()
- Super(...)

super()

super() It is used for calling super class default constructor from the context of derived class constructor.

Super keyword used to call base class constructor

```
class Employee
```

```
{
Employee()
{
System.out.println("Employee class Constructor");
}
}
```

```

class HR extends Employee
{
HR()
{
super(); //will invoke or call parent class constructor
System.out.println("HR class Constructor");
}
}
class Supercons
{
public static void main(String[] args)
{
HR obj=new HR();
}
}
Output

```

Employee class Constructor

HR class Constructor

Note: super() is added in each class constructor automatically by compiler.

super(...) (with parameters) it is used for calling super class parameterized constructor from the context of derived class constructor explicitly.

```

class Person
{
int id;
String name;
Person(int id,String name)
{
this.id=id;
this.name=name;
}
}
class Emp extends Person
{
float salary;
Emp(int id,String name,float salary)
{
super(id,name);//reusing parent constructor
this.salary=salary;
}
void display()
{
System.out.println(id+" "+name+" "+salary);
}
}
class TestSuper5
{
public static void main(String[] args)

```

```

{
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}
}

```

Note: When the super class has a parameterized constructor and if you want to use that constructor in a sub class constructor, you have to call the super class constructor explicitly otherwise the compiler reports an error.

Final keyword

It is used to make a variable as a constant, restrict a method from overriding, and can restrict inheritance. In java language final keyword can be used in following way.

- Final at variable level
- Final at method level
- Final at class level



Final Keyword

- Restrict Changing value of variable
- Restrict method Overriding
- Restrict Inheritance

Final at variable level

Final keyword is used to make a variable as a constant. This is similar to const in other languages. A variable declared with the final keyword cannot be modified by the program after initialization. This is useful to universal constants, such as "PI".

Final Keyword in java Example

```

public class Circle
{
public static final double PI=3.14159;

```

```

public static void main(String[] args)
{
System.out.println(PI);
}
}

```

Final at method level

It makes a method final, meaning that sub classes cannot override this method. The compiler checks and gives an error if you try to override the method.

When we want to restrict overriding, then make a method as a final.

Example

```

public class A
{
public void fun1()
{
.....
}
public final void fun2()

```

```

{
.....
}

}
class B extends A
{
public void fun1()
{
.....
}
public void fun2()
{
// it gives an error because we cannot override final method
}
}

```

Example of final keyword at method level

Example

```

class Employee
{
final void disp()
{
System.out.println("Hello Good Morning");
}
}
class Developer extends Employee
{
void disp()
{
System.out.println("How are you ?");
}
}
class FinalDemo
{
public static void main(String args[])
{
Developer obj=new Developer();
obj.disp();
}
}

```

Output

It gives an error

Final at class level

It makes a class final, meaning that the class cannot be inheriting by other classes. When we want to restrict inheritance then make class as a final.

Example

```

public final class A
{
.....
}

```

```

.....
}
public class B extends A
{
// it gives an error, because we cannot inherit final class
}
Example of final keyword at class level
Example
final class Employee
{
int salary=10000;
}
class Developer extends Employee
{
void show()
{
System.out.println("Hello Good Morning");
}
}
class FinalDemo
{
public static void main(String args[])
{
Developer obj=new Developer();
Developer obj=new Developer();
obj.show();
}
}
Output
It gives an error

```

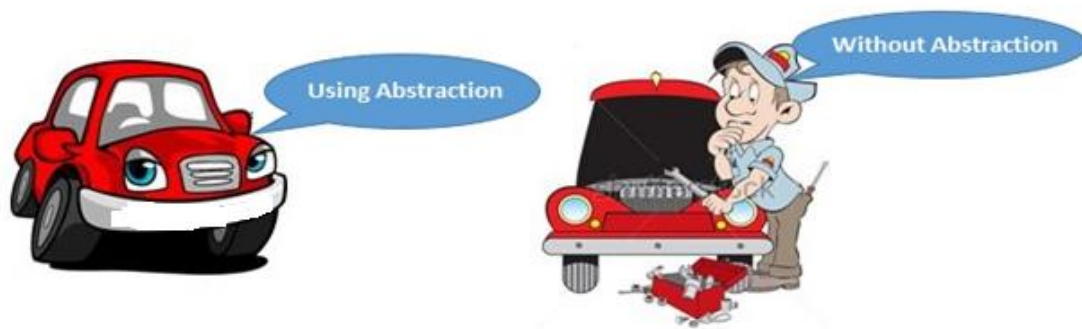
Abstraction

Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.

Hiding of data is known as data abstraction. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

Real Life Example of Abstraction in Java

Abstraction shows only important things to the user and hides the internal details, for example, when we ride a bike, we only know about how to ride bikes but cannot know how it work? And also we do not know the internal functionality of a bike.



Another real life example of Abstraction is ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement...etc. but we can't know internal details about ATM.



Real Life Example of Abstraction

Note: Data abstraction can be used to provide security for the data from the unauthorized methods.

Note: In Java language data abstraction can achieve using class.

Example of Abstraction

```
class Customer
{
int account_no;
float balance_Amt;
String name;
int age;
String address;
void balance_inquiry()
{
/* to perform balance inquiry only account number
is required that means remaining properties
are hidden for balance inquiry method */
}
void fund_Transfer()
{
/* To transfer the fund account number and
balance is required and remaining properties
are hidden for fund transfer method */
}
```

```
}
```

How to achieve Abstraction?

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (Achieve 100% abstraction)

Difference between Encapsulation and Abstraction

Encapsulation is not providing full security because we can access private member of the class using reflection API, but in case of Abstraction we can't access static, abstract data member of a class.

Abstract class

We know that every Java program must start with a concept of class that is without the class concept there is no Java program perfect.

In Java programming we have two types of classes they are

- Concrete class
- Abstract class

Concrete class in Java

A concrete class is one which is containing fully defined methods or implemented methods.

Example

```
class Helloworld
```

```
{  
void display()  
{  
System.out.println("Good Morning.....");  
}  
}
```

Here Helloworld class is containing a defined method and object can be created directly.

Create an object

```
Helloworld obj=new Helloworld();
```

```
obj.display();
```

Every concrete class has specific features and these classes are used for specific requirement, but not for common requirement.

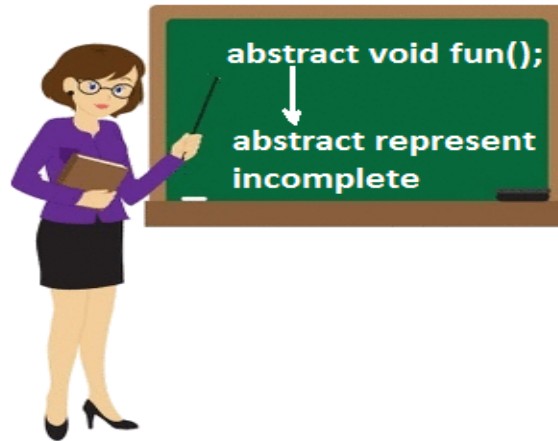
If we use concrete classes for fulfill common requirements than such application will get the following limitations.

- Application will take more amount of memory space (main memory).
- Application execution time is more.
- Application performance is decreased.

To overcome above limitation you can use abstract class.

Abstract class in Java

A class that is declared with abstract keyword is known as abstract class. An abstract class is one which is containing some defined methods and some undefined methods. In java programming undefined methods are known as un-implemented or abstract methods.



Syntax

```
abstract class className  
{  
.....  
}
```

Example

```
abstract class A  
{  
.....  
}
```

If any class has any abstract method then that class becomes an abstract class.

Example

```
class Vehicle  
{  
abstract void Bike();  
}
```

Class Vehicle is become an abstract class because it has abstract Bike() method.

Make class as abstract class

To make the class as abstract class, whose definition must be preceded by an abstract keyword.

Example

```
abstract class Vehicle  
{  
.....  
}
```

Abstract method

An abstract method is one which contains only declaration or prototype but it never contains body or definition. In order to make any undefined method as abstract whose declaration must be predefined by abstract keyword.

Syntax

```
abstract ReturnType methodName(List of formal parameter)
```

Example

```
abstract void sum();  
abstract void diff(int, int);
```

Example of abstract class

```
abstract class Vehicle  
{
```



```

    abstract void speed(); // abstract method
}
class Bike extends Vehicle
{
    void speed()
    {
        System.out.println("Speed limit is 40 km/hr..");
    }
    public static void main(String args[])
    {
        Vachile obj = new Bike(); //indirect object creation
        obj.speed();
    }
}

```

Output

Speed limit is 40 km/hr..

Create an Object of abstract class

An object of abstract class cannot be created directly, but it can be created indirectly. It means you can create an object of abstract derived class. You can see in above example Example

Vachile obj = new Bike(); //indirect object creation

Important Points about abstract class

- Abstract class of Java always contains common features.
- Every abstract class participates in inheritance.
- Abstract class definitions should not be made as final because abstract classes always participate in inheritance classes.
- An object of abstract class cannot be created directly, but it can be created indirectly.
- All the abstract classes of Java makes use of polymorphism along with method overriding for business logic development and makes use of dynamic binding for execution logic.

Advantage of abstract class

- Less memory space for the application
- Less execution time
- More performance

Why abstract classes have no abstract static method?

In abstract classes we have the only abstract instance method, but not containing abstract static methods because every instance method is created for performing repeated operation where as static method is created for performing a onetime operations in other word every abstract method is instance but not static.

Abstract base class

An abstract base class is one which is containing physical representation of abstract methods which are inherited by various sub classes.

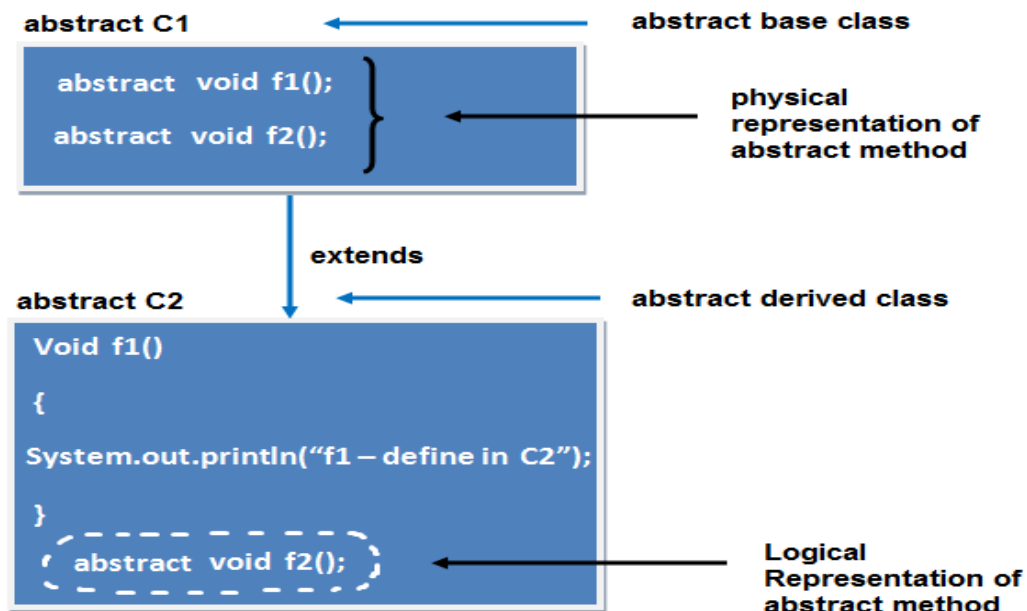
Abstract derived class

An abstract derived class is one which is containing logic representation of abstract methods which are inherited from abstract base class with respect to both abstract base class and abstract derived class one cannot create objects directly, but we can create their objects indirectly both abstract base class and abstract derived class are always reusable by various sub classes.

When the derived class inherits multiple abstract methods from abstract base class and if the derived class is not defined at least one abstract method then the derived class is known as

abstract derived class and whose definition must be made as abstract by using abstract keyword

If the derived class defined all the abstract methods which are inherited from abstract Base class, then the derived class is known as concrete derived class.



Example of abstract class having method body

abstract class Vehicle

```
{
    abstract void speed();
    void mileage()
    {
        System.out.println("Mileage is 60 km/ltr..");
    }
}
class Bike extends Vehicle
{
    void speed()
    {
        System.out.println("Speed limit is 40 km/hr..");
    }
    public static void main(String args[])
    {
        Vehicle obj = new Bike();
        obj.speed();
        obj.mileage();
    }
}
```

Output

Mileage is 60 km/ltr..

Speed limit is 40 km/hr..

Example of abstract class having constructor, data member, methods

abstract class Vehicle

```

{
int limit=40;
Vehicle()
{
System.out.println("constructor is invoked");
}
void getDetails()
{
System.out.println("it has two wheels");
}
abstract void run();
}

```

```

class Bike extends Vehicle
{
void run()
{
System.out.println("running safely..");
}
public static void main(String args[])
{
Vehicle obj = new Bike();
obj.run();
obj.getDetails();
System.out.println(obj.limit);
}
}

```

Output

constructor is invoked

running safely..

it has two wheels

40

Difference between Abstract class and Concrete class

Concrete class	Abstract class
A Concrete class is used for specific requirement	Abstract class is used to fulfill a common requirement.
Object of concrete class can create directly.	Object of an abstract class can not create directly (can create indirectly).
Concrete class containing fully defined methods or implemented method.	Abstract class has both undefined method and defined method.

Interface

Interface is similar to class which is collection of public static final variables (constants) and abstract methods.

The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

Why we use Interface?

- It is used to achieve fully abstraction.
- By using Interface, you can achieve multiple inheritance in java.
- It can be used to achieve loose coupling.

Properties of Interface

- It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

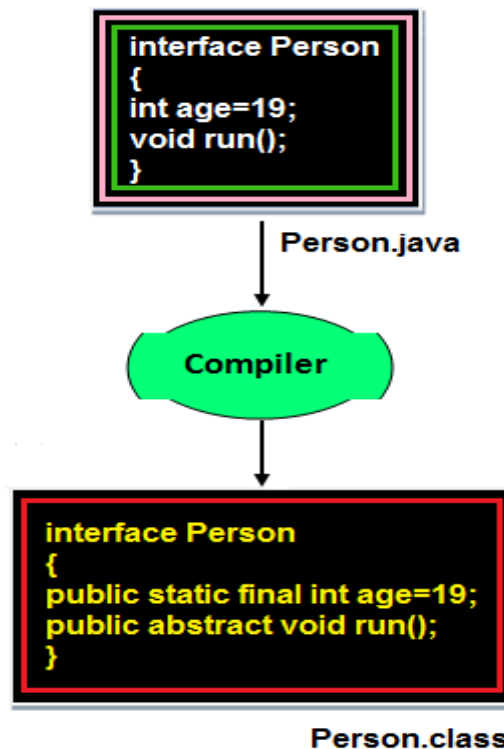
How interface is similar to class?

Whenever we compile any Interface program it generate .class file. That means the byte code of an interface appears in a .class file.

How interface is different from class?

- You cannot instantiate an interface.
- It does not contain any constructors.
- All methods in an interface are abstract.
- Interface cannot contain instance fields. Interface only contains public static final variables.
- Interface is cannot extended by a class; it is implemented by a class.
- Interface can extend multiple interfaces. It means interface support multiple inheritance

Behavior of compiler with Interface program



In the above image when we compile any interface program, by default compiler added public static final before any variable and public abstract before any method. Because Interface is design for fulfill universal requirements and to achieve fully abstraction.

Declaring Interfaces:

The interface keyword is used to declare an interface.

Example

```

interface Person
{
    datatype variablename=value;
    //Any number of final, static fields
    returntype methodname(list of parameters or no parameters)
    //Any number of abstract method declarations
}

```

Explanations

In the above syntax Interface is a keyword interface name can be user defined name the default signature of variable is public static final and for method is public abstract. JVM will add implicitly public static final before data members and public abstract before methods.

Example

public static final datatype variable name=value; ----> for data member
 public abstract returntype methodname(parameters)---> for method

Implementing Interfaces:

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```

interface Person
{
    void run();
}
class Employee implements Person
{
    public void run()
    {
        System.out.println("Run fast");
    }
}

```

When we use abstract and when Interface

If we do not know about any things about implementation just we have requirement specification then we should be go for Interface

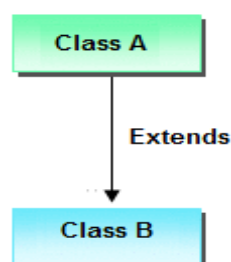
If we are talking about implementation but not completely (partially implemented) then we should be go for abstract

Rules for implementation interface

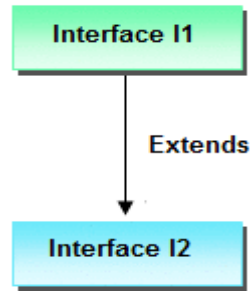
- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

Relationship between class and Interface

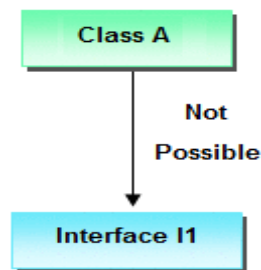
- Any class can extends another class



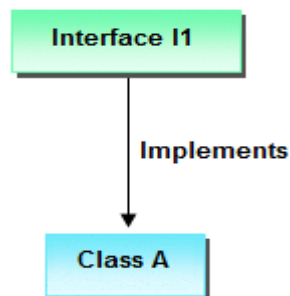
- Any Interface can extends another Interface.



- Any class can Implements another Interface



- Any Interface cannot extend or Implements any class.



Difference between Abstract class and Interface

	Abstract class	Interface
1	It is collection of abstract method and concrete methods.	It is collection of abstract method.
2	There properties can be reused commonly in a specific application.	There properties commonly usable in any application of java environment.
3	It does not support multiple inheritance.	It supports multiple inheritance.
4	Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
5	Which may contain either variable or constants.	Which should contains only constants.
6	The default access specifier of abstract class methods are default.	There default access specifier of interface method are public.
7	These class properties can be reused in other class using extend keyword.	These properties can be reused in any other class using implements keyword.
8	Inside abstract class we can take constructor.	Inside interface we cannot take any constructor.

9	For the abstract class there is no restriction like initialization of variable at the time of variable declaration.	For the interface it should be compulsory to initialization of variable at the time of variable declaration.
10	There are no any restriction for abstract class variable.	For the interface variable cannot declare variable as private, protected, transient, volatile.
11	There are no any restriction for abstract class method modifier that means we can use any modifiers.	For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized.

Example of Interface

interface Person

```
{
void run(); // abstract method
}
```

class A implements Person

```
{
public void run()
{
System.out.println("Run fast");
}
public static void main(String args[])
{
A obj = new A();
obj.run();
}
}
```

Output

Run fast

Multiple Inheritance using interfaces

Example

interface Developer

```
{
void disp();
}
```

interface Manager

```
{
void show();
}
```

class Employee implements Developer, Manager

```
{
public void disp()
{
System.out.println("Hello Good Morning");
}
public void show()
{
System.out.println("How are you ?");
}
```

```

public static void main(String args[])
{
Employee obj=new Employee();
obj.disp();
obj.show();
}
}

```

Output

Hello Good Morning

How are you ?

Marker or tagged interface

An interface that has no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Example

//Way of writing Serializable interface

```

public interface Serializable
{
}

```

Why interface have no constructor?

Because, constructor are used for eliminate the default values by user defined values, but in case of interface all the data members are public static final that means all are constant so no need to eliminate these values.

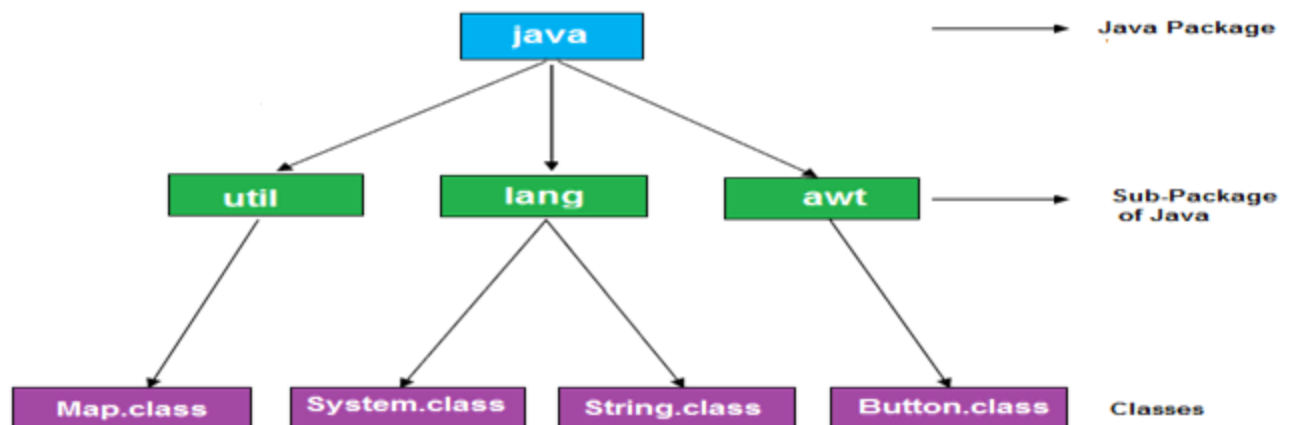
Other reason because constructor is like a method and it is concrete method and interface does not have concrete method it have only abstract methods that's why interface have no constructor.

Packages

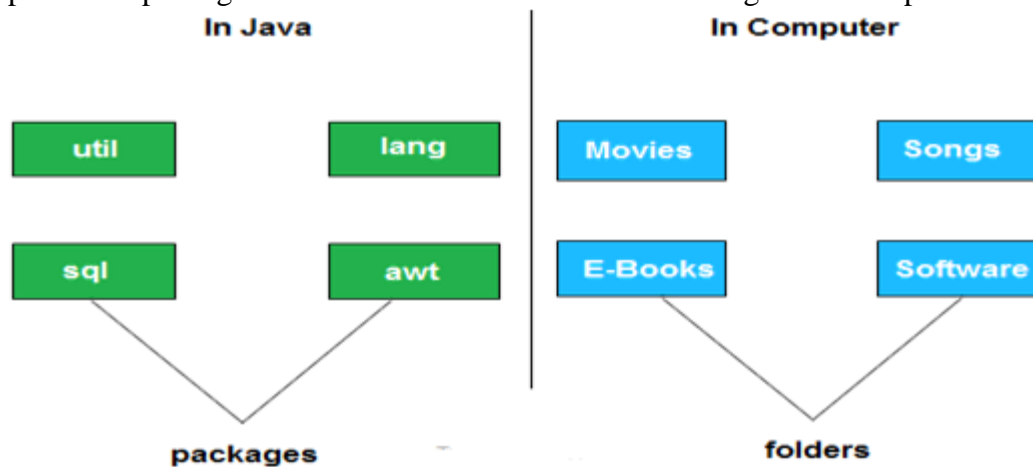
A package is a collection of similar types of classes, interfaces and sub-packages.

Purpose of package

The purpose of package concept is to provide common classes and interfaces for any program separately. In other words if we want to develop any class or interface which is common for most of the java programs than such common classes and interfaces must be placed in a package.



Packages in Java are the way to organize files when a project has many modules. Same like we organized our files in Computer. For example we store all movies in one folder and songs in other folder, here also we store same type of files in a particular package for example in awt package have all classes and interfaces for design GUI components.



Advantage of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained
- Application development time is less, because reuse the code
- Application memory space is less (main memory)
- Application execution time is less
- Application performance is enhance (improve)
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

Types of package

Packages are classified into two types which are given below.

1. Predefined or built-in package
2. User defined package

Predefined or built-in package

These are the packages which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.

User defined package

If any package is designed by the user that is known as a user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement.

Rules to create user defined package

- package statement should be the first statement of any package program.
- Choose an appropriate class name or interface name and whose modifier must be public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any main())
- modifier of constructor of the class which is present in the package must be public. (This is not applicable in case of interface because interface has no constructor.)

- The modifier of method of class or interface which is present in the package must be public (This rule is optional in case of interface because interface methods by default public)
- Every package program should be save either with public class name or public Interface name

<code>//Sum.java</code>	→	Save package program with 'public' class name
<code>package MyPackage</code>	→	First statement of java program is package
<code>public class Sum</code>	→	class modifier must be 'public'
<code>{</code>		
<code>public Sum()</code>	→	constructor modifier must be 'public'
<code>{</code>		
<code>System.out.println("Sum class constructor");</code>		
<code>}</code>		
<code>public void show()</code>	→	method modifier must be 'public'
<code>{</code>		
<code>System.out.println("Sum class method");</code>		
<code>}</code>		
<code>}</code>		

Compile package programs

For compilation of package program first we save program with public className.java and it compile using below syntax:

Syntax

`javac -d . className.java`

Syntax

`javac -d path className.java`

Explanations: In above syntax "-d" is a specific tool which is tell to java compiler create a separate folder for the given package in given path. When we give specific path then it create a new folder at that location and when we use . (dot) then it crate a folder at current working directory.

Note: Any package program can be compile but can not be execute or run. These program can be executed through user defined program which are importing package program.

Example of package program

Package program which is save with A.java and compile by `javac -d . A.java`

Example

```
package mypack;
public class A
{
public void show()
{
System.out.println("Sum method");
```

```
}  
}
```

Import above class in below program using import packageName.className

Example

```
import mypack.A;  
public class Hello  
{  
    public static void main(String arg[])  
    {  
        A a=new A();  
        a.show();  
        System.out.println("show() class A");  
    }  
}
```

Explanations: In the above program first we create Package program which is save with A.java and compiled by "javac -d . A.java". Again we import class "A" in class Hello using "import mypack.A;" statement.

Difference between Inheritance and package

Inheritance concept always used to reuse the feature within the program between class to class, interface to interface and interface to class but not accessing the feature across the program.

Package concept is to reuse the feature both within the program and across the programs between class to class, interface to interface and interface to class.

Difference between package keyword and import keyword

Package keyword is always used for creating the undefined package and placing common classes and interfaces.

import is a keyword which is used for referring or using the classes and interfaces of a specific package.

Fill in the Blanks

1. To inherit a class,_____keyword is used.
2. Super keyword can be used in_____number of ways.
3. To disallow a method from being overridden_____modifier is specified.
4. All the classes in Java are subclasses of_____Class.
5. Reference variable of type_____can refer to an object of any other class.
6. _____are the containers for the classes.
7. Packages are stored in_____manner.
8. Fully abstracted class can be called as_____.
9. The interface does not_____any implementation.
10. A class can implement more than_____interfaces.
11. A_____is an abstraction that either produces or consumes information.
12. Java defines_____and_____type of Streams.
13. Partially implemented class can be called as_____.
14. Fully Implemented class can be called as_____.
15. Java supports_____number of access modes.
16. Java supports_____number of access specifiers.
17. The variables declared in an interface are implicitly_____and_____.
18. _____keyword is used for implementing an interface.

19. Completely unimplemented class can be called as _____
20. One interface can inherit another by the use of the keyword _____.

Short Answer Questions		
S. No	Questions	Blooms Taxonomy Level
1	Define Inheritance.	Understand
2	List various types of inheritances in java.	Remember
3	Define static binding.	Understand
4	Identify the use of “super” keyword	Understand
5	Summarize the use of “final” keyword with inheritance.	Understand
6	List various methods in Object class.	Remember
7	Describe abstract class.	Remember
8	Interpret various member access rules in java.	Understand
9	Define method overriding.	Understand
10	Explain different Types of Packages	Understand
11	Define interface	Understand
12	List the advantages of Package.	Remember
13	Identify the keyword used for creating the package.	Understand
14	Define a package?	Understand
15	State various steps for creating and importing packages.	Remember
16	Define abstract method.	Understand
17	Summarize the steps to implement an interface	Understand
18	List advantages of inheritance.	Remember
19	Define CLASSPATH.	Understand

Long Answer Questions		
S. No	Questions	Blooms Taxonomy Level
1	Exemplify the “this” and “super” keywords usage in java.	Understand
2	List different types of inheritances in java with example.	Remember
3	Discuss various methods of Object class.	Understand
4	Illustrate the Use of “Super” keyword in method overriding with example.	Understand
5	Discuss the importance of package. Demonstrate with program.	Understand
6	Compare and Contrast interfaces and Abstract classes.	Understand
7	Demonstrate dynamic binding with an example.	Understand
8	List out the some of the standard overloaded methods in java.	Remember
9	Describe Abstraction in java using abstract class with an example.	Remember
10	Explain about interface with an example.	Understand
11	Define multiple inheritances with suitable example.	Understand
12	Discuss in detail creating and importing package in java.	Understand
13	Compare and contrast overloading and overriding methods.	Understand
14	Explain different ways to extending interfaces with an example.	Understand

15	Differentiate between class and interface.	Understand
16	Discuss the importance of final keyword in java with a program..	Understand
17	Explain the benefits of inheritance with an example.	Understand
18	Describe various member access rules and explain with an example.	Understand
19	Discuss the role of classpath in packages.	Understand

Problem Solving and Critical Thinking Questions		Blooms Taxonomy Level
S. No	Questions	
1	<p>Analyze the program and give output</p> <pre> class Animal { void eat() { System.out.println("eating..."); } } class Dog extends Animal { void bark() { System.out.println("barking..."); } } class TestInheritance { public static void main(String args[]) { Dog d=new Dog(); d.bark(); d.eat(); } } </pre>	Understand
2	<p>Identify the output of the following program.</p> <pre> interface Sample { int x=12; void show(); default void display() { System.out.println("default method of interface"); } Static void print(String str) { System.out.println("Static method of interface:"+str); } } </pre>	Understand
3	<p>Predict output of the program?</p> <pre> class A </pre>	Understand

	<pre> { public A() { System.out.println("NewA"); } } class B extends A { public B() { super(); System.out.println("New B"); } } </pre>	
4	<p>Discuss the output of the following program?</p> <pre> interface MyInterface { public void method1(); public void method2(); } class XYZ implements MyInterface { public void method1() { System.out.println("implementation of method1"); } public void method2() { System.out.println("implementation of method2"); } public static void main(String arg[]) { MyInterface obj = new XYZ(); obj. method1(); } } </pre>	Understand
5	<p>Interpret output of following program</p> <pre> class A { final public int GetResult(int a, int b) { return 0; } } class B extends A { public int GetResult(int a, int b) {return 1; } } public class Test { public static void main(String args[]) </pre>	Understand

	<pre> { B b = new B(); System.out.println("x = " + b.GetResult(0, 1)); } } </pre>	
6	<p>Analyze the output of the program?</p> <pre> class Super { public int i = 0; public Super(String text) { i = 1; } } class Sub extends Super { public Sub(String text) { i = 2; } public static void main(String args[]) { Sub sub = new Sub("Hello"); System.out.println(sub.i); } } </pre>	Remember
7	<p>Identify the output of the program?</p> <pre> interface Count { short counter = 0; void countUp(); } public class TestCount implements Count { public static void main(String [] args) { TestCount t = new TestCount(); t.countUp(); } public void countUp() { for (int x = 6; x>counter; x--, ++counter) { System.out.print(" " + counter); } } } </pre>	Understand
8	<p>Analyze and find out the output of the program?</p> <pre> public class Test { public int aMethod() </pre>	Understand

	<pre> { static int i = 0; i++; return i; } public static void main(String args[]) { Test test = new Test(); test.aMethod(); int j = test.aMethod(); System.out.println(j); } } </pre>	
9	<p>Illustrate a java program to create an abstract class named Shape that contains two integers and an empty method named print Area().provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contains only the method print Area () that prints the area of the given shape.</p>	Remember
10	<p>Predict out the output of the program?</p> <pre> package mypack class Book { String bookname; String author; Book(String b, Stringc) { this.bookname = b; this.author = c; } public void show() { System.out.println(bookname+" "+ author); } } class test { public static void main(String[] args) { Book bk = new Book("java","Herbert"); bk.show(); } } </pre>	Understand

Review Questions

1. What is the base class of all classes?
2. Does Java support multiple inheritance?
3. How to define a constant variable in Java?

4. Can a main() method be declared final?
5. What is a package?
6. Which package is imported by default?
7. Can a class declared as private be accessed outside its package?
8. Can a class be declared as protected?
9. What is the access scope of a protected method?
10. What is the purpose of declaring a variable as final?
11. What is the impact of declaring a method as final?
12. I don't want my class to be inherited by any other class. What should I do?
13. Can you give few examples of final classes defined in Java API?
14. What is an Abstract Class and what is its purpose?
15. Can an abstract class be declared final?
16. What is use of an abstract variable?
17. Can you create an object of an abstract class?
18. Can an abstract class be defined without any abstract methods?
19. Class C implements Interface I containing method m1 and m2 declarations. Class C has provided implementation for method m2. Can I create an object of Class C?
20. Can a method inside an Interface be declared as final?
21. Can an Interface implement another Interface?
22. Can an Interface extend another Interface?
23. Can a Class extend more than one Class?
24. Why is an Interface able to extend more than one Interface but a Class can't extend more than one Class?
25. Can an Interface be final?
26. Can a class be defined inside an Interface?
27. Can an Interface be defined inside a class?
28. What is a Marker Interface?
29. Which object oriented Concept is achieved by using overloading and overriding?
30. Can we define private and protected modifiers for variables in interfaces?
31. What modifiers are allowed for methods in an Interface?
32. What is a local, member and a class variable?
33. What is an abstract method?

UNIT III

UNIT WISE PLAN

UNIT-III: Exception Handling and Multi Threading		Planned Hours: 09	
S. No.	Topic Learning Outcomes	Cos	Blooms Levels
1	Understand the impact of exception handling to avoid abnormal termination of program using checked and unchecked exceptions.	CO4	L2
2	Demonstrate the user defined exceptions by exception handling keywords (try, catch, throw, throws and finally).	CO4	L3
3	Use multithreading concepts to develop inter process communication.	CO4	L3
4	Understand and implement concepts on file streams and operations in java programming for a given application programs	CO4	L2

Exception Handling in Java

The Exception Handling in Java is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

What is Exception in Java

Dictionary Meaning: Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IO, SQL, Remote etc.

Advantage of Exception Handling

The core advantage of exception handling is to maintain the normal flow of the application.

An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Handling the Exception

We use five keywords for Handling the Exception

- try
- catch
- finally
- throws
- throw

Syntax for handling the exception

```
try
{
    // statements causes problem at run time
}
catch(type of exception-1 object-1)
{
    // statements provides user friendly error message
}
catch(type of exception-2 object-2)
{
    // statements provides user friendly error message
}
finally
{
    // statements which will execute compulsory
}
```

1. try block

inside try block we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

Important points about try block

- If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.
- After executing appropriate catch block, even though we use run time statement, CPU control never goes to try block to execute the rest of the statements.
- Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

Syntax

```
try
{
    .....
}
/* Here no other statements are allowed
between try and catch block */
catch()
{
    ....
}
```

- Each and every try block must contains at least one catch block. But it is highly recommended to write multiple catch blocks for generating multiple user friendly error messages.

- One try block can contains another try block that is nested or inner try block can be possible.

Syntax

```
try
{
.....
try
{
.....
}
}
```

2. catch block

Inside catch block we write the block of statements which will generates user friendly error messages.

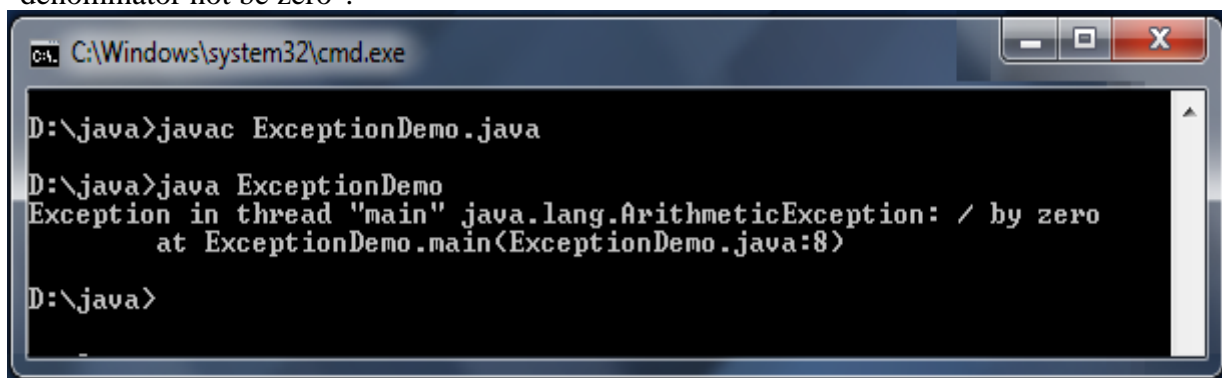
catch block important points

- Catch block will execute exception occurs in try block.
- You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.
- At a time only one catch block will execute out of multiple catch blocks.
- in catch block you declare an object of sub class and it will be internally referenced by JVM.

Example without Exception Handling

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
ans=a/0;
System.out.println("Denominator not be zero");
}
}
```

Abnormally terminate program and give a message like below, this error message is not understandable by user so we convert this error message into user friendly error message, like "denominator not be zero".



```
C:\Windows\system32\cmd.exe

D:\java>javac ExceptionDemo.java

D:\java>java ExceptionDemo
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExceptionDemo.main(ExceptionDemo.java:8)

D:\java>
```

Example with Exception Handling

```
class ExceptionDemo
{
public static void main(String[] args)
{
```

```

int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
}
}

```

Output

Denominator not be zero

Multiple catch block

You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.

Example

```

import java.util.*;
class ExceptionDemo
{
public static void main(String[] args)
{
int a, b, ans=0;
Scanner s=new Scanner(System.in);
System.out.println("Enter any two numbers: ");
try
{
a=s.nextInt();
b=s.nextInt();
ans=a/b;
System.out.println("Result: "+ans);
}
catch(ArithmeticException ae)
{
System.out.println("Denominator not be zero");
}
catch(Exception e)
{
System.out.println("Enter valid number");
}
}
}

```

Output

Enter any two number: 5 0

Denominator not be zero

3. finally block

Inside finally block we write the block of statements which will relinquish (released or close or terminate) the resource (file or database) where data store permanently.

finally block important points

- Finally block will execute compulsory
- Writing finally block is optional.
- You can write finally block for the entire java program
- In some of the circumstances one can also write try and catch block in finally block.

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
finally
{
System.out.println("I am from finally block");
}
}
```

Output

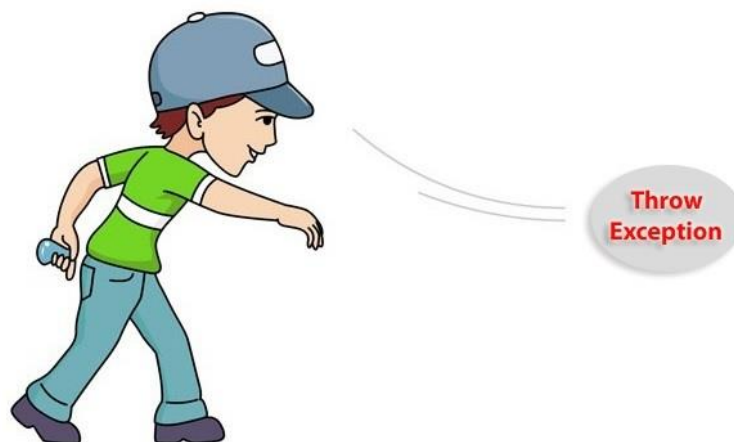
Denominator not be zero
I am from finally block

4. throw

throw is a keyword in java language which is used to throw any user defined exception to the same signature of method in which the exception is raised.

Note: throw keyword always should exist within method body.

whenever method body contain throw keyword than the call method should be followed by throws keyword.



Syntax

```
class className
{
```

```
returntype method(...) throws Exception_class
{
throw(Exception obj)
}
}
```

5. throws

throws is a keyword in java language which is used to throw the exception which is raised in the called method to its calling method throws keyword always followed by method signature.

Example

```
returnType methodName(parameter)throws Exception_class....
```

```
{
.....
}
```

Difference between throw and throws

	throw	throws
1	throw is a keyword used for hitting and generating the exception which are occurring as a part of method body	throws is a keyword which gives an indication to the specific method to place the common exception methods as a part of try and catch block for generating user friendly error messages
2	The place of using throw keyword is always as a part of method body.	The place of using throws is a keyword is always as a part of method heading
3	When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading	When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body.

Example of throw and throws

```
// save by DivZero.java

package pack;

public class DivZero
{
public void division(int a, int b)throws ArithmeticException
{
if(b==0)
{
ArithmeticException ae=new ArithmeticException("Does not enter zero for Denominator");
throw ae;
}
else
{
int c=a/b;
System.out.println("Result: "+c);
}
}
}
```

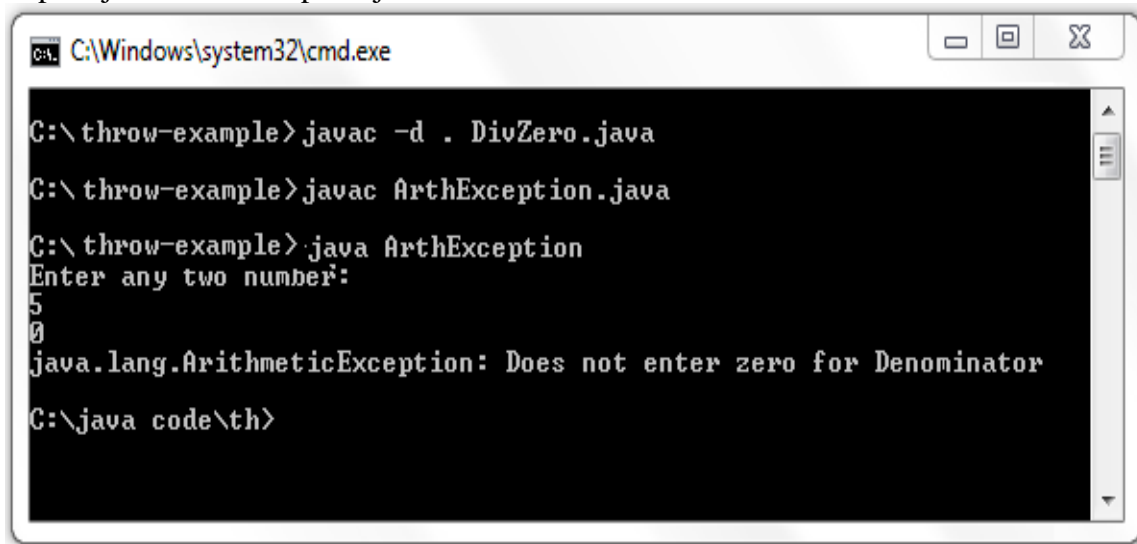
Compile: javac -d . DivZero.java

```
// save by ArthException.java

import pack.DivZero;
import java.util.*;

class ArthException
{
public static void main(String args[])
{
System.out.println("Enter any two number: ");
Scanner s=new Scanner(System.in);
try
{
int a=s.nextInt();
int b=s.nextInt();
DivZero dz=new DivZero();
dz.division(a, b);
}
catch(Exception e)
{
System.err.println(e);
}
}
}
```

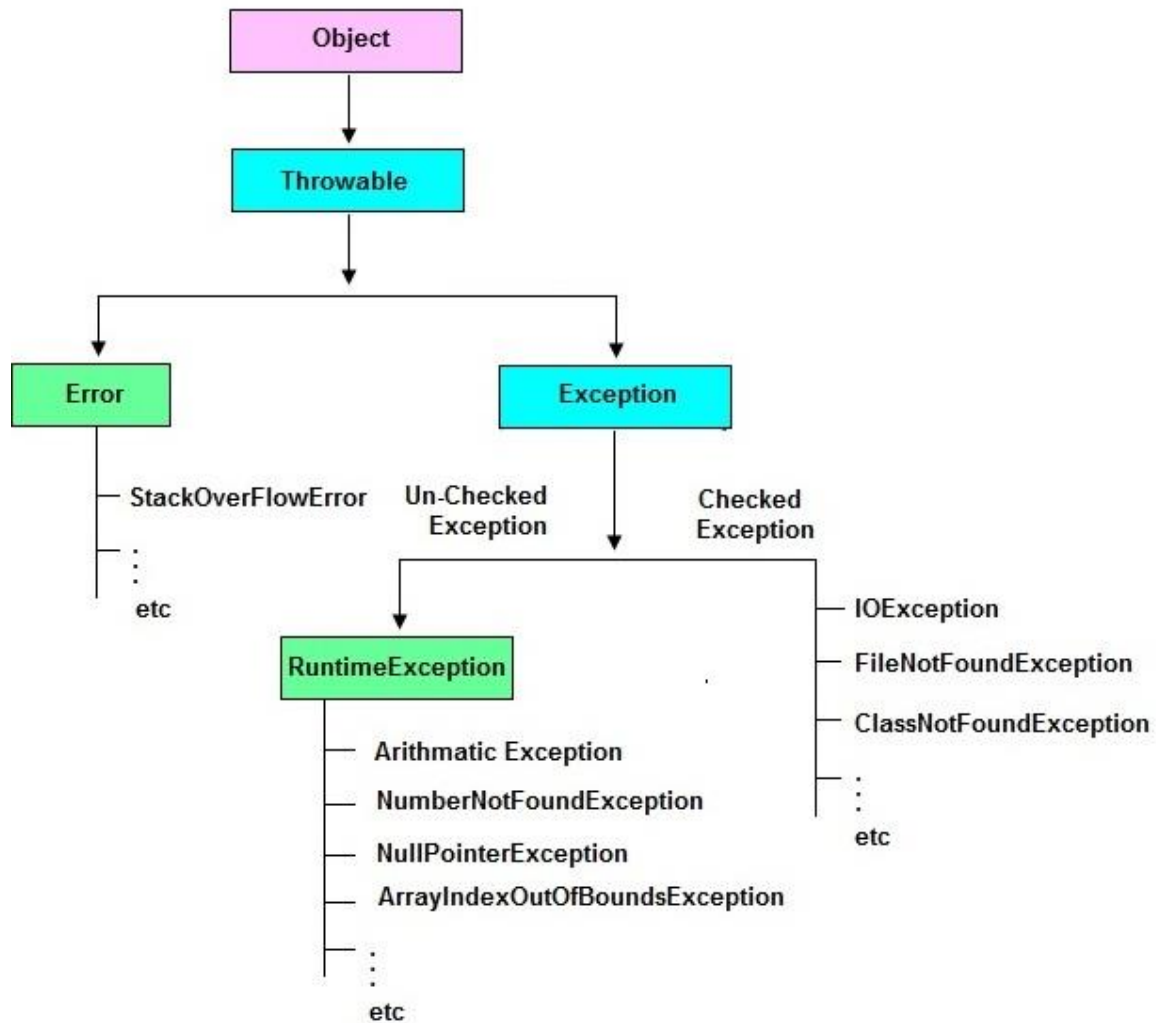
Compile: javac ArthException.java



```
C:\Windows\system32\cmd.exe

C:\>throw-example>javac -d . DivZero.java
C:\>throw-example>javac ArthException.java
C:\>throw-example>java ArthException
Enter any two number:
5
0
java.lang.ArithmeticException: Does not enter zero for Denominator
C:\>java code\th>
```

Hierarchy of Exception classes



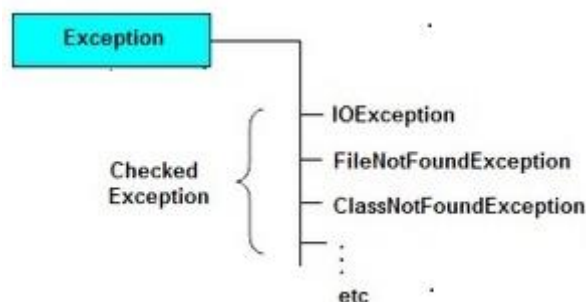
Types of Exceptions

- Checked Exception
- Un-Checked Exception

Checked Exception

Checked Exception are the exception which checked at compile-time. These exception are directly sub-class of java.lang.Exception class.

Only for remember: Checked means checked by compiler so checked exception are checked at compile-time.



Checked Exception Classes

- FileNotFoundException
- ClassNotFoundException
- IOException

- InterruptedException

FileNotFoundException

If the given filename is not available in a specific location (in file handling concept) then FileNotFoundException will be raised. This exception will be thrown by the FileInputStream, FileOutputStream, and RandomAccessFile constructors.

ClassNotFoundException

If the given class name is not existing at the time of compilation or running of program then ClassNotFoundException will be raised. In other words this exception is occurred when an application tries to load a class but no definition for the specified class name could be found.

IOException

This exception is raised whenever problem occurred while writing and reading the data in the file. This exception is occurred due to following reason;

- When try to transfer more data but less data are present.
- When try to read data which is corrupted.
- When try to write on file but file is read only.

InterruptedException

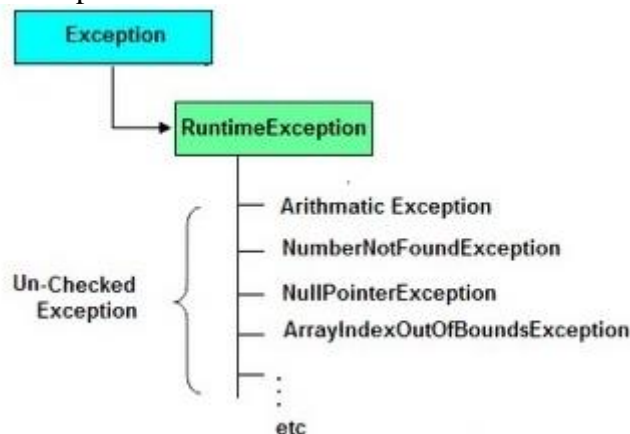
This exception is raised whenever one thread is disturb the other thread. In other words this exception is thrown when a thread is waiting, sleeping, or otherwise occupied, and the thread is interrupted, either before or during the activity.

Un-Checked Exception

Un-Checked Exception are the exception both identifies or raised at run time. These exception are directly sub-class of java.lang.RuntimeException class.

Note: In real time application mostly we can handle un-checked exception.

Only for remember: Un-checked means not checked by compiler so un-checked exception are checked at run-time not compile time.



Un-Checked Exception Classes

- ArithmeticException
- ArrayIndexOutOfBoundsException
- StringIndexOutOfBoundsException
- NumberFormatException
- NullPointerException
- NoSuchMethodException
- NoSuchFieldException

ArithmeticException

This exception is raised because of problem in arithmetic operation like divide by zero. In other words this exception is thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero".

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
}
}
```

ArrayIndexOutOfBoundsException

This exception will be raised whenever given index value of an array is out of range. The index is either negative or greater than or equal to the size of the array.

Example

```
int a[]=new int[5];
a[10]=100; //ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException
```

This exception will be raised whenever given index value of string is out of range. The index is either negative or greater than or equal to the size of the array.

Example

```
String s="Hello";
s.charAt(3);
s.charAt(10); // Exception raised
charAt() is a predefined method of string class used to get the individual characters based on index value.
```

NumberFormatException

This exception will be raised whenever you trying to store any input value in the unauthorized datatype.

Example: Storing string value into int datatype.

Example

```
int a;
a="Hello";
Example
String s="hello";
int i=Integer.parseInt(s); //NumberFormatException
NoSuchMethodException
```

This exception will be raised whenever calling method is not existing in the program.

NullPointerException

A NullPointerException is thrown when an application is trying to use or access an object whose reference equals to null.

Example

```
String s=null;
```

```
System.out.println(s.length());//NullPointerException
```

StackOverflowException

This exception throw when full the stack because the recursion method are stored in stack area.

Difference between checked Exception and un-checked Exception

	Checked Exception	Un-Checked Exception
1	checked Exception are checked at compile time	un-checked Exception are checked at run time
3	e.g. FileNotFoundException, NumberFormatException etc.	e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Difference between Error and Exception

	Error	Exception
1	Can't be handle.	Can be handle.
2	Example: NoSuchMethodError OutOfMemoryError	Example: ClassNotFoundException NumberFormatException

Custom Exception in Java

If any exception is design by the user known as user defined or Custom Exception. Custom Exception is created by user.

Rules to design user defined Exception

1. Create a package with valid user defined name.
2. Create any user defined class.
3. Make that user defined class as derived class of Exception or RuntimeException class.
4. Declare parametrized constructor with string variable.
5. call super class constructor by passing string variable within the derived class constructor.
6. Save the program with public class name.java

```
// AgeException.java  
  
package pack;  
  
public class AgeException extends Exception  
{  
    public AgeException(String s)  
    {  
        super(s);  
    }  
}
```

Example

```
// save by AgeException.java  
package nage;
```

```
public class AgeException extends Exception  
{  
    public AgeException(String s)  
    {  
        super(s);  
    }  
}
```

Compile: javac -d . AgeException.java

Example

```
// save by CheckAge.java  
package nage;
```

```
public class CheckAge  
{  
    public void verify(int age)throws AgeException  
    {  
        if (age>0)  
        {  
            System.err.print("valid age");  
        }  
        else  
        {  
            AgeException ae=new AgeException("Invalid age");  
            throw(ae);  
        }  
    }  
}
```

Compile: javac -d . CheckAge.java

Example

```
// save by VerifyAgeException
```

```
import nage.AgeException;  
import nage.CheckAge;  
import java.util.*;
```

```
public class VerifyAgeException  
{  
    public static void main(String args[])  
    {  
        int a;  
        System.out.println("Enter your age");  
        Scanner s=new Scanner(System.in);  
        a=s.nextInt();  
        try  
        {  
            CheckAge ca=new CheckAge();
```

```

ca.verify(a);
}
catch(AgeException ae)
{
System.err.println("Age should not be -ve");
}
catch(Exception e)
{
System.err.println(e);
}
}
}
}

```

Compile: javac VerifyAgeException.java

```

C:\Windows\system32\cmd.exe

C:\verify-age>javac -d . AgeException.java
C:\verify-age>javac -d . CheckAge.java
C:\verify-age>javac VerifyAgeException.java
C:\verify-age>java VerifyAgeException
Enter your age
-4
Age should not be -ve
C:\verify-age>

```

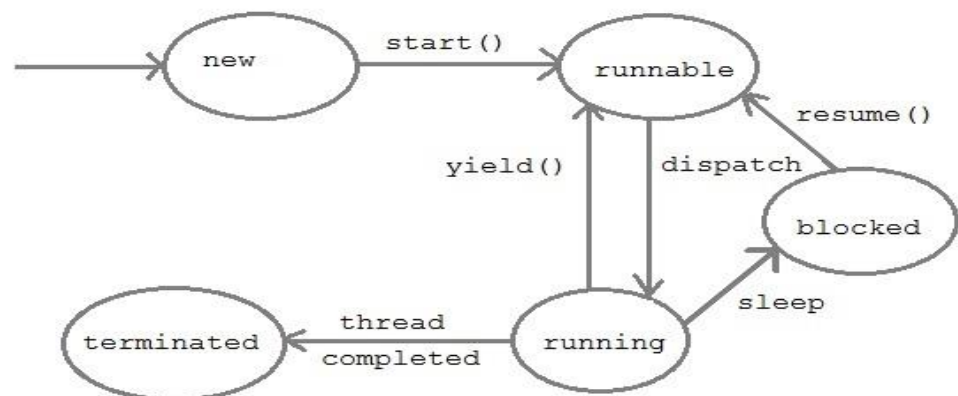
Multithreading in Java

Multithreading in java is a process of executing multiple threads simultaneously. The aim of multithreading is to achieve the concurrent execution.

Thread

Thread is a lightweight components and it is a flow of control. In other words a flow of control is known as thread.

Life cycle of a Thread



1. New : A thread begins its life cycle in the new state. It remains in this state until the start() method is called on it.
2. Runnable : After invocation of start() method on new thread, the thread becomes runnable.
3. Running : A thread is in running state if the thread scheduler has selected it.
4. Waiting : A thread is in waiting state if it waits for another thread to perform a task. In this stage the thread is still alive.
5. Terminated : A thread enter the terminated state when it complete its task.

The *main* thread

When we run any java program, the program begins to execute its code starting from the main method. Therefore, the JVM creates a thread to start executing the code present in main method. This thread is called as main thread. Although the main thread is automatically created, you can control it by obtaining a reference to it by calling `currentThread()` method.

Two important things to know about main thread are,

- It is the thread from which other threads will be produced.
- main thread must be always the last thread to finish execution.

```
class MainThread
{
    public static void main(String[] args)
    {
        Thread t=Thread.currentThread();
        t.setName("MainThread");
        System.out.println("Name of thread is "+t);
    }
}
```

Result:*Name of thread is Thread[MainThread,5,main]*

Thread Priorities

Every thread has a priority that helps the operating system determine the order in which threads are scheduled for execution. In java thread priority ranges between 1 to 10,

- MIN-PRIORITY (a constant of 1)
- MAX-PRIORITY (a constant of 10)

By default every thread is given a NORM-PRIORITY(5). The main thread always have NORM-PRIORITY.

Note: Thread priorities cannot guarantee that a higher priority thread will always be executed first than the lower priority thread. The selection of the threads for execution depends upon the thread scheduler which is platform dependent.

Thread Class

Thread class is the main class on which Java's Multithreading system is based. Thread class, along with its companion interface Runnable will be used to create and run threads for utilizing Multithreading feature of Java.

Constructors of Thread class

1. Thread ()

2. Thread (*String str*)
3. Thread (*Runnable r*)
4. Thread (*Runnable r, String str*)

Thread class also defines many methods for managing threads. Some of them are,

Method	Description
setName()	to give thread a name
getName()	return thread's name
getPriority()	return thread's priority
isAlive()	checks if thread is still running or not
join()	Wait for a thread to end
run()	Entry point for a thread
sleep()	suspend thread for a specified time
start()	start a thread by calling run() method

Some Important points to Remember

1. When we extend Thread class, we cannot override setName() and getName() functions, because they are declared final in Thread class.
2. While using sleep(), always handle the exception it throws.

static void sleep(long milliseconds) throws InterruptedException

Thread Class Methods Example Program

```
class UserDefinedThread extends Thread
{
public void run()
{
}
}
class ThreadMethods
{
public static void main(String args[])
{
UserDefinedThread ut=new UserDefinedThread();
System.out.println("Initial Name of Thread:"+ut.getName());
System.out.println("Initial Priority of Thread:"+ut.getPriority());
System.out.println("Initial Daemon State of Thread:"+ut.isDaemon());
System.out.println("Initial Aliveness of Thread:"+ut.isAlive());
ut.setName("SREC");
ut.setPriority(10);
ut.setDaemon(true);
ut.start();
System.out.println("Post Name of Thread:"+ut.getName());
System.out.println("Post Priority of Thread:"+ut.getPriority());
System.out.println("Post Daemon State of Thread:"+ut.isDaemon());
System.out.println("Post Aliveness of Thread:"+ut.isAlive());
}
}
```

Result:

```
Initial Name of Thread:Thread-0
Initial Priority of Thread:5
Initial Daemon State of Thread:false
Initial Aliveness of Thread:false
Post Name of Thread:SREC
Post Priority of Thread:10
Post Daemon State of Thread:true
Post Aliveness of Thread:false
```

Creating a thread

Java defines two ways by which a thread can be created.

1. By implementing the Runnable interface.
2. By extending the Thread class.

1. Implementing the Runnable Interface

The easiest way to create a thread is to create a class that implements the runnable interface. After implementing runnable interface, the class needs to implement the run () method, which is of form,

```
public void run()
```

- run() method introduces a concurrent thread into your program. This thread will end when run() method terminates.
- You must specify the code that your thread will execute inside run() method.
- run() method can call other methods, can use other classes and declare variables just like any other normal method.

```
class MyThread implements Runnable
{
    public void run()
    {
        System.out.println("concurrent thread started running..");
    }
}
```

```
class MyThreadDemo
{
    public static void main( String args[] )
    {
        MyThread mt = new MyThread();
        Thread t = new Thread(mt);
        t.start();
    }
}
```

concurrent thread started running..

To call the run() method, start() method is used. On calling start(), a new stack is provided to the thread and run() method is called to introduce the new thread into the program.

Note: If you are implementing Runnable interface in your class, then you need to explicitly create a Thread class object and need to pass the Runnable interface implemented class object as a parameter in its constructor.

2. Extending Thread class

This is another way to create a thread by a new class that extends Thread class and create an instance of that class. The extending class must override run () method which is the entry point of new thread.

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("concurrent thread started running..");
    }
}
```

```
class MyThreadDemo
{
    public static void main( String args[] )
    {
        MyThread mt = new MyThread();
        mt.start();
    }
}
```

```
}  
}
```

concurrent thread started running..

In this case also, we must override the run() and then use the start() method to run the thread. Also, when you create MyThread class object, Thread class constructor will also be invoked, as it is the super class, hence MyThread class object acts as Thread class object.

Joining threads

Sometimes one thread needs to know when other thread is terminating.

In java, isAlive() and join() are two different methods that are used to check whether a thread has finished its execution or not.

The isAlive() method returns true if the thread upon which it is called is still running otherwise it returns false.

final boolean isAlive()

But, join() method is used more commonly than isAlive(). This method waits until the thread on which it is called terminates.

final void join() throws InterruptedException

Using join() method, we tell our thread to wait until the specified thread completes its execution. There are overloaded versions of join() method, which allows us to specify time for which you want to wait for the specified thread to terminate.

final void join(long milliseconds) throws InterruptedException

As we know, the main thread must always be the last thread to finish its execution, we can use Thread join() method to ensure that all the threads created by the program have been terminated before the execution of the main thread.

Example of thread without `join()` method

```
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("r1 ");
        try {
            Thread.sleep(500);
        }
        catch (InterruptedException ie){ }
        System.out.println("r2 ");
    }
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t2.start();
    }
}
```

```
r1
r1
r2
r2
```

In this above program two thread t1 and t2 are created. t1 starts first and after printing "r1" on console thread t1 goes to sleep for 500 ms. At the same time Thread t2 will start its process and print "r1" on console and then go into sleep for 500 ms. Thread t1 will wake up from sleep and print "r2" on console similarly thread t2 will wake up from sleep and print "r2" on console. So you will get output like r1 r1 r2 r2

Example of thread with join() method

```
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("r1 ");
        try {
            Thread.sleep(500);
        } catch (InterruptedException ie) { }
        System.out.println("r2 ");
    }
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();

        try{
            t1.join();    //Waiting for t1 to finish
        } catch (InterruptedException ie){}

        t2.start();
    }
}
```

```
r1
r2
r1
r2
```

In this above program join() method on thread t1 ensures that t1 finishes its process before thread t2 starts.

Specifying time with join()

If in the above program, we specify time while using join() with t1, then t1 will execute for that time, and then t2 will join it.

```
t1.join(1500);
```

Doing so, initially t1 will execute for 1.5 seconds, after which t2 will join it.

Synchronization

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax:

```
synchronized (object)
```

```
{  
//statement to be synchronized  
}
```

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

Why we use Synchronization?

If we do not use synchronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Consider an example, suppose we have two different threads T1 and T2, T1 starts execution and save certain values in a file *temporary.txt* which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file *temporary.txt* (*temporary.txt* is the shared resource). Now obviously T1 will return wrong result.

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using *temporary.txt* file, this file will be locked (LOCK mode), and no other thread will be able to access or modify it until T1 returns.

Using Synchronized Methods

Using Synchronized methods is a way to accomplish synchronization. But lets first see what happens when we do not use synchronization in our program.

Example with no Synchronization

```
class First
{
    public void display(String msg)
    {
        System.out.print ("["+msg);
        try
        {
            Thread.sleep(1000);
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
        System.out.println ("]");
    }
}

class Second extends Thread
{
    String msg;
    First fobj;
    Second (First fp,String str)
    {
        fobj = fp;
        msg = str;
        start();
    }
    public void run()
    {
        fobj.display(msg);
    }
}

public class Syncro
{
    public static void main (String[] args)
    {
        First fnew = new First();
        Second ss = new Second(fnew, "welcome");
        Second ss1= new Second (fnew,"new");
        Second ss2 = new Second(fnew, "programmer");
    }
}
```

```
[welcome [ new [ programmer]
]
]
```

In the above program, object fnew of class First is shared by all the three running threads(ss, ss1 and ss2) to call the shared method(*void* display). Hence the result is unsynchronized and such situation is called Race condition.

Synchronized Keyword

To synchronize above program, we must *synchronize* access to the shared display() method, making it available to only one thread at a time. This is done by using keyword synchronized with display() method.

synchronized void display (String msg)

Using Synchronized block

If you have to synchronize access to an object of a class or you only want a part of a method to be synchronized to an object then you can use synchronized block for it.

```
class First
{
    public void display(String msg)
    {
        System.out.print ("["+msg);
        try
        {
            Thread.sleep(1000);
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
        System.out.println ("]");
    }
}

class Second extends Thread
{
    String msg;
    First fobj;
    Second (First fp,String str)
    {
        fobj = fp;
        msg = str;
        start();
    }
    public void run()
    {
        synchronized(fobj)    //Synchronized block
        {
            fobj.display(msg);
        }
    }
}
```



```

public class Syncro
{
    public static void main (String[] args)
    {
        First fnew = new First();
        Second ss = new Second(fnew, "welcome");
        Second ss1= new Second (fnew,"new");
        Second ss2 = new Second(fnew, "programmer");
    }
}

```

```

[welcome]
[new]
[programmer]

```

Because of synchronized block this program gives the expected output.

Difference between synchronized keyword and synchronized block

When we use synchronized keyword with a method, it acquires a lock in the object for the whole method. It means that no other thread can use any synchronized method until the current thread, which has invoked it's synchronized method, has finished its execution.

Synchronized block acquires a lock in the object only between parentheses after the synchronized keyword. This means that no other thread can acquire a lock on the locked object until the synchronized block exits. But other threads can access the rest of the code of the method.

Which is more preferred - Synchronized method or synchronized block?

In Java, synchronized keyword causes a performance cost. A synchronized method in Java is very slow and can degrade performance. So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

Inter Thread Communication

Java provides benefits of avoiding thread pooling using inter-thread communication.

The wait(), notify(), and notifyAll() methods of Object class are used for this purpose. These methods are implemented as final methods in Object, so that all classes have them. All the three method can be called only from within a synchronized context.

- wait() tells calling thread to give up monitor and go to sleep until some other thread enters the same monitor and call notify.
- notify() wakes up a thread that called wait() on same object.
- notifyAll() wakes up all the thread that called wait() on same object.

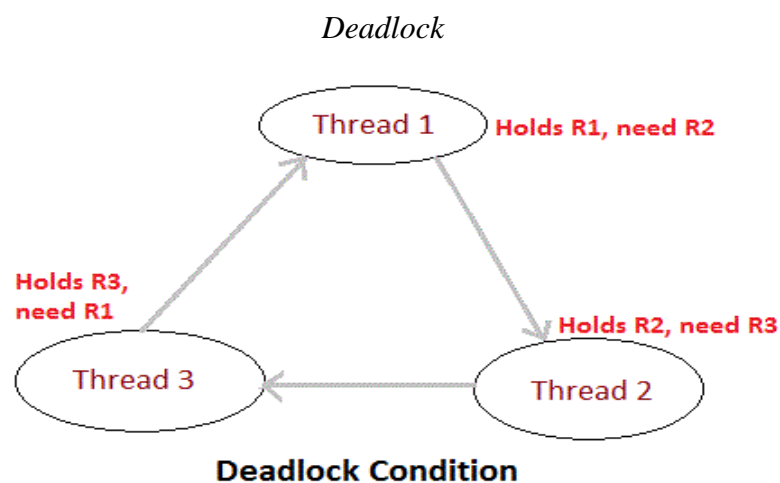
Difference between wait() and sleep()

wait()	sleep()
called from synchronized block	no such requirement
monitor is released	monitor is not released

gets awake when notify() or notifyAll() method is called.	does not get awake when notify() or notifyAll() method is called
not a static method	static method
wait() is generally used on condition	sleep() method is simply used to put your thread on sleep.

Thread Pooling

Pooling is usually implemented by loop i.e. to check some condition repeatedly. Once condition is true appropriate action is taken. This wastes CPU time.



Deadlock is a situation of complete Lock, when no thread can complete its execution because lack of resources. In the above picture, Thread 1 is holding a resource R1, and need another resource R2 to finish execution, but R2 is locked by Thread 2, which needs R3, which in turn is locked by Thread 3. Hence none of them can finish and are stuck in a deadlock.

ThreadGroup in Java

Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.

Note: Now suspend(), resume() and stop() methods are deprecated.

Java thread group is implemented by *java.lang.ThreadGroup* class.

Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

No.	Constructor	Description
1)	ThreadGroup(String name)	creates a thread group with given name.
2)	ThreadGroup(ThreadGroup parent, String name)	creates a thread group with given parent group and name.

Important methods of ThreadGroup class

There are many methods in ThreadGroup class. A list of important methods are given below.

No.	Method	Description
-----	--------	-------------

1)	int activeCount()	returns no. of threads running in current group.
2)	int activeGroupCount()	returns a no. of active group in this thread group.
3)	void destroy()	destroys this thread group and all its sub groups.
4)	String getName()	returns the name of this group.
5)	ThreadGroup getParent()	returns the parent of this group.
6)	void interrupt()	interrupts all threads of this group.
7)	void list()	prints information of this group to standard console.

Let's see a code to group multiple threads.

1. ThreadGroup tg1 = new ThreadGroup("Group A");
2. Thread t1 = new Thread(tg1,new MyRunnable(),"one");
3. Thread t2 = new Thread(tg1,new MyRunnable(),"two");
4. Thread t3 = new Thread(tg1,new MyRunnable(),"three");

Now all 3 threads belong to one group. Here, tg1 is the thread group name, MyRunnable is the class that implements Runnable interface and "one", "two" and "three" are the thread names.

Now we can interrupt all threads by a single line of code only.

1. Thread.currentThread().getThreadGroup().interrupt();

ThreadGroup Example

File: ThreadGroupDemo.java

```
public class ThreadGroupDemo implements Runnable{
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String[] args) {
        ThreadGroupDemo runnable = new ThreadGroupDemo();
        ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");

        Thread t1 = new Thread(tg1, runnable,"one");
        t1.start();
        Thread t2 = new Thread(tg1, runnable,"two");
        t2.start();
        Thread t3 = new Thread(tg1, runnable,"three");
        t3.start();

        System.out.println("Thread Group Name: "+tg1.getName());
        tg1.list();

    }
}
```

Output:

one

two

three

Thread Group Name: Parent ThreadGroup

java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]

Thread[one,5,Parent ThreadGroup]

Thread[two,5,Parent ThreadGroup]

Thread[three,5,Parent ThreadGroup]

Fill in the Blanks

1. An _____ is an object that is generated when a run time error occurs.
2. Thread is a _____
3. Program statements that must be monitored are kept in _____ block.
4. _____ block must immediately followed by a try block
5. Any Exception that is thrown out of a method must be specified by _____ clause.
6. To throw an Exception manually, _____ clause is used.
7. Any code that absolutely must be executed before a method returns is put in a _____ block.
8. All Exception types are subclasses of the built-in class _____.
9. Throwable have _____ & _____ as sub classes.
10. Every try block must associate with a _____ block.
11. The _____ statement can be nested in Exception Handling.
12. Each try statement requires at least one _____ or a _____ clause.
13. Using thread's priority to decide when to switch from one running thread to the next is known as _____.
14. _____ occurs when two threads have a circular dependency on a pair of synchronized objects.
15. _____ in java is a service provider thread that provides services to the user thread.
16. Java thread group is implemented by _____ class.
17. There are _____ number of constructors are available for ThreadGroup class.
18. The thread is in _____ state after invocation of start() method
19. A thread is in terminated or dead state when its _____ method exits.
20. _____ class constructor allocates a new thread object.

Short Answer Questions		
S. No	Questions	Blooms Taxonomy Level
1	Define Exception.	Understand
2	Distinguish between exception and error.	Understand
3	Describe the benefits of exception handling.	Remember
4	State the classification of exceptions.	Remember
5	Define checked exceptions.	Understand
6	State the use of try and catch blocks.	Remember
7	Define built in exception.	Understand
8	Define thread in java.	Understand
9	Compare and contrast between process and thread.	Understand
10	List the various ways of creating thread.	Remember
11	Define unchecked exceptions.	Understand
12	Describe the various states of threads.	Remember
13	List the different ways to create a thread.	Remember
14	Differentiate throw and finally.	Understand
15	Define inter-thread communication.	Understand
16	Explain about the alive() and join() method	Understand

17	Interpret the different thread priorities	Understand
18	Distinguish between throw and throws.	Understand
19	Define wait() state of the thread	Understand
20	Describe about “thread class implements Runnable interface”	Remember

Short Answer Questions		
S. No	Questions	Blooms Taxonomy Level
1	Explain briefly about exception handling mechanism with suitable examples.	Understand
2	Describe try, catch , and finally keywords with an example	Remember
3	Illustrate use of throws keyword with a program	Remember
4	Define a exception called “NotEqualException” that is thrown when a float value is not equal to 3.14. write a program that uses the above user	Understand
5	Differentiate checked and unchecked exceptions.	Understand
6	Exemplify the different type of exception.	Understand
7	Illustrate built in exceptions with suitable example.	Understand
8	Describe the producer consumer problem with an example	Remember
9	Explain with an example how java performs thread synchronization.	Understand
10	Differentiate multiprocessing and multithreading with a program.	Understand
11	Explain briefly about the life cycle of a thread with an example.	Understand
12	Interpret various methods of thread class.	Understand
13	Describe a java program using thread priorities.	Remember
14	Explain Daemon threads with an example.	Understand
15	Exemplify the behavior of thread using thread class methods.	Understand
16	Illustrate the process of creating thread by implementing Runnable interface	Remember

Problem Solving and Critical Thinking Questions		
S. No	Questions	Blooms Taxonomy Level
1	<p>Analyze the output of program</p> <pre> public class TestMultipleCatchBlock { public static void main(String args[]) { try{ int a[]=new int[5]; a[5]=30/0; } catch(ArithmeticException e) { System.out.println("task1 is completed"); } catch(ArrayIndexOutOfBoundsException e) </pre>	Understand

	<pre> { System.out.println("task 2 completed"); } catch(Exception e) { System.out.println("common task completed"); } System.out.println("rest of the code..."); } } </pre>	
2	<p>Analyze the program and find out the output</p> <pre> ? public class Test { public static void aMethod() throws Exception { try { throw new Exception(); } finally { System.out.print("finally "); } } public static void main(String args[]) { try { aMethod(); } catch (Exception e) { System.out.print("exception "); } System.out.print("finished"); } } </pre>	Understand
3	<p>Identify the output of the following program? class s1 implements Runnable</p> <pre> { int x = 0, y = 0; int addX() { x++; return x; } int addY() { y++; return y; } } </pre>	Understand

	<pre> } public void run() { for(int i = 0; i < 10; i++) System.out.println(addX() + " " + addY()); } public static void main(String args[]) { s1 run2 = new s1(); Thread t1 = new Thread(run1); Thread t2 = new Thread(run2); t1.start(); t2.start(); } </pre>	
4	<p>Interpret the output of following program?</p> <pre> class Exceptions { public static void main(String[] args) { String languages[] = { "C", "C++", "Java", "Perl", "Python" }; try { for (int c = 1; c <= 5; c++) { System.out.println(languages[c]); } } catch (Exception e) { System.out.println(e); } } } </pre>	Understand
5	<p>Analyze the output of the below program?</p> <pre> class Allocate { public static void main(String[] args) { try { long data[] = new long[10000000000]; } catch (Exception e) { System.out.println(e); } Finally { System.out.println("finally block will execute always."); } } } </pre>	Understand

	<pre> } } </pre>	
6	<p>Identify the output of the program?</p> <pre> class MyThread extends Thread { public static void main(String [] args) { MyThread t = new MyThread(); Thread x = new Thread(t); x.start(); } public void run() { for(int i = 0; i < 3; ++i) { System.out.print(i + ".."); } } } </pre>	Understand
7	<p>Interpret the output of the program?</p> <pre> public class RTExcept { public static void throwit () { System.out.print("throwit "); throw new RuntimeException(); } public static void main(String [] args) { try { System.out.print("hello "); throwit(); } catch (Exception re) { System.out.print("caught "); } finally { System.out.print("finally "); } System.out.println("after "); } } </pre>	Understand
8	<p>Analyze the program and find the output</p> <pre> public class NFE { public static void main(String [] args) { String s = "42"; try { </pre>	Remember

	<pre> s = s.concat(".5"); double d = Double.parseDouble(s); s = Double.toString(d); int x = (int) Math.ceil(Double.valueOf(s).doubleValue()); System.out.println(x); } catch (NumberFormatException e) { System.out.println("bad number"); } } </pre>	
9	<p>Identify the output of the program?</p> <pre> class MyThread extends Thread { MyThread() { System.out.print(" MyThread"); } public void run() { System.out.print(" bar"); } public void run(String s) { System.out.println(" baz"); } } public class TestThreads { public static void main (String [] args) { Thread t = new MyThread() { public void run() { System.out.println(" foo"); } } t.start(); } } </pre>	Understand
10	<p>Analyze the output of the program?</p> <pre> class implements Runnable { int x, y; public void run() { for(int i = 0; i < 1000; i++) synchronized(this) </pre>	Remember

	<pre> { x = 12; y = 12; } System.out.print(x + " " + y + " "); } public static void main(String args[]) { s run = new s(); Thread t1=new Thread(run); Thread t2=new Thread(run); t1.start(); t2.start(); } } </pre>	
--	---	--

Review Questions

1. What value does read() return when it has reached the end of a file?
2. Can a Byte object be cast to a double value?
3. What is an object's lock and which object's have locks?
4. What is an Exception
5. What is the purpose of the **throw** and **throws** keywords?
6. How can you handle an exception?
7. How can you catch multiple exceptions?
8. What is the difference between a checked and an unchecked exception?
9. What is the difference between an exception and error?
10. What are the exception handling keywords in java?
11. What are the important methods of Exception class
12. What are different scenarios causing "Exception in thread main"?
13. What is difference between final, finally and finalize in Java?
14. What happens when exception is thrown by main method?
15. Can we have an empty catch block?
16. What is the difference between Process and Thread?
17. What are the benefits of multi-threaded programming?
18. What is difference between user Thread and daemon Thread?
19. How can we create a Thread in Java?
20. What are different states in lifecycle of Thread?
21. Can we call run() method of a Thread class?
22. How can we pause the execution of a Thread for specific time?
23. What do you understand about Thread Priority?
24. What is context-switching in multi-threading?
25. How can we make sure main() is the last thread to finish in Java Program?
26. How does thread communicate with each other?
27. Why thread communication methods wait(), notify() and notifyAll() are in Object class?

28. Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?
29. Why Thread sleep() and yield() methods are static?
30. How can we achieve thread safety in Java?
31. Which is more preferred – Synchronized method or Synchronized block?
32. How to create daemon thread in Java?
33. What is Thread Group? Why it's advised not to use it?
34. What is Deadlock? How to analyze and avoid deadlock situation?
35. What is Thread Pool? How can we create Thread Pool in Java?
36. What will happen if we don't override Thread class run() method?

UNIT IV

UNIT WISE PLAN

UNIT-IV: Applets, Event Handling and AWT		Planned Hours: 09	
S. No.	Topic Learning Outcomes	Cos	Blooms Levels
1	Understand the difference between Applications and Applets	CO5	L2
2	Develop applications with some event handling	CO5	L6
3	Use AWT controls in GUI Programming	CO5	L6
4	Use different layouts (Flow Layout, Boarder Layout, Grid Layout, Card Layout) to position the controls for developing graphical user interface applications.	CO5	L6

Applet in Java

- Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as a part of a web document.
- After a user receives an applet, the applet can produce a graphical user interface. It has limited access to resources so that it can run complex computations without introducing the risk of viruses or breaching data integrity.
- Any applet in Java is a class that extends the java.applet.Applet class.
- An Applet class does not have any main() method. It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.
- JVM creates an instance of the applet class and invokes init() method to initialize an Applet.

A Simple Applet

```
import java.awt.*;
import java.applet.*;
public class Simple extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("A simple Applet", 20, 20);
    }
}
```

Every Applet application must import two packages - java.awt and java.applet. java.awt.* imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user (either directly or indirectly) through the AWT. The AWT contains support for a window-based, graphical user interface. java.applet.* imports the applet package, which contains the class Applet. Every applet that you create must be a subclass of Applet class.

The class in the program must be declared as public, because it will be accessed by code that is outside the program. Every Applet application must declare a paint() method. This method is defined by AWT class and must be overridden by the applet. The paint() method is called each time when an applet needs to redisplay its output. Another

important thing to notice about applet application is that, execution of an applet does not begin at main() method. In fact an applet application does not have any main() method.

Advantages of Applets

1. It takes very less response time as it works on the client side.
2. It can be run on any browser which has JVM running in it.

Differences between Applets and Applications

An applet runs under the control of a browser, whereas an application runs stand-alone, with the support of a virtual machine. As such, an applet is subjected to more stringent security restrictions in terms of file and network access, whereas an application can have free reign over these resources.

Applets are great for creating dynamic and interactive web applications, but the true power of Java lies in writing full blown applications. With the limitation of disk and network access, it would be difficult to write commercial applications (though through the user of server based file systems, not impossible). However, a Java application has full network and local file system access, and its potential is limited only by the creativity of its developers.

Types of Applets: Web pages can contain two types of applets which are named after the location at which they are stored.

1. Local Applet
2. Remote Applet

Local Applets: A local applet is the one that is stored on our own computer system.

When the Web-page has to find a local applet, it doesn't need to retrieve information from the Internet. A local applet is specified by a path name and a file name as shown below in which the codebase attribute specifies a path name, whereas the code attribute specifies the name of the byte-code file that contains the applet's code.

```
<applet codebase="MyAppPath" code="MyApp.class" width=200 height=200> </applet>
```

Remote Applets: A remote applet is the one that is located on a remote computer system. This computer system may be located in the building next door or it may be on the other side of the world. No matter where the remote applet is located, it's downloaded onto our computer via the Internet. The browser must be connected to the Internet at the time it needs to display the remote applet. To reference a remote applet in Web page, we must know the applet's URL (where it's located on the Web) and any attributes and parameters that we need to supply. A local applet is specified by a url and a file name as shown below.

```
<applet codebase="http://www.srecwarangal.ac.in" code="MyApp.class" width=200 height=200> </applet>
```

Applet class

Applet class provides all necessary support for applet execution, such as initializing and destroying of applet. It also provides methods that load and display images and methods that load and play audio clips.

Applet Class Hierarchy

As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Life cycle of applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life

cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. public void init(): is used to initialize the Applet. It is invoked only once.
2. public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Note: The stop() method is always called before destroy() method.

Running of applet programs

Applet program can run in two ways.

- ☐ Using html (in the web browser)
- ☐ Using appletviewer tool (in applet window)

Running of applet using html

In general no java program can directly execute on the web browser except markup language like

html, xml etc.

Html supports a predefined tag called <applet> to load the applet program on the browser window.

Example of applet program to run applet using html

Java code, JavaApp.java

```
import java.applet.*;
import java.awt.*;
public class JavaApp extends Applet
{
    public void paint(Graphics g)
    {
        Font f=new Font("Arial",Font.BOLD,30);
        g.setFont(f);
        setForeground(Color.red);
        setBackground(Color.white);
        g.drawString("Student",200,200);
    }
}
```

Html code, myapplet.html

```
<html>
<title> AppletEx</Title>
<body>
<applet code="JavaApp.class"
height="70%"
width="80%">
```

```
</applet>
</body>
</html>
```

Running of applet using appletviewer

Some browser does not support <applet> tag so that Sun Microsystems was introduced a special

tool called appletviewer to run the applet program.

In this Scenario java program should contains <applet> tag in the commented lines so that appletviewer tools can run the current applet program.

Example of Applet

```
import java.applet.*;
import java.awt.*;
/*<applet code="LifeApp.class" height="500",width="800">
</applet>*/
public class LifeApp extends Applet
{
String s= " ";
public void init()
{
s=s+ " int ";
}
public void start()
{
s=s+ "start ";
}
public void stop()
{
s=s+ "stop ";
}
public void destroy()
{
s=s+ " destroy ";
}
public void paint(Graphics g)
{
Font f=new Font("Arial",Font.BOLD,30);
setBackgroundColor(Color."red");
g.setFont(f);
g.drawString(s,200,250);
}
}
```

Execution of applet program

```
javac LifeApp.java
```

```
appletviewer LifeApp.java
```

Note: init() always execute only once at the time of loading applet window and also it will be executed if applet is restarted.

Passing Parameter in Java Applet

Java applet has the feature of retrieving the parameter values passed from the html page. So, you can pass the parameters from your html page to the applet embedded in your page. The

param tag(<param name="" value=""></param>) is used to pass the parameters to an applet. For

the illustration about the concept of applet and passing parameter in applet, a example is given below.

In this example, we will see what has to be done in the applet code to retrieve the value from parameters. Value of a parameter passed to an applet can be retrieved using `getParameter()` function. E.g. code:

```
String strParameter = this.getParameter("Message");
```

Printing the value:

Then in the function `paint (Graphics g)`, we prints the parameter value to test the value passed from html page. Applet will display "Hello! Java Applet". if no parameter is passed to the applet

else it will display the value passed as parameter. In our case applet should display "Welcome in

Passing parameter in java applet example." message.

Here is the code for the Java Program :

```
import java.applet.*;
```

```
import java.awt.*;
```

```
public class appletParameter extends Applet
```

```
{
```

```
private String strDefault = "Hello! Java Applet.";
```

```
public void paint(Graphics g)
```

```
{
```

```
String strParameter = this.getParameter("Message");
```

```
if (strParameter == null)
```

```
strParameter = strDefault;
```

```
g.drawString(strParameter, 50, 25);
```

```
}
```

```
}
```

Here is the code for the html program:

```
<html>
```

```
<head>
```

```
<title>passing parameter in java applet</title>
```

```
</head>
```

```
<body>
```

```
this is the applet:<p>
```

```
<applet code="appletParameter.class" width="800" height="100">
```

```
<param name="message" value="welcome in passing parameter in java applet example.">
```

```
</applet>
```

```
</body>
```

```
</html>
```

There is the advantage that if need to change the output then you will have to change only the value of the param tag in html file not in java code.

Compile the program:

```
javac appletParameter.java
```

Output after running the program:

To run the program using `appletviewer`, go to command prompt and type `appletviewer`

`appletParameter.html` Appletviewer will run the applet for you and and it should show output like

Welcome in Passing parameter in java applet example. Alternatively you can also run this example from your favorite java enabled browser.

Event Handling

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven.

Event describes the change in state of any object.

For Example: Pressing a button, entering a character in Textbox, Clicking or dragging a mouse, etc.

Java uses the Delegation Event model technique to handle events in GUI (Graphical User Interface), which has the following components and process of event handling.

Components of Event Handling

Event handling has three main components,

- Event: An event is a change in state of an object.
- Events Source: Event source is an object that generates an event.
- Event Listener: A listener is an object that listens to the event. A listener gets notified when an event occurs.

How Events are handled?

A source generates an Event and sends it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return.

Events are supported by a number of Java packages, like java.util, java.awt and java.awt.event.

Important Event Classes and Interface

Event Classes Description Listener Interface

ActionEvent generated when button is pressed, menuitem is selected, list-item is double clicked ActionListener

MouseEvent

generated when mouse is dragged, moved, clicked, pressed or released and also when it enters or exit a component

MouseListener

KeyEvent generated when input is received from keyboard KeyListener

ItemEvent generated when check-box or list item is clicked ItemListener

TextEvent generated when value of textarea or textfield is changed TextListener

MouseEvent generated when mouse wheel is moved MouseWheelListener

WindowEvent

generated when window is activated, deactivated, deiconified, iconified, opened or closed

WindowListener

ComponentEvent generated when component is hidden, moved, resized or set visible ComponentEventListener

ContainerEvent generated when component is added or removed from container ContainerListener

AdjustmentEvent generated when scroll bar is manipulated AdjustmentListener

FocusEvent generated when component gains or loses
keyboard focus FocusListener

Example of Key Event Handling

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class Test extends Applet implements KeyListener
{
    String msg="";
    public void init()
    {
        addKeyListener(this);
    }
    public void keyPressed(KeyEvent k)
    {
        showStatus("KeyPressed");
    }
    public void keyReleased(KeyEvent k)
    {
        showStatus("KeyReleased");
    }
    public void keyTyped(KeyEvent k)
    {
        msg = msg+k.getKeyChar();
        repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString(msg, 20, 40);
    }
}
```

HTML code :

```
<applet code="Test" width=300, height=100 ></applet>
```

Example of Mouse Event Handling

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="Mouse" width=500 height=500>
</applet>
*/
```

```
public class Mouse extends Applet
implements MouseListener,MouseMotionListener
{
    int X=0,Y=20;
    String msg="MouseEvents";
    public void init()
    {
        addMouseListener(this);
        addMouseMotionListener(this);
    }
}
```

```

setBackground(Color.black);
setForeground(Color.red);
}
public void mouseEntered(MouseEvent m)
{
setBackground(Color.magenta);
setStatus("Mouse Entered");
repaint();
}
public void mouseExited(MouseEvent m)
{
setBackground(Color.black);
setStatus("Mouse Exited");
repaint();
}
public void mousePressed(MouseEvent m)
{
X=10;
Y=20;
msg="S R";
setBackground(Color.green);
repaint();
}
public void mouseReleased(MouseEvent m)
{
X=10;
Y=20;
msg="Engineering";
setBackground(Color.blue);
repaint();
}
public void mouseMoved(MouseEvent m)
{
X=m.getX();
Y=m.getY();
msg="College";
setBackground(Color.white);
setStatus("Mouse Moved");
repaint();
}
public void mouseDragged(MouseEvent m)
{
msg="CSE";
setBackground(Color.yellow);
setStatus("Mouse Moved"+m.getX()+" "+m.getY());
repaint();
}
public void mouseClicked(MouseEvent m)
{
msg="Students";

```

```

setBackground(Color.pink);
showStatus("Mouse Clicked");
repaint();
}
public void paint(Graphics g)
{
g.drawString(msg,X,Y);
}
}

```

Adapter classes

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

The adapter classes are found in java.awt.event, java.awt.dnd and javax.swing.event packages. The Adapter classes with their corresponding listener interfaces are given below

```

java.awt.event Adapter classes
Adapter class Listener interface
WindowAdapter WindowListener
KeyAdapter KeyListener
MouseAdapter MouseListener
MouseMotionAdapter MouseMotionListener
FocusAdapter FocusListener
ComponentAdapter ComponentListener
ContainerAdapter ContainerListener
HierarchyBoundsAdapter HierarchyBoundsListener

```

Adapter Class Example

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="AdapterDemo" width=300 height=300>
</applet>*/
public class AdapterDemo extends Applet
{
public void init()
{
addMouseListener(new MyMouseAdapter(this));
}
}
class MyMouseAdapter extends MouseAdapter
{
AdapterDemo ad;
public MyMouseAdapter(AdapterDemo ad)
{
this.ad=ad;
}
public void mouseClicked(MouseEvent me)
{
ad.showStatus("Mouse Clicked");
}
}

```

```
}
```

What if you don't want your event-handling class to inherit from an adapter class? For example, suppose you write an applet, and you want your Applet subclass to contain some code to handle mouse events. Since the Java language doesn't permit multiple inheritance, your class can't extend both the Applet and MouseAdapter classes. The solution is to define an inner class -- a class inside of your Applet subclass -- that extends the MouseAdapter class,

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="InnerClassDemo" width=300 height=300>
</applet>*/
public class InnerClassDemo extends Applet
{
    public void init()
    {
        addMouseListener(new MyMouseAdapter());
    }
    class MyMouseAdapter extends MouseAdapter
    {
        public void mousePressed(MouseEvent me)
        {
            showStatus("Mouse Pressed");
        }
    }
}
```

Inner classes work well even if your event handler needs access to private instance variables from the enclosing class. As long as you don't declare an inner class to be static, an inner class can refer to instance variables and methods just as if its code is in the containing class.

Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time. They are like local classes except that they do not have a name.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="AnonymousInnerClassDemo" width=300 height=300>
</applet>*/
public class AnonymousInnerClassDemo extends Applet
{
    public void init()
    {
        addMouseListener(new MouseAdapter()
        {
            public void mousePressed(MouseEvent me)
            {
                showStatus("Mouse Pressed");
            }
        });
    }
}
```

}

AWT

Controls are components that allow a user to interact with your application in various ways—for example; a commonly used control is the push button.

Control Fundamentals

The AWT supports the following types of controls:

- Labels
- Push buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text editing

These controls are subclasses of *Component*.

Adding and Removing Controls

To include a control in a window, you must add it to the window. To do this, you must first create an instance of the desired control and then add it to a window by calling `add()`, which is defined by *Container*. The `add()` method has several forms. It has this general form:

Component add(Component compObj)

Here, *compObj* is an instance of the control that you want to add. A reference to *compObj* is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.

Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call `remove()`. This method is also defined by *Container*. It has this general form:

void remove(Component obj)

Here, *obj* is a reference to the control you want to remove. You can remove all controls by calling `removeAll()`.

Responding to Controls

Except for labels, which are passive, all controls generate events when they are accessed by the user. For example, when the user clicks on a push button, an event is sent that identifies the push button. In general, your program simply implements the appropriate interface and then registers an event listener for each control that you need to monitor. Once a listener has been installed, events are automatically sent to it.

Labels

The easiest control to use is a label. A *label* is an object of type *Label*, and it contains a string, which it displays. Labels are passive controls that do not support any interaction with the user. *Label* defines the following constructors:

Label() throws HeadlessException

Label(String str) throws HeadlessException

Label(String str, int how) throws HeadlessException

The first version creates a blank label. The second version creates a label that contains the string specified by *str*. This string is left-justified. The third version creates a label that contains the string specified by *str* using the alignment specified by *how*. The value of *how* must be one of these three constants: *Label.LEFT*, *Label.RIGHT*, or *Label.CENTER*.

You can set or change the text in a label by using the `setText()` method. You can obtain the current label by calling `getText()`. These methods are shown here:

```
void setText(String str)
```

```
String getText( )
```

For `setText()`, `str` specifies the new label. For `getText()`, the current label is returned. You can set the alignment of the string within the label by calling `setAlignment()`. To obtain the current alignment, call `getAlignment()`. The methods are as follows:

```
void setAlignment(int how)
```

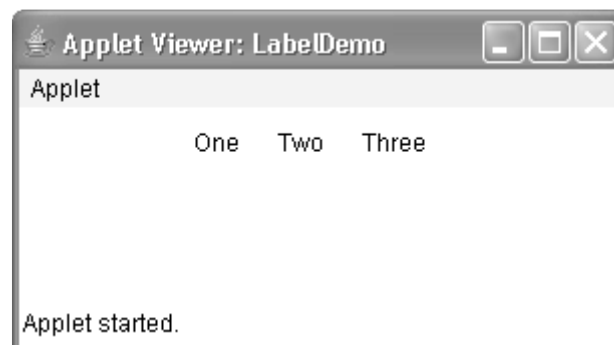
```
int getAlignment( )
```

Here, `how` must be one of the alignment constants shown earlier.

The following example creates three labels and adds them to an applet window:

```
// Demonstrate Labels
import java.awt.*;
import java.applet.*;
/*
<applet code="LabelDemo" width=300 height=200>
</applet>
*/
public class LabelDemo extends Applet {
    public void init() {
        Label one = new Label("One");
        Label two = new Label("Two");
        Label three = new Label("Three");
        // add labels to applet window
        add(one);
        add(two);
        add(three);
    }
}
```

Here is the window created by the LabelDemo applet.



Buttons

A *push button* is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type `Button`. `Button` defines these two constructors:

```
Button( ) throws HeadlessException
```

```
Button(String str) throws HeadlessException
```

The first version creates an empty button. The second creates a button that contains `str` as a label.

After a button has been created, you can set its label by calling `setLabel()`. You can retrieve its label by calling `getLabel()`. These methods are as follows:

```
void setLabel(String str)
```

`String getLabel()`

Here, *str* becomes the new label for the button.

Handling Buttons

Each time a button is pressed, an action event is generated. This is sent to any listeners that previously registered an interest in receiving action event notifications from that component. Each listener implements the `ActionListener` interface. That interface defines the `actionPerformed()` method, which is called when an event occurs. An `ActionEvent` object is supplied as the argument to this method. It contains both a reference to the button that generated the event and a reference to the *action command string* associated with the button.

By default, the action command string is the label of the button. Usually, either the button reference or the action command string can be used to identify the button. (You will soon see examples of each approach.)

Here is an example that creates three buttons labeled “Yes”, “No”, and “Undecided”. Each time one is pressed, a message is displayed that reports which button has been pressed. In this version, the action command of the button (which, by default, is its label) is used to determine which button has been pressed. The label is obtained by calling the `getActionCommand()` method on the `ActionEvent` object passed to `actionPerformed()`.

```
// Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250 height=150>
</applet>
*/
public class ButtonDemo extends Applet implements ActionListener {
    String msg = "";
    Button yes, no, maybe;
    public void init() {
        yes = new Button("Yes");
        no = new Button("No");
        maybe = new Button("Undecided");
        add(yes);
        add(no);
        add(maybe);
        yes.addActionListener(this);
        no.addActionListener(this);
        maybe.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae) {
        String str = ae.getActionCommand();
        if(str.equals("Yes")) {
            msg = "You pressed Yes.";
        }
        else if(str.equals("No")) {
            msg = "You pressed No.";
        }
    }
}
```

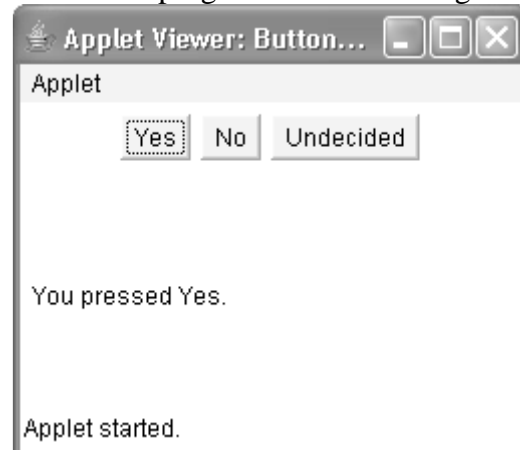


```

else {
    msg = "You pressed Undecided.";
}
repaint();
}
public void paint(Graphics g) {
    g.drawString(msg, 6, 100);
}
}

```

Sample output from the ButtonDemo program is shown in Figure below



As mentioned, in addition to comparing button action command strings, you can also determine which button has been pressed, by comparing the object obtained from the `getSource()` method to the button objects that you added to the window. To do this, you must keep a list of the objects when they are added. The following applet shows this approach:

```

// Recognize Button objects.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonList" width=250 height=150>
</applet>
*/
public class ButtonList extends Applet implements ActionListener {
    String msg = "";
    Button bList[] = new Button[3];
    public void init() {
        Button yes = new Button("Yes");
        Button no = new Button("No");
        Button maybe = new Button("Undecided");
        // store references to buttons as added
        bList[0] = (Button) add(yes);
        bList[1] = (Button) add(no);
        bList[2] = (Button) add(maybe);
        // register to receive action events
        for(int i = 0; i < 3; i++) {
            bList[i].addActionListener(this);
        }
    }
}

```

```

public void actionPerformed(ActionEvent ae) {
    for(int i = 0; i < 3; i++) {
        if(ae.getSource() == bList[i]) {
            msg = "You pressed " + bList[i].getLabel();
        }
    }
    repaint();
}

public void paint(Graphics g) {
    g.drawString(msg, 6, 100);
}
}

```

In this version, the program stores each button reference in an array when the buttons are added to the applet window. (Recall that the `add()` method returns a reference to the button when it is added.) Inside `actionPerformed()`, this array is then used to determine which button has been pressed.

For simple programs, it is usually easier to recognize buttons by their labels. However, in situations in which you will be changing the label inside a button during the execution of your program, or using buttons that have the same label, it may be easier to determine which button has been pushed by using its object reference. It is also possible to set the action command string associated with a button to something other than its label by calling `setActionCommand()`. This method changes the action command string, but does not affect the string used to label the button. Thus, setting the action command enables the action command and the label of a button to differ.

Check Boxes

A *check box* is a control that is used to turn an option on or off. It consists of a small box that can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group. Check boxes are objects of the `Checkbox` class.

`Checkbox` supports these constructors:

```

Checkbox( ) throws HeadlessException
Checkbox(String str) throws HeadlessException
Checkbox(String str, boolean on) throws HeadlessException
Checkbox(String str, boolean on, CheckboxGroup cbGroup) throws HeadlessException
Checkbox(String str, CheckboxGroup cbGroup, boolean on) throws HeadlessException

```

The first form creates a check box whose label is initially blank. The state of the check box is unchecked. The second form creates a check box whose label is specified by *str*. The state of the check box is unchecked. The third form allows you to set the initial state of the check box. If *on* is true, the check box is initially checked; otherwise, it is cleared. The fourth and fifth forms create a check box whose label is specified by *str* and whose group is specified by *cbGroup*. If this check box is not part of a group, then *cbGroup* must be null. (Check box groups are described in the next section.) The value of *on* determines the initial state of the check box.

To retrieve the current state of a check box, call `getState()`. To set its state, call `setState()`. You can obtain the current label associated with a check box by calling `getLabel()`. To set the label, call `setLabel()`. These methods are as follows:

```

boolean getState( )
void setState(boolean on)

```

```
String getLabel( )  
void setLabel(String str)
```

Here, if *on* is true, the box is checked. If it is false, the box is cleared. The string passed in *str* becomes the new label associated with the invoking check box.

Handling Check Boxes

Each time a check box is selected or deselected, an item event is generated. This is sent to any listeners that previously registered an interest in receiving item event notifications from that component. Each listener implements the `ItemListener` interface. That interface defines the `itemStateChanged()` method. An `ItemEvent` object is supplied as the argument to this method. It contains information about the event (for example, whether it was a selection or-deselection). The following program creates four check boxes. The initial state of the first box is checked. The status of each check box is displayed. Each time you change the state of a check box, the status display is updated.

```
// Demonstrate check boxes.  
import java.awt.*;  
import java.awt.event.*;  
import java.applet.*;  
/*  
<applet code="CheckboxDemo" width=250 height=200>  
</applet>  
*/  
public class CheckboxDemo extends Applet implements ItemListener {  
    String msg = "";  
    Checkbox winXP, winVista, solaris, mac;  
    public void init() {  
        winXP = new Checkbox("Windows XP", null, true);  
        winVista = new Checkbox("Windows Vista");  
        solaris = new Checkbox("Solaris");  
        mac = new Checkbox("Mac OS");  
        add(winXP);  
        add(winVista);  
        add(solaris);  
        add(mac);  
        winXP.addItemListener(this);  
        winVista.addItemListener(this);  
        solaris.addItemListener(this);  
        mac.addItemListener(this);  
    }  
    public void itemStateChanged(ItemEvent ie) {  
        repaint();  
    }  
    // Display current state of the check boxes.  
    public void paint(Graphics g) {  
        msg = "Current state: ";  
        g.drawString(msg, 6, 80);  
        msg = " Windows XP: " + winXP.getState();  
        g.drawString(msg, 6, 100);  
        msg = " Windows Vista: " + winVista.getState();  
        g.drawString(msg, 6, 120);  
    }  
}
```

```

msg = " Solaris: " + solaris.getState();
g.drawString(msg, 6, 140);
msg = " Mac OS: " + mac.getState();
g.drawString(msg, 6, 160);
}
}

```

Sample output is shown in Figure



CheckboxGroup

It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called *radio buttons*, because they act like the station selector on a car radio—only one station can be selected at any one time. To create a set of mutually exclusive check boxes, you must first define the group to which they will belong and then specify that group when you construct the check boxes. Check box groups are objects of type `CheckboxGroup`. Only the default constructor is defined, which creates an empty group.

You can determine which check box in a group is currently selected by calling `getSelectedCheckbox()`. You can set a check box by calling `setSelectedCheckbox()`.

These methods are as follows:

```

Checkbox getSelectedCheckbox( )
void setSelectedCheckbox(Checkbox which)

```

Here, *which* is the check box that you want to be selected. The previously selected check box will be turned off.

Here is a program that uses check boxes that are part of a group:

```

// Demonstrate check box group.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CBGroup" width=250 height=200>
</applet>
*/
public class CBGroup extends Applet implements ItemListener {
String msg = "";
    Checkbox winXP, winVista, solaris, mac;

```

```

CheckboxGroup cbg;
public void init() {
    cbg = new CheckboxGroup();
    winXP = new Checkbox("Windows XP", cbg, true);
    winVista = new Checkbox("Windows Vista", cbg, false);
    solaris = new Checkbox("Solaris", cbg, false);
    mac = new Checkbox("Mac OS", cbg, false);
    add(winXP);
    add(winVista);
    add(solaris);
    add(mac);
    winXP.addItemListener(this);
    winVista.addItemListener(this);
    solaris.addItemListener(this);
    mac.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie) {
    repaint();
}
// Display current state of the checkboxes.
public void paint(Graphics g) {
    msg = "Current selection: ";
    msg += cbg.getSelectedCheckbox().getLabel();
    g.drawString(msg, 6, 100);
}
}

```

Output generated by the CBGroup applet is shown in Figure. Notice that the checkboxes are now circular in shape.



TextField

The TextField class implements a single-line text-entry area, usually called an *edit control*. Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections. TextField is a subclass of TextComponent.

TextField defines the following constructors:

TextField() throws *HeadlessException*
TextField(int numChars) throws *HeadlessException*

TextField(String str) throws *HeadlessException*

TextField(String str, int numChars) throws *HeadlessException*

The first version creates a default text field. The second form creates a text field that is *numChars* characters wide. The third form initializes the text field with the string contained in *str*. The fourth form initializes a text field and sets its width.

TextField (and its superclass TextComponent) provides several methods that allow you to utilize a text field. To obtain the string currently contained in the text field, call *getText()*. To set the text, call *setText()*. These methods are as follows:

String getText()

void setText(String str)

Here, *str* is the new string.

The user can select a portion of the text in a text field. Also, you can select a portion of text under program control by using *select()*. Your program can obtain the currently selected text by calling *getSelectedText()*. These methods are shown here:

String getSelectedText()

void select(int startIndex, int endIndex)

getSelectedText() returns the selected text. The *select()* method selects the characters beginning at *startIndex* and ending at *endIndex*–1. You can control whether the contents of a text field may be modified by the user by calling *setEditable()*. You can determine editability by calling *isEditable()*. These methods are shown here:

boolean isEditable()

void setEditable(boolean canEdit)

isEditable() returns true if the text may be changed and false if not. In *setEditable()*, if *canEdit* is true, the text may be changed. If it is false, the text cannot be altered. There may be times when you will want the user to enter text that is not displayed, such as a password. You can disable the echoing of the characters as they are typed by calling *setEchoChar()*. This method specifies a single character that the TextField will display when characters are entered (thus, the actual characters typed will not be shown). You can check a text field to see if it is in this mode with the *echoCharIsSet()* method. You can retrieve the echo character by calling the *getEchoChar()* method. These methods are as follows:

void setEchoChar(char ch)

boolean echoCharIsSet()

char getEchoChar()

Here, *ch* specifies the character to be echoed.

Handling a TextField

Since text fields perform their own editing functions, your program generally will not respond to individual key events that occur within a text field. However, you may want to respond when the user presses ENTER. When this occurs, an action event is generated.

Here is an example that creates the classic user name and password screen:

// Demonstrate text field.

import java.awt.;*

import java.awt.event.;*

import java.applet.;*

*/**

<applet code="TextFieldDemo" width=380 height=150>

</applet>

**/*

public class TextFieldDemo extends Applet

implements ActionListener {

```

TextField name, pass;
public void init() {
    Label namep = new Label("Name: ", Label.RIGHT);
    Label passp = new Label("Password: ", Label.RIGHT);
    name = new TextField(12);
    pass = new TextField(8);
    pass.setEchoChar('?');
    add(namep);
    add(name);
    add(passp);
    add(pass);
    // register to receive action events
    name.addActionListener(this);
    pass.addActionListener(this);
}
// User pressed Enter.
public void actionPerformed(ActionEvent ae) {
    repaint();
}
public void paint(Graphics g) {
    g.drawString("Name: " + name.getText(), 6, 60);
    g.drawString("Selected text in name: "
        + name.getSelectedText(), 6, 80);
    g.drawString("Password: " + pass.getText(), 6, 100);
}
}

```

Sample output from the TextFieldDemo applet is shown in Figure



TextArea

Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called TextArea. Following are the constructors for TextArea:

```

TextArea( ) throws HeadlessException
TextArea(int numLines, int numChars) throws HeadlessException
TextArea(String str) throws HeadlessException
TextArea(String str, int numLines, int numChars) throws HeadlessException
TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException

```

Here, *numLines* specifies the height, in lines, of the text area, and *numChars* specifies its width, in characters. Initial text can be specified by *str*. In the fifth form, you can specify the scroll

bars that you want the control to have. *sBars* must be one of these values:

`SCROLLBARS_BOTH` `SCROLLBARS_NONE`

`SCROLLBARS_HORIZONTAL_ONLY` `SCROLLBARS_VERTICAL_ONLY`

`TextArea` is a subclass of `TextComponent`. Therefore, it supports the `getText()`, `setText()`, `getSelectedText()`, `select()`, `isEditable()`, and `setEditable()` methods described in the preceding section.

`TextArea` adds the following methods:

`void append(String str)`

`void insert(String str, int index)`

`void replaceRange(String str, int startIndex, int endIndex)`

The `append()` method appends the string specified by *str* to the end of the current text. `insert()`

inserts the string passed in *str* at the specified index. To replace text, call `replaceRange()`. It replaces the characters from *startIndex* to *endIndex*-1, with the replacement text passed in *str*.

Text areas are almost self-contained controls. Your program incurs virtually no management overhead. Text areas only generate got-focus and lost-focus events. Normally, your program simply obtains the current text when it is needed.

The following program creates a `TextArea` control:

```
// Demonstrate TextArea.
```

```
import java.awt.*;
```

```
import java.applet.*;
```

```
/*
```

```
<applet code="TextAreaDemo" width=300 height=250>
```

```
</applet>
```

```
*/
```

```
public class TextAreaDemo extends Applet {
```

```
    public void init() {
```

```
        String val =
```

```
        "Java SE 6 is the latest version of the most\n" +
```

```
        "widely-used computer language for Internet programming.\n" +
```

```
        "Building on a rich heritage, Java has advanced both\n" +
```

```
        "the art and science of computer language design.\n\n" +
```

```
        "One of the reasons for Java's ongoing success is its\n" +
```

```
        "constant, steady rate of evolution. Java has never stood\n" +
```

```
        "still. Instead, Java has consistently adapted to the\n" +
```

```
        "rapidly changing landscape of the networked world.\n" +
```

```
        "Moreover, Java has often led the way, charting the\n" +
```

```
        "course for others to follow.";
```

```
        TextArea text = new TextArea(val, 10, 30);
```

```
        add(text);
```

```
    }
```

```
}
```

```
public void init() {
```

```
    String val =
```

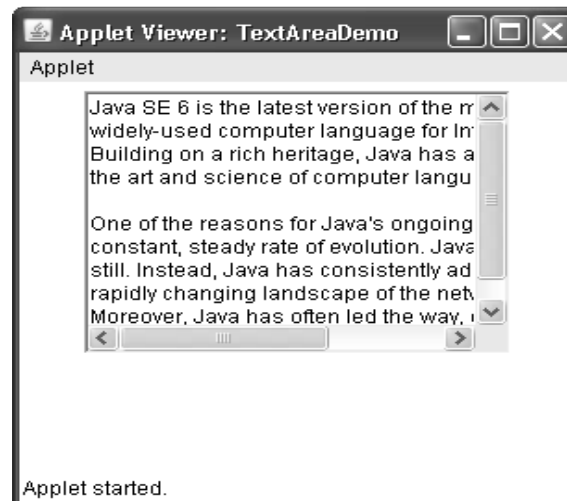
```
    "Java SE 6 is the latest version of the most\n" + "widely-used computer language for Internet\nprogramming.\n" + "Building on a rich heritage, Java has advanced both\n" + "the art and
```



```

science of computer language design.\n\n" + "One of the reasons for Java's ongoing success
is its\n" + "constant, steady rate of evolution. Java has never stood\n" + "still. Instead, Java
has consistently adapted to the\n" +
"rapidly changing landscape of the networked world.\n" + "Moreover, Java has often led the
way, charting the\n" + "course for others to follow.";
TextArea text = new TextArea(val, 10, 30);
add(text);
}

```



Choice Controls

The Choice class is used to create a *pop-up list* of items from which the user may choose. Thus, a Choice control is a form of menu. When inactive, a Choice component takes up only enough space to show the currently selected item. When the user clicks on it, the whole list of choices pops up, and a new selection can be made. Each item in the list is a string that appears as a left-justified label in the order it is added to the Choice object. Choice only defines the default constructor, which creates an empty list.

To add a selection to the list, call `add()`. It has this general form:

```
void add(String name)
```

Here, *name* is the name of the item being added. Items are added to the list in the order in which calls to `add()` occur. To determine which item is currently selected, you may call either `getSelectedItem()` or `getSelectedIndex()`. These methods are shown here:

```
String getSelectedItem()
```

```
int getSelectedIndex()
```

The `getSelectedItem()` method returns a string containing the name of the item.

`getSelectedIndex()` returns the index of the item. The first item is at index 0. By default, the first item added to the list is selected. To obtain the number of items in the list, call `getItemCount()`. You can set the currently selected item using the `select()` method with either a zero-based integer index or a string that will match a name in the list. These methods are shown here:

```
int getItemCount()
```

```
void select(int index)
```

```
void select(String name)
```

Given an index, you can obtain the name associated with the item at that index by calling `getItem()`, which has this general form:

```
String getItem(int index)
```

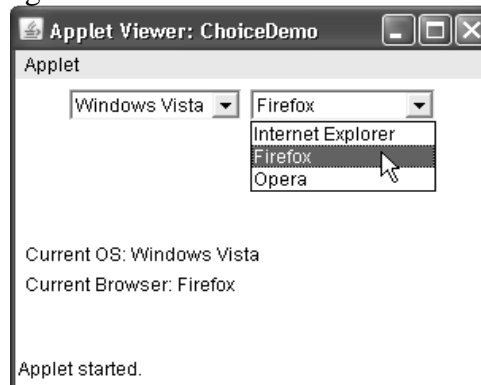
Here, *index* specifies the index of the desired item.

Handling Choice Lists

Each time a choice is selected, an item event is generated. This is sent to any listeners that previously registered an interest in receiving item event notifications from that component. Each listener implements the `ItemListener` interface. That interface defines the `itemStateChanged()` method. An `ItemEvent` object is supplied as the argument to this method. Here is an example that creates two Choice menus. One selects the operating system. The other selects the browser.

```
// Demonstrate Choice lists.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ChoiceDemo" width=300 height=180>
</applet>
*/
public class ChoiceDemo extends Applet implements ItemListener {
    Choice os, browser;
    String msg = "";
    public void init() {
        os = new Choice();
        browser = new Choice();
        // add items to os list
        os.add("Windows XP");
        os.add("Windows Vista");
        os.add("Solaris");
        os.add("Mac OS");
        // add items to browser list
        browser.add("Internet Explorer");
        browser.add("Firefox");
        browser.add("Opera");
        // add choice lists to window
        add(os);
        add(browser);
        // register to receive item events
        os.addItemListener(this);
        browser.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie) {
        repaint();
    }
    // Display current selections.
    public void paint(Graphics g) {
        msg = "Current OS: ";
        msg += os.getSelectedItem();
        g.drawString(msg, 6, 120);
        msg = "Current Browser: ";
        msg += browser.getSelectedItem();
        g.drawString(msg, 6, 140);
    }
}
```

Sample output is shown in Figure



Using Lists

The List class provides a compact, multiple-choice, scrolling selection list. Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible window. It can also be created to allow multiple selections. List provides these constructors:

List() throws HeadlessException

List(int numRows) throws HeadlessException

List(int numRows, boolean multipleSelect) throws HeadlessException

The first version creates a List control that allows only one item to be selected at any one time. In the second form, the value of *numRows* specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed). In the third form, if *multipleSelect* is true, then the user may select two or more items at a time. If it is false, then only one item may be selected. To add a selection to the list, call *add()*. It has the following two forms:

void add(String name)

void add(String name, int index)

Here, *name* is the name of the item added to the list. The first form adds items to the end of the list. The second form adds the item at the index specified by *index*. Indexing begins at zero. You can specify *-1* to add the item to the end of the list. For lists that allow only single selection, you can determine which item is currently selected by calling either *getSelectedItem()* or *getSelectedIndex()*. These methods are shown here:

String getItemSelected()

int getSelectedIndex()

The *getSelectedItem()* method returns a string containing the name of the item. If more than one item is selected, or if no selection has yet been made, null is returned. *getSelectedIndex()* returns the index of the item. The first item is at index 0. If more than one item is selected, or if no selection has yet been made, *-1* is returned. For lists that allow multiple selection, you must use either *getSelectedItems()* or *getSelectedIndexes()*, shown here, to determine the current selections:

String[] getSelectedItems()

int[] getSelectedIndexes()

getSelectedItems() returns an array containing the names of the currently selected items. *getSelectedIndexes()* returns an array containing the indexes of the currently selected items. To obtain the number of items in the list, call *getItemCount()*. You can set the currently selected item by using the *select()* method with a zero-based integer index. These methods are shown here:

```
int getItemCount( )  
void select(int index)
```

Given an index, you can obtain the name associated with the item at that index by calling `getItem()`, which has this general form:

```
String getItem(int index)
```

Here, *index* specifies the index of the desired item.

Handling Lists

To process list events, you will need to implement the `ActionListener` interface. Each time a List item is double-clicked, an `ActionEvent` object is generated. Its `getActionCommand()` method can be used to retrieve the name of the newly selected item. Also, each time an item is selected or deselected with a single click, an `ItemEvent` object is generated. Its `getStateChange()` method can be used to determine whether a selection or deselection triggered this event. `getItemSelectable()` returns a reference to the object that triggered this event. Here is an example that converts the Choice controls in the preceding section into List components, one multiple choice and the other single choice:

```
// Demonstrate Lists.  
import java.awt.*;  
import java.awt.event.*;  
import java.applet.*;  
/*  
<applet code="ListDemo" width=300 height=180>  
</applet>  
*/  
public class ListDemo extends Applet implements ActionListener {  
    List os, browser;  
    String msg = "";  
    public void init() {  
        os = new List(4, true);  
        browser = new List(4, false);  
        // add items to os list  
        os.add("Windows XP");  
        os.add("Windows Vista");  
        os.add("Solaris");  
        os.add("Mac OS");  
        // add items to browser list  
        browser.add("Internet Explorer");  
        browser.add("Firefox");  
        browser.add("Opera");  
        browser.select(1);  
        // add lists to window  
        add(os);  
        add(browser);  
        // register to receive action events  
        os.addActionListener(this);  
        browser.addActionListener(this);  
    }  
    public void actionPerformed(ActionEvent ae) {  
        repaint();  
    }  
}
```

```

}
// Display current selections.
public void paint(Graphics g) {
    int idx[];
    msg = "Current OS: ";
    idx = os.getSelectedIndexes();
    for(int i=0; i<idx.length; i++)
        msg += os.getItem(idx[i]) + " ";
    g.drawString(msg, 6, 120);
    msg = "Current Browser: ";
    msg += browser.getSelectedItem();
    g.drawString(msg, 6, 140);
}
}

```

Sample output generated by the ListDemo applet is shown in



Figure

Managing Scroll Bars

Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically. A scroll bar is actually a composite of several individual parts. Each end has an arrow that you can click to move the current value of the scroll bar one unit in the direction of the arrow. The current value of the scroll bar relative to its minimum and maximum values is indicated by the *slider box* (or *thumb*) for the scroll bar. The slider box can be dragged by the user to a new position. The scroll bar will then reflect this value. In the background space on either side of the thumb, the user can click to cause the thumb to jump in that direction by some increment larger than 1. Typically, this action translates into some form of page up and page down. Scroll bars are encapsulated by the Scrollbar class.

Scrollbar defines the following constructors:

Scrollbar() throws HeadlessException

Scrollbar(int style) throws HeadlessException

Scrollbar(int style, int initialValue, int thumbSize, int min, int max) throws HeadlessException

The first form creates a vertical scroll bar. The second and third forms allow you to specify the orientation of the scroll bar. If *style* is Scrollbar.VERTICAL, a vertical scroll bar is created. If *style* is Scrollbar.HORIZONTAL, the scroll bar is horizontal. In the third form of the constructor, the initial value of the scroll bar is passed in *initialValue*. The number of units represented by the height of the thumb is passed in *thumbSize*. The minimum and maximum values for the scroll bar are specified by *min* and *max*.

If you construct a scroll bar by using one of the first two constructors, then you need to

set its parameters by using `setValues()`, shown here, before it can be used:

```
void setValues(int initialValue, int thumbSize, int min, int max)
```

The parameters have the same meaning as they have in the third constructor just described.

To obtain the current value of the scroll bar, call `getValue()`. It returns the current setting. To set the current value, call `setValue()`. These methods are as follows:

```
int getValue( )
```

```
void setValue(int newValue)
```

Here, *newValue* specifies the new value for the scroll bar. When you set a value, the slider

box inside the scroll bar will be positioned to reflect the new value. You can also retrieve the minimum and maximum values via `getMinimum()` and `getMaximum()`, shown here:

```
int getMinimum( )
```

```
int getMaximum( )
```

They return the requested quantity. By default, 1 is the increment added to or subtracted from the scroll bar each time it is scrolled up or down one line. You can change this increment by calling `setUnitIncrement()`. By default, page-up and page-down increments are 10. You can change this value by calling `setBlockIncrement()`. These methods are shown here:

```
void setUnitIncrement(int newIncr)
```

```
void setBlockIncrement(int newIncr)
```

Handling Scroll Bars

To process scroll bar events, you need to implement the `AdjustmentListener` interface. Each time a user interacts with a scroll bar, an `AdjustmentEvent` object is generated. Its `getAdjustmentType()` method can be used to determine the type of the adjustment. The types of adjustment events are as follows:

BLOCK_DECREMENT A page-down event has been generated.

BLOCK_INCREMENT A page-up event has been generated.

TRACK An absolute tracking event has been generated.

UNIT_DECREMENT The line-down button in a scroll bar has been pressed.

UNIT_INCREMENT The line-up button in a scroll bar has been pressed.

The following example creates both a vertical and a horizontal scroll bar. The current settings of the scroll bars are displayed. If you drag the mouse while inside the window, the coordinates of each drag event are used to update the scroll bars. An asterisk is displayed at the current drag position.

```
// Demonstrate scroll bars.
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import java.applet.*;
```

```
/*
```

```
<applet code="SBDemo" width=300 height=200>
```

```
</applet>
```

```
*/
```

```
public class SBDemo extends Applet implements AdjustmentListener, MouseMotionListener {
```

```
String msg = "";
```

```
Scrollbar vertSB, horzSB;
```

```
public void init() {
```

```
int width = Integer.parseInt(getParameter("width"));
```

```
int height = Integer.parseInt(getParameter("height"));
```

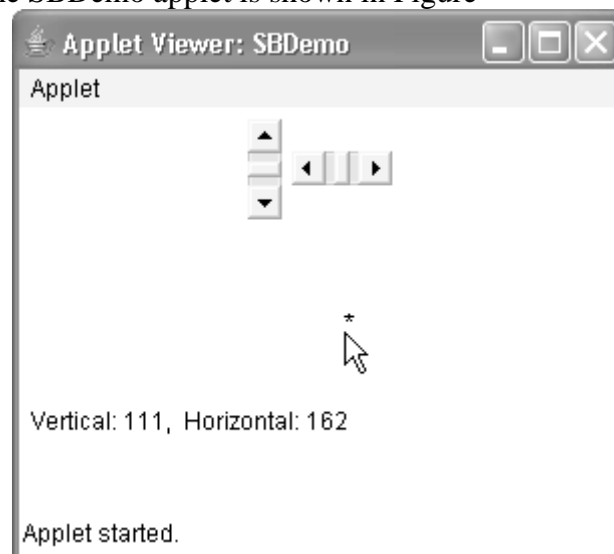
```
vertSB = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, height);
```

```

horzSB = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, width);
add(vertSB);
add(horzSB);
// register to receive adjustment events
vertSB.addAdjustmentListener(this);
horzSB.addAdjustmentListener(this);
addMouseMotionListener(this);
}
public void adjustmentValueChanged(AdjustmentEvent ae) {
repaint();
}
// Update scroll bars to reflect mouse dragging.
public void mouseDragged(MouseEvent me) {
int x = me.getX();
int y = me.getY();
vertSB.setValue(y);
horzSB.setValue(x);
repaint();
}
// Necessary for MouseMotionListener
public void mouseMoved(MouseEvent me) {
}
// Display current value of scroll bars.
public void paint(Graphics g) {
msg = "Vertical: " + vertSB.getValue();
msg += ", Horizontal: " + horzSB.getValue();
g.drawString(msg, 6, 160);
// show current mouse drag position
g.drawString("*", horzSB.getValue(),
vertSB.getValue());
}
}

```

Sample output from the SBDemo applet is shown in Figure



Menu Bars and Menus

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. This concept is implemented in the AWT by the following classes: `MenuBar`, `Menu`, and `MenuItem`. In general, a menu bar contains one or more `Menu` objects. Each `Menu` object contains a list of `MenuItem` objects. Each `MenuItem` object represents something that can be selected by the user. Since `Menu` is a subclass of `MenuItem`, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items. These are menu options of type `CheckboxMenuItem` and will have a check mark next to them when they are selected.

To create a menu bar, first create an instance of `MenuBar`. This class only defines the default constructor. Next, create instances of `Menu` that will define the selections displayed on the bar. Following are the constructors for `Menu`:

`Menu() throws HeadlessException`

`Menu(String optionName) throws HeadlessException`

`Menu(String optionName, boolean removable) throws HeadlessException`

Here, *optionName* specifies the name of the menu selection. If *removable* is true, the menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar. (Removable menus are implementation-dependent.) The first form creates an empty menu. Individual menu items are of type `MenuItem`. It defines these constructors:

`MenuItem() throws HeadlessException`

`MenuItem(String itemName) throws HeadlessException`

`MenuItem(String itemName, MenuShortcut keyAccel) throws HeadlessException`

Here, *itemName* is the name shown in the menu, and *keyAccel* is the menu shortcut for this item. You can disable or enable a menu item by using the `setEnabled()` method. Its form is shown here:

`void setEnabled(boolean enabledFlag)`

If the argument *enabledFlag* is true, the menu item is enabled. If false, the menu item is disabled.

You can determine an item's status by calling `isEnabled()`. This method is shown here:

`boolean isEnabled()`

`isEnabled()` returns true if the menu item on which it is called is enabled. Otherwise, it returns false.

You can change the name of a menu item by calling `setLabel()`. You can retrieve the current name by using `getLabel()`. These methods are as follows:

`void setLabel(String newName)`

`String getLabel()`

Here, *newName* becomes the new name of the invoking menu item. `getLabel()` returns the current name.

You can create a checkable menu item by using a subclass of `MenuItem` called `CheckboxMenuItem`. It has these constructors:

`CheckboxMenuItem() throws HeadlessException`

`CheckboxMenuItem(String itemName) throws HeadlessException`

`CheckboxMenuItem(String itemName, boolean on) throws HeadlessException`

Here, *itemName* is the name shown in the menu. Checkable items operate as toggles. Each time one is selected, its state changes. In the first two forms, the checkable entry is unchecked. In the third form, if *on* is true, the checkable entry is initially checked. Otherwise, it is cleared. You can obtain the status of a checkable item by calling `getState()`. You can set it to a known state by using `setState()`. These methods are shown here:

`boolean getState()`

`void setState(boolean checked)`

If the item is checked, `getState()` returns true. Otherwise, it returns false. To check an item, pass true to `setState()`. To clear an item, pass false. Once you have created a menu item, you must add the item to a Menu object by using `add()`, which has the following general form:

`MenuItem add(MenuItem item)`

Here, *item* is the item being added. Items are added to a menu in the order in which the calls to `add()` take place. The *item* is returned. Once you have added all items to a Menu object, you can add that object to the menu bar by using this version of `add()` defined by MenuBar:

`Menu add(Menu menu)`

Here, *menu* is the menu being added. The *menu* is returned. Menus only generate events when an item of type MenuItem or CheckboxMenuItem is selected. They do not generate events when a menu bar is accessed to display a drop-down menu, for example. Each time a menu item is selected, an ActionEvent object is generated.

By default, the action command string is the name of the menu item. However, you can specify a different action command string by calling `setActionCommand()` on the menu item. Each time a check box menu item is checked or unchecked, an ItemEvent object is generated. Thus, you must implement the ActionListener and/or ItemListener interfaces in order to handle these menu events. The `getItem()` method of ItemEvent returns a reference to the item that generated this event. The general form of this method is shown here:

`Object getItem()`

Following is an example that adds a series of nested menus to a pop-up window. The item selected is displayed in the window. The state of the two check box menu items is also displayed.

```
// Illustrate menus.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MenuDemo" width=250 height=250>
</applet>
*/
// Create a subclass of Frame.
class MenuFrame extends Frame {
    String msg = "";
    CheckboxMenuItem debug, test;
    MenuFrame(String title) {
        super(title);
        // create menu bar and add it to frame
        MenuBar mbar = new MenuBar();
        setMenuBar(mbar);
        // create the menu items
        Menu file = new Menu("File");
        MenuItem item1, item2, item3, item4, item5;
        file.add(item1 = new MenuItem("New..."));
        file.add(item2 = new MenuItem("Open..."));
        file.add(item3 = new MenuItem("Close"));
        file.add(item4 = new MenuItem("-"));
        file.add(item5 = new MenuItem("Quit..."));
        mbar.add(file);
        Menu edit = new Menu("Edit");
        MenuItem item6, item7, item8, item9;
```

```

edit.add(item6 = new MenuItem("Cut"));
edit.add(item7 = new MenuItem("Copy"));
edit.add(item8 = new MenuItem("Paste"));
edit.add(item9 = new MenuItem("-"));
Menu sub = new Menu("Special");
MenuItem item10, item11, item12;
sub.add(item10 = new MenuItem("First"));
sub.add(item11 = new MenuItem("Second"));
sub.add(item12 = new MenuItem("Third"));
edit.add(sub);
// these are checkable menu items
debug = new CheckboxMenuItem("Debug");
edit.add(debug);
test = new CheckboxMenuItem("Testing");
edit.add(test);
mbar.add(edit);
// create an object to handle action and item events
MyMenuHandler handler = new MyMenuHandler(this);
// register it to receive those events
item1.addActionListener(handler);
item2.addActionListener(handler);
item3.addActionListener(handler);
item4.addActionListener(handler);
item5.addActionListener(handler);
item6.addActionListener(handler);
item7.addActionListener(handler);
item8.addActionListener(handler);
item9.addActionListener(handler);
item10.addActionListener(handler);
item11.addActionListener(handler);
item12.addActionListener(handler);
debug.addItemListener(handler);
test.addItemListener(handler);
// create an object to handle window events
MyWindowAdapter adapter = new MyWindowAdapter(this);
// register it to receive those events
addWindowListener(adapter);
}
public void paint(Graphics g) {
g.drawString(msg, 10, 200);
if(debug.getState())
g.drawString("Debug is on.", 10, 220);
else
g.drawString("Debug is off.", 10, 220);
if(test.getState())
g.drawString("Testing is on.", 10, 240);
else
g.drawString("Testing is off.", 10, 240);
}
}

```

```

class MyWindowAdapter extends WindowAdapter {
    MenuFrame menuFrame;
    public MyWindowAdapter(MenuFrame menuFrame) {
        this.menuFrame = menuFrame;
    }
    public void windowClosing(WindowEvent we) {
        menuFrame.setVisible(false);
    }
}

class MyMenuHandler implements ActionListener, ItemListener {
    MenuFrame menuFrame;
    public MyMenuHandler(MenuFrame menuFrame) {
        this.menuFrame = menuFrame;
    }
    // Handle action events.
    public void actionPerformed(ActionEvent ae) {
        String msg = "You selected ";
        String arg = ae.getActionCommand();
        if(arg.equals("New..."))
            msg += "New.";
        else if(arg.equals("Open..."))
            msg += "Open.";
        else if(arg.equals("Close"))
            msg += "Close.";
        else if(arg.equals("Quit..."))
            msg += "Quit.";
        else if(arg.equals("Edit"))
            msg += "Edit.";
        else if(arg.equals("Cut"))
            msg += "Cut.";
        else if(arg.equals("Copy"))
            msg += "Copy.";
        else if(arg.equals("Paste"))
            msg += "Paste.";
        else if(arg.equals("First"))
            msg += "First.";
        else if(arg.equals("Second"))
            msg += "Second.";
        else if(arg.equals("Third"))
            msg += "Third.";
        else if(arg.equals("Debug"))
            msg += "Debug.";
        else if(arg.equals("Testing"))
            msg += "Testing.";
        menuFrame.msg = msg;
        menuFrame.repaint();
    }
    // Handle item events.
    public void itemStateChanged(ItemEvent ie) {
        menuFrame.repaint();
    }
}

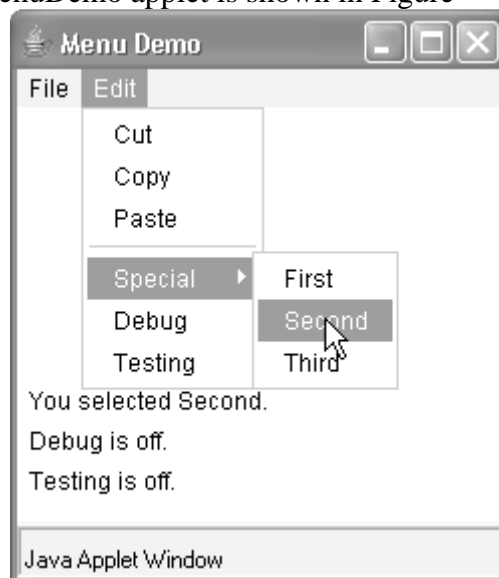
```

```

}
}
// Create frame window.
public class MenuDemo extends Applet {
    Frame f;
    public void init() {
        f = new MenuFrame("Menu Demo");
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        setSize(new Dimension(width, height));
        f.setSize(width, height);
        f.setVisible(true);
    }
    public void start() {
        f.setVisible(true);
    }
    public void stop() {
        f.setVisible(false);
    }
}
}

```

Sample output from the MenuDemo applet is shown in Figure



Understanding Layout Managers

All of the components that we have shown so far have been positioned by the default layout manager. A layout manager automatically arranges your controls within a window by using some type of algorithm.

Each Container object has a layout manager associated with it. A layout manager is an instance of any class that implements the `LayoutManager` interface. The layout manager is set by the `setLayout()` method. If no call to `setLayout()` is made, then the default layout manager is used. Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it. The `setLayout()` method has the following general form:

```
void setLayout(LayoutManager layoutObj)
```

Here, *layoutObj* is a reference to the desired layout manager. If you wish to disable the layout manager and position components manually, pass null for *layoutObj*. If you do this, you will need to determine the shape and position of each component manually, using the `setBounds()` method defined by `Component`. Each layout manager keeps track of a list of components that are stored by their names. The layout manager is notified each time you add a component to a container. Java has several predefined `LayoutManager` classes, which are described next.

FlowLayout

`FlowLayout` is the default layout manager. `FlowLayout` implements a simple layout style, which is similar to how words flow in a text editor. The direction of the layout is governed by the container's component orientation property, which, by default, is left to right, top to bottom. Therefore, by default, components are laid out line-by-line beginning at the upper-left corner. In all cases, when a line is filled, layout advances to the next line. A small space is left between each component, above and below, as well as left and right. Here are the constructors for `FlowLayout`:

`FlowLayout()`

`FlowLayout(int how)`

`FlowLayout(int how, int horz, int vert)`

The first form creates the default layout, which centers components and leaves five pixels of space between each component. The second form lets you specify how each line is aligned. Valid values for *how* are as follows:

`FlowLayout.LEFT`

`FlowLayout.CENTER`

`FlowLayout.RIGHT`

`FlowLayout.LEADING`

`FlowLayout.TRAILING`

These values specify left, center, right, leading edge, and trailing edge alignment, respectively.

The third constructor allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively.

// Use left-aligned flow layout.

`import java.awt.*;`

`import java.awt.event.*;`

`import java.applet.*;`

`/*`

`<applet code="FlowLayoutDemo" width=250 height=200>`

`</applet>`

`*/`

`public class FlowLayoutDemo extends Applet`

`implements ItemListener {`

`String msg = "";`

`Checkbox winXP, winVista, solaris, mac;`

`public void init() {`

// set left-aligned flow layout

`setLayout(new FlowLayout(FlowLayout.LEFT));`

`winXP = new Checkbox("Windows XP", null, true);`

`winVista = new Checkbox("Windows Vista");`

`solaris = new Checkbox("Solaris");`

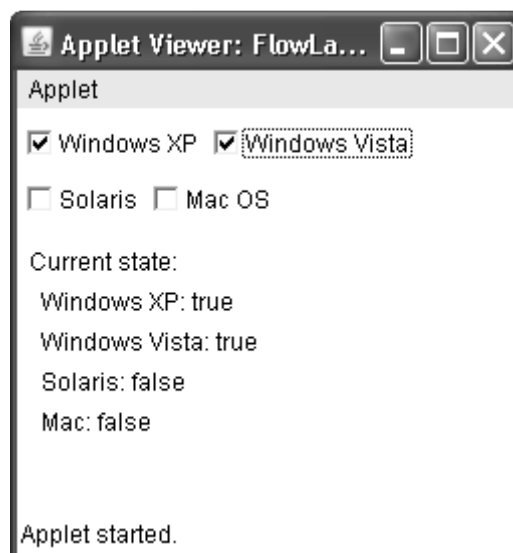
`mac = new Checkbox("Mac OS");`

```

add(winXP);
add(winVista);
add(solaris);
add(mac);
// register to receive item events
winXP.addItemListener(this);
winVista.addItemListener(this);
solaris.addItemListener(this);
mac.addItemListener(this);
}
// Repaint when status of a check box changes.
public void itemStateChanged(ItemEvent ie) {
    repaint();
}
// Display current state of the check boxes.
public void paint(Graphics g) {
    msg = "Current state: ";
    g.drawString(msg, 6, 80);
    msg = " Windows XP: " + winXP.getState();
    g.drawString(msg, 6, 100);
    msg = " Windows Vista: " + winVista.getState();
    g.drawString(msg, 6, 120);
    msg = " Solaris: " + solaris.getState();
    g.drawString(msg, 6, 140);
    msg = " Mac: " + mac.getState();
    g.drawString(msg, 6, 160);
}
}

```

Here is sample output generated by the FlowLayoutDemo applet.



BorderLayout

The BorderLayout class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center. The

four sides are referred to as north, south, east, and west. The middle area is called the center. Here are the constructors defined by BorderLayout:

BorderLayout()

BorderLayout(int horz, int vert)

The first form creates a default border layout. The second allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively. BorderLayout defines the following constants that specify the regions:

BorderLayout.CENTER

BorderLayout.SOUTH

BorderLayout.EAST

BorderLayout.WEST

BorderLayout.NORTH

When adding components, you will use these constants with the following form of `add()`, which is defined by Container:

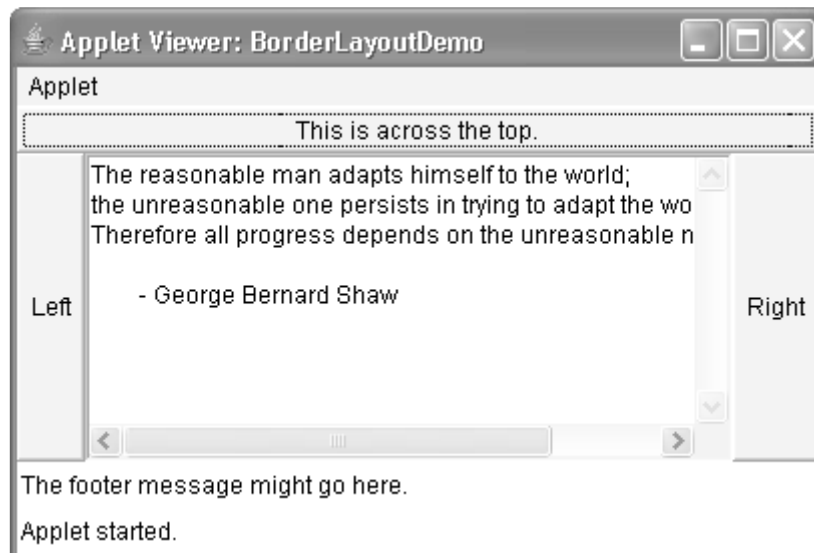
`void add(Component compObj, Object region)`

Here, *compObj* is the component to be added, and *region* specifies where the component will be added.

Here is an example of a BorderLayout with a component in each layout area:

```
// Demonstrate BorderLayout.
import java.awt.*;
import java.applet.*;
import java.util.*;
/*
<applet code="BorderLayoutDemo" width=400 height=200>
</applet>
*/
public class BorderLayoutDemo extends Applet {
    public void init() {
        setLayout(new BorderLayout());
        add(new Button("This is across the top."),
            BorderLayout.NORTH);
        add(new Label("The footer message might go here."),
            BorderLayout.SOUTH);
        add(new Button("Right"), BorderLayout.EAST);
        add(new Button("Left"), BorderLayout.WEST);
        String msg = "The reasonable man adapts " +
            "himself to the world;\n" +
            "the unreasonable one persists in " +
            "trying to adapt the world to himself.\n" +
            "Therefore all progress depends " +
            "on the unreasonable man.\n\n" +
            "- George Bernard Shaw\n\n";
        add(new TextArea(msg), BorderLayout.CENTER);
    }
}
```

Sample output from the BorderLayoutDemo applet is shown here:



GridLayout

GridLayout lays out components in a two-dimensional grid. When you instantiate a GridLayout, you define the number of rows and columns. The constructors supported by GridLayout are shown here:

`GridLayout()`

`GridLayout(int numRows, int numColumns)`

`GridLayout(int numRows, int numColumns, int horz, int vert)`

The first form creates a single-column grid layout. The second form creates a grid layout with the specified number of rows and columns. The third form allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively. Either *numRows* or *numColumns* can be zero. Specifying *numRows* as zero allows for unlimited-length columns. Specifying *numColumns* as zero allows for unlimited-length rows.

Here is a sample program that creates a 4×4 grid and fills it in with 15 buttons, each labeled with its index:

// Demonstrate GridLayout

import java.awt.;*

import java.applet.;*

*/**

<applet code="GridLayoutDemo" width=300 height=200>

</applet>

**/*

public class GridLayoutDemo extends Applet {

static final int n = 4;

public void init() {

setLayout(new GridLayout(n, n));

setFont(new Font("SansSerif", Font.BOLD, 24));

for(int i = 0; i < n; i++) {

for(int j = 0; j < n; j++) {

*int k = i * n + j;*

if(k > 0)

add(new Button("" + k));

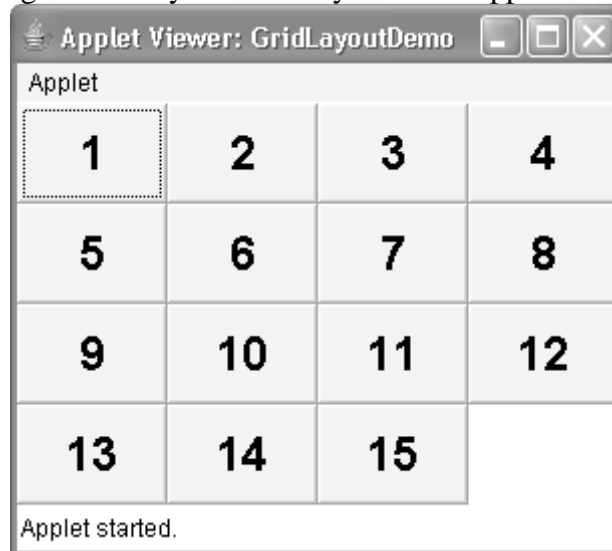
}

}

}

!

Following is the output generated by the GridLayoutDemo applet:



CardLayout

The CardLayout class is unique among the other layout managers in that it stores several different layouts. Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time. This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input. You can prepare the other layouts and have them hidden, ready to be activated when needed. CardLayout provides these two constructors:

CardLayout()

CardLayout(int horz, int vert)

The first form creates a default card layout. The second form allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively. Use of a card layout requires a bit more work than the other layouts. The cards are typically held in an object of type Panel. This panel must have CardLayout selected as its layout manager. The cards that form the deck are also typically objects of type Panel. Thus, you must create a panel that contains the deck and a panel for each card in the deck. Next, you add to the appropriate panel the components that form each card. You then add these panels to the panel for which CardLayout is the layout manager. Finally, you add this panel to the window. Once these steps are complete, you must provide some way for the user to select between cards. One common approach is to include one push button for each card in the deck. When card panels are added to a panel, they are usually given a name. Thus, most of the time, you will use this form of add() when adding cards to a panel:

void add(Component panelObj, Object name)

Here, *name* is a string that specifies the name of the card whose panel is specified by *panelObj*. After you have created a deck, your program activates a card by calling one of the following methods defined by CardLayout:

void first(Container deck)

void last(Container deck)

void next(Container deck)

void previous(Container deck)

void show(Container deck, String cardName)

Here, *deck* is a reference to the container (usually a panel) that holds the cards, and *cardName* is the name of a card. Calling first() causes the first card in the deck to be shown.

To show the last card, call `last()`. To show the next card, call `next()`. To show the previous card, call `previous()`. Both `next()` and `previous()` automatically cycle back to the top or bottom of the deck, respectively. The `show()` method displays the card whose name is passed in `cardName`. The following example creates a two-level card deck that allows the user to select an operating system. Windows-based operating systems are displayed in one card. Macintosh and Solaris are displayed in the other card.

// Demonstrate CardLayout.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CardLayoutDemo" width=300 height=100>
</applet>
*/
public class CardLayoutDemo extends Applet
implements ActionListener, MouseListener {
    Checkbox winXP, winVista, solaris, mac;
    Panel osCards;
    CardLayout cardLO;
    Button Win, Other;
    public void init() {
        Win = new Button("Windows");
        Other = new Button("Other");
        add(Win);
        add(Other);
        cardLO = new CardLayout();
        osCards = new Panel();
        osCards.setLayout(cardLO); // set panel layout to card layout
        winXP = new Checkbox("Windows XP", null, true);
        winVista = new Checkbox("Windows Vista");
        solaris = new Checkbox("Solaris");
        mac = new Checkbox("Mac OS");
        // add Windows check boxes to a panel
        Panel winPan = new Panel();
        winPan.add(winXP);
        winPan.add(winVista);
        // add other OS check boxes to a panel
        Panel otherPan = new Panel();
        otherPan.add(solaris);
        otherPan.add(mac);
        // add panels to card deck panel
        osCards.add(winPan, "Windows");
        osCards.add(otherPan, "Other");
        // add cards to main applet panel
        add(osCards);
        // register to receive action events
        Win.addActionListener(this);
        Other.addActionListener(this);
        // register mouse events
        addMouseListener(this);
    }
}
```

```

}
// Cycle through panels.
public void mousePressed(MouseEvent me) {
    cardLO.next(osCards);
}
// Provide empty implementations for the other MouseListener methods.
public void mouseClicked(MouseEvent me) {
}
public void mouseEntered(MouseEvent me) {
}
public void mouseExited(MouseEvent me) {
}
public void mouseReleased(MouseEvent me) {
}
public void actionPerformed(ActionEvent ae) {
    if(ae.getSource() == Win) {
        cardLO.show(osCards, "Windows");
    }
    else {
        cardLO.show(osCards, "Other");
    }
}
}
}
}

```

Here is the output generated by the CardLayoutDemo applet. Each card is activated by pushing its button. You can also cycle through the cards by clicking the mouse.



GridBagLayout

Although the preceding layouts are perfectly acceptable for many uses, some situations will require that you take a bit more control over how the components are arranged. A good way to do this is to use a grid bag layout, which is specified by the `GridBagLayout` class. What makes the grid bag useful is that you can specify the relative placement of

components by specifying their positions within cells inside a grid. The key to the grid bag is that each component can be a different size, and each row in the grid can have a different number of columns. This is why the layout is called a *grid bag*. It's a collection of small grids joined together._

Fill in the Blanks

1. ____ is a special type of program that is embedded in the webpage to generate the dynamic content.
2. Applets run under ____
3. Applet Life Cycle has ____ number of methods.
4. public void paint method provides ____ Class object that can be used to draw various shapes.
5. ____ is used to draw the specified string.
6. ____ is used draw the specified image.
7. The ____ method is called when the applet needs to be removed.
8. An ____ is an object that describes a state change in a source.
9. A ____ is an object that generates an event.
10. A ____ is an object that is notified when an event occurs.
11. ____ class is the superclass for all events.
12. To handle mouse events ____ & ____ interfaces must be implemented.
13. To handle keyboard events ____ interface must be implemented.
14. An ____ class provides an empty implementation of all methods in an event listener interface.
15. ____ is a class defined in another class.
16. The AWT classes are contained in the ____ package.
17. All graphics are drawn relative to a ____.
18. ____ are the passive controls which do not have any interaction with the user.
19. A ____ is a control that is used to turn an option on or off.
20. ____ is the default layout manager.

Short Answer Questions		
S.No	Questions	Blooms Taxonomy Levels
1	Define AWT class hierarchy.	Understand
2	List various events for handling mouse events.	remember
3	Define layout management.	Understand
4	State the Event Listeners	remember
5	Illustrate layout manager types in AWT.	remember
6	Describe Events and Event sources.	remember
7	Compare and contrast Event sources and Listeners.	Understand
8	Define Delegation event model.	Understand
9	List out various events used for handling a button click.	remember
10	Define adapter class?	Understand
11	Distinguish between applet and application?	remember
12	Illustrate the life cycle of an Applet.	remember
13	Describe applet security issues?	remember

Long Answer Questions		
S.No	Questions	Blooms Taxonomy Levels
1	Paraphrase Events, Event sources and Event classes.	Understand
2	Explain in detail about hierarchy for AWT.	Understand
3	Exemplify handling a button clicks in AWT.	Understand
4	Explain in detail about Layout management.	Understand
5	Illustrate mouse handling events with an example.	remember
6	Describe parameters passing to an applet with a program.	remember
7	Explain the differences between applets and applications	Understand
8	Compare and contrast Swing and AWT in java.	Understand
9	Explain in detail about Event sources and Listeners	Understand
10	Define an applet that receives an integer in one text field and computes its factorial value and returns it in another text field, when the button named “compute” is clicked	Understand
11	Explain briefly about Adapter classes.	Understand
12	Exemplify the importance of Delegation Event Model on Event Handling	Understand

Problem Solving and Critical Thinking Questions		
S.No	Questions	Blooms Taxonomy Levels
1	Identify the output of the program <pre> import java.awt.*; import java.applet.*; public class GridLayoutDemo extends Applet { static final int n=5; public void init() { setLayout(new GridLayout(n,n)); setFont (new Font (“SamsSerof”, Font.BOLD, 24)); for (int j=0l j<n; j++) { int k= I * n + j; if(k>00) Add(new button (“” + k0); } } } </pre>	Understand
2	Identify the output of following code <pre> Public void actionPerformed(ActionEvent e) { if(e.getSource()== b1) { int x= Integer.parseInt(t1.getText()); </pre>	Understand

	<pre> int y= Integer.parseInt(t2.getText()) ; int sum= X+Y; t3.setText(""+sum); } </pre>	
3	<p>Explain the output of the following program?</p> <pre> import java.applet.*; import java.awt.*; public class Main extends Applet { public void paint(Graphics g) { g.drawString("Welcome in Java Applet.",40,20); } } <HTML> <HEAD> </HEAD> <BODY> <div > <APPLET CODE="Main.class" WIDTH="800" HEIGHT="500"> </APPLET> </div> </BODY> </HTML> </pre>	Understand
4	<p>Predict output in following code?</p> <pre> public void actionPerformed(ActionEvent ae) { try{ num = Integer.parseInt(input.getText()); sum = sum+num; input.setText(""); output.setText(Integer.toString(sum)); lbl.setForeground(Color.blue); lbl.setText("Output of the second Text Box : "+ output.getText()); } catch(NumberFormatException e) { lbl.setForeground(Color.red); lbl.setText("Invalid Entry!"); } } </pre>	Understand
5	<p>Analyze the program output.</p> <pre> import java.awt.*; class Frame1 extends Frame { </pre>	Understand

	<pre> Frame1() { setTitle("demo"); setSize(200,200); setVisible(true); setLayout(newFlowLayout()); Label l1= new Label("java"); Label l2= new Label("j2ee"); add(l1); add(l2); } } Class Labeldemo { Public static void main(String args()); { Frame1 f= new Frame(); } } </pre>	
6	<p>Identify the program output</p> <pre> import java.awt.*; import java.applet.*; public class satusdemo extends Applet { Public void init() { setBackground(Color.red); } Public void paint(Graphics g) { g.drawString("this is in the applet window" 10,20)" showStatus("this is the status window message"); } } </pre>	Understand
7	<p>Identify the program output</p> <pre> Public void mouseClicked(MouseEvent me) { Mousex-=0; Mousey=10; Msg= "mouse clicked" Repaint(); } Public void mouseEntered(MouseEvent me) { Mousex-=0; Mousey=10; Msg= "mouse entered" Repaint(); } </pre>	Understand

8	<p>Identify the program output</p> <pre> Public void mouseClicked(MouseEvent me) { Mousex-=0; Mousey=10; Msg= "mouse clicked" Repaint(); } Public void mouseEntered(MouseEvent me) { Mousex-=0; Mousey=10; Msg= "mouse entered" Repaint(); } </pre>	Understand
---	--	------------

Review Questions

1. What Is An Applet? Should Applets Have Constructors?
2. What Is The Order Of Method Invocation In An Applet?
3. What Are The Applets Life Cycle Methods?
4. What Is The Sequence For Calling The Methods By Awt For Applets?
5. When An Applet Is Terminated, what Sequence Of Method Calls Takes Place
6. How Do Applets Differ From Applications?
7. Can We Pass Parameters To An Applet From Html Page To An Applet?
8. Which Classes And Interfaces Does Applet Class Consist?
9. Which Method Is Used To Output A String To An Applet? Which Function Is This Method Included In?
10. When Is Update Method Called?
11. What Are The Attributes Of Applet Tags?
12. How Can We Determine The Width And Height Of A Applet?
13. What Are The Methods That Control An Applet's On-screen Appearance?
14. Can You Write A Java Class That Could Be Used Both As An Applet As Well As An Application?
15. What are the component and container class?
16. What is the difference between the paint() and repaint() method?
17. What interface is extended by AWT event listener?
18. What is a container in a GUI?
19. What is the default layout for Applet?
20. Name Components subclasses that support painting?
21. What is the difference between Grid and GridbagLayout?
22. What is a layout manager?
23. Where the CardLayout is used?
24. Which container may contain a menu bar?
25. What are the types of checkboxes?
26. What is paint method?
27. What is the purpose of repaint method?
28. What are the subclasses of the Container class?
29. Which containers use a Border layout as their default layout?
30. What is the difference between choice and list?

31. What is the difference between a window and a frame?
32. Which method is method to set the layout of a container?
33. What are the default layouts for a applet, a frame and a panel?
34. What is the difference between a Scrollbar and a Scrollpane?
35. Which method will cause a Frame to be displayed?
36. Explain the use of update method?
37. What are the subclass of textcomponent class?
38. When is update method called?
39. When do you use codebase in applets?

UNIT V

UNIT WISE PLAN

UNIT-V: Swings		Planned Hours: 09	
S. No.	Topic Learning Outcomes	Cos	Blooms Levels
1	Distinguish applets and swings	CO5	L4
2	Explain MVC Architecture	CO5	L2
3	Use swing controls in GUI Programming	CO5	L6
4	Use graphical IDE for design and implementation of real time applications in java.	CO5	L6

Java Swing Tutorial

Java Swings is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

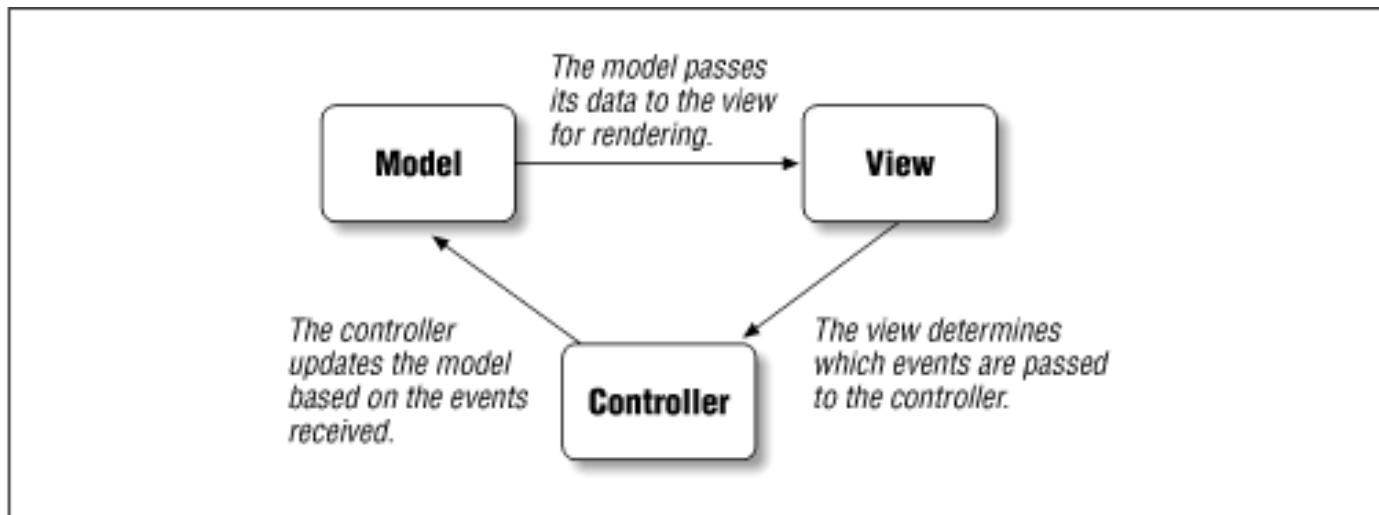
Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent.	Java swing components are platform-independent.
2)	AWT components are heavyweight.	Swing components are lightweight.
3)	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
4)	AWT provides fewer components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

The Model-View-Controller Architecture

Swing uses the *model-view-controller architecture* (MVC) as the fundamental design behind each of its components. Essentially, MVC breaks GUI components into three elements. Each of these elements plays a crucial role in how the component behaves.



Model

The model encompasses the state data for each component. There are different models for different types of components. For example, the model of a scrollbar component might contain information about the current position of its adjustable “thumb,” its minimum and maximum values, and the thumb’s width (relative to the range of values). A menu, on the other hand, may simply contain a list of the menu items the user can select from. Note that this information remains the same no matter how the component is painted on the screen; model data always exists independent of the component’s visual representation.

View

The view refers to how you see the component on the screen. For a good example of how views can differ, look at an application window on two different GUI platforms. Almost all window frames will have a titlebar spanning the top of the window. However, the titlebar may have a close box on the left side (like the older MacOS platform), or it may have the close box on the right side (as in the Windows 95 platform). These are examples of different types of views for the same window object.

Controller

The controller is the portion of the user interface that dictates how the component interacts with events. Events come in many forms — a mouse click, gaining or losing focus, a keyboard event that triggers a specific menu command, or even a directive to repaint part of the screen. The controller decides how each component will react to the event—if it reacts at all.

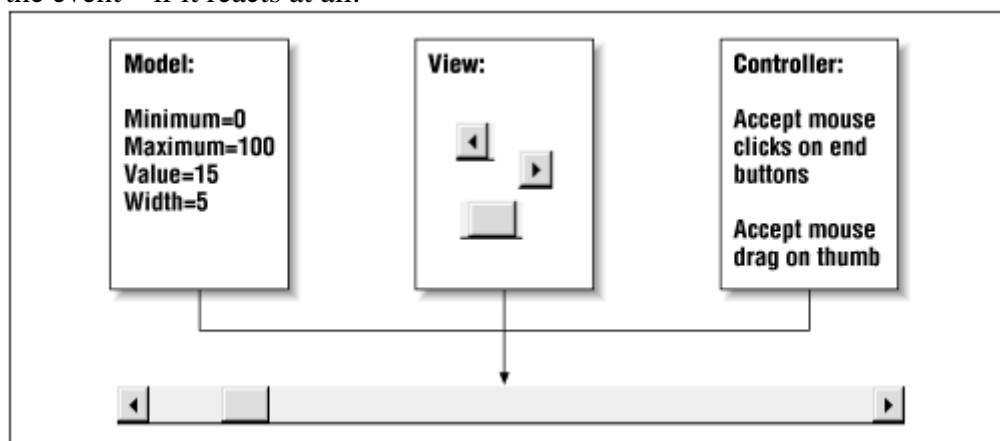
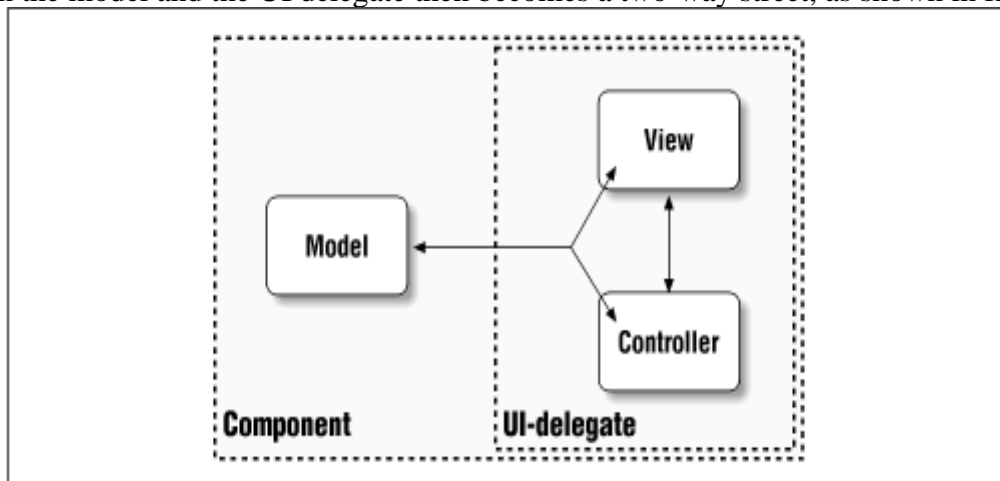


Figure shows how the model, view, and controller work together to create a scrollbar component. The scrollbar uses the information in the model to determine how far into the scrollbar to render the thumb and how wide the thumb should be. Note that the model specifies this information relative to the minimum and the maximum. It does not give the position or width of the thumb in screen pixels—the view calculates that. The view determines exactly where and how to draw the scrollbar, given the proportions offered by the model. The view knows whether it is a horizontal or vertical scrollbar, and it knows exactly how to shadow the end buttons and the thumb. Finally, the controller is responsible for handling mouse events on the component. The controller knows, for example, that dragging the thumb is a legitimate action for a scroll bar, and pushing on the end buttons is acceptable as well. The result is a fully functional MVC scrollbar.

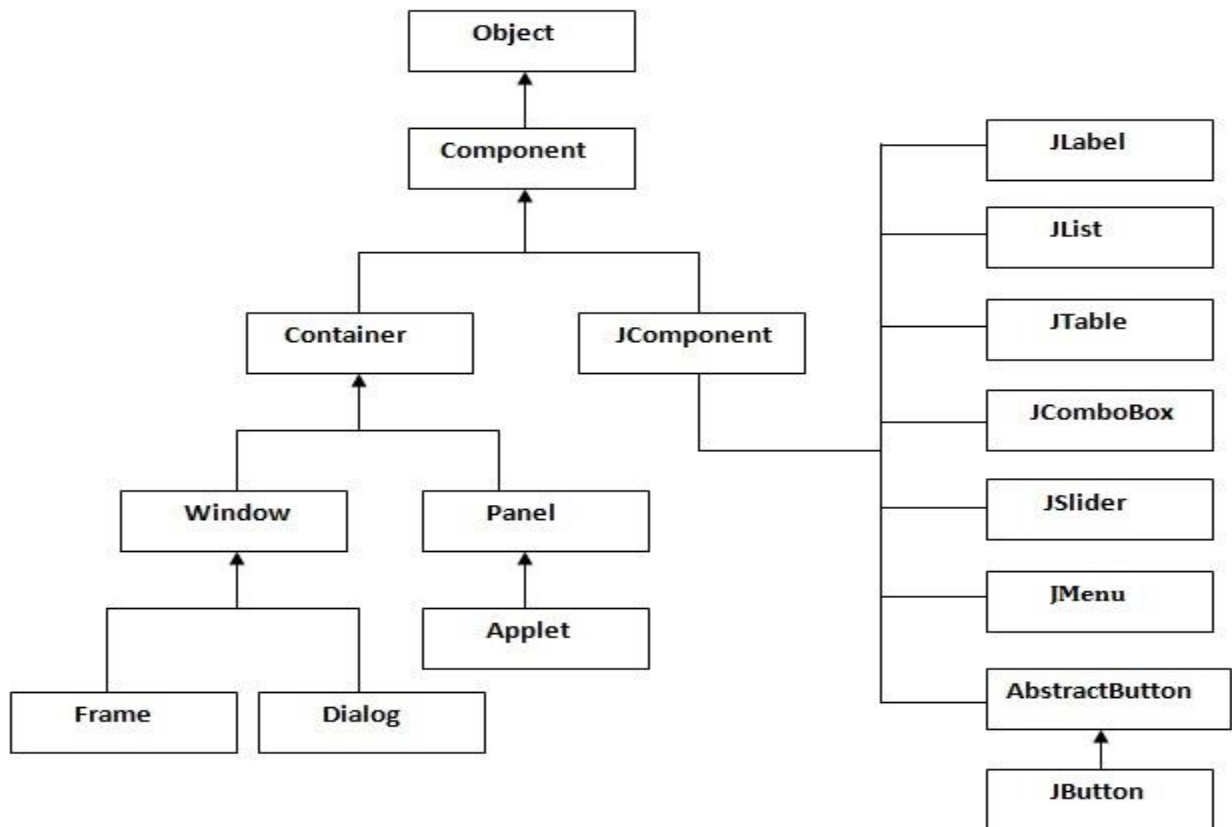
MVC in Swing

Swing actually makes use of a simplified variant of the MVC design called the *model-delegate*. This design combines the view and the controller object into a single element that draws the component to the screen and handles GUI events known as the *UI delegate*. Bundling graphics capabilities and event handling is somewhat easy in Java, since much of the event handling is taken care of in AWT. As you might expect, the communication between the model and the UI delegate then becomes a two-way street, as shown in fig



Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main (), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```

import javax.swing.*;

public class FirstSwingExample {
    public static void main(String[] args) {
        JFrame f=new JFrame();//creating instance of JFrame

        JButton b=new JButton("click");//creating instance of JButton
  
```

```
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
```

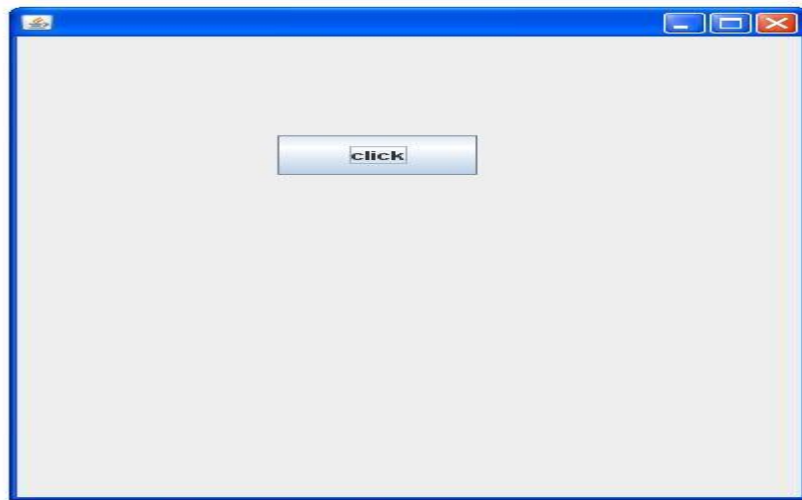
```
f.add(b);//adding button in JFrame
```

```
f.setSize(400,500);//400 width and 500 height
```

```
f.setLayout(null);//using no layout managers
```

```
f.setVisible(true);//making the frame visible
```

```
}  
}
```



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
import javax.swing.*;
```

```
public class Simple {
```

```
    JFrame f;
```

```
    Simple(){
```

```
        f=new JFrame();//creating instance of JFrame
```

```
        JButton b=new JButton("click");//creating instance of JButton
```

```
        b.setBounds(130,100,100, 40);
```

```
        f.add(b);//adding button in JFrame
```

```
        f.setSize(400,500);//400 width and 500 height
```

```
        f.setLayout(null);//using no layout managers
```

```
        f.setVisible(true);//making the frame visible
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        new Simple();
```

```
    }
```

```
}
```

The setBounds(int xaxis, int yaxis, int width, int height) is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
    JFrame f;
    Simple2(){
        JButton b=new JButton("click");//create button
        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame
        setSize(400,500);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String[] args) {
        new Simple2();
    }
}
```

Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. public class JButton extends AbstractButton implements Accessible

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton Example

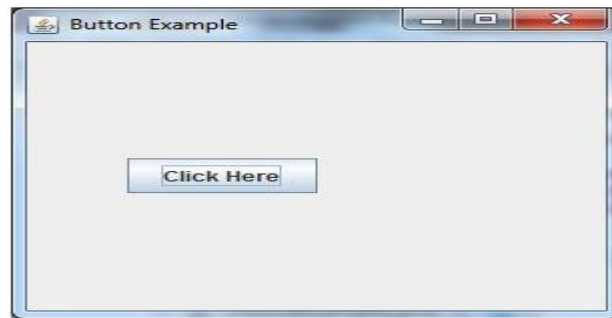
```
import javax.swing.*;
public class ButtonExample {
    public static void main(String[] args) {
```



```

JFrame f=new JFrame("Button Example");
JButton b=new JButton("Click Here");
b.setBounds(50,100,95,30);
f.add(b);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}

```

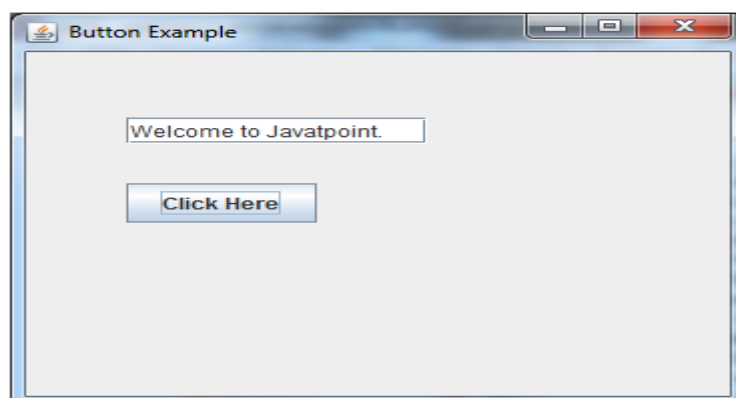


Java JButton Example with ActionListener

```

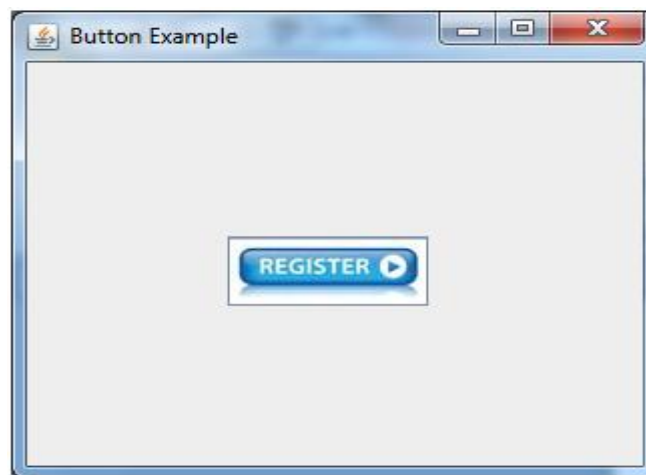
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
    tf.setText("Welcome to Javatpoint.");
    }
});
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}

```



Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample{
    ButtonExample(){
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton(new ImageIcon("D:\\icon.png"));
        b.setBounds(100,100,100, 40);
        f.add(b);
        f.setSize(300,400);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        new ButtonExample();
    }
}
```



Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

1. public class JLabel extends JComponent implements SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

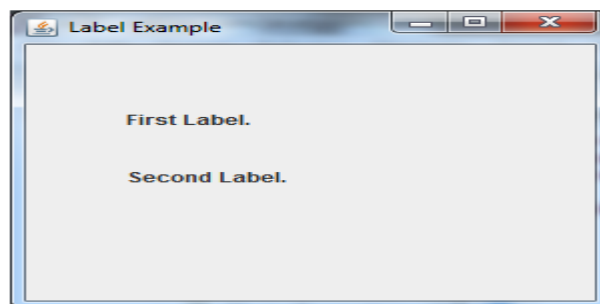
Commonly used Methods:

Methods	Description
---------	-------------

String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example

```
import javax.swing.*;
class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



Java JLabel Example with ActionListener

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener{
    JTextField tf; JLabel l; JButton b;
    LabelExample(){
        tf=new JTextField();
        tf.setBounds(50,50, 150,20);
        l=new JLabel();
        l.setBounds(50,100, 250,20);
        b=new JButton("Find IP");
        b.setBounds(50,150,95,30);
        b.addActionListener(this);
    }
}
```

```

        add(b);add(tf);add(l);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        try{
            String host=tf.getText();
            String ip=java.net.InetAddress.getByName(host).getHostAddress();
            l.setText("IP of "+host+" is: "+ip);
        }catch(Exception ex){System.out.println(ex);}
    }
    public static void main(String[] args) {
        new LabelExample();
    }
}

```



Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

1. public class JTextField extends JTextComponent implements SwingConstants

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

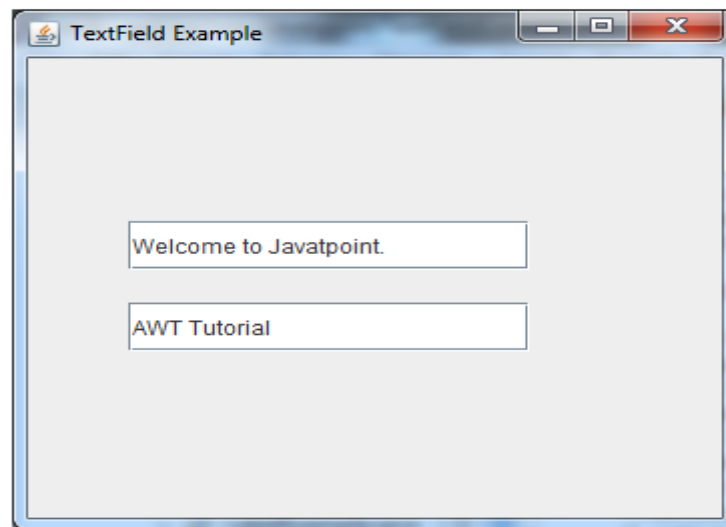
Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.

void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Java JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Welcome to Javatpoint.");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("AWT Tutorial");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



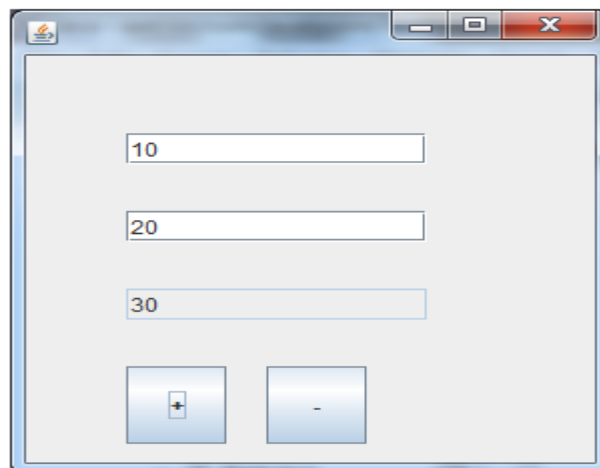
Java JTextField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener{
    JTextField tf1,tf2,tf3;
    JButton b1,b2;
    TextFieldExample(){
        JFrame f= new JFrame();
        tf1=new JTextField();
        tf1.setBounds(50,50,150,20);
        tf2=new JTextField();
```

```

tf2.setBounds(50,100,150,20);
tf3=new JTextField();
tf3.setBounds(50,150,150,20);
tf3.setEditable(false);
b1=new JButton("+");
b1.setBounds(50,200,50,50);
b2=new JButton("-");
b2.setBounds(120,200,50,50);
b1.addActionListener(this);
b2.addActionListener(this);
f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public void actionPerformed(ActionEvent e) {
    String s1=tf1.getText();
    String s2=tf2.getText();
    int a=Integer.parseInt(s1);
    int b=Integer.parseInt(s2);
    int c=0;
    if(e.getSource()==b1){
        c=a+b;
    }else if(e.getSource()==b2){
        c=a-b;
    }
    String result=String.valueOf(c);
    tf3.setText(result);
}
public static void main(String[] args) {
    new TextFieldExample();
} }

```



Java JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

JTextArea class declaration

Let's see the declaration for javax.swing.JTextArea class.

1. public class JTextArea extends JTextComponent

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

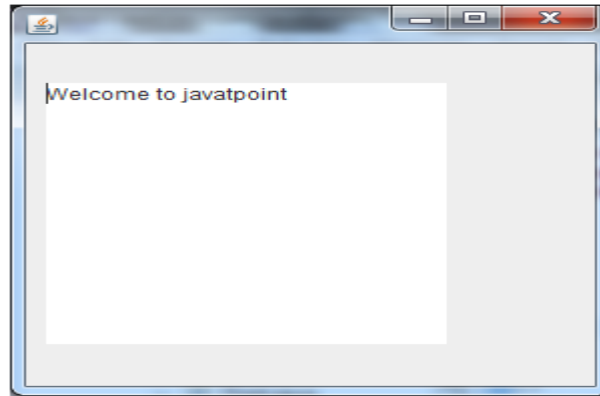
Commonly used Methods:

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Java JTextArea Example

```
import javax.swing.*;
public class TextAreaExample
{
    TextAreaExample(){
        JFrame f= new JFrame();
        JTextArea area=new JTextArea("Welcome to javatpoint");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaExample();
    }
}
```

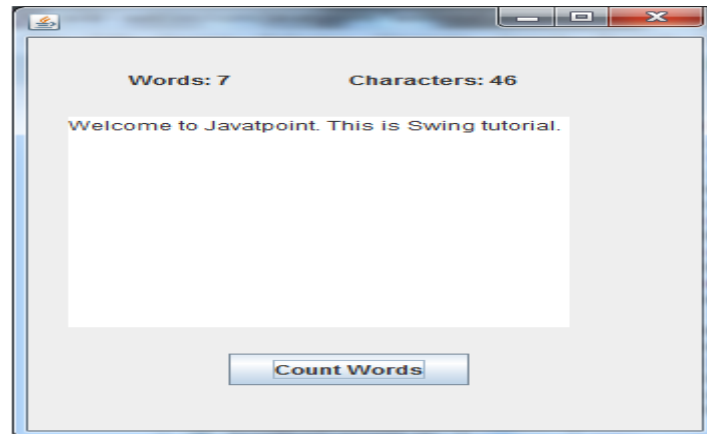
Output:



Java JTextArea Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener{
    JLabel l1,l2;
    JTextArea area;
    JButton b;
    TextAreaExample() {
        JFrame f= new JFrame();
        l1=new JLabel();
        l1.setBounds(50,25,100,30);
        l2=new JLabel();
        l2.setBounds(160,25,100,30);
        area=new JTextArea();
        area.setBounds(20,75,250,200);
        b=new JButton("Count Words");
        b.setBounds(100,300,120,30);
        b.addActionListener(this);
        f.add(l1);f.add(l2);f.add(area);f.add(b);
        f.setSize(450,450);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        String text=area.getText();
        String words[]=text.split("\\s");
        l1.setText("Words: "+words.length);
        l2.setText("Characters: "+text.length());
    }
    public static void main(String[] args) {
        new TextAreaExample();
    }
}
```

Output:



Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on". It inherits JToggleButton class.

JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

1. public class JCheckBox extends JToggleButton implements Accessible

Commonly used Constructors:

Constructor	Description
JCheckBox()	Creates an initially unselected check box button with no text, no icon.
JCheckBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

Java JCheckBox Example

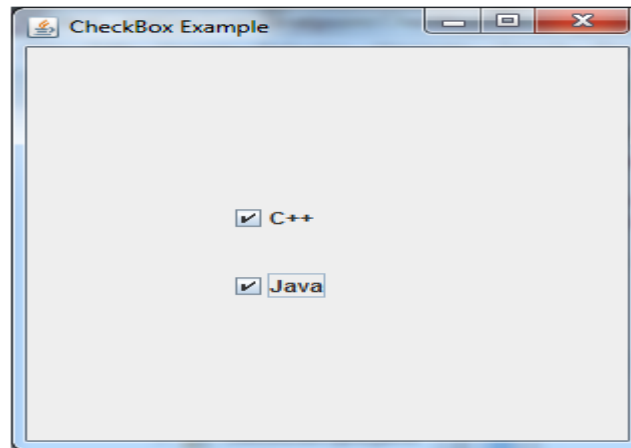
```
import javax.swing.*;
public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
    }
}
```

```

        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckBoxExample();
    }
}

```

Output:



Java JCheckBox Example with ItemListener

```

import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JCheckBox checkbox1 = new JCheckBox("C++");
        checkbox1.setBounds(150,100, 50,50);
        JCheckBox checkbox2 = new JCheckBox("Java");
        checkbox2.setBounds(150,150, 50,50);
        f.add(checkbox1); f.add(checkbox2); f.add(label);
        checkbox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("C++ Checkbox: "
                    + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });
        checkbox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("Java Checkbox: "
                    + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}

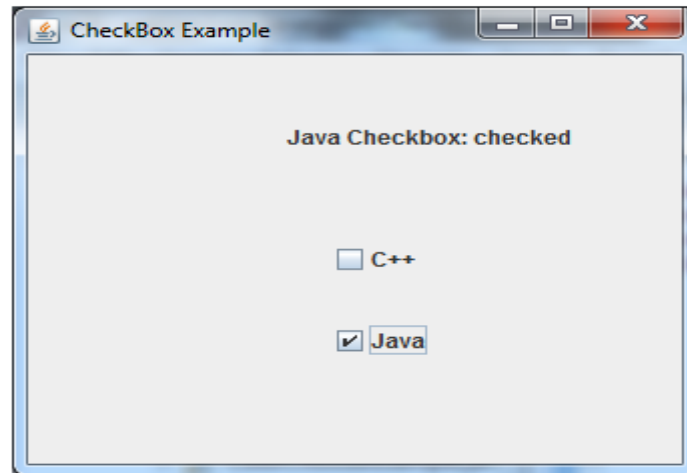
```

```

    }
    public static void main(String args[])
    {
        new CheckBoxExample();
    }
}

```

Output:



Java JCheckBox Example: Food Order

```

import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample extends JFrame implements ActionListener{
    JLabel l;
    JCheckBox cb1,cb2,cb3;
    JButton b;
    CheckBoxExample(){
        l=new JLabel("Food Ordering System");
        l.setBounds(50,50,300,20);
        cb1=new JCheckBox("Pizza @ 100");
        cb1.setBounds(100,100,150,20);
        cb2=new JCheckBox("Burger @ 30");
        cb2.setBounds(100,150,150,20);
        cb3=new JCheckBox("Tea @ 10");
        cb3.setBounds(100,200,150,20);
        b=new JButton("Order");
        b.setBounds(100,250,80,30);
        b.addActionListener(this);
        add(l);add(cb1);add(cb2);add(cb3);add(b);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e){
        float amount=0;
        String msg="";
        if(cb1.isSelected()){
            amount+=100;

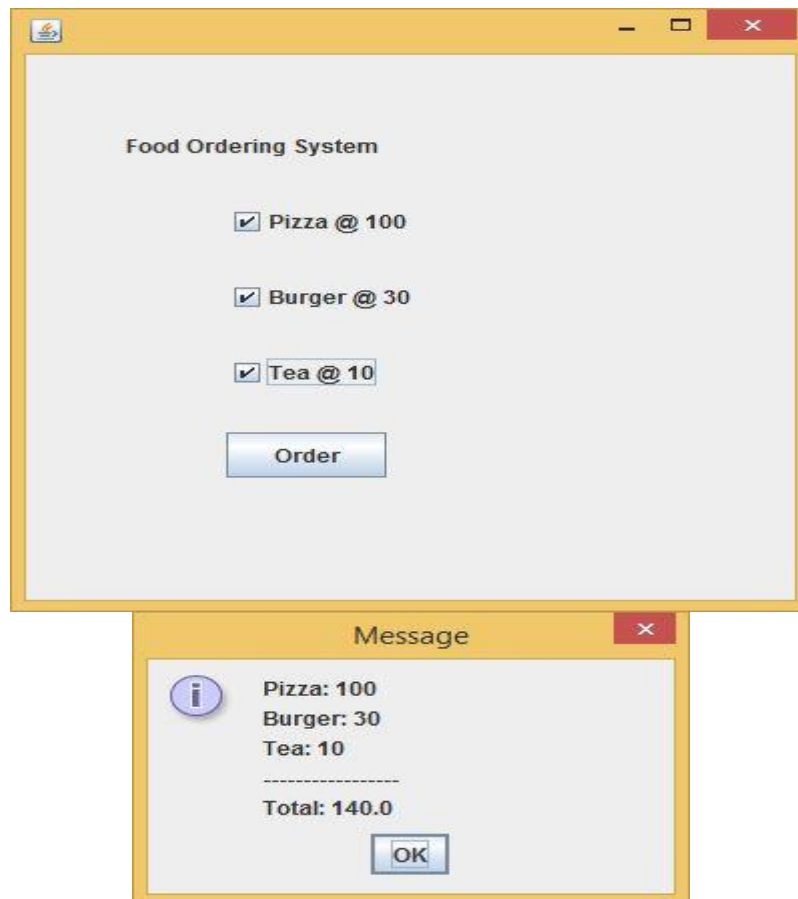
```

```

        msg="Pizza: 100\n";
    }
    if(cb2.isSelected()){
        amount+=30;
        msg+="Burger: 30\n";
    }
    if(cb3.isSelected()){
        amount+=10;
        msg+="Tea: 10\n";
    }
    msg+="-----\n";
    JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
}
public static void main(String[] args) {
    new CheckBoxExample();
}
}

```

Output:



Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

1. public class JRadioButton extends JButton implements Accessible

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

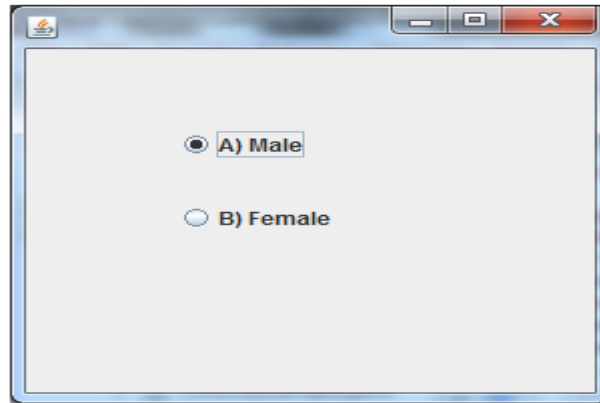
Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JRadioButton Example

```
import javax.swing.*;
public class RadioButtonExample {
    JFrame f;
    RadioButtonExample(){
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);bg.add(r2);
        f.add(r1);f.add(r2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new RadioButtonExample();
    }
}
```

Output:



Java JRadioButton Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;

class RadioButtonExample extends JFrame implements ActionListener{
    JRadioButton rb1,rb2;
    JButton b;

    RadioButtonExample(){
        rb1=new JRadioButton("Male");
        rb1.setBounds(100,50,100,30);
        rb2=new JRadioButton("Female");
        rb2.setBounds(100,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);bg.add(rb2);
        b=new JButton("click");
        b.setBounds(100,150,80,30);
        b.addActionListener(this);
        add(rb1);add(rb2);add(b);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e){
        if(rb1.isSelected()){
            JOptionPane.showMessageDialog(this,"You are Male.");
        }
        if(rb2.isSelected()){
            JOptionPane.showMessageDialog(this,"You are Female.");
        }
    }

    public static void main(String args[]){
        new RadioButtonExample();
    }
}
```

Output:



Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

1. public class JComboBox extends JComponent implements ItemSelectable, ListDataListener, ActionListener, Accessible

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Java JComboBox Example

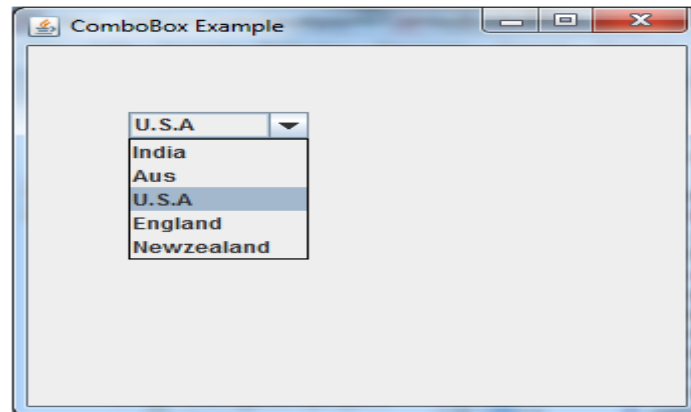
```
import javax.swing.*.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        String country[]={ "India","Aus","U.S.A","England","Newzealand"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50,90,20);
    }
}
```

```

        f.add(cb);
        f.setLayout(null);
        f.setSize(400,500);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ComboBoxExample();
    }
}

```

Output:



Java JComboBox Example with ActionListener

```

import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JButton b=new JButton("Show");
        b.setBounds(200,100,75,20);
        String languages[]={ "C","C++","C#","Java","PHP" };
        final JComboBox cb=new JComboBox(languages);
        cb.setBounds(50, 100,90,20);
        f.add(cb); f.add(label); f.add(b);
        f.setLayout(null);
        f.setSize(350,350);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "Programming language Selected: "
                    + cb.getItemAt(cb.getSelectedIndex());
                label.setText(data);
            }
        });
    }
}

```

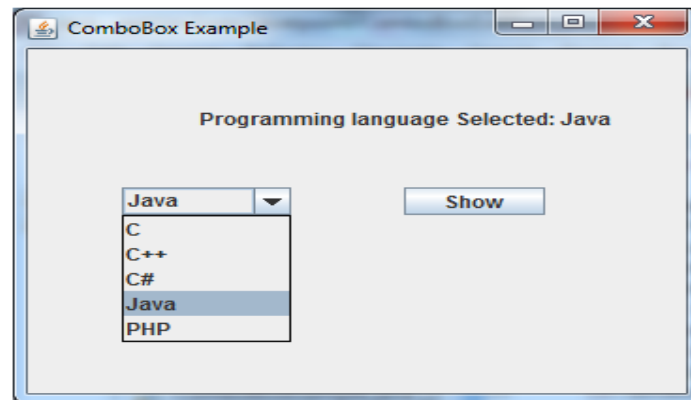


```

public static void main(String[] args) {
    new ComboBoxExample();
}
}

```

Output:



Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

Let's see the declaration for javax.swing.JTable class.

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

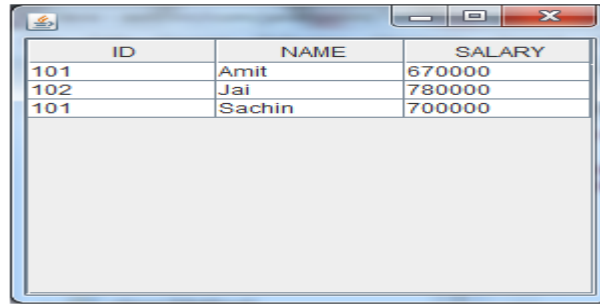
Java JTable Example

```

import javax.swing.*.*;
public class TableExample {
    JFrame f;
    TableExample(){
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };
        String column[]={ "ID","NAME","SALARY"};
        JTable jt=new JTable(data,column);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300,400);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new TableExample();
    }
}

```

Output

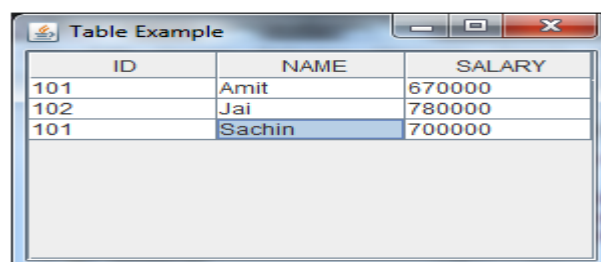


ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Java JTable Example with ListSelectionListener

```
import javax.swing.*.*;
import javax.swing.event.*;
public class TableExample {
    public static void main(String[] a) {
        JFrame f = new JFrame("Table Example");
        String data[][]= { {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };
        String column[]={"ID","NAME","SALARY"};
        final JTable jt=new JTable(data,column);
        jt.setCellSelectionEnabled(true);
        ListSelectionModel select= jt.getSelectionModel();
        select.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        select.addListSelectionListener(new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                String Data = null;
                int[] row = jt.getSelectedRows();
                int[] columns = jt.getSelectedColumns();
                for (int i = 0; i < row.length; i++) {
                    for (int j = 0; j < columns.length; j++) {
                        Data = (String) jt.getValueAt(row[i], columns[j]);
                    }
                }
                System.out.println("Table element selected is: " + Data);
            }
        });
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300, 200);
        f.setVisible(true);
    }
}
```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

If you select an element in column NAME, name of the element will be displayed on the console:

Table element selected is: Sachin

Java JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration

Let's see the declaration for javax.swing.JList class.

1. public class JList extends JComponent implements Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Java JList Example

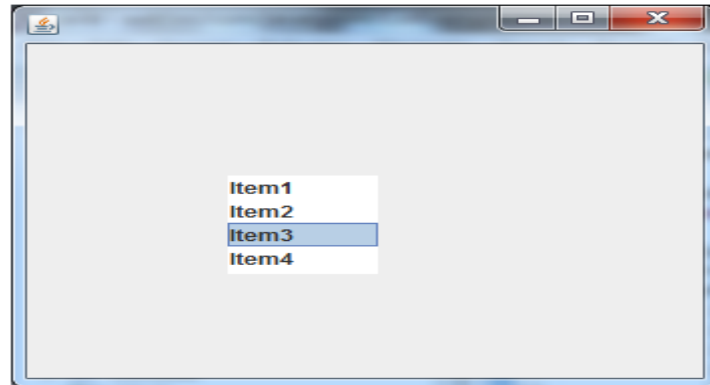
```
import javax.swing.*.*;
public class ListExample
{
    ListExample(){
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

```

    }
    public static void main(String args[])
    {
        new ListExample();
    }
}

```

Output:



Java JList Example with ActionListener

```

import javax.swing.*;
import java.awt.event.*;
public class ListExample
{
    ListExample(){
        JFrame f= new JFrame();
        final JLabel label = new JLabel();
        label.setSize(500,100);
        JButton b=new JButton("Show");
        b.setBounds(200,150,80,30);
        final DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("C");
        l1.addElement("C++");
        l1.addElement("Java");
        l1.addElement("PHP");
        final JList<String> list1 = new JList<>(l1);
        list1.setBounds(100,100, 75,75);
        DefaultListModel<String> l2 = new DefaultListModel<>();
        l2.addElement("Turbo C++");
        l2.addElement("Struts");
        l2.addElement("Spring");
        l2.addElement("YII");
        final JList<String> list2 = new JList<>(l2);
        list2.setBounds(100,200, 75,75);
        f.add(list1); f.add(list2); f.add(b); f.add(label);
        f.setSize(450,450);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "";
                if (list1.getSelectedIndex() != -1) {
                    data = "Programming language Selected: " + list1.getSelectedValue();
                }
            }
        });
    }
}

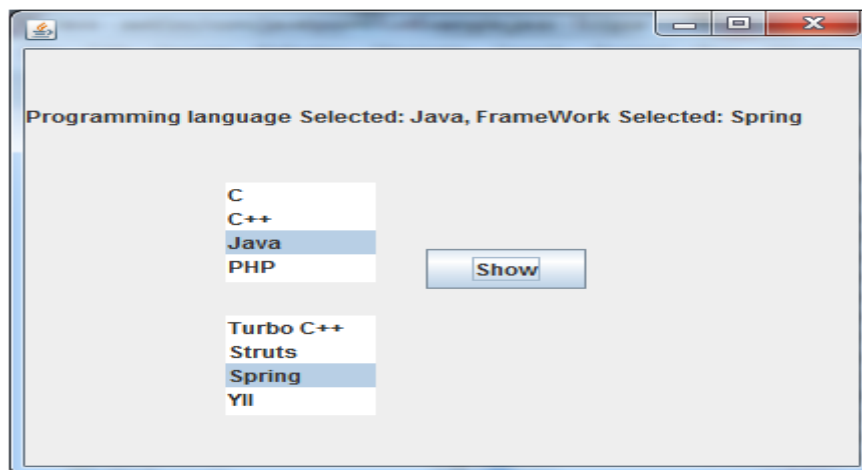
```

```

        label.setText(data);
    }
    if(list2.getSelectedIndex() != -1){
        data += ", FrameWork Selected: ";
        for(Object frame :list2.getSelectedValues()){
            data += frame + " ";
        }
    }
    label.setText(data);
}
});
}
public static void main(String args[])
{
    new ListExample();
}}

```

Output:



Java JTree

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

JTree class declaration

Let's see the declaration for javax.swing.JTree class.

1. public class JTree extends JComponent implements Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

Java JTree Example

```

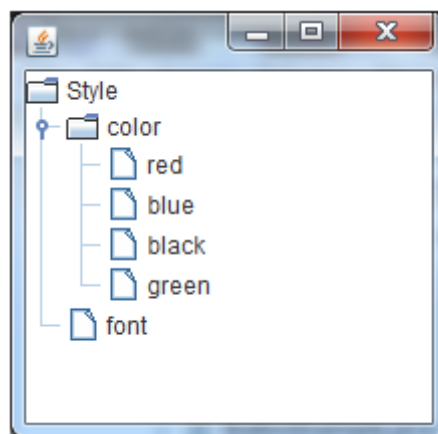
import javax.swing.*.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample {
    JFrame f;

```

```

TreeExample(){
    f=new JFrame();
    DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
    DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
    DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
    style.add(color);
    style.add(font);
    DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
    DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
    DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");
    DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");
    color.add(red); color.add(blue); color.add(black); color.add(green);
    JTree jt=new JTree(style);
    f.add(jt);
    f.setSize(200,200);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TreeExample();
}
}
Output:

```



Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

Let's see the declaration for javax.swing.JTabbedPane class.

public class JTabbedPane extends JComponent implements Serializable, Accessible, SwingConstants

Commonly used Constructors:

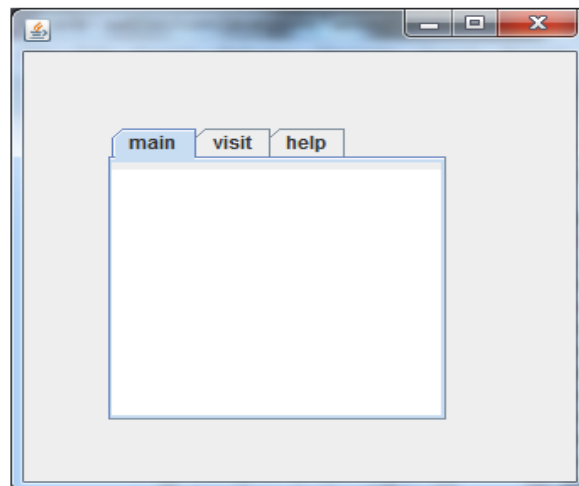
Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.

JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.
--	---

Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample {
    JFrame f;
    TabbedPaneExample(){
        f=new JFrame();
        JTextArea ta=new JTextArea(200,200);
        JPanel p1=new JPanel();
        p1.add(ta);
        JPanel p2=new JPanel();
        JPanel p3=new JPanel();
        JTabbedPane tp=new JTabbedPane();
        tp.setBounds(50,50,200,200);
        tp.add("main",p1);
        tp.add("visit",p2);
        tp.add("help",p3);
        f.add(tp);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new TabbedPaneExample();
    }
}
```

Output:



Java JScrollPane

A JScrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Constructors

Constructor	Purpose
-------------	---------

JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

Useful Methods

Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
void	setCorner(String, Component)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in JScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
Component	getCorner(String)	
void	setViewportView(Component)	Set the scroll pane's client.

JScrollPane Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class JScrollPaneExample {
    private static final long serialVersionUID = 1L;

    private static void createAndShowGUI() {

        // Create and set up the window.
        final JFrame frame = new JFrame("Scroll Pane Example");

        // Display the window.
```



```

frame.setSize(500, 500);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// set flow layout for the frame
frame.getContentPane().setLayout(new FlowLayout());

JTextArea textArea = new JTextArea(20, 20);
JScrollPane scrollableTextArea = new JScrollPane(textArea);

scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_S
CROLLBAR_ALWAYS);
scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROL
LBAR_ALWAYS);

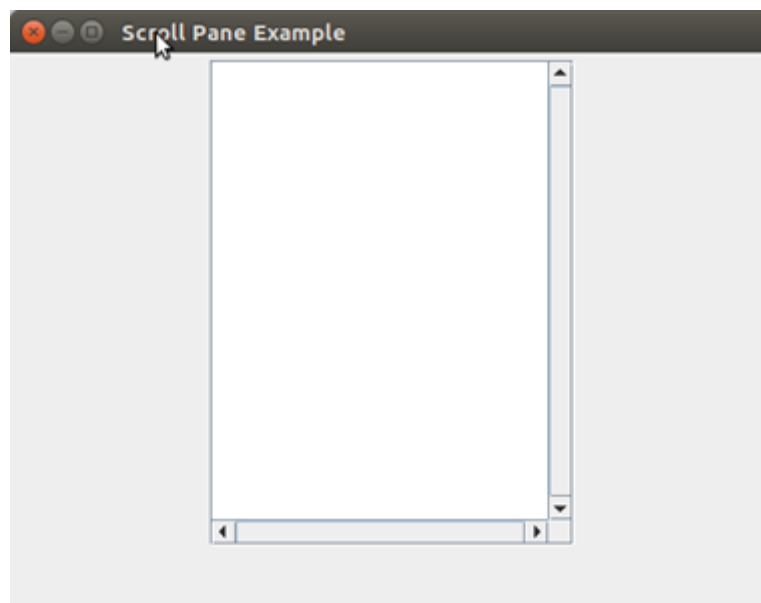
frame.getContentPane().add(scrollableTextArea);
}
public static void main(String[] args) {

    javax.swing.SwingUtilities.invokeLater(new Runnable() {

        public void run() {
            createAndShowGUI();
        }
    });
}
}

```

Output:



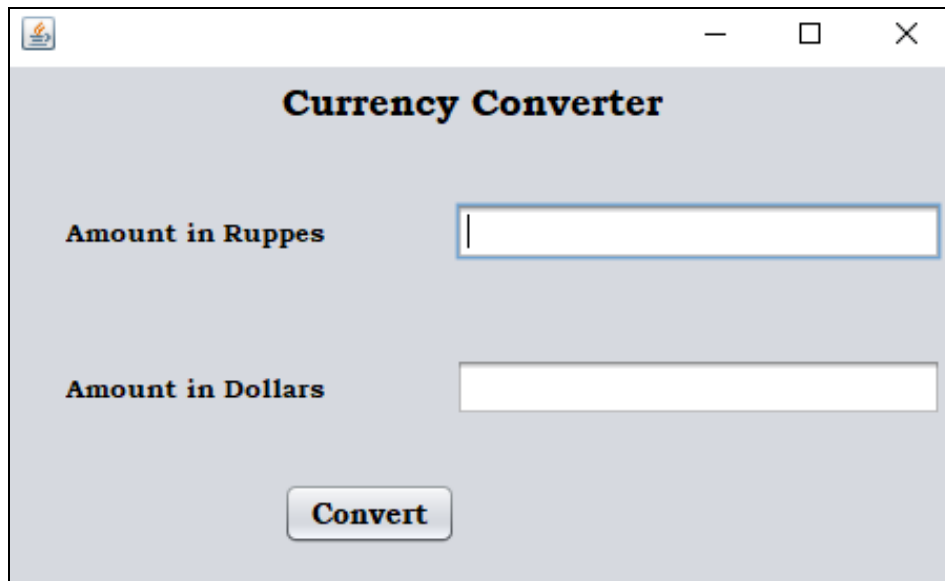
Fill in the Blanks

1. The _____ package provides classes for java swing API such as JButton, JTextField etc.
2. Java swing components are platform_____
3. _____ method creates a button with no text and icon.
4. _____ method creates an unselected radio button with specified text.
5. The _____ class is used to create a text area.
6. _____ method is used to remove all the items from the list.
7. _____ method creates a table with empty cells.
8. _____ method is used to set the graphics current font to the specified font.
9. The _____ are used to arrange components in a particular manner.
10. The BorderLayout is used to arrange the components in ____ number of regions.
11. MVC Stands for____,_____ &_____
12. A _____ provides a space where a component can be located.
13. Component represents an object with _____ representation.
14. A _____ is the abstract base class for the non menu user-interface controls of SWING.
15. A _____ is a base class for all swing UI components.
16. _____ are generated as result of user interaction with the graphical user interface components.
17. The _____ is an object on which event occurs.
18. _____ is responsible for generating response to an event.
19. _____ is the root class from which all event state objects shall be derived.
20. _____ interface is responsible to handle events.

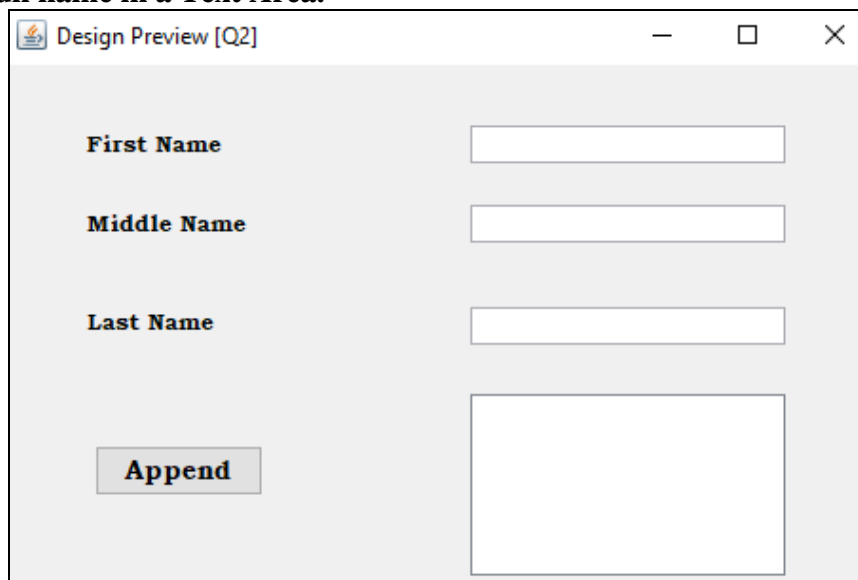
Short Answer Questions		
S.No	Questions	Blooms Taxonomy Level
1	Differentiate AWT and Swings.	Understand
2	Explain MVC Architecture.	Understand
3	Write a Java Program to create a button using Swings.	Understand
4	Explain JButton Class	Understand
5	Describe JApplet	Understand
6	Write a Java Program to create Icons and Labels using Swings.	Understand
7	Write a Java Program to create text fields using Swings.	Understand
8	Write a Java Program to create Check boxes using Swings.	Understand
9	Write a Java Program to create Radio buttons using Swings.	Understand
10	Write a Java Program to create Tabbed Panes using Swings.	Understand
11	List out swing components.	remember

Long Answer Questions

1. **Develop a Java Applet Program which has the following GUI, converts the value of a currency entered in Rupees to Dollar.**



2. Develop a Java Applet Program which has the following GUI, accepts First Name, Middle Name and Last Names into three different Text Fields and Prints the full name in a Text Area.



3. Develop a Java Applet Program which has the following GUI.
Write the code for
 - a) To display the series of odd or even number(depending on Starting Number and Last Number entered in TextField 1 and TextField 2 respectively)in the Text Area on click of "Display the Series" Button.
For Example:
If the starting number is 5 and last number is 11 then text area should contain 5,7,9,11.
If the starting number is 2 and last number is 10 then text area should contain 2,4,6,8,10.
 - b) To clear both the text fields and text area by click on "Reset" button.
 - c) To terminate the application by click on "Clear" button

Enter Starting Numer

Enter Last Number

Series

Display the Series

Reset

Clear

4. Develop a Java Applet Program which has the following GUI.

English

Analytical Skills

General Knowledge

Total

Grade

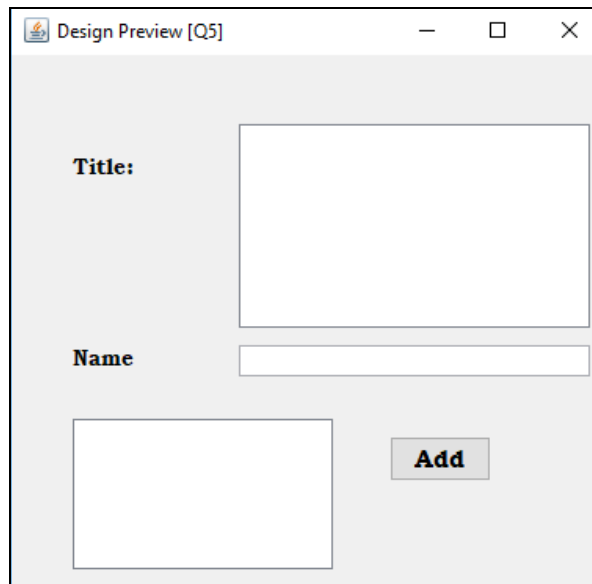
Get Total

Get Grade

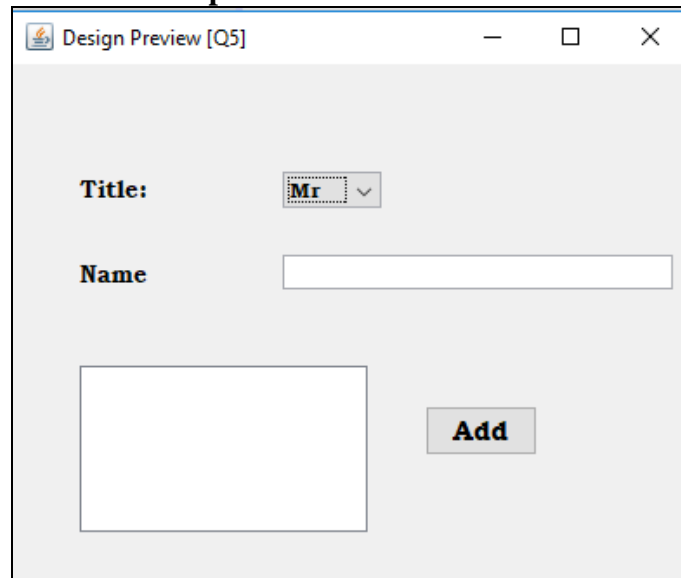
Exit

Write the code for

- a) To calculate the total marks obtained and display in text field4 on click of command button “Get Total”
 - b) To calculate the grade obtained and display in text field5 on click of command button “Get Grade”
 - c) To terminate the application by click on “Exit” button
- 5. Develop a Java Applet Program which has the following GUI, adds the selected item from the List labeled Title (should contain Mr, Miss, Ms) to the Name entered in text field and prints in the text area provided.**



6. Develop a Java Applet Program which has the following GUI, adds the selected item from the Choice (contains Mr, Miss, Ms) to the Name entered in text field and prints in the text area provided.



7. Develop a Java Program which has the following GUI, adds the selected information from all the controls and prints in the text area provided.

Design Preview [Q7]

Name

Gender ☐ Male ☐ Female

Hobbies ☐ Reading ☐ Sports
☐ Painting ☐ Technology

8. Develop a Java Applet Program which has the following GUI, calculates and displays Interest Amount taking values of Principal amount, ROI, Time.

Design Preview [Q8]

Prinipal:

Rate-Of-Interest

Time(Years):

Interest:

9. Develop a Java Program which has the following GUI, checks whether a number entered in a text field is positive or not and display on a Label provided

Design Preview [Q9]

Enter a number

Check

10. Develop a Java program with the below GUI, verify the criteria for admission in army.

If the person is male with greater than or equal to 175cm of height then he is eligible for admission in army. For ladies height criteria is 160cm. or more.

Enter your height in cm:

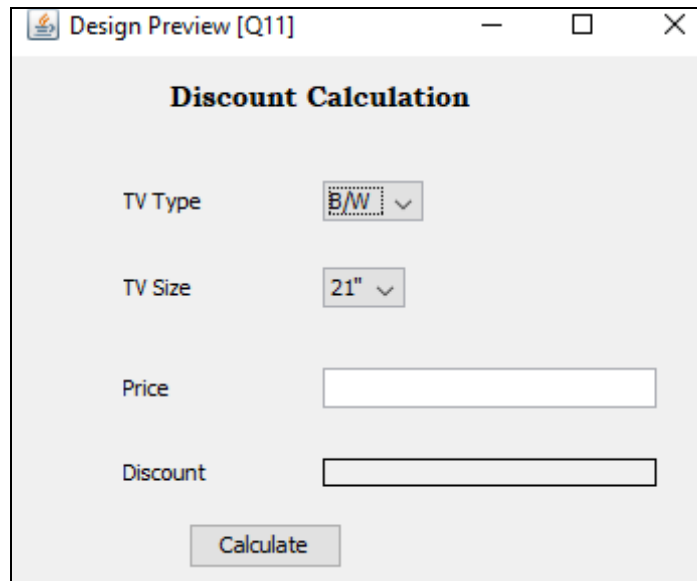
Gender: Male

Check

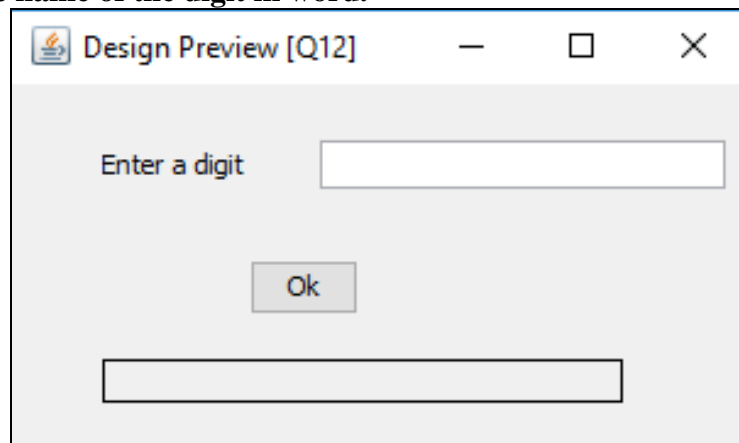
11. Develop a Java program with the below GUI, calculates discount amount.

This Program accepts size, type and price of a TV set and calculates discount based on the policy given in table below.

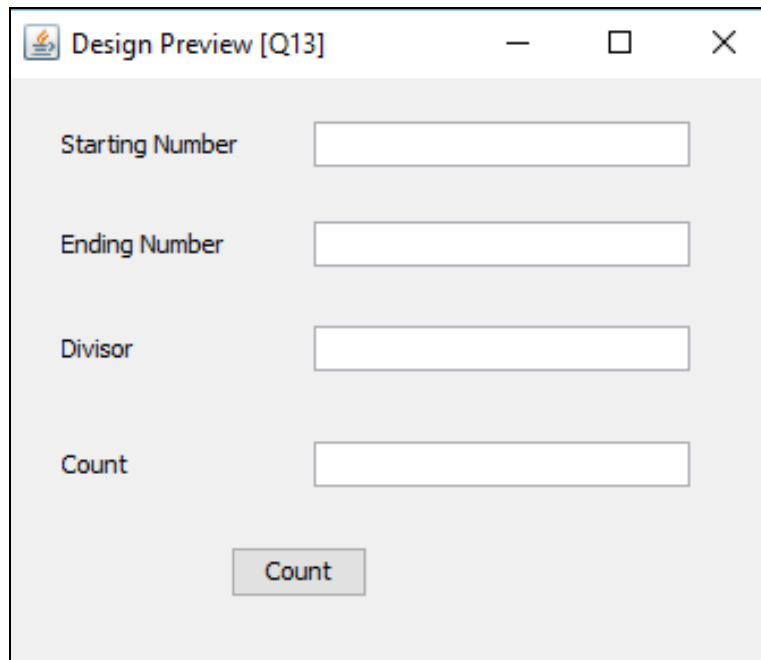
Type	Size	Discount
B/W	21''	10%
B/W	27''	15%
Color	21''	20%
Color	27''	25%



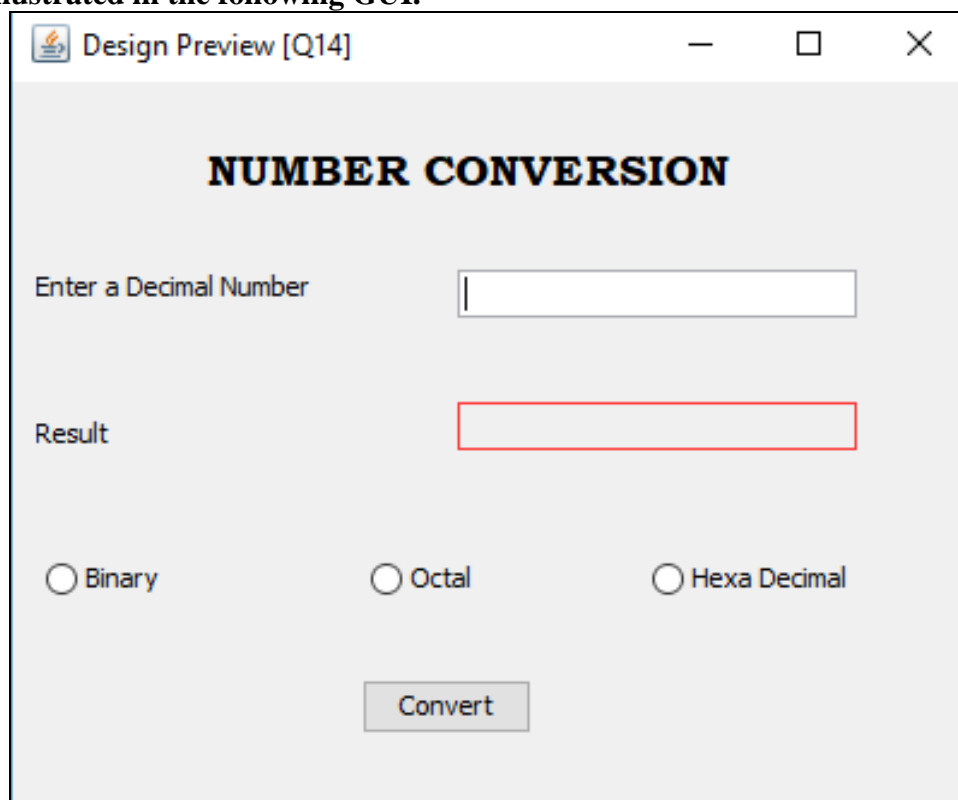
12. Write a Java Program to create the following GUI, which accepts a number and prints the name of the digit in word.



13. Develop a java program to create the following GUI, which accepts a start number and end number and a digit and counts the numbers of between start and end number divisible by entered digit using for loop.



14. Develop a java application which converts the entered a number in a text field into binary or octal or hexadecimal number by using radio button as illustrated in the following GUI.



Review Questions

1. What are differences between Swing and AWT?
2. Why Swing components are called lightweight component?
3. Does Swing is thread safe?

4. Which package has light weight component?
5. Why swing is not thread safe?
6. What do heavy weight components mean?
7. How do you classify Swing Components?
8. What is MVC?
9. What is the use of JFC in Java Swing?
10. Why is Model-View-Controller Architecture used in Swing?
11. What are the different types of layout managers used in Swing?
12. What is the design pattern that Java uses for all Swing components ?

Previous Mid and External Theory Exams Question Papers



II B.Tech. I Semester I – Internal Examinations, AUGUST– 2018

OBJECT ORIENTED PROGRAMMING THROUGH JAVA

Date: 08– 08 – 2018 (AN)

[CSE]

Max. Marks: 20 ::

Time: 1 ½ Hours

PART – A

Note: Answer ALL Questions.

10 x ½

1. A method can be implemented in several classes along the inheritance chain. The JVM decides which method is invoked at runtime. Such mechanism known as _____.
a)abstraction b)inheritance c) both a&b d) none
2. _____used inside a constructor to invoke another constructor of the same class.
a)super b)lower c)this d)that
3. Object oriented features are
a)encapsulation b)inheritance c)object d)all
4. _____ variables store values for the variables in a common memory location.
a) Dynamic b) local c)static d) final
5. A String object is mutable? (yes / No)
6. Role of garbage collector is _____.
7. Errors that are detected by the compiler are called ____.
8. Methods for Scanner object _____.
9. In java, polymorphism means _____.
10. Instance of a class known as _____.

PART – B

Note: Answer Any THREE Questions.

3 x 5

1. a) Develop a program that calculates the energy needed to heat water from an initial temperature to a final temperature. Your program should prompt the user to enter the amount of water in kilograms and the initial and final temperatures of the water. The formula to compute the energy is
$$Q = M * (\text{finalTemperature} - \text{initialTemperature}) * 4184.$$
where M is the weight of water in kilograms, temperatures are in degrees Celsius, and energy Q is measured in joules.
b) What is interface? How to create it and access it?
2. a) What do you mean by abstract method and concrete method? Can we make an instance of an abstract class? Justify your answer with an example?
b) Explain Method Overriding with suitable example.
3. a) What is package? How to create package?
b) Develop a program that reads in ten numbers and displays distinct numbers (i.e., if a number appears multiple times, it is displayed only once). (Hint: Read a number and store it to an array if it is new. If the number is already in the array, ignore it.) After the input, the array contains the distinct numbers. Here is the sample run of the program:
Output : Enter ten numbers: 1 2 3 2 1 6 3 4 5 2
The distinct numbers are: 1 2 3 6 4 5

4. a) Design a class named Triangle . The class contains:
 - Three double data fields named side1, side2, side3 with default values 1 to denote three sides of the triangle.
 - A no-arg constructor that creates a default Triangle.
 - A constructor that creates a Triangle with the specified side1, side2, & side3.
 - A method named getArea() that returns the area of this triangle.
 - A method named getPerimeter() that returns the perimeter of this triangle
 Develop a test program that prompts the user to enter three sides of the triangle. The program should display the area, perimeter.
 - b) Exemplify method overloading with suitable example.
5. a) What are the different forms of inheritance? Explain.
 - b) Develop a code for below scenario.



III B.Tech II Semester I – Internal Examinations, JANUARY– 2018
 OBJECT ORIENTED PROGRAMMING CONCEPTS THROUGH JAVA
 [ME & ECE]

Date: 18 – 01 – 2018
 ½ Hours

Max. Marks: 20 :: Time: 1

PART – A

Note: Answer ALL Questions.

10 x ½

1. Which one of the following is NOT a correct variable name?
 - a) 2bad b) zero c) theLastValueButOne d) year2000
2. What does the following program output?


```
class question4
{
public static void main ( String[] args )
{
int unitCost = 8;
int items = 5;
System.out.println("total cost: " + (unitCost * items) );
}
}
```

 - a) total cost: + 40 b) total cost: 8*5 c) total cost: 40 d) "total cost: " 40
3. What is an object?
 - a) An object is a chunk of memory that can contain data along with methods to process that data.
 - b) An object is a tiny box of memory that contains only data.
 - c) "Object" is another word for "program". d) An object is a description of a potential class.
4. Why is the main() method special in a Java program?
 - a) It is where the Java interpreter starts the whole program running.
 - b) Only the main() method may create objects. c) Every class must have a main() method.

- d) The main() method must be the only static method in a program.
5. What is the output of the following program fragment?
- ```
int j;
for (j = 0; j < 5; j++)
{ System.out.print(j + " "); }
System.out.println();
```
- a) 1 2 3 4 5                      b) 0 1 2 3 4                      c) 0 1 2 3 4 5                      d) j j j j j
6. When the access modifier is omitted from the definition of a member of a class (instance variable or method) the member has ..... ?
- a) default access              b) public access              c) private access              d) universal access
7. Does a subclass inherit both member variables and methods?
- a) No—only member variables are inherited.              b) No—only methods are inherited.
- c) Yes—both are inherited.              d) Yes—but only one or the other are inherited.
8. Which one of the following is NOT a correct arithmetic expression?
- a) alpha                              b) (alpha / momentum) - 12.4
- c) alpha ( / momentum - 12.4 )              d) ((alpha / momentum) - 12.4)
9. A class Car and its subclass Yugo both have a method run() which was written by the programmer as part of the class definition. If junker refers to an object of type Yugo, what will the following code do?
- ```
junker.show();
```
- a) The show() method defined in Yugo will be called.
- b) The show() method defined in Car will be called.
- c) The compiler will complain that run() has been defined twice.
- d) Overloading will be used to pick which run() is called.
10. Complete the program so that it writes "Good-by" on the computer monitor.
- ```
class Bye {
 public static void main (String[] args) {
 System.out.println(_____); }}
```

#### PART – B

Note: Answer Any THREE Questions.

3 x 5

- Develop a Java Program to find Area of Geometric figures using Method Overloading.
- Develop a program to read integer variable (rupees) and convert into dollars.
- Discuss static members and non members of a class.
- Develop a Java program to print the number of days in a month based on the month number and year using switch statement.
- Summarize Java Buzzwords.



II B.Tech. I Semester II – Internal Examinations, Oct– 2017

**OOPS THROUGH JAVA**

[CSE]

Date: 13 – 10 – 2017 (AN)

Max. Marks: 20 ::

Time: 1 ½ Hours

**PART – A**

Note: Answer ALL Questions.

10 x ½

1. \_\_\_\_\_ block must immediately followed by a try block.  
a) finally                      b) catch                      c) static                      d) a or b
2. The thread is in \_\_\_\_\_ state after invocation of start() method.  
a) running                      b) sleeping                      c) blocked                      d) stopped
3. \_\_\_\_\_ is the default layout manager  
a) Grid                      b) Card                      c) Flow                      d) Border
4. Which life cycle method of an applet provided by java.awt.Component class?  
a) public void paint(Graphics g) b) public void destroy() c) public void stop( ) d) public void init()
5. Which of these methods are used to register a mouse event?  
a) MouseListener()              b) addListener()              c) addMouseListener() d) eventMouseListener()
6. Java swing components are platform \_\_\_\_\_ .
7. MVC Stands for \_\_\_\_\_, \_\_\_\_\_ & \_\_\_\_\_.
8. An \_\_\_\_\_ is an object that describes a state change in a source.
9. \_\_\_\_\_ occurs when two threads have a circular dependency on a pair of synchronized objects.
10. To throw an Exception manually, \_\_\_\_\_ keyword is used.

**PART – B**

Note: Answer Any THREE Questions.

3 x 5

1. Distinguish various Layout Managers with Example Programs
- 2 a) Differentiate AWT and Swings  
b) Describe Adapter, Inner & Anonymous Classes.
- 3 Explain Exceptional Handling Mechanism with all the keywords used in that.
- 4 Discriminate the methods to create a user defined thread with examples
- 5 a) Illustrate applet life cycle with a program  
b) Explain Mouse Events with a program



II B.Tech. I Semester II – Internal Examinations, OCT – 2016

**OOPS THROUGH JAVA**

[CSE]

---

**PART – A****Note:** Answer ALL Questions.

10 x ½

1. What is the name of the method used to start a thread execution?  
a) init();                      b) start();                      c) run();                      d) resume();
2. Which method must be defined by a class implementing the *java.lang.Runnable* interface?  
a) void run()                      b) public void run()                      c) public void start()                      d) void run(int priority)
3. Which of these class is not a member class of java.io package?  
a) String                      b) StringReader                      c) Writer                      d) File
4. Which of these methods can be used to output a sting in an applet?  
a) display()                      b) print()                      c) drawString()                      d) transient()
5. Which methronds are utilized to control the access to an object in multi threaded programming  
a) Asynchrnized methods                      b) Synchronized methods  
c) Serialized methods                      d) None of above
6. AWT stands for \_\_\_\_\_.
7. J in JAPPLET stands for \_\_\_\_\_.
8. stream is a \_\_\_\_\_.
9. this is a \_\_\_\_\_.
10. Applet is a \_\_\_\_\_.

**PART – B****Note:** Answer Any THREE Questions.

3 x 5

1. Write a short note on adapter classes with an example.
2. In developing a GUI we use MVC architecture –Justify.
3. Compare Applets with Swings.
4. DataInputStream and Scanner classes are used for accepting data from input devices which is effective-justify.
5. Differentiate Sting and StringBuffer with an example.



**PART – A****Note:** Answer ALL Questions.

10 x ½

1. \_\_\_\_\_ is the mechanism that binds together code & data  
a) Encapsulation      b) Polymorphism      c) Abstraction      d) Inheritance
2. Java's Character Data type follows \_\_\_\_\_ code  
a) Unicode      b) ASCII      c) EBCDIC      d) None
3. \_\_\_\_\_ keyword must be preceded by finalize () method  
a) Protected      b) public      c) private      d) None of the above
4. Partially implemented/defined class can be called as \_\_\_\_\_  
a) Abstract class      b) interface      c) concrete class      d) None of the above
5. To disallow a method from overridden \_\_\_\_\_ modifier is used  
a) final      b) static      c) abstract      d) None of the above
6. JVM is an interpreter for \_\_\_\_\_.
7. To inherit a class \_\_\_\_\_ keyword is used.
8. Method signature does not include \_\_\_\_\_.
9. Constructor can be called \_\_\_\_\_ number of times in its lifetime.
10. A final variable must be \_\_\_\_\_ when it is declared.

**PART – B****Note:** Answer Any THREE Questions.

3 x 5

1. Summarize java buzzwords.
2. Discriminate various types of inheritance with example programs.
3. Define Constructor and give example program to illustrate Constructor Overloading.
4. Define & illustrate dynamic method dispatch with an example program.
5. a) Differentiate method overloading, method overriding.  
b) Outline the procedure of creating & accessing a package.



Code No.: CS104

II B.TECH. I SEM. (RA15) Regular Examinations, OCT/NOV – 2018

OBJECT ORIENTED PROGRAMMING CONCEPTS

(CSE)

Time: 3 Hours

Max. Marks:

70

**PART – A**

Answer ALL questions

All questions carry equal marks

10 x 2



1. Java is platform-independent and portable – Justify.
2. List the primitive and non-primitive data types used in Java.
3. Write short notes on Polymorphism.
4. What is the significance of the CLASSPATH environment variable in creating/using a package?
5. List out the different types of exception.
6. Give syntax for creating a thread using a class and an interface.
7. List out the differences between applets and applications.
8. List out methods of Mouse Listener.
9. What are the applications of an applet?
10. What are the limitations of AWT?

PART – B

Answer any FIVE questions  
All questions carry equal marks

5 x 10

1. a) Explain the features of object-oriented programming.  
b) What is Overloading? Explain Method overloading with an example.
2. a) What is a class and Object? Explain how constructors are defined in java for a class with example.  
b) Describe this keyword and garbage collection in java.
3. What are the differences between classes and interfaces? Explain with an example.
4. Discuss about different Byte Streams available.
5. Describe the concept of exception handling mechanism with an example.
6. Explain the life cycle of a Thread. Write a program to create a Thread.
7. Explain about keyboard handling events with program.
8. Explain the following Swing components in detail.  
i) JLabels            ii) JComboBoxes            iii) TabbedPane



Code No.: CS104

IV B.Tech. I Sem. (RA15) Regular Examinations, Oct./Nov., – 2018

OBJECT ORIENTED PROGRAMMING CONCEPTS THROUGH JAVA

(EEE)

Time: 3 Hours

Max. Marks: 70

PART – A

Answer ALL questions  
All questions carry equal marks

10 x 2

1. Write about garbage collection?
2. Differentiate between abstraction and information hiding.
3. What is the difference between an interface and an abstract class?
4. How to create and use a package in Java program?
5. Define multi threading.
6. Why to use finally block in java exception handling.
7. Write the differences between application and applet program.

8. What is source and listener in java event handling?
9. What are the different types of controls available in AWT?
10. List the features of swings.

PART – B

Answer any FIVE questions  
All questions carry equal marks

5 x 10

1. Explain the important features of Java.
2. Explain different types of inheritances with examples.
3. Give a detail note on interfaces and packages in java with examples.
4. What is thread synchronization? Discuss with an example.
5. Differentiate method overloading with method overriding with examples
6. Write a Java program to create a combo box which includes list of subjects. Copy the subjects in text field on click using applet.
7. Write a java program using listeners for handling keyboard events.
8. Construct an application to explain the use of JTabbedPane.



Code No.: CS104

III B.Tech. II Sem. (RA15) Regular Examinations, April - 2018  
**OBJECTED ORIENTED PROGRAMMING CONCEPTS THROUGH JAVA**  
(ME & ECE)

Time: 3 Hours

Max. Marks: 70

PART – A

Answer ALL questions  
All questions carry equal marks

10 x 2

1. Explain about commands javac, java with examples.
2. What is Type Conversion? Give an example for it.
3. In how many ways a user-defined package can be compiled? Explain.
4. Can interface be instantiated? Justify.
5. List out various keywords for Exception Handling.
6. Explain about Thread priorities.
7. What are the different methods present in the applet class?
8. What is the need of adapter classes.
9. List out the classes in AWT package.
10. What is a container? Explain briefly.

PART – B

Answer any FIVE questions  
All questions carry equal marks

5 x 10

1. a) Explain about Arrays in java with example.  
b) Write a java program to perform matrix multiplication.
2. a) Explain the concept of extending an interface.

- b) Explain how multiple inheritance can be achieved in Java with example program.
3. Explain about creating your own exception in Java with example.
4. a) Explain about the ways to create an applet with example.  
b) How to pass parameters to an applet? Explain with an example.
5. What is constructor? Explain about types of constructors with example.
6. Explain how a package is created and accessed with various access specifiers.
7. a) Write a Java program to check whether the given String is a palindrome or not.  
b) Write a Java program to sort an array of strings in ascending order.
8. Write a Java program to implement the following components.  
i) Label                      ii) Button                      iii) TextField                      iv) Checkbox



Code No.: CS206

II B.TECH. I SEM. (RA11) Supplementary Examinations, OCT/NOV – 2017

### OBJECT ORIENTED PROGRAMMING CONCEPTS

(CSE)

Time: 3 Hours

Max. Marks:

70

#### PART – A

Answer ALL questions

All questions carry equal marks

10 x 2

1. Explain briefly the garbage collection in java.
2. List different access specifiers in java. Explain briefly.
3. What is the significance of super keyword?
4. Define abstract class.
5. How to set class path in java.
6. Differentiate between class and interface.
7. Define thread.
8. Write the advantages of user defined exceptions.
9. What is Adapter class?
10. What is AWT?

#### PART – B

Answer any FIVE questions

All questions carry equal marks

5 x 10

1. Explain the architecture of JVM with neat diagram.
2. Explain dynamic method dispatch with an example program.
3. How to create and import a package. Explain different access controls in packages.
4. Differentiate between String and StringBuffer class.
5. Explain in detail about applet life cycle.
6. Write a program to demonstrate mouse events handling.

7. Differentiate between a class and an interface with an example.
8. a) Differentiate between applet and application. [3]  
b) Write a program to demonstrate how to pass the parameters to an applet. [7]



**II B.Tech. I Semester II – Internal Examinations, OCTOBER – 2018**  
**OBJECT ORIENTED PROGRAMMING THROUGH JAVA**

Date: 05 – 10 – 2018 (AN)

[CSE]

Max. Marks: 20 ::

Time: 1 ½ Hours

**PART – A**

Note: Answer ALL Questions.

10 x ½

1. \_\_\_\_\_ is an object that describes a state change in a source  
a) Applet                      b) Event                      c) Exception                      d) none
2. \_\_\_\_\_ classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface  
a) String class                      b) adapter class                      c) both a & b                      d) none
3. Built-in Exception is / are:  
a) NumberFormatException                      b) ArrayIndexOutOfBoundsException  
c) IllegalArgumentException                      d) All
4. \_\_\_\_\_ Block will execute whether or not an exception is thrown.  
a) Throws                      b) finally                      c) try                      d) none
5. \_\_\_\_\_ is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.
6. \_\_\_\_\_ Components are lightweight.
7. MVC means \_\_\_\_\_.
8. Constructors of List control \_\_\_\_\_.
9. Daemon thread is \_\_\_\_\_.
10. \_\_\_\_\_ Method used to get thread name.

**PART – B**

Note: Answer Any THREE Questions.

3 x 5

1. a) What is an event? Explain the event delegation model.  
b) Write a java program to handle keyboard events.
2. a) What is exception? Explain the exception handling mechanism.  
b) Explain File Input and FileOutputStream with suitable example.
3. a) Explain border and grid layout managers briefly.  
b) Explain about applet life cycle with a suitable example.
4. a) Define a thread? Give syntax for creating a thread using a class and an interface.  
b) Explain multithreaded environment with suitable example.
5. Design a swing application that creates registration form of a student.



IV B.Tech. I Semester II – Internal Examinations, November – 2018

OBJECT ORIENTED PROGRAMMING CONCEPTS THROUGH JAVA

Date: 22 – 11 – 2018 (FN)

[EEE]

Max. Marks: 20 ::

Time: 1 ½ Hours

### PART – A

Note: Answer ALL Questions.

10 x ½

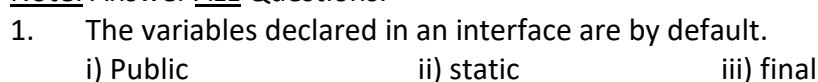
1. When Exceptions in Java does arise in code sequence?  
a) Run Time      b) Compilation Time      c) Can Occur Any Time      d) None of the mentioned
2. Which of these keywords is not a part of exception handling?  
a) try      b) finally      c) throw      d) catch
3. What is multi threaded programming?  
a) It is a process in which two or more parts of same process run simultaneously  
b) It is a process in which a single process can access information from many sources  
c) It is a process in which two different processes run simultaneously  
d) It is a process in which many different processes are able to access same information
4. What does AWT stands for?  
a) All Window Tools    b) All Writing Tools    c) Abstract Window Toolkit    d) Abstract Writing Toolkit
5. Which of these functions is called to display the output of an applet?  
a) display()      b) paint()      c) displayApplet()      d) PrintApplet()
6. A \_\_\_\_\_ is a collection of classes and interfaces.
7. \_\_\_\_\_ keyword must be used to handle the exception thrown by try block in some rational manner.



- PART – B

 $3 \times 5$ 

- 



2. \_\_\_\_\_ keyword is used to implement an interface.  
 a) 1,2                                  b) 1,3                                  c) 2,3                                  d) 1,2,3
3. \_\_\_\_\_ package is by default available for all the java programs.  
 a) Extends                                  b) this                                  c) implements                                  d) super
4. \_\_\_\_\_ package is by default available for all the java programs.  
 a) Java.io                                  b) java.lang                                  c) java.util                                  d) None of the above
5. Java doesn't support \_\_\_\_\_ inheritance.  
 a) Multiple                                  b) Single                                  c) Multi level                                  d) Hierarchical
6. Access specifier provides only class level accessibility.  
 a) Public                                  b) protected                                  c) private                                  d) none of the above
7. Java's Char Datatype follows \_\_\_\_\_ Code.
8. To retrieve a character from a given String \_\_\_\_\_ method of String class is used.
9. \_\_\_\_\_ keyword must be preceded by finalize method.
10. One interface can inherit another interface by using \_\_\_\_\_ keyword.
11. JVM Stands for \_\_\_\_\_.

### PART – B

Note: Answer Any THREE Questions.

3 x 5

1. Summarize Java Buzzwords.
2. Define Constructor with an example? Construct a java program to implement Constructor Overloading.
3. Discriminate various types of inheritance with the help of example programs.
4. Differentiate abstract classes and interfaces and illustrate them with the help of example programs.
5. Illustrate the procedure of Package Creation and Importing with the help of programs.

