II Year –II Semester

Subject Code	Subject Name	L	T	P	C
R19CSE-PC2201	Java Programming	3	1	0	4

Course Objectives:

- To understand the structure and environment of Java.
- To implement the relationship between objects.
- To apply data hiding strategy in objects.
- To implement text processing and error handling.
- To organize data using different data structures.
- To create multi threaded graphical user interface applications.

Course Outcomes:

1. Understand the environment of JRE and Control Statements.
2. Implement real world objects using class Hierarchy.
3. Implement generic data structures for iterating distinct objects.
4. Implement error handling through exceptions and file handling through streams.
5. Design thread-safe GUI applications for data communication between objects.

Unit I: Java Environment and Program Structure

History of Java, Features, Applications, Java Installation - JDK and JRE, JVM Architecture, OOPS Principles, Class and Object, Naming Convention, Data Types, Type Casting, Type Conversion, Wrapper classes, Operators, instance of operator, Command Line Arguments, Decision making, Arrays, and Looping statements.

Course Outcomes: Student will be able to

1. Understand architecture of Java Virtual Machine.(L2)
2. Understand the structure of java program and its environment. (L2)

Unit II: Class Hierarchy & Data Hiding

Property, Method, Constructor, Inheritance (IS-A) , Aggregation and Composition (HAS-A), this and super, static and initialize blocks, Method overloading and overriding, static and final keywords, Types of Inheritance, Compile time and Runtime Polymorphism, Access Specifiers and scope, packages and access modifiers, Abstract class, Interface, Interface Inheritance, Achieving Multiple Inheritance, Class casting, Object Cloning, Inner Classes.

Course Outcomes: Student will be able to

1. Understand the class hierarchy and their scope. (L2)

2. Implement relationship between objects. (L3)

3. Understand data hiding and nested classes. (L2)

4. Implement data type casting and cloning of objects. (L3)

Unit III: Strings and Collections

String: Methods, StringBuffer and StringBuilder, StringTokenizer, Collections: Exploring java.util.*, Scanner, Iterable, Collection Hierarchy, Set, List, Queue and Map, Comparable and Comparator, Iterators: foreach, Enumeration, Iterator and ListIterator.

Course Outcomes: Student will be able to

1. Understand the usage of String and its properties and methods.(L2)

2. Understand data structures and Iterators. (L2)

3. Create the data structures and implement different utility classes. (L3)

Unit IV: IO and Error Handling

IO Streams: Exploring java.io.*, Character and Byte Streams, Reading and Writing,

Serialization and De-serialization, Error Handling: Error vs Exception, Exception hierarchy, Types of Exception, Exception handlers, User defined exception, Exception propagation.

Course Outcomes: Student will be able to

1. Understand character and byte streams. (L2)

2. Understand the hierarchy of errors and exceptions. (L2)

3. Implement data streams and exception handlers. (L3)

Unit V: Threads and GUI

Multi-Threading: Process vs Thread, Thread Life Cycle, Thread class and Runnable

Interface, Thread synchronization and communication. GUI: Component, Container, Applet, Applet Life Cycle, Event delegation model, Layouts, Menu, MenuBar, MenuItem.

Course Outcomes: Student will be able to

1. Understand the Thread Life Cycle and its scheduling.(L2)

2. Implement the synchronization of threads. (L2)

3. Create graphical components using Abstract window toolkit. (L3)

TEXT BOOKS:

1. The complete Reference Java, 8th edition, Herbert Schildt, TMH.

2. Programming in JAVA, Sachin Malhotra, SaurabhChoudary, Oxford.

3. Introduction to java programming, 7th edition by Y Daniel Liang, Pearson.

4. Java: How to Program, 9th Edition (Deitel) 9th Edition.

5. Core Java: An Integrated Approach, Java 8 by R. Nageswara Rao.

REFERENCE BOOKS:

1. Swing: Introduction, JFrame, JApplet, JPanel, Componets in Swings, Layout

Managers

2. Swings, JList and JScrollPane, Split Pane, JTabbedPane, JTree, JTable, Dialog

Box.

UNIT-1

FEATURES of java:

1. Simple
2. Platform independent
3. Architectural neutral
4. Portable
5. Multi threading
6. Distributed
7. Networked
8. Robust
9. Dynamic
10. Secured
11. High performance
12. Interpreted
13. Object Oriented Programming Language

1. Simple:
JAVA is free from pointers hence we can achieve less development time and lessexecution time JAVA contains user friendly syntax's for developing JAVA 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).

3Architectural Neutral:A language or technology is said to be *architectural neutral* which can run on any available processors in the real world. The languages like C,Cpp are treated as *architectural dependent*. The language like *JAVA* can run on any of the processor irrespective of their *architecture* and *vendor*.

4.Portable: A portable language is one which can run on all operating systems and on all processors irrespective their architectures
i.e Portable=Architectural neutral + Platform Independent

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

6. Distributed: Using this language we can create distributed applications.

7. Robust: Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages.

8. Dynamic: It supports Dynamic memory allocation due to this memory wastage is reduce and improve performance of the application. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation,

9. Secure: java API contains security related concepts due to these security related concepts java is one of the secure language.

10. High performance: It have high performance because of following reasons;

- ✓ **Garbage collector**, collect the unused memory space and improve the performance of the 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 executing the program.

11. Object Oriented: It supports OOP's concepts because of this it is most secure language

12. Networked: The basic aim of networking is to share the data on multiple machine either on same network or different network

13. **Interpreted:** Java is one of the interpreted programming language.

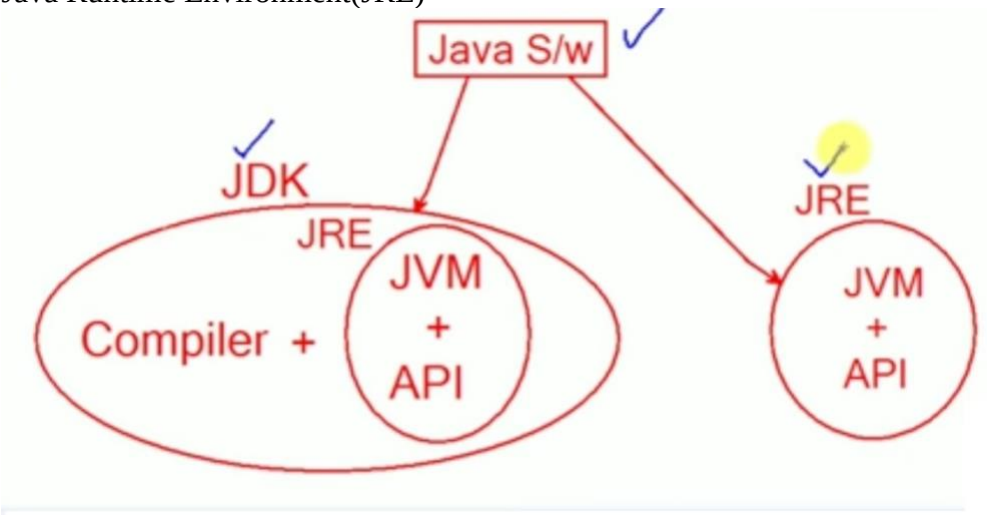
Applications of java

- Desktop Applications such as acrobat reader, media player etc.
- Web Applications such as online library management, www.amazon.com, etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games, etc.

<u>Two types of java softwares</u>

i. Java Development Kit(JDK)

ii. Java Runtime Environment(JRE)



- ✓ On the developer machine JDK must install
- ✓ On client machine JRE must install

JDK used for developing, compiling, executing and modify existing java Applications. JDK contains development tools like compiler and java run time environment (JRE).

Java runtime environment (JRE) stands for java run time environment. Using JRE we only run java applications. JRE contains JVM and API(set of library files)

JVM stands for Java Virtual machine JVM is responsible for executing java byte code. JVM contains interpreter and JIT. Interpreter is used for running java byte code. JIT helps the interpreter to run java program fast.

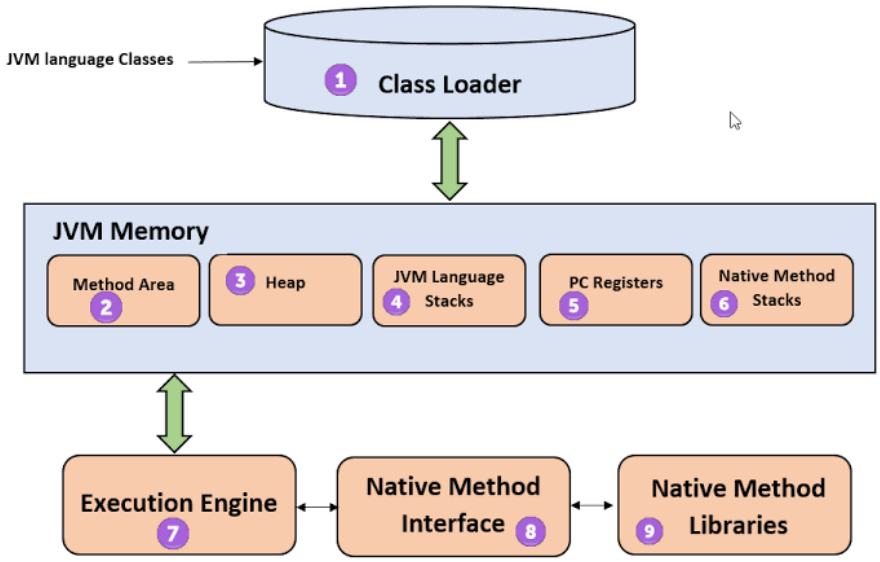
JIT is Just in time compiler increases the performance of the java program by executing fast.

JAVA VIRTUAL MACHINE(JVM)

Java virtual machine (JVM) is the heart of the entire java program execution process.

We compile the program by using java compiler and we run the program using JVM. JVM is the software program written in c++ used to provide runtime environment of java program.

It is responsible for taking .class file and converting each byte code instruction into machine language instruction that can be executed by micro processor.



1) ClassLoader :

- ✓ The class loader is a subsystem used for loading class files.
- ✓ It is responsible for three activities
- ✓ Loading: The Class loader loads the "*.class"* file
- ✓ Linking: it verifies the .class file and allocates memory for class variables
- ✓ initialization : in this phase all the static variables are assigned their values.

2) Method Area :

Method area is the memory block which stores class name, methods and variables information in the java program

3) Heap :

In this memory area objects are created.

4) JVM language Stacks or java stacks :

Method code is stored on method area but while running a method, it needs some more memory to store data and results. This memory is allotted on java stacks. Java stacks is the memory area where java methods are executed

5) PC Registers :

PC register store the address of the instruction which is currently being executed

6) Native Method Stacks :

Java methods are executed in java stacks whereas native methods (such as C/C++ functions) are executed native method stacks.

7) Execution Engine :

Execution engine convert the byte code into machine understandable language

It is a type of software consists of interpreter, jit compiler and garbage collector

Interpreter executes the byte code line by line.

Just in time (JIT) compiler increases the speed of execution of interpreter

Garbage collector is software used to remove unused objects

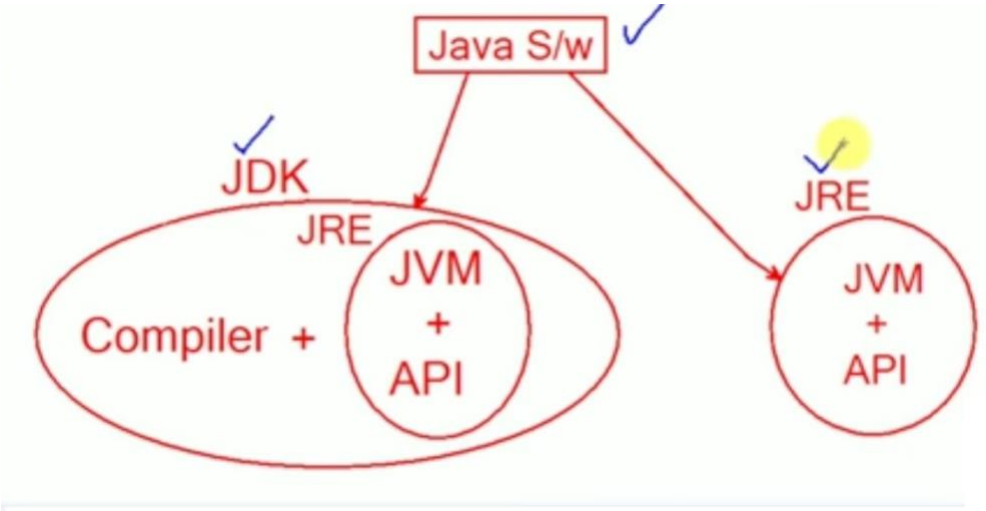
8) Native Method interface :

Native Method Interface allows Java code which is running in a JVM by call by native libraries (C/C++ libraries)

9) Native Method Libraries

Native Libraries is a collection of the Libraries(C/C++) which are needed by the Execution Engine to run java program.

What is JRE. What happen when JRE is not available? Write about the relation in between JRE and JVM.



Java runtime environment (JRE) stands for java run time environment. Using JRE we only run java applications. JRE contains JVM and API(set of library files)

JDK=JRE +Compiler Tools

JRE=JVM+API

If JRE is not available then we cannot run the program

- ✓ On the developer machine JDK must install
- ✓ On client machine JRE must install

JDK used for developing, compiling, executing and modify existing java Applications. JDK contains development tools like compiler and java run time environment (JRE).

JVM stands for Java Virtual machine JVM is responsible for executing java byte code. JVM contains interpreter and JIT. Interpreter is used for running java byte code. JIT helps the interpreter to run java program fast.

JIT is Just in time compiler increases the performance of the java program by executing fast.

OOPS(Object Oriented Programming System) Principles

Object-Oriented Programming is a methodology to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

1. Object
2. Class
3. Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism

1.Object

Object is an physical entity which contains properties and behavior. For example, a chair, pen, table, keyboard, bike, etc. Object is a physical entity. An object contains an address and takes up some space in memory.

We can create object by new operator

Classname objname=new Classname();

Example Car benz=new Car()

benjz is a object and Car is class

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

2.Class

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Using Class a model or blue print we can create 'n' number of objects

Class contains two parts namely properties and behaviour.

Syntax for defining a CLASS:

```
Class <clsname>

{

Variable declaration;//properties
Methods definition;//behaviour
};
```

3.Data Abstraction:

"*Data abstraction is a mechanism of retrieving the essential details without dealing with background details*".

Advantages of Abstraction :

- ✓ It reduces the complexity of viewing the things.
- ✓ only important details are provided to the user.

4.Encapsulation :

The process of binding data members and methods into a single unit are known as encapsulation.

- *Data encapsulation* is basically used for achieving data/information hiding i.e., security.

5.Inheritance:

- *Inheritance* is the process of taking the features (*data members + methods*) from one *class* to another *class*.
- The *class* which is giving the features is known as base/parent class.
- The *class* which is taking the features is known as derived/child/sub class.

6.Polymorphism:

Polymorphism is a process of representing "*one form in many forms*".

Poly *means many.*

morphism *means forms*

Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Class and object

Class	**object**
Class is a group of elements having common properties and behavior	Among the group of elements if any individual element having physical properties and physical behavior then that individual element is called is an object.
Class is virtual	Object is real
Class is virtual encapsulation of properties and behavior.	Object is physical encapsulation of properties and behavior.
Class is a blueprint or model for creating objects	Object is an instance of the class
Syntax: class <classname> { }	Syntax: Classname objname=new Classname()

Naming Conventions

Naming conventions are Rules to be followed when giving names to the identifier

Class Level Naming Conventions

1.Class name must not be a keyword

2.Class name should begin with capital letter (Single word)

First letter of every word of class name should begin with capital letter(Multiple word)

Ex

Student----(Single word)

BranchName----(multiple word)

Method Level Naming Conventions

1.Method name must not be a keyword

2.First letter of method name should begin with lowercase letter(Single word)

3.First letter & first word should be lowercase and First letter of every other word must be capital(multiple word)

Example

read() (Single word)

readName(Multiple word)

nextLine(Multiple word)

Variable level Naming Conventions

1.Variable name must not be a keyword

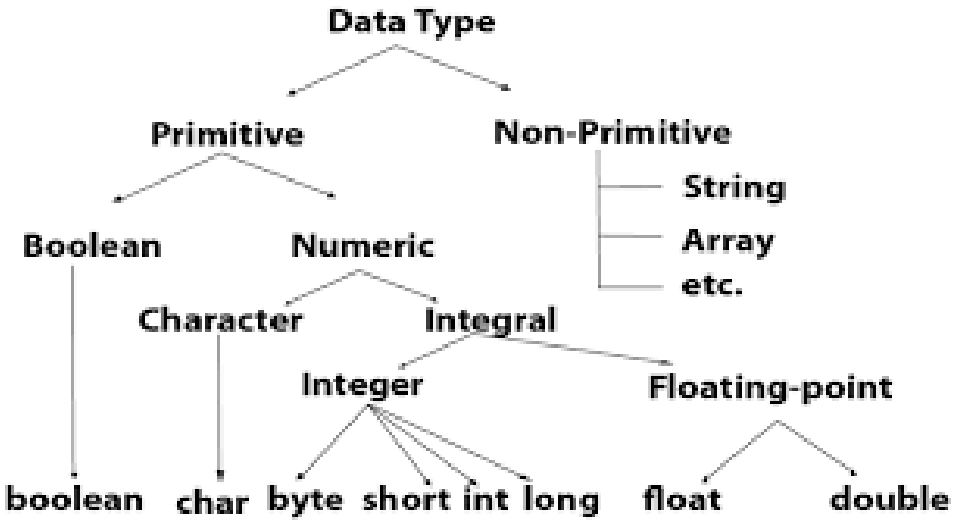
2.Allowed variables in java

1. a to z
2. A to Z
3. 0 to 9
4. _ (underscore)
5. $ (dollar)

3.variables can't start with digit

4.variables are case sensitive

Java Data Types



1.Primitive/Fundamental Data Type:Primary Data Type stores single value.
Java supports eight primitive data types: **byte**, **short**, **int**, **long**, **float**, **double**, **char** and **boolean**.
These eight data types are further classified into four groups:

1. Integer data type:These data types stores integer value

   Ex: byte,short,int,long

   Floating data type:These data types stores decimal values

2. Ex :float,double
3. Character data type:this data type stores single character value
4. Boolean data type:It stores Boolean values like true or false

Primitive data types

Data Type	Size	Description
Byte	1 byte	Stores whole numbers from -128 to 127
Short	2 bytes	Stores whole numbers from -32,768 to 32,767
Int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
Long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
Double	8 bytes	Stores fractional numbers. Sufficient for storing 15

		decimal digits
Boolean	1 bit	Stores true or false values
Char	2 bytes	Stores a single character/letter or ASCII values

2.Non-Primitive/Derived Data Types: It stores multiple value of same type Example :Arrays,String

3.User friendly data types:It stores multiple values of different types Example: classes, interfaces

Why character datatype in java takes two bytes of information

Character data type of java takes 2 bytes of storing single character because java is based on Unicode system it supports 18 international language to support all characters in 18international languages it takes two bytes of information

Type Conversion and Type Casting:

Type conversion: Conversion of lower data type to higher data type is known as TypeConversion or automatic conversion or implicit type casting or Widening. Internallly Compiler is responsible to converting one data type to another data type automatically.

Type casting: Conversion of higher data type to lower data type is known as TypeCasting or Explicit type casting or Narrowing .Type casting is done by the programmer by using cast operator. Here programmer is responsible for converting one data type to another data type by using cast operator

TypeConversionAndCasting.java

```
class TypeConversionAndCasting
{
public static void main(String args[])
{
char x='A';
int y=x;// converting lower data type to higher data type
//type conversion or widening  or implicit casting or automaticconversion
//here compiler is responsible for converting one data type to another data type
System.out.print(y);
```

```
double a=1.23345555566677;
float b=(float)a;// narrowing // programmer manual conversion or
//explicit type casting
System.out.println(b);
}
}
```

Output:

```
javac TypeConversionAndCasting.java
java TypeConversionAndCasting
converting char 'A' to int=65
converting double to float=1.2334555
```

Wrapper Classes

The **wrapper class in Java** provides the mechanism ***to convert primitive into object and object into primitive***.

Since J2SE 5.0, **auto boxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as auto boxing and the automatic conversion of object into primitive is known as unboxing.

Java is an object-oriented programming language, so we need to deal with objects many times so we need to use wrapper classes for converting primitives into objects and objects into primitive.

The eight classes of the ***java.lang*** package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	**Wrapper class**
boolean	Boolean
char	Character
byte	Byte
short	Short

Int	Integer
long	Long
float	Float
double	Double

WrapperExample.java

```
//Wrapper classes example autoboxing and unboxing
class WrapperExample
{
public static void main(String args[])
{
Char ch=’A’;
Character  j=ch;//auto boxing from 1.5 onwards converting primitive into object

System.out.println(j);//auto boxing from 1.5 onwards converting primitive into object
Character c1=new Character(’A’);
char c2=c1;////un boxing from 1.5 onwards converting object into premitive
System.out.println(l);//auto boxing from 1.5 onwards converting object into premitive
}
}
```

javac WrapperExample.java

java WrapperExample

20

20

30

30

Operators

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- ✓ Unary Operator
- ✓ Arithmetic Operator
- ✓ Shift Operator
- ✓ Relational Operator
- ✓ Bitwise Operator
- ✓ Logical Operator
- ✓ Ternary Operator
- ✓ Assignment Operator.

Operator Type	Category	Precedence
Unary	Postfix	***X++(post increment)*** ***x—(post decrement)*** ***if x=5 and a=x++;*** then x=6; and a=5;
	Prefix	++***x(pre increment)*** ***--x(pre decrement)*** If x=5; and a=++x; then a=6; and x=6;
Arithmetic	Multiplicative	*
	Additive	+
	Substraction	-
	Division	/
	Modulus	%

Shift	Shift	<<>>
Relational	Comparison	<><= >=
	Equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	Ternary	? :
Assignment	Assignment	= += -= *= /= %=

instanceof operator

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Simple.java

```
class Simple{

 public static void main(String args[]){

 Simple s=new Simple();

 System.out.println(s instanceof Simple);//true
```

```
 }

}
```

javac Simple.java

java Simple

true

Command Line Arguments in Java:

If any input value is passed through the command prompt at the time of running of the program is known as **command line argument** by default every command line argument will be treated as string value and those are stored in a string array of main() method

Syntax for Compile and Run CMD programs

Compile By -> Javac Mainclass.java
Run By -> Java Mainclass value1 value2 value3 .....
Example for Command line Arguments

DecimaltoBinary.java

```
class DecimaltoBinary
{
public static void main(String args[])
{
int num=Integer.parseInt(args[0]);
String str=Integer.toBinaryString(num);
System.out.println(str);
}
}
```

Output

javac DecimaltoBinary.java

java DecimaltoBinary 10

1010

Control Flow

Control Statements are the statements which alter the flow of execution of the program.

Control Statements can be divided into three categories, namely

- Selection statements or decision making statements
- Iteration statements or looping statements
- Jump statements

Selection statement are if,if/else and Switch statements

Iteration Statements are for loop,while,do-while statements

Jump statement are continue,break statements

i)Selection statements

If/else statement syntax

```
if (condition)
{
   first statement;
}
else
{
   second statement;
}
```

Example

```
class Ifstatement

{

public static void main(String args[])

{

int i=-5;

if(i>0)

{
```

```
System.out.println("positive number");
}
else if(i<0)
System.out.println("Negative number");
else
System.out.println("number is zero");
}
}
```

Output:

```
Javac Ifstatement.java
Java Ifstatement
Negative number
```

Switch statement syntax

```
Switch(choice)
{
Case 1:
        Statements;
        break;
Case 2:
        Statements;
        break;
Case 3:
        Statements;
        break;
.....
.....
Case n:
        Statements;
        break;

Default:
break;
}
```

Example

```
class Switch_Statement
{// class begin
public static void main(String args[])

{// main method

int choice=4;
String day;
```

```
switch(choice)
{
case 1:
        day="Monday";
        break;
case 2:
        day="Tuesday";
        break;
case 3:
        day="Wednesday";
        break;
case 4:
        day="Thursday";
        break;
case 5:
        day="Friday";
        break;
case 6:
        day="Saturday";
        break;
case 7:
        day="Sunday";
        break;
default:

      day="Invalid day";
}
System.out.println("Selected choice is "+day);
}
}
```

Output:

```
Javac Switch_statement.java
Java Switch_statement
Selected choice is Thursday
```

ii)Iteration Statements: are for loop,while,do-while statements

For:

The for loop in java is used to iterate and evaluate a code multiple times.

Syntax:

for(initialization;condition;increment/decrement)

example

```
class Forloop_Demo
{ // class begin
public static void main(String args[])
{//main method
for(int i=1;i<=10;i++)
```

```
{//for loop
System.out.println(i)
}

}// end of main method

}// end of class
```

Output :

javac Forloop_Demo.java:
java Forloop_Demo:
1 2 3 4 5 6 7 8 9 10
While loop :
Syntax:

```
while(condition)
{
Statements;
}
```

Example:

```
class While_Statement
{// class begin
public static void main(String args[])
{// main method
int i=0;
while(i<=10)
{//while loop starts
System.out.println(i);
i++;
}

}// end of main method
}// end of class
```

Output :
javac While_Statement.java :
java While_Statement :
1 2 3 4 5 6 7 8 9 10
Foreach:
The traversal of elements in an array can be done by the for-each loop
Syntax:

```
Foreach(initialization:array)
{
Statements;
}
```

Example:

```
Class Foreach
```

```
{
public static void main(String args[])
{
Int arr[]={10,20,30,40,50}';
for(int i : arr)
{
System.out.println(i);
}

}// end of main method
}// end of class forEach
```

Output :
javac Foreach.java :
java Foreach :
1 2 3 4 5 6 7 8 9 10

iii.Jump statement are continue,break statements :
Break :
The break statement in java is used to terminate a loop and break the current flow of the program.
Syntax:

```
break;
```

Example:

```
class Break_Example
{// class Break_Example begin
public static void main(String args[])

{// main method
for(int i=1;i<=10;i++)
{
if(i==5)
{
break;
}
System.out.println(i);
}

}// end of main method
}// end of class
```

Output :
javac Break_Example.java
java Break_Example
1 2 3 4

Continue :
To jump to the next iteration of the loop, we make use of the continue statement. This statement continues the current flow of the program and skips a part of the code at the specified condition.

Syntax:

Continue;

Example

```
class Continue_Example

{

public static void main(String args[])

{

        for(int i=1;i<=10;i++) {

        if(i==5)

        {

        continue;

                }

                System.out.println(i);

        }

}

}
```

Output

javac Continue_Example.java

java Continue_Example

1 2 3 4 6 7 8 9 10

return

return keyword used to **exit** from a method, with or without a value.

Syntax: return

Example

```
class Return
{
int display(int n)
{
return n*n;
}
}
class Return_demo
{
public static void main(String args[])
{
Return obj=new Return();
int x=obj.display(10);// method callinf
System.out.println(x);
}
}
```

Output

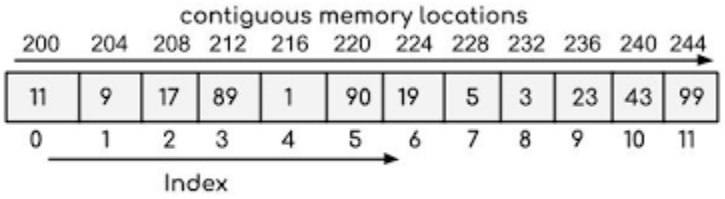
```
javac Return_demo.java
java Return_demo
100
```

Array in java:
Array is a collection of similar type of data. It is fixed in size means that you can't increase the size of array at run time. It is a collection of homogeneous data elements. It stores the value on the basis of the index value.
Advantage of Array:
One variable can store multiple value:
The main advantage of the array is we can represent multiple value under the same name.

Random access:
We can retrieve any data from array with the help of the index value.
Disadvantage of Array:
The main limitation of the array is **Size Limit** when once we declare array there is no chance to increase and decrease the size of an array according to our requirement, Hence memory point of view array concept is not recommended to use. To overcome this limitation in Java introduce the collection concept.
Note: At the time of array declaration we cannot specify the size of the array. For Example int[5] a; this is wrong.
Syntax Array in Java:
1. int[][] a;
2. int a[][];
3. int [][]a;
4. int[] a[];
5. int[] []a;
6. int []a[];

Array creation:
Every array in a Java is an object, Hence we can create array by using **new** keyword.
Single dimensional Arrays:
Syntax :
int[] arr = new int[10]; // declare, instantiate
or
int[] arr = {10,20,30,40,50};//declare instantiate, initialize
Multi dimensional arrays
int[][] arr=new int[2][3];// declare instantiate
or
int [][] arr={{1,2},{2,3,4,},{4,5,6}};// declare instantiate ,initialize

How the memory allocation of arrays in java. Is array object? If yes explain.
Yes Array is an object in java
Memory is stored as contagious memory locations in heap .



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

int b[]=new int[10];//single dimensional arrays

```
b[0]=1;
b[1]=2;
b[2]=3;
b[3]=4;
System.out.println("array b values");
for(int i=0;i<b.length;i++)
{
System.out.println(b[i]);

}
int d[][]=new int[2][3];//multi  dimensional arrays
d[0][0]=23;
d[0][1]=34;
d[0][2]=56;
d[1][0]=3;
d[1][1]=4;
d[1][2]=6;

System.out.println("array d values");
for(int i=0;i<d.length;i++)
{
        for(int j=0;j<d[i].length;j++)
        {
        System.out.println(d[i][j]);
        }

}
}
}
javac ArrayExample.java
java ArrayExample

array b values
1
2
3
4
0
0
0
0
0
0
array d values
23
34
```

56
3
4
6

Contrast ternary operator and if condition ?. Explain with example.

ternary operator	if condition
Java ternary operator is the only conditional operator that takes three operands. It's a one-liner replacement for if-then-else statement and used a lot in Java programming.	If Condition to specify a block of code to be executed, if a specified condition is true Use else to specify a block of code to be executed, if the same condition is false
Syntax: *var=(condition)?exp-True :exp-False;*	Syntax: `if (condition)` `{` `  first statement;` `}` `else` `{` `  second statement;` `}`
Example: int time = 20; String result = (time < 18) ? "Good day." : "Good evening."; System.out.println(result);	Example: int time = 20; if (time < 18) { System.out.println("Good day."); } else { System.out.println("Good evening."); }

Illustrate arrays to display a matrix 2X2. Input 5, 6, 7, 8.

Matrix.java(2X2)

```
import java.util.Scanner;
class Matrix
{
public static void main(String args[])
{
Scanner sc=new Scanner(System.in);
int a[][]=new int[2][2];
System.out.println("enter the matrix input values");
for(int i=0;i<a.length;i++)
{
        for(int j=0;j<a[i].length;j++)
        {
```

```
        a[i][j]=sc.nextInt();
        }
}
for(int i=0;i<a.length;i++)
{
        for(int j=0;j<a[i].length;j++)
        {
        System.out.println(a[i][j]);
        }
}
}
}
```

```
javac Matrix.java
java Matrix
5 6 7 8
5
6
7
8
```

UNIT-2

Property or Variable or Attribute or Class member :

Variables are used for storing values.

There are different **types** of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers without decimals, such as 123 or -123
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- boolean - stores values with two states: true or false.

Syntax:

< data Type><variable name> = <value>

Rules of declaring variable :

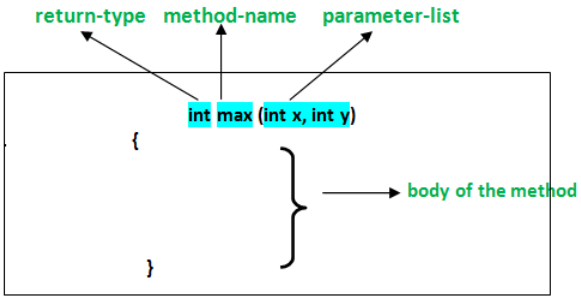
- A variable name can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and two special characters such as underscore and dollar Sign.
- The first character must be a letter.
- Blank spaces cannot be used in variable names.
- Java keywords cannot be used as variable names.
- Variable names are case-sensitive.

Method :

A method is a collection of statements that perform some specific task

Methods allow us to **reuse** the code without retyping the code.

Syntax of method :



The return type : The data type of the value returned by the method or void if does not return a value.

Method Name : Name of the method

Parameter list : parameters are separated by comas

Method body :

it is enclosed between braces

MethodOverloading.java:

```
class MethodOverloading
{
        void add(int a,int b)
        {
        int c=a+b;
        System.out.println(c);
        }
```

```
        void add(float a,float b)
        {
        float c=a+b;
        System.out.println(c);
        }

        void add(double a,double b)
        {
        double c=a+b;
        System.out.println(c);
        }
public static void main(String args[])
{
MethodOverloading m=new MethodOverloading();
m.add(10,20);
m.add(10.5,34.6);
m.add(1.3f,3.4f);
}
}
```

Output :

```
javac MethodOverLoading.java
java MethodOverloading
30
45.1
4.7
```

Constructor:

Constructors are meant for initializing the object. Constructor is a special type of method that is used to initialize object values.

Constructor is invoked at the time of object creation. It constructs the values i.e. data for the object that is why it is known as constructor.

Constructor is just like the instance method but it does not have any explicit return type.

Constructor name must be same as its class name. Constructor should not return any value even void also.

Constructors are called automatically whenever an object is creating.

Types of Constructors:

There are two types of constructors:-

1. Default constructor (no-argument constructor)
2. Parameterized constructor

1. Default constructor (no-argument constructor):-

A constructor is one which will not take any parameter.

A constructor that have no parameter is known as default constructor.

Syntax:-

```
class < class name >
{
```

```
        classname() //default constructor
        {
                Block of statements;
                .....;
                .....;
        }
        .....;
        .....;
};
```

2. Parameterized Constructor:- A constructor is one which takes some parameters.

Syntax:-

```
class < class name >
{
        classname(list of parameters) //paramiterized constructor
        {
                Block of statements;
                .....;
                .....;
        }
        .....;
        .....;
};
```

Constructor overloading:

is a technique in Java in which a class can have any number of constructors such that each constructor differ with their parameters

Constructors with diff arguments is known as Constructor Overloading

Constructor over loading example

Student.java :

```
class Student
{
int rollNumber;
String branchName;
        Student()
        {
        rollNumber=100;
        branchName="CSE";
        System.out.println(rollNumber);
        System.out.println(branchName);
        }
        Student(int rollNumber)
        {
        this.rollNumber=rollNumber;
        branchName="CSE";
        System.out.println(rollNumber);
        System.out.println(branchName);
```

```
        }
        Student(int rollNumber,String branchName)
        {
        this.rollNumber=rollNumber;
        this.branchName=branchName;
        System.out.println(rollNumber);
        System.out.println(branchName);

        }
public static void main(String args[])
{
        Student ravi=new Student();
        System.out.println("-----------");
        Student seetha=new Student(101);
        System.out.println("-----------");
        Student balu=new Student(102,"CSE");

}
}
```

Output:

```
javac Student.java
java Student
100
CSE
-----------
101
CSE
-----------
102
CSE
```

Inheritance(IS A Relation-ship): The process of taking the features(data members and class) from one class to another class is known as inheritance
The *class* which is giving the features is known as base/parentclass.
The *class* which is taking the features is known as derived/child/subclass.
Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Advantages of INHERITANCE:

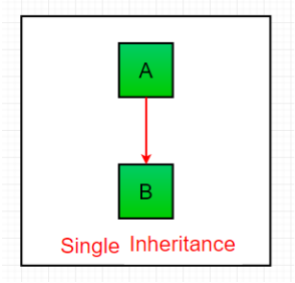
I. Application development time is veryless.
II. Redundancy (repetition) of the code is reducing. Hence we can get less memory cost and consistent results.

Types of INHERITANCES :

Based on taking the features from base class to the derived class, in JAVA we have five types of inheritances. They are as follows:

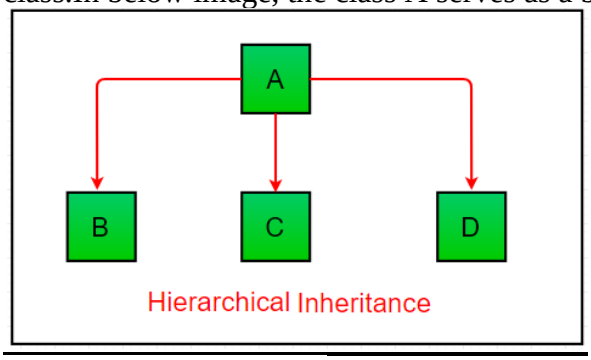
Single Inheritance :

In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.



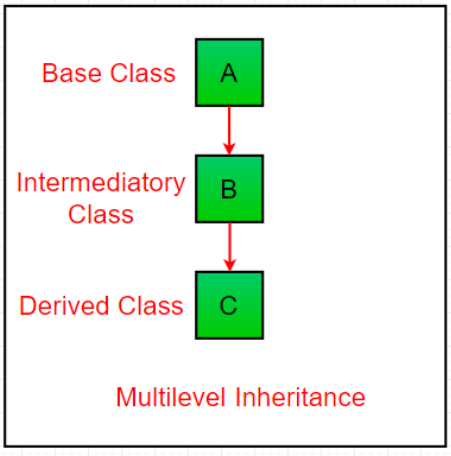
Heirarcial Inheritance :

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.



Multilevel Inheritance :

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.



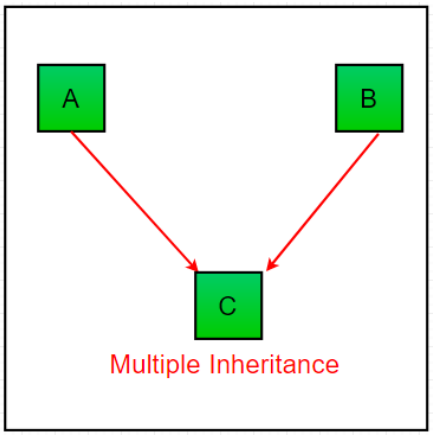
Multiple Inheritance :

In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes.

Java does **not** support multiple inheritance with classes.

In java, we can achieve multiple inheritance only through Interfaces.

In image below, Class C is derived from class A and B.

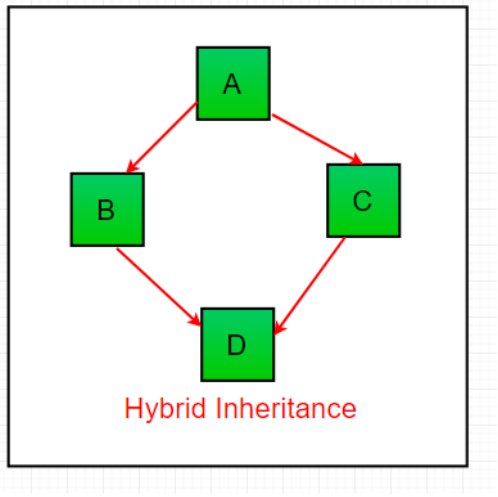


Hybrid Inheritance:

It is a mixture of two or more of the above types of inheritance.

Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes.

In java, we can achieve hybrid inheritance only through Interfaces.



Singleinheritance.java :

```
class Mobile_Version1
{
        void smsservice()
        {
        System.out.println("sms service");
        }
        void callingservice()
        {
        System.out.println(" calling service");
        }
}
class Mobile_Version2 extends Mobile_Version1
{
        void cameraservice()
```

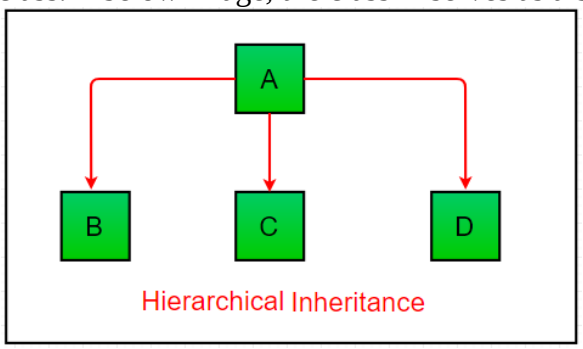
```
        {
        System.out.println("camera service");
        }
}
class Single_inheritance
{
public static void main(String args[])
{
Mobile_Version2 m=new Mobile_Version2();
System.out.println("Calling Mobile_Version1 class methods through Mobile_Version2 object ");
m.cameraservice();
m.smsservice();
m.callingservice();
}
}
```

Output :

```
Calling Mobile_Version1 class methods through Mobile_Version2 object
camera service
sms service
calling service
```

Heirarcial Inheritance :

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.



Heirarcial_inheritance.java :

```
class MobileVersion1
{
void smsservice()
 {
        System.out.println("sms service");
 }
void calling_service()
 {
        System.out.println("calling service");
 }
}
class MobileVersion2 extends MobileVersion1
```

```
{
	voidcameraservice()
	{
	System.out.println("camera service");
	}
}
class MobileVersion3 extends MobileVersion1
{
	voidfingerprint_service()
	{
	System.out.println("finger print service");
	}
}
class MobileVersion4 extends MobileVersion1
{
	void facerecognization_service()
	{
	System.out.println("facerecognization service");
	}
}
class Heirarcial_inheritance
{
public static void main(String args[])
{
MobileVersion1 m1=new MobileVersion1();
System.out.println("mobile version1 services");
m1.calling_service();
m1.smsservice();

MobileVersion2 m2=new MobileVersion2();
System.out.println("\nmobile version2 services:-");
m2.calling_service();
m2.smsservice();
m2.cameraservice();
MobileVersion3 m3=new MobileVersion3();
System.out.println("\nmobile version3 services:-");
m3.calling_service();
m3.smsservice();
m3.fingerprint_service();
MobileVersion4 m4=new MobileVersion4();
System.out.println("\nmobile version4 services:-");
m4.calling_service();
m4.smsservice();
m4.facerecognization_service();
}
```

```
}
```

Output :

javac Heirarcial_inheritance.java
java Heirarcial_inheritance

```
mobile version1 services
calling service
sms service
mobile version2 services:-
calling service
sms service
camera service
mobile version3 services:-
calling service
sms service
finger print service
mobile version4 services:-
calling service
sms service
facerecognization service
```

This Keyword :

The main purpose of **using this keyword** is to differentiate the formal parameter and class members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity

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 .variablename-this is used for calling current class variable
- this() (this is used for calling current class constructor
- this.methodname()-this is used to call current class method

 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)

Student.java :

```
class Student
{
int rollNumber;
String branchName;
        Student(int rollNumber,String branchName)
        {
        this.rollNumber=rollNumber;
        this.branchName=branchName;
        }

public static void main(String args[])
{
Student s1=new Student(100,"CSE");
```

```
System.out.println(s1.rollNumber);
System.out.println(s1.branchName);
}
}
```

```
javac Student.java
java Student
100
CSE
```

Super Keyword :

Super keyword is placing an important role in three places. They are at variable level, at method level and at constructor level.

Super at variable level :

Whenever we inherit the base class variables into derived class, there is a possibility that base class variables are similar to derived class members.

In order to distinguish the base class variables with derived class variables in the derived class, the base class members will be preceded by a keyword super.

Syntax for super at variable level super.variablename;

Super at method level :

Whenever we inherit the base class methods into the derived class, there is a possibility that base class methods are similar to derived methods.

To differentiate the base class methods with derived class methods in the derived class, the base class methods must be preceded by a keyword super.

Syntax :for super at method level: super.methodname();

Super at Constructor Level :

super keyword can also be used to invoke the parent class construct**or**

Syntax super();

MobileVersion2.java :

```
class MobileVersion1
{
String modelName="Nokia1600";
int price=2000;

        public void display()
        {
        System.out.println("MobileVerison1 features");
        System.out.println(modelName);
        System.out.println(price);
        }
}
class MobileVersion2 extends MobileVersion1
{
String modelName="Redmi 10";
int price=10000;

        public void display()
        {
```

```
        System.out.println("MobileVerison2 features");
        System.out.println(modelName);
        System.out.println(price);
        System.out.println("MobileVerison1 features");
        System.out.println(super.modelName);
        System.out.println(super.price);
        super.display();
        }

public static void main(String args[])
{
MobileVersion2 m1=new MobileVersion2();
m1.display();
}

}
javac MobileVersion2.java
java MobileVersion2
MobileVerison2 features
Redmi 10
10000
MobileVerison1 features
Nokia1600
2000
MobileVerison1 features
Nokia1600
2000
```

How to use super keyword in child class to access constructor in parent class with example.

```
//Accessing Super class constructor from child class
class MobileVersion1
{
int a=20;
MobileVersion1()
{
System.out.println("Super class constructor");
}
        void callingService()
        {
        System.out.println("calling serivce");
        }

        void smsService()
        {
        System.out.println("sms serivce");

        }
```

```
}
class MobileVersion2  extends MobileVersion1
{
MobileVersion2()
{
super();//Accessing super class constructor from child class
System.out.println("Sub class constructor");
}
        void cameraService()
        {
        System.out.println("camera service");
        }
public static void main(String args[])
{
MobileVersion2 m1=new MobileVersion2();
m1.cameraService();
m1.callingService();
m1.smsService();
System.out.println(m1.a);
}
}
```

Output :

```
javac MobileVersion2.java
java MobileVersion2
Super class constructor
Sub class constructor
camera service
calling serivce
sms serivce
20
```

What the difference between 'extends' and 'implements' where can we use these keywords.

S.No.	Extends	Implements
1.	By using "extends" keyword a class can inherit another class, or an interface can inherit other interfaces	By using "implements" keyword a class can implement an interface
2.	It is not compulsory that subclass that extends a superclass override all the methods in a superclass.	It is compulsory that class implementing an interface has to implement all the methods of that interface.
3.	Only one superclass can be extended by a class.	A class can implement any number of an interface at a time
4.	Any number of interfaces can be extended by interface.	An interface can never implement any other interface

Association (Has a Relation-ship) :

Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.

Two forms of Associations are Composition and **Aggregation**

Aggregation is a kind of association

Aggregation vs Composition

1. **Dependency:** Aggregation implies a relationship where the child **can exist independently** of the parent. For example, Bank and Employee, delete the Bank and the Employee still exist. whereas Composition implies a relationship where the child **cannot exist independent** of the parent. Example: Human and heart, heart don't exist separate to a Human
2. **Type of Relationship:** Aggregation relation is **"has-a"** and composition is **"part-of"** relation.

Aggregation Example (has a relation ship)

```
class Author
{
  String authorName;
  int age;
  String place;

  // Author class constructor
  Author(String name, int age, String place)
  {
    this.authorName = name;
    this.age = age;
    this.place = place;
  }
}

class  Book
{
  String name;
  int price;
  // author details
  Author auther;
  Book(String n, int p, Author auther)
  {
    this.name = n;
    this.price = p;
    this.auther = auther;
  }
  public static void main(String[] args) {
    Author auther = new Author("John", 42, "USA");
    Book b = new Book("Java for Begginer", 800, auther);
    System.out.println("Book Name: "+b.name);
    System.out.println("Book Price: "+b.price);
    System.out.println("----------Auther Details----------");
    System.out.println("Auther Name: "+b.auther.authorName);
```

```
    System.out.println("Auther Age: "+b.auther.age);
    System.out.println("Auther place: "+b.auther.place);

  }
}
```

Output :

```
javac Book.java
java Book
Book Name: Java for Begginer
Book Price: 800
----------Auther Details----------
Auther Name: John
Auther Age: 42
Auther place: USA
```

MethodOverRiding :

When super class and sub class method has same name the subclass method overrides the super class method is known as method over riding.

What are the benefits of method overriding in java.

The main advantage of method overriding is that it allows the super class to define methods which would be common to all of the subclasses and it also allows the subclasses to define their own specific implementation of some or all of those methods.

```
/*When super class and sub class contains
same method overriding super class method with the sub class
is knwon  method over riding
*/
  class MobileVersion1
{
          public   void hotSpot()
          {
          System.out.println("5 meters");
          }

}
class MobileVersion2 extends MobileVersion1
{
          public void hotSpot()
          {
          System.out.println("10 meters");
          }
  public static    void main(String args[])
{
MobileVersion2 m=new MobileVersion2();
m.hotSpot();
}
```

}

Output :

javac MobileVersion2.java
java MobileVersion2
10 meters

Static keyword :

In Java, **static keyword** is mainly used for memory management. It can be used with variables, methods, blocks and nested classes. It is a keyword which is used to share the same variable or method of a given class.

Basically, static is used for a constant variable or a method that is same for every instance of a class. The main method of a class is generally labeled static.

In order to create a static member (block, variable, method, nested class), you need to precede its declaration with the keyword ***static***. When a member of the class is declared as static, it can be accessed before the objects of its class are created, and without any object reference.

Static block :

Java supports a special block, called static block which can be used for static variable initializations of a class. This code inside static block is executed only once. Static block calling before main method

Syntax :

static
{
//static block
}

Static Variable :

When you declare a variable as static, then a single copy of the variable is created and divided among all objects at the class level. Static variables are global variables. Basically, all the instances of the class share the same static variable. Static variables can be created at class-level only.

Generally Static variables are called by
Classname.variablename;

Static method :

When a method is declared with the ***static*** keyword, it is known as a static method. The most common example of a static method is the ***main( )*** method.

We can access static methods using classname
Static methods can be called by classname.methodname()

Syntax :

Classname.methodname();

Static keyword example

```
//static  Block Level will execute before main
// static varaible can be called byclassname.variable
//staticmethod canbe called byclassname.methodname()
//static  Block Level will execute before main
class AccountHolder
{
```

```
long accountNumber;
static String bankName="SBI";
static long pinCode=530016L;
        static
        {
        System.out.println("library files loading");
        }
        static void pinCode()
        {
        System.out.println(pinCode);
        }

public static void main(String args[])
{
System.out.println(AccountHolder.bankName);
AccountHolder.pinCode();
}
}
```

Output :

```
javac AccountHolder.java
java AccountHolder
library files loading
SBI
530016
```

Polymorphism :

Illustrate dynamic binding(Dynamic polymorphism) and static binding(Static polymorphism) in java.

Polymorphism: **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Static polymorphism or Compile time polymorphism or early binding

In static polymorphism method invocation is determined at compile time

Static polymorphism is known as early binding because method invocation is determined early by the compiler at the compile time

StaticPolymorphism.java :

```
class StaticPolymorphism
{

        public static void add(int a ,int b)
        {
        int c=a+b;
```

```
        System.out.println(c);
        }

        public static void  add(double a,double b)
        {
        double c=a+b;
        System.out.println(c);
        }
public static void main(String args[])
{
StaticPolymorphism s=new StaticPolymorphism();
s.add(10,20);
s.add(10.4,23.5);
}
}
```

```
javac StaticPolymorphism.java
java StaticPolymorphism
30
33.9
```

Dynamic Polymorphism or run time polymorphism or late binding
method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.
dynamic polymorphism is known as late binding because method invocation is determined late at runtime by the jvm.

Method Overriding is an example of Dynamic polymorphism :

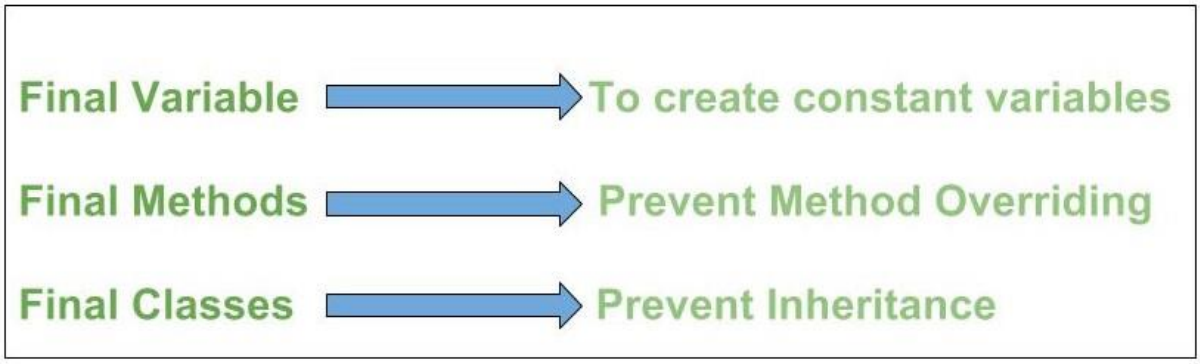
MobileVersion2.java

```
 class MobileVersion1
{
         public   void hotSpot()
        {
        System.out.println("5 meters");
        }

}
class MobileVersion2 extends MobileVersion1
{
        public void hotSpot()
        {
        System.out.println("10 meters");
        }
 public static    void main(String args[])
{
MobileVersion2 m=new MobileVersion2();
m.hotSpot();
}
```

```
}
```
Output
```
javac MobileVersion2.java
java MobileVersion2
10 meters
```

Final Keyword :

final keyword is used in different contexts. First of all, ***final*** is a non-access modifier applicable **only to a variable, a method or a class**.Following are different contexts where final is used



Final variable:

When a variable is declared with ***final*** keyword, its value can't be modified, essentially, a constant.

Final_Variable.java

```
class Final_Variable
{
public static void main(String args[])
{
final int pi=3.14;
int a=20;//complete time error final variable cannot be reintiantiated
}
}
```

Output :

compile time error

Final method:

When a method declared as final method cannot be overridden by the sub class

Final_Method.java

```
 class MobileVersion1
{
        final public   void hotSpot()
        {
        System.out.println("5 meters");
        }

}
class MobileVersion2 extends MobileVersion1
```

```
{
        public void hotSpot()
        {
        System.out.println("10 meters");
        }
  public static    void main(String args[])
{
MobileVersion2 m=new MobileVersion2();
m.hotSpot();
}

}
```

Output :

```
compile time error
MobileVersion2.java:15: error: hotSpot() in MobileVersion2 cannot override hotSpot() in
MobileVersion1
      public void hotSpot()
          overridden method is final
```

Final class:

When ever a parent class declared as final class its child class or sub class cannot be inherit the parent class

Child.java

```
final class Parent
{//parent class

}
class Child extends Parent

{//child class


public static void main(String args[])
{
Child c=new Child();
}
}
```

Output :

```
javac   Final_Class.java
java Final_Class
Compile time error
Final class cannot be inherited
```

Access modifiers or Access specifiers in java

Access specifiers define the boundary and scope to access the method, variable, and class etc. Java has defines four type of access specifiers such as:-

1. public

2. private
3. protected
4. default

These access specifiers are defines the scope for variables/methods. If you are not defining any access specifier so it will be as 'default' access specifier for variables/methods.

1. public access specifier:- The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
Ex:-
public int number;

2. private access specifier:- The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
3. protected access specifier:- The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. default access specifier:- Actually, there is no default access modifier; the absence of a modifier is treated as default. The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default..
Classes belonging to other packages cannot access. That is why default access modifier is known as package level access.

Access Specifiers/modifiers scope in java.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Package:
Package is a collection of similar classes,interfaces, and sub packages
simply package is a directory or folder

uses of packages are :
1) Java package is used to categorize the classes
and interfaces so that they can be easily maintained.

2) Java package provides access protection.
Types of packages :
There are two types of packages are there
in java

1.built in packages:
2. user defined packages
Built in packages: these are predefined packages
which are present after installing jdk
example
import java.io.*;
import java.applet.*
import java.sql.*;
user defined packages :these are defined by the user
creating user defined package
syntax:
package packagename;
how to compile package program in java
javac -d . filename.java
-d creates package
how to run package program
java packagename.classname
importing userdefined packages :

1. Import packagename.*(imports allclasses in the package)
2. Import packagename.classname;(import particular class name)

CSE.java:

```
package college;
class CSE
{
public static void main(String args[])
{
System.out.println("CSE is the branch is part of college");
}
}
```

output:
javac -d . CSE.java
java College.CSE
CSE is the branch is part of college
Abstract class:
Abstraction is a process of hiding the implementation details and showing only functionality to the user.
There are two types of classes present in java
1.Concrete classes
2.Abstract classes
A concrete class is one which is containing fully defined methods or implemented method.
A class that is declared with abstract keyword, is known as **abstract class**.
- It can have abstract and non-abstract methods.
- It cannot be instantiated. We cannot create objects to abstract class

Abstract class can contain 0 or more abstract methods

We can achieve 0 to 100% abstraction using abstract class;
Syntax:
abstract class <classname>
{
}
Abstract methods: A method which is declared as abstract and does not have implementation is known as an abstract method.An abstract method methodis an incomplete method

Abstract method syntax :
abstract returntype methodname();

AbstractBankDemo.java :

```
abstract class Bank
{
public abstract void getRateOfInterest();
        public void withdraw(int balance,int debit)
        {
        int withdraw=balance-debit;
        }
}
class SBI extends Bank
{
        public void getRateOfInterest()
        {
        System.out.println("SBI Rate of interest is 6%");
        }
}


class HDFC extends Bank
{
        public void getRateOfInterest()
        {
        System.out.println("SBI Rate of interest is 8%");
        }
}
class AbstractBankDemo
{

public static void main(String args[])
{
SBI s=new SBI();
s.getRateOfInterest();
HDFC h=new HDFC();
h.getRateOfInterest();
```

```
}
}
```

Output :

```
javac AbstractBankDemo.java
java AbstractBankDemo
Interest rate of SBI 6%
Interest rate of HDFC 8%
```

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.

- It is used to achieve fully abstraction(100% abstraction).
- By using Interface, you can achieve multiple inheritance in java.
- 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.
  - You can not instantiate an interface which means we cannot create object directly to interface.
  - It does not contain any constructors.
  - All methods in an interface are abstract.
  - Interface can not contain instance fields. Interface only contains public static final variables.
  - Interface is can not extended by a class; it is implemented by a class.
  - Interface can extend multiple interfaces. It means interface support multiple inheritance

  Interface Syntax:

```
interface <interfacename>
{
 public static final datatype variablename=value;
 //Any number of final, static fields
 abstract returntype methodname(list of parameters or no parameters);
 //Any number of abstract method declarations
}
```

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.

• Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
5) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
6) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

How Multiple inheritance achieved in java :

Child.java :

```
interface Mother
{
 public static final float height=5.2f;
public abstract void height();
}
interface Father
{
public static final float height=6.2f;
public abstract void height();


}
class Child implements Mother,Father
{
public void height()
{
System.out.println((Mother.height+Father.height)/2);

}
public static void main(String args[])
{
Child c=new Child();
c.height();
}
}
```

javac Child.java
java Child
5.7
What is interface inheritance in Java? How it can be inherited? Explain with example.
Or
How to access an interface from an interface write in brief with example to access.
Interface Inheritence :
Interface extends another interface is known as interface inheritance
BankDemo.java :

```
interface Bank
{
public abstract void withdrawService();
public abstract void depositService();
}
interface SBI extends Bank
{
public abstract void loanService();
}
class BankDemo implements SBI
{
        public void withdrawService()
        {
        System.out.println("withdraw service implementation");
        }
        public void depositService()
        {
        System.out.println("deposit service implementation");
        }
        public void loanService()
        {
        System.out.println("loan service implementation");
        }
public static void main(String args[])
{
BankDemo b=new BankDemo();
b.withdrawService();
b.depositService();
b.loanService();
}

}
```

Output :
javac BankDemo.java
java BankDemo
java BankDemo
withdraw service implementation

deposit service implementation
loan service implementation

Functional Interface :

A functional interface is an interface that contains only one abstract method. They can have only one functionality to exhibit.
Runnable, ***ActionListener***, ***Comparable*** are some of the examples of functional interfaces.
Before Java 8, we had to create anonymous inner class objects or implement these interfaces.
It can have any number of default, static methods but can contain only one abstract method.

FunctionalInterfaceExample.java

```
interface sayable{
    void say(String msg);
}
public class FunctionalInterfaceExample implements sayable{
    public void say(String msg){
        System.out.println(msg);
    }
    public static void main(String[] args) {
        FunctionalInterfaceExample fie = new FunctionalInterfaceExample();
        fie.say("Hello there");
    }
}
```

Output :

javac FunctionalInterfaceExample.java
java FunctionalInterfaceExample
Hello there

How do you declare static method in interface and write it's syntax. How to access it form main method.

Static Methods in **Interface** are those methods, which are defined in the interface with the keyword static. Unlike other methods in Interface, these static methods contain the complete definition of the function and since the definition is complete and the method is static, therefore these methods cannot be overridden or changed in the implementation class.

```
// Java program to demonstrate
// static method in Interface.

interface NewInterface {

    // static method
    static void hello()
    {
        System.out.println("Hello, New Static Method Here");
    }

    // Public and abstract method of Interface
    void overrideMethod(String str);
```

```
}

// Implementation Class
public class InterfaceDemo implements NewInterface {

   public static void main(String[] args)
   {
      InterfaceDemo interfaceDemo = new InterfaceDemo();

      // Calling the static method of interface
      NewInterface.hello();

      // Calling the abstract method of interface
      interfaceDemo.overrideMethod("Hello, Override Method here");
   }

   // Implementing interface method

   @Override
   public void overrideMethod(String str)
   {
      System.out.println(str);
   }
}
```

Output:

Hello, New Static Method Here
Hello, Override Method here

Class Casting :

A process of converting one data type to another is known as **Typecasting**
In java, there are two types of casting namely up casting and down casting as follows:

Upcasting :

Assigning the object of subclass to parent class reference is known as Upcasting
Upcasting is done by the system implicitly

DownCasting :

Downcasting: assigning the Object or object reference of super class to the sub class reference is known as downcasting
Downcasting cannot done implicitly
Downcastingmust done by the programmer explicitly
Downcasting always need upcasting

ClassCasting.java :

```
class Mobile
{
                                public void calling()
                                {
                                System.out.println("Calling from nokia 1600");
                                }
```

```
                                public void smsService()
                                {
                                System.out.println("smsservice");
                                }
}
class Samsung extends Mobile
{
                                public void camera()
                                {
                                System.out.println("camera");
                                }
}
class ClassCasting
{
public static void main(String args[])
{
System.out.println("Upcasting");
Samsung s=new Samsung();
Mobile m=s;//assigning child class referece to parentclass type
m.calling();
m.smsService();
System.out.println("Downcasting");
Mobile m1=new Samsung();
Samsung s1=(Samsung)m1;// assigning parent reference to sub class type
s1.camera();
s1.calling();
}
}
```

Output :

```
javac ClassCasting.java
java ClassCasting
Upcasting
Calling from nokia 1600
smsservice
Downcasting
camera
Calling from nokia 1600
```

Object Cloning in java :

The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.:

The **clone() method** is defined in the Object class. Syntax of the clone() method is as follows:

1. protected Object clone() throws CloneNotSupportedException

Student.java :

```
class Student implements Cloneable
{
String rollNumber;
int age;
                                        Student(String rollNumber,int age)
                                        {
                                        this.rollNumber=rollNumber;
                                        this.age=age;
                                        }
                                        public Object clone()throws CloneNotSupportedException
                                        {
                                        return super.clone();
                                        }
public static void main(String args[])throws CloneNotSupportedException
{
Student s1=new Student("20kd1a1567",22);
System.out.println("S2 hash code="+s1.hashCode());
Student s2=(Student)s1.clone();
System.out.println(s2.rollNumber);
System.out.println(s2.age);
System.out.println("s2 hash code="+s2.hashCode());
}
}
```

OUTPUT :

```
S2 hash code=746292446
20kd1a1567
22
s2 hash code=140435067
```

Nested Classes:

Java nested class is a class which is declared inside the another class.

We use inner classes to logically group classes in one place so that it can be more readable and maintainable.

Nested class cannot exit independently without outer class

advantages:

nested classes enables you to group classes in a single pace.

It creates more read ability of the code.

There are two types of nested classes in java

1.Non Static nested class

2.Static nested class

Non static nested class: non static inner class present inside outer class is known as Non static nested class

Regular inner class or non static class object creation :

```
Outer_Class outer = new Outer_Class();
Outer_Class.Inner_Class inner = outer.new Inner_Class();
```

NonStatic nested class again divided into three types
1.regular inner class:non static inner class present inside outer classknown as regualra inner class
Regular innerclass can access private,static,non static memebrs of outer class
2.local method inner class: inner class present inside local method of outer class is known as local method inner class
3Anonymous inner class:inner class which has no name is known as Anonymous inner class

Static inner class :

static class present inside the outer class is known as static inner classStatic inner can can access only static members of outer class. It cannot access non static memebrs of outer class

static class object creation :

OuterClass.StaticNestedClass nestedObject =new OuterClass.StaticNestedClass();

Regular inner class example :

Outer.java :

```
class Outer
{
 class Inner
{
        public void m1()
        {
        System.out.println(" regular Innerclass method");
        }

}
public static void main(String args[])
{
Outer o=new Outer();
Outer.Inner i=o.new Inner();
i.m1();
}
}
```

```
javac Outer.java
Java Outer
regular Innerclass method
```

Static inner class example :

```
class OuterClass
{
static int x=10;
        static class InnerClass
        {

                public void display()
                {
                System.out.println(x);
                }
        }
public static void main(String arg[])
```

```
{

OuterClass.InnerClass o=new OuterClass.InnerClass();
o.display();
}
}
```
java OuterClass.java
java OuterClass
10

UNIT-3 Strings and Collections

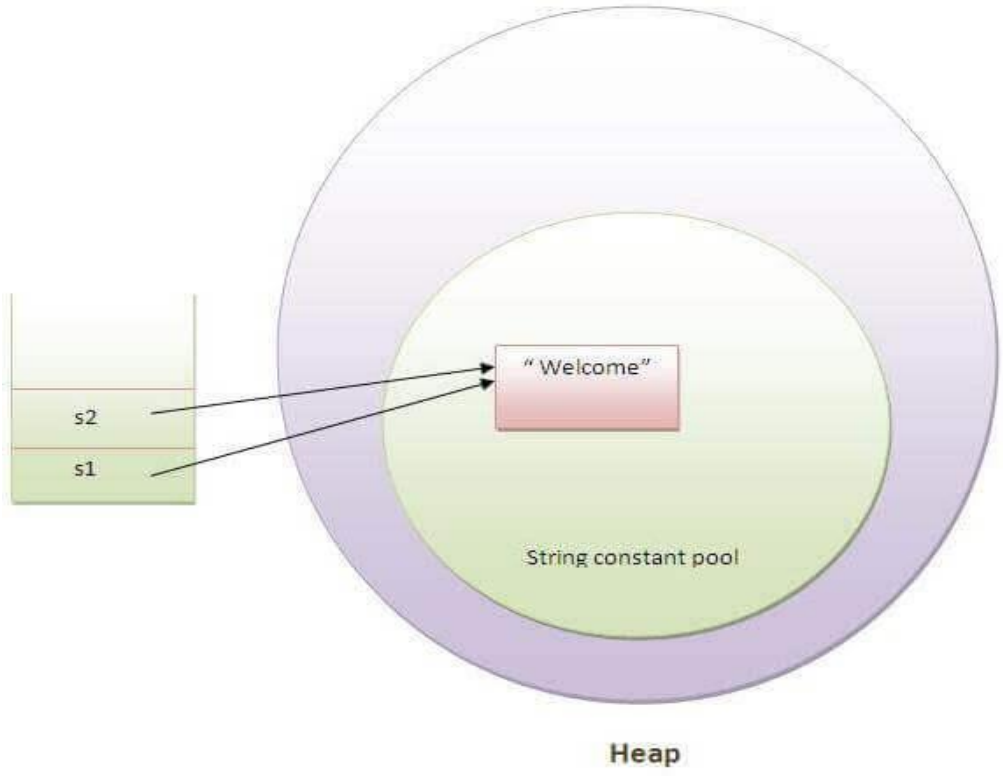
String

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 a string object.
String is a sequence of characters. In java, objects of String are immutable which means a constant and cannot be changed once created.
There are two ways to create string in Java:

- *String literal*
  String s = “GeeksforGeeks”;
- Using *new* keyword
  String s = new String (“GeeksforGeeks”);
  The Java String is immutable which means contents of the string cannot be changed.
  Whenever we change any string, a new instance is created.
  For creating mutable strings, we use StringBuffer and StringBuilder classes.

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

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a 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, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

1. String s=new String("Welcome");//creates two objects and one reference variable
   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 a heap (non-pool).

Java String class methods :
The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1	charAt(int index)	returns char value for the particular index
2	length()	returns string length
3	substring(int beginIndex)	returns substring for given begin index.
4	substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
5	contains(CharSequence s)	returns true or false after matching the sequence of char value.
6	equals(Object another)	checks the equality of string with the given object.
7	isEmpty()	checks if string is empty.
8	concat(String str)	concatenates the specified string.
9	replace(char old, char new)	replaces all occurrences of the specified char value.
10	equalsIgnoreCase(String another)	compares another string. It doesn't check case.
11	indexOf(String substring)	returns the specified substring index.
13	toLowerCase()	returns a string in lowercase.
14	toUpperCase()	returns a string in uppercase.
15	trim()	removes beginning and ending spaces of this string.
16	valueOf(int value)	converts given type into string. It is an overloaded method.

Immutable String in Java :
In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.
Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

```
1. class Testimmutablestring{
2.  public static void main(String args[]){
3.    String s="Sachin";
4.    s.concat(" Tendulkar");//concat() method appends the string at the end
5.    System.out.println(s);//will print Sachin because strings are immutable objects
6.  }
7. }
```

Output:Sachin

Why string objects are immutable in java?

Because java uses the concept of string literal.Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

Java String compare :

There are three ways to compare string in java:

1. By equals() method
2. By = = operator
3. By compareTo() method

1) String compare by equals() method

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- equals(Object another) compares this string to the specified object.
- equalsIgnoreCase(String another) compares this String to another string, ignoring case.

2) String compare by == operator

The = = operator compares references not values.

3) String compare by compareTo() method

The String compareTo() method compares values lexicographically based on ascii values and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

- s1 == s2 :0
- s1 > s2 :positive value
- s1 < s2 :negative value

StringBuffer :

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.

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.

String Buffer Methods :

length( ) and capacity( ): The length of a StringBuffer can be found by the length( ) method, while the total allocated capacity can be found by the capacity( ) method.

append(String s): It is used to add text at the end of the existence text. Here are a few of its forms:

insert(int index, String str): It is used to insert text at the specified index position. These are a few of its forms:

reverse( ): It can reverse the characters within a StringBuffer object using reverse( ).This method returns the reversed object on which it was called.

delete(int startIndex,int endIndex):

is used to delete the string from specified startIndex and endIndex.

ensureCapacity(int minimum capacity):

is used to ensure the capacity at least equal to the given minimum.

indexOf(String substring)	returns the specified substring index.

replace(int startIndex,int endIndex,String str):

is used to replace the string from specified startIndex and endIndex.

Difference between String and String Buffer :

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.

StringBuilder

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.

Constructor	Description
StringBuilder()	creates an empty string Builder with the initial capacity of 16.

Important methods of StringBuilder class :

Method	Description
append(String s)	is used to append the specified string with this string.
insert(int offset, String s)	is used to insert the specified string with this string at the specified position.
replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
reverse()	is used to reverse the string.
capacity()	is used to return the current capacity.
ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
charAt(int index)	is used to return the character at the specified position.

length()	is used to return the length of the string i.e. total number of characters.
substring(int beginIndex)	is used to return the substring from the specified beginIndex.
substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

Difference between String Buffer and String Builder :

	String Buffer	String Builder
1)	StringBuffer is *synchronized* i.e. thread safe.It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* thanStringBuilder.	StringBuilder is *more efficient* than StringBuffer.

StringTokenizer :

The **java.util.StringTokenizer** class allows you to break a string into tokens.
It is simple way to break string.
StringTokenizer is present in util package
Methods of StringTokenizer class
The 6 useful methods of StringTokenizer class are as follows:

Public method	Description
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object.
String nextToken(String delim)	returns the next token based on the delimeter.
boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

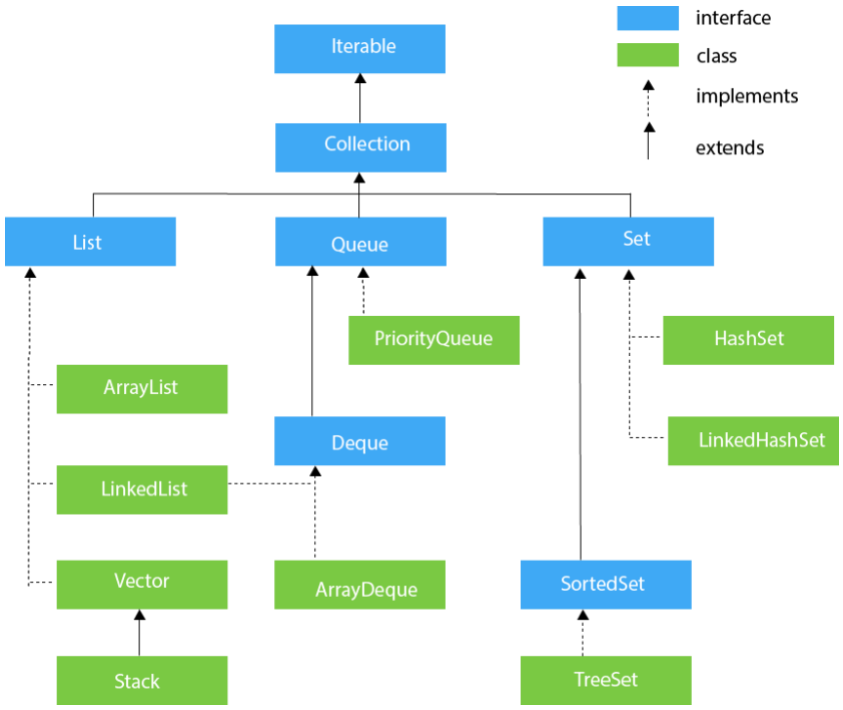
StringTokeniser.java

```
import java.util.StringTokenizer;
class StringTokeniserDemo
{
public static void main(String args[])
{
StringTokenizer str=new StringTokenizer("Java is Object Oriented Programming Language ");
System.out.println("no of tokens="+str.countTokens());

        while(str.hasMoreTokens())
        {
        System.out.println(str.nextToken());
        }
```

}
}
Output :
java Stringtokeniser
no of tokens=4
Java
is
Object
Oriented
Programming
Language
Collections in Java :
A Collection represents a single unit of objects, i.e., a group.
The **Collection in Java** is a framework that provides architecture to store and manipulate the group of objects.
Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
Advantage of collections over arrays is arrays are fixed in size we cannot increased or decreased size based on our requirement but collections are growable in nature based on our requirement we can increase or decrease size
Frame work:
provides readymade architecture. It represents a set of classes and interfaces.
The **Collection framework** represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

Hierarchy of Collection Framework ;
The **java.util** package contains all the classes and interfaces for the Collection framework.



Iterable Interface :

The Iterable **interface** is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

Methods of Iterable Interface

1. Iterator<T> iterator()

It returns the iterator over the elements of type T.

Methods of Collection interface :

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	add(E e)	It is used to insert an element in this collection.
2	remove(Object element)	It is used to delete an element from the collection.
3	size()	It returns the total number of elements in the collection.
4	clear()	It removes the total number of elements from the collection.
5	contains(Object element)	It is used to search an element.
6	public Iterator iterator()	It returns an iterator.
7	equals(Object element)	It matches two collections.

List Interface: It is present in java.util package.

List interface is the child interface of Collection interface.
It inhibits a list type data structure in which we can store the ordered collection of objects.
It can have duplicate values.
List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
There are various methods in List interface that can be used to insert, delete, and access the elements from the list.
The classes that implement list interface are:
ArrayList

- The ArrayList class implements the List interface.
- It uses a dynamic array to store elements
- It can store the duplicate element of different data types.
- The ArrayList class maintains the insertion order
- ArrayList is non-synchronized.
- Data accessing is fast compare to linked list

ArrayListDemo.java :

```
import java.util.ArrayList;
import java.util.Iterator;
class ArrayListDemo
{
public static void main(String args[])
{
ArrayList <String> al=new ArrayList<String>();
al.add("peter");
al.add("laxmi");
al.add("nani");
al.add("nani");
al.add("khan");
Iterator<String> i=al.iterator();
        while(i.hasNext())
        {
        System.out.println(i.next());
        }

  }

}
```

Output:

```
peter
laxmi
nani
nani
khan
```

LinkedList :

- LinkedList implements the Collection interface.
- It uses a doubly linked list internally to store the elements.
- It can store the duplicate elements.

- It maintains the insertion order and is not synchronized.
- In LinkedList, the insertion and deletion is fast compare to arraylist because no shifting is required.

LinkedListDemo.java :

```
import java.util.LinkedList;
import java.util.Iterator;
class LinkedListDemo
{
public static void main(String args[])
{
LinkedList <String> ll=new LinkedList<String>();
ll.add("nina");
ll.add("nina");
ll.add("rama");
ll.add("krishna");

Iterator i=ll.iterator();

        while(i.hasNext())
        {
        System.out.println(i.next());
        }
}
}
```

Output :

```
javac LinkedListDemo.java
java LinkedListDemo
nina
nina
rama
krishna
```

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.
However, there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) insertion and deletion is slower than LinkedList because all the bits are shifted in memory.	Insertion and deletion is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) ArrayList is **better accessing of data**	LinkedList is **better for insertion and deletion of**

data.

Vector :

- Vector uses a dynamic array to store the data elements. It is similar to ArrayList.
- It implements list interface
- It allows duplicate values
- It maintains insertion order
- It is synchronized

VectorDemo.java :

```
import java.util.Vector;
import java.util.Iterator;
class VectorDemo
{
public static void main(String args[])
{

Vector <String> v=new Vector<String>();
v.add("rama");
v.add("sita");
v.add("rama");
v.add("laxman");

Iterator i=v.iterator();
        while(i.hasNext())
        {
        System.out.println(i.next());
        }

}
}
```

```
javac VectorDemo.java
java VectorDemo
rama
sita
rama
laxman
```

ArrayList and Vector both implements List interface and maintains insertion order.
However, there are many differences between ArrayList and Vector classes that are given below.

Difference between ArrayList and Vector :

ArrayList	**Vector**

1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.

Stack :

- The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack.
- The stack contains all of the methods of Vector class since Stack is sub class to Vector class
- It provides its methods like push(), peek(), which defines its properties.
- It is a synchronized implementation.

StackDemo.java :

```
import java.util.*;
class StackDemo
{
public static void main(String args[])
{
Stack <Integer>  s=new Stack <Integer>();
s.push(10);
s.push(10);
s.push(20);
s.push(30);
s.pop();
Iterator i=s.iterator();
        while(i.hasNext())
        {
        System.out.println(i.next());
        }
}
}
```

Output :

```
javac  StackDemo.java
java StackDemo
10
10
20
```

Set Interface :

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set Interface :
Set Interface in Java is present in java.util package. It extends the Collection interface. which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

HashSet :

- HashSet class implements Set Interface. It represents the collection that uses a hash table for storage.
- Hashing is used to store the elements in the HashSet.
- It contains unique items.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- It does not allow duplicate values
- We can insert at most one null value in HashSet
- **HashSet in Java is not** thread safe as it is **not synchronized** by default.

HashSetDemo.java :

```
import java.util.*;
class HashSetDemo
{
public static void main(String args[])
{

HashSet <String> hs=new HashSet <String>();
hs.add("rama");
hs.add("sita");
hs.add("rama");
hs.add("laxman");

Iterator i=hs.iterator();
        while(i.hasNext())
        {
        System.out.println(i.next());
        }

}
}
```

Output :

```
javac HashSetDemo.java
java HashSetDemo
rama
sita
laxman
```

LinkedHashSet :
LinkedHashSet class represents the hash table and LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface.

- Java LinkedHashSet class contains unique elements only like HashSet.
- Java LinkedHashSet class is non synchronized.
- Java LinkedHashSet class maintains insertion order.
- We can add atmost one null value in linkedhash set

LinkedHashSetDemo.java :

```
import java.util.LinkedHashSet;
import java.util.Iterator;
class LinkedHashSetDemo
{
public static void main(String args[])
{
LinkedHashSet <String> v=new LinkedHashSet<String>();
v.add("rama");
v.add("sita");
v.add("");
v.add("rama");
v.add("laxman");
Iterator<String> i=v.iterator();
        while(i.hasNext())
        {
        System.out.println(i.next());
        }
}
}
```

Output :

```
rama
sita
laxman
```

Sorted Set Interface :

- The Sorted Set interface present in java.util package extends the Set interface.
- This interface contains the methods inherited from the Set interface
- It stores all the elements in this interface to be stored in a sorted manner.

Tree Set Class :

TreeSet class implements the Sorted Set interface that uses a tree for storage.
Like HashSet, TreeSet also contains unique elements.
It can allow at most one null value
However, the access and retrieval time of TreeSet is quite fast.
The elements in TreeSet stored in ascending order.

TreeSetDemo.java :

```
import java.util.TreeSet;
import java.util.Iterator;
class TreeSetDemo
{
public static void main(String args[])
```

```
{
TreeSet <String> t=new TreeSet<String>();
t.add("rama");
t.add("laxman");
t.add("sita");
t.add("dasaradh");
t.add("ravan");
t.add("ravan");
t.add("");
t.add("");
Iterator<String> i=t.iterator();
while(i.hasNext())
{
System.out.println(i.next());
}
}
}
```

```
javac TreeSetDemo.java
java TreeSetDemo
dasaradh
laxman
rama
ravan
sita
```

Map Interface in java :

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

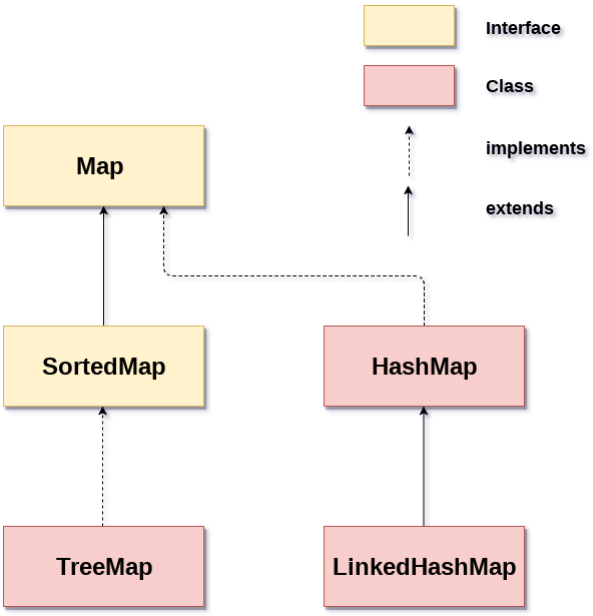
A Map doesn't allow duplicate keys, but you can have duplicate values.

A Map is useful if you have to search, update or delete elements on the basis of a key.

Some of the methods in map interface

put(Object key, Object value)	It is used to insert an entry in the map.
remove(Object key)	It is used to delete an entry for the specified key.
clear()	It is used to reset the map.

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap. The hierarchy of Java Map is given below:



Map.Entry interface

getKey()	It is used to obtain a key.
getValue()	It is used to obtain value.

SortedMap Interface:

SortedMap is an interface in the **collection framework**. This interface extends the **Map interface** and provides a total ordering of its elements (elements can be traversed in sorted order of keys). The class that implements this interface is **TreeMap**.

The main characteristic of a SortedMap is that it orders the keys by their natural ordering.

HashMap :

- Java **HashMap** class implements the Map interface which allows us *to store key and value pair*, where keys should be unique.
- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap is non-synchronized.
- Java HashMap maintains no order.

Some Methods of hash map

put(Object key,Object value)	It is used to insert an entry in the map.
clear()	It is used to remove all of the mappings from this map.
remove(Object key)	It is used to delete an entry for the specified key.
entrySet()	It is used to return a collection view of the mappings contained in this map.

LinkedHashMap :

Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface It inherits HashMap class and implements the Map interface.

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains unique elements.
- Java LinkedHashMap may have one null key and multiple null values.
- Java LinkedHashMap is non synchronized.
- Java LinkedHashMap maintains insertion order.

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

- Java TreeMap contains values based on the key.
- Java TreeMap contains only unique elements.
- Java TreeMap cannot have a null key but can have multiple null values.
- Java TreeMap is non synchronized.
- Java TreeMap maintains ascending order.

HashMapDemo.java

```
import java.util.*;
class HashMapDemo {

   public static void main(String args[])
   {

      // creating a hashmap
      HashMap<Integer, String> hm = new HashMap<Integer, String>();

      // putting elements
      hm.put(01, "rama");
      hm.put(03, "sita");
      hm.put(04, "laxman");
      hm.put(02, "daardh");

      System.out.println("Iterate over original HashMap");

      // printing hashmap
      for (Map.Entry<Integer, String> entry :
          hm.entrySet()) {
         System.out.println(entry.getKey() + " => "
                        + ": " + entry.getValue());
      }
   }
}
```

OutPut :

```
Iterate over original HashMap
01 => : rama
02 => : daardh
03 => : sita
```

04 => : laxman

Queue Interface :

- Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed.
- Queue is implemented by priority queue.

PriorityQueue :

- The PriorityQueue class implements the Queue interface.
- It holds the elements or objects which are to be processed by their priorities.

The example below explains the following basic operations of the priority queue.

- add(E element): This method inserts the specified element into this priority queue.
- peek(): This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
- poll(): This method retrieves and removes the head of this queue, or returns null if this queue is empty.

PriorityQueueDemo.java :

```
import java.util.PriorityQueue;
import java.util.Iterator;
class PriorityQueueDemo
{
public static void main(String args[])
{

PriorityQueue<String>    pq=new PriorityQueue <String>();
pq.add("gita");
pq.add("ravi");
pq.add("sita");
pq.add("nani");
Iterator<String> i=pq.iterator();
        while(i.hasNext())
        {
        System.out.println(i.next());
        }
}
}
```

Output

```
javac PriorityQueueDemo.java
java PriorityQueueDemo
gita
nani
sita
ravi
```

Deque Interface :

- Deque interface extends the Queue interface.
- In Deque, we can remove and add the elements from both the side.
- Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

ArrayDeque :

- ArrayDeque class implements the Deque interface.
- It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.
- ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.
- ArrayDeque is not thread safe it is non synchronized
- *ArrayDeque* **is doubles its size when it gets filled.**

Methods of Java Deque Interface

Method	Description
add(object)	It is used to insert the specified element into this deque and return true upon success.
remove()	It is used to retrieves and removes the head of this deque.
poll()	It is used to retrieves and removes the head of this deque, or returns null if this deque is empty.
element()	It is used to retrieves, but does not remove, the head of this deque.
peek()	It is used to retrieves, but does not remove, the head of this deque, or returns null if this deque is empty.

addFirst():This method add the first element from a Deque

addLast():This method add the last element from a Deque

removeFirst():This method removes the first element from a Deque
removeLast()This method removes the last element from a Deque

ArrayDequeDemo.java :

```
import java.util.ArrayDeque;
import java.util.Iterator;
class ArrayDequeDemo
{
public static void main(String args[])
{

ArrayDeque <String> ad=new ArrayDeque <String>();
ad.add("rama");
ad.add("sita");
ad.add("rama");
ad.add("laxman");
```

```
ad.addLast("peter");
ad.addFirst("peter");

Iterator i=ad.iterator();
        while(i.hasNext())
        {
        System.out.println(i.next());
        }

}
}
```

Output :

```
javac ArrayDequeDemo.java
java ArrayDequeDemo
java ArrayDequeDemo
peter
rama
sita
rama
laxman
peter
```

Iterators in Collection

- ForEach
- Iterator
- Listiterator
- Enumeration

ForEach loop:

for-each loop can be used for traversing the elements of a collection. But this can only be used if we don't want to modify the contents of a collection and we don't want any **reverse** access.

Syntax

The syntax of Java for-each loop consists of data_type with the variable followed by a colon (:), then array or collection.

```
1. for(data_type variable : array | collection){
2. //body of for-each loop
3. }
```

ForEachDemo.java :

```
import java.util.LinkedList;
class ForEachDemo
{
public static void main(String args[])
{
LinkedList <String>ll=new LinkedList<String>();
ll.add("rama");
```

```
ll.add("sita");
ll.add("laxman");
ll.add("ravana");

        for(String str:ll)
        {
        System.out.println(" "+str);
        }
}
}
```

Output :

```
 rama
 sita
 laxman
 ravana
```

Enumeration: The Enumeration interface defines the methods by which you can enumerate the elements in a collection object.

Sr.No.	Method & Description
1	**hasMoreElements( )** it must return true while there are still more elements to extract, and false when all the elements have been enumerated.
2	**nextElement( )** This returns the next object in the enumeration object.

EnumerationDemo.java :

```
import java.util.Enumeration;
import java.util.Vector;
class EnumerationDemo
{
public static void main(String args[])
{
Vector <String>v=new Vector<String>();
v.add("rama");
v.add("sita");
v.add("laxman");

Enumeration e=v.elements();
while(e.hasMoreElements())
{
System.out.println(e.nextElement());

}
}

}
```

Output:
rama
sita
laxman

Iterator interface :
Iterator interface provides the facility of iterating the elements in a forward direction only.
Methods of Iterator interface
There are only three methods in the Iterator interface. They are:

No.	Method	Description
1	public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is less used.

IteratorDemo.java :
//displaying even numbers in the vector collection object

```
import java.util.Iterator;
import java.util.Vector;
class IteratorDemo
{
public static void main(String args[])
{
Vector <Integer>v=new Vector<Integer>();
v.add(10);
v.add(12);
v.add(13);
Iterator i=v.iterator();
while(i.hasNext())
{
Integer n=(Integer)i.next();
if(n%2==0)
System.out.println(n);
else
i.remove();
}
}
}
```

Output:
10
12
ListIterator : It is a iterator which is used to traverse all types of lists such as ArrayList, Vector, LinkedList, Stack etc.
ListIterator extends iterator interface

ListIterator is a bi-directional iterator. For this functionality, it has two kinds of methods:

1. Forward direction iteration :

- **hasNext():** This method returns true when the list has more elements to traverse while traversing in the forward direction
- **next():** This method returns the next element of the list and advances the position of the cursor.

2. Backward direction iteration :

- **hasPrevious():** This method returns true when the list has more elements to traverse while traversing in the reverse direction
- **previous():** This method returns the previous element of the list and shifts the cursor one position backwards.

ListIteratorDemo.java :

```
import java.util.ListIterator;
import java.util.Vector;
class ListIteratorDemo
{
public static void main(String args[])
{
Vector <Integer>v=new Vector<Integer>();
v.add(10);
v.add(12);
v.add(13);
v.add(20);
ListIterator l=v.listIterator();
System.out.println("accessing elements in forward direction");
        while(l.hasNext())
        {
        System.out.println(l.next());
        }
System.out.println("accessing elements in backward direction");
        while(l.hasPrevious())
        {
        System.out.println(l.previous());
        }

}

}
```

Output :

```
accessing elements in forward direction
10
12
13
20
```

accessing elements in backward direction
20
13
12
10

Comparable in java :
Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.
public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

Collections class :
Collections class provides static methods for sorting the elements of collections. If collection elements are of Set or Map, we can use TreeSet or TreeMap. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.
Method of Collections class for sorting List elements :
public void sort(List list): It is used to sort the elements of List. List elements must be of the Comparable type.

Student.java :

```
import java.util.Collections;
import java.util.ArrayList;
import java.lang.Comparable;
class Student implements Comparable<Student>
{
int age;
String name;
int marks;
Student(String name,int age,int marks)
{
this.age=age;
this.name=name;
this.marks=marks;

}
public int compareTo(Student s)
{

if(this.age>s.age)return 1;//code for  age, sorting integers
```

```
else if(this.age<s.age)return -1;
return 0;

//return this.name.compareTo(s.name);// code   for names sorting string objects
}
public static void main(String args[])
{
ArrayList<Student>al=new ArrayList<Student>();
Student s1=new Student("dhoni",37,100);
Student s2=new Student("pant",23,70);
Student s3=new Student("jadeja",31,80);
Student s4=new Student("bumrah",27,0);
al.add(s1);
al.add(s2);
al.add(s3);
al.add(s4);
Collections.sort(al);
for(Student st:al)
{
System.out.println(st.age+","+st.name);
}
}
}
```

```
javac Student.java
Java Student
23,pant
27,bumrah
31,jadeja
37,dhoni
```

Comparator in java :

Java Comparator interface is used to order the objects of a user-defined class.

This interface is found in java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member of a class, for example, rollno, name, age or anything else.

Methods of Java Comparator Interface :

Method	Description
public int compare(Object obj1, Object obj2)	It compares the first object with the second object.
public boolean equals(Object obj)	It is used to compare the current object with the specified object.

Collections class :

Collections **class provides static methods for sorting the elements of a collection. If collection elements are of Set or Map, we can use TreeSet or TreeMap. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements also.**

Method of Collections class for sorting List elements

public void sort(List list, Comparator c): is used to sort the elements of List by the given Comparator.

ComparatorMain.java :

```
import java.util.Collections;
import java.util.ArrayList;
import java.util.Comparator;

class Student
{
int age;
String name;
int marks;
        Student(int age,String name,int marks)
        {
        this.age=age;
        this.name=name;
        this.marks=marks;
        }

}

class AgeComparator implements Comparator
{
        public int compare(Object o1,Object o2)
        {
        Student s1=(Student)o1;
        Student s2=(Student)o2;
        if(s1.age>s2.age) return 1;
        else if(s1.age<s2.age) return -1;
        else return 0;
        }
}

class NameComparator implements Comparator
{
        public int compare(Object o1,Object o2)
        {
        Student s1=(Student)o1;
        Student s2=(Student)o2;

        return s1.name.compareTo(s2.name);
```

```
        }
}


public class ComparatorMain
{
public static void main(String args[])
{
ArrayList <Student>al=new ArrayList<Student>();

al.add(new Student(21,"mahesh",78));
al.add(new Student(31,"giresh",88));
al.add(new Student(29,"saho",68));
al.add(new Student(21,"pawan",100));
al.add(new Student(32,"hemesh",98));

Collections.sort(al,new AgeComparator());
System.out.println("sort by age");
for(Student s:al)
{
System.out.println(s.age+","+s.name+","+s.marks);

}
System.out.println("Sort by name");
Collections.sort(al,new NameComparator());
for(Student s:al)
{
System.out.println(s.age+","+s.name+","+s.marks);

}
}
}
```

Difference between Comparable and Comparator :

Comparable and Comparator both are interfaces and can be used to sort collection elements. However, there are many differences between Comparable and Comparator interfaces that are given below.

Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element of a class such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements of a class such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.

4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

List the examples for Set, List in java. Can we add set to list in Java?

Yes we can add set to list in java

The **addAll()** method of **java.util.Collections** class is used to add all of the specified elements to the specified collection. Elements to be added may be specified individually or as an array.

Tester.java :

```
import java.util.*;
public class Tester{
 public static void main(String[] args){
 ArrayList<String> al=new ArrayList<String>();
 al.add("peter");
 al.add("prakash");
 HashSet<String> hs=new HashSet<String>();
 hs.add("karthik");
 hs.add("yugandhar");
 al.addAll(hs);
 Iterator itr=al.iterator();
 while(itr.hasNext())
  System.out.println(itr.next());
}
}
```

Output

peter
prakash
yugandhar
karthik

In Java 1.4, both the String and the StringBuffer classes implements java.lang.CharSequence interface, which is a standard interface for querying the length of and extracting characters and subsequences from a readable sequence of characters. It can be explained with the help of example given below:

```
class A
{
public void dumpSeq(CharSequence cs)
{
System.out.println("length = " + cs.length());
System.out.println("first char = " + cs.charAt(0));
System.out.println("string = " + cs);

}
}
```

```
class program {
public static void main(String args[])
{
String s = "test";
A a=new A();
a.dumpSeq(s);
}
}
```

Output :

length = 4
first char = t
string = test

Diff between for and for each :

for Loop vs foreach Loop	
The for loop is a control structure for specifying iteration that allows code to be repeatedly executed.	The foreach loop is a control structure for traversing items in an array or a collection.
Element Retrieving	
A for loop can be used to retrieve a particular set of elements.	The foreach loop cannot be used to retrieve a particular set of elements.
Readability	
The for loop is harder to read and write than the foreach loop.	The foreach loop is easier to read and write than the for loop.
Usage	
The for loop is used as a general purpose loop.	The foreach loop is used for arrays and collections.

Foreach Syntax :

The syntax of Java for-each loop consists of data_type with the variable followed by a colon (:), then array or collection.

```
for(data_type variable : array | collection){
//body of for-each loop  }
```

Forloop Syntax:

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```

What Scanner class and write a program to read various data kinds of data?

Scanner Class in java :

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

How to get Java Scanner :

To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

```
Scanner in = new Scanner(System.in);
```

1)**double nextDouble():**It scans the next token of the input as a double.
2)**float nextFloat():**It scans the next token of the input as a float.
2)**int nextInt():**It scans the next token of the input as an Int.

3)**String nextLine():**It is used to get the input string that was skipped of the Scanner object.
4)**long nextLong():**It scans the next token of the input as a long.
5)**short nextShort():**It scans the next token of the input as a short.

```
import java.util.*;
public class ScannerExample {
public static void main(String args[]){
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        System.out.println("Name is: " + name);
        in.close();
        }
}
```

Output:

```
Enter your name:vicky
Name is: vicky
```

Discuss java.util.* package :
Answer: Explain about Scanner class, StringTokenizer and few Collection Hierarchy Classes and interfaces

Illustrate Java Program to Add Characters to a String.
Adding Character to String can be done in following ways
1.Using + operator
i.At the end
ii. At the beginning
2.Using substring() method

```
// Java Program to Add Characters to a String
// At the End
// Main class
public class GFG {

    // Main driver method
    public static void main(String args[])
    {

        // Input character and string
        char a = 's';
        String str = "ComputerScience";

        // Inserting at the end
        String str2 = str + a;
```

```
        // Print and display the above string
        System.out.println(str2);
    }
}
```

Output :

ComputerSciences

UNIT-4

IO Streams: IO Streams is a part of java library.IO Streams deals with all predefined classes present in java.io package. The classes present in java.io package together are called IO Streams.

Stream:

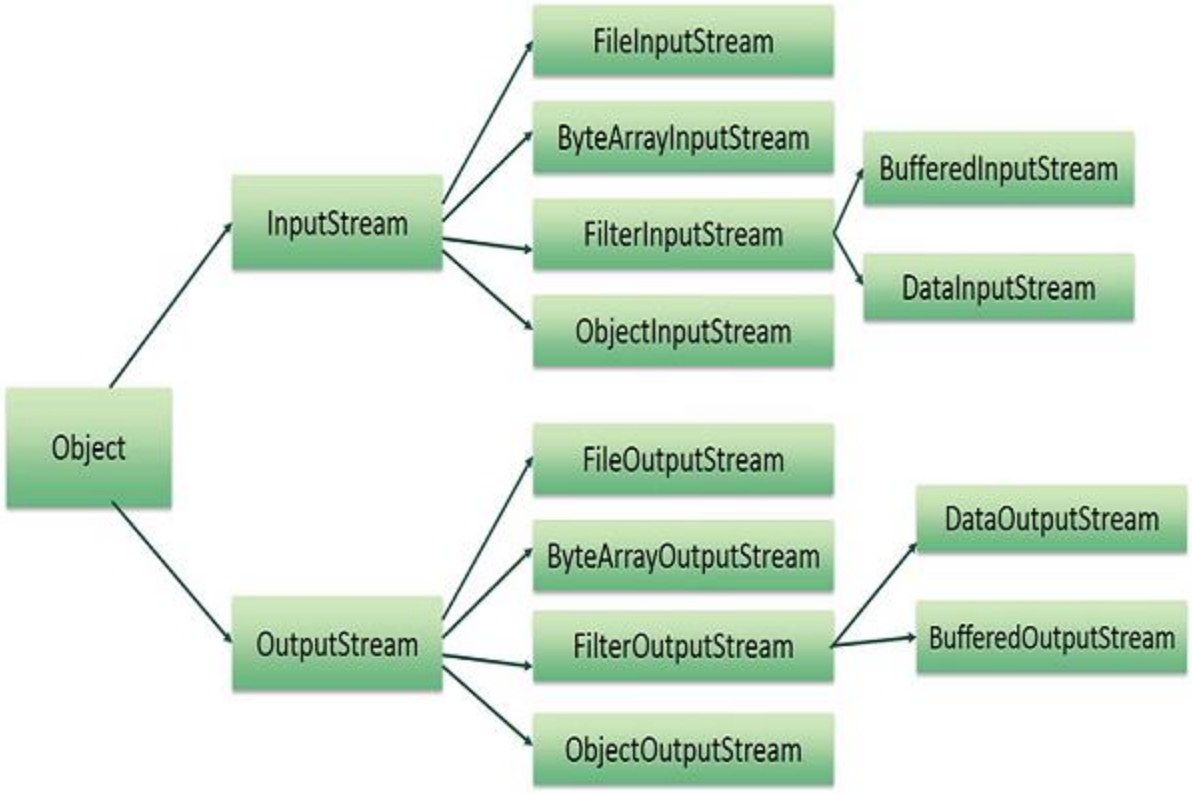
Stream is a flow of data or sequence of data that carries from location to another location.

A stream is a logical connection between java program and a file

There are two kinds of Streams −

- **InPutStream** − The InputStream is used to read data from a source.
- **OutPutStream** − The OutputStream is used for writing data to a destination.

The different input and output stream classes in java



Different Input Stream classes

InputStream : This class is used to read the data. It is the super class for all the InputStream classes.

FileInputStream: This class is used to read the data from the file

ByteArrayInputStream: This class is used to read the data from the ByteArray

FilterInputStream: This class implements the InputStream. It contains different sub classes as BufferedInputStream, DataInputStream

DataInputStream: This class is used to read primitive data

BufferInputStream: This class is used to read data from the Buffer.

Different Output Stream classes

OutputStream: This class is used to write the data. It is the super class for all the InputStream classes.

File OutputStream: This class is used to write the data from the file

ByteArrayOutputStream: This class is used to write the data from the ByteArray

FilterOutputStream: This class implements the InputStream. It contains different sub classes as BufferedOutputStream, DataOutputStream

DataOutputStream: This class is used to read primitive data

BufferOutputStream: This class is used to read data from the Buffer.

Two forms of data based on type of data in the Streams :

1.**Character Streams**: The Stream that allows us to send data in the Character format is Character Stream.

Character streams: It carries the data in the form of character stream

It can read and write 2 bytes i.e 16bits of data. In other words, it processes data character by character.

Character streams are available from 1.1

2.**Byte Streams:** The Stream that allows us to send data in the byte format is byte stream.

It can read and write 1 byte i.e 8 bit of data. In other words, it processes data byte by byte.

In other words, we can say that **ByteStream** classes read/write the data of 8-bits. We can store video, audio, characters, etc., by using **ByteStream** classes.

Byte stream are available in java 1.0

Byte Stream: It carries the data in the form of byte stream

ByteStream: Data is in the form of Bytes .
Api of ByteStreams ends with Stream

Some of the examples of bytestreams

FileInputStream:Reads the data of file in the form of Streams
FileOutputStream: Write's the data of file in the form of Streams
BufferedInputStream:Reads the data in the form of Bytes and stores the data in Buffer
BufferedOutputStream: Writes the data in the form of Bytes and stores the data in Buffer

Elaborate read and write operations of File in JAVA?

CopyFileusingByteStreams.java

```
import java.io.*;
class CopyFileusingByteStreams {

  public static void main(String args[])  {

    try {
      FileInputStream in = new FileInputStream("\\E:input.txt");
```

```
      FileOutputStream  out = new FileOutputStream("\\E:output.txt");

        int c;
        while ((c = in.read()) != -1) {
          out.write(c);
        }
          out.close();
          in.close();
          System.out.println("file copy successfull");

      }
catch(Exception e){
     System.out.println(e);}
       }
  }
```

```
javac CopyFileusingByteStreams.java
java CopyFileusingByteStreams

file copy successfull
```

Character Sreams: Data is in the form of Characters
API of Character Stream ends with Readers and Writer
Some of the examples of Character streams

FileReader:Reads the data from a file
FileWriter:Writes the data from the file
BufferedReader:reads the data and stores in the buffer
BufferedWriter:write the data in to a buffer

CopyFileusingCharacterStreams.java

```
import java.io.*;
class CopyFileusingCharacterStreams {

   public static void main(String args[])   {

      try {
        FileReader in = new FileReader("\\E:input.txt");
       FileWriter  out = new FileWriter("\\E:output.txt");

        int c;
        while ((c = in.read()) != -1) {
          out.write(c);
        }
          out.close();
          in.close();
          System.out.println("file copy successfull");
```

```
      }
catch(Exception e){
     System.out.println(e);}
       }
  }
```

javac CopyFileusingCharacterStreams.java

java CopyFileusingCharacterStreams

file copy successfull

Byte Streams	Character Streams
The Stream that allows us to send data in the byte format is byte stream.	The Stream that allows us to send data in the chracter format is chracter stream.
It can read and write 1 byte i.e 8 bit of data. In other words, it processes data byte by byte.	It can read and write 2 byte i.e 16 bit of data. In other words, it processes data chracter by character.
Byte stream are available in java 1.0	Character stream are available in java 1.1
We can store video, audio, characters, etc., by using **ByteStream** classes.	We can store text data by using **CharacterStream** classes.
FileInputStream,FileOutputStreamare examples	FileReader,FileWriter classes are examples

Exception: An exception is an unexpected event that occurred during the execution of a program, and disrupts the normal flow of instructions.

Exception Handling

Exception handling is a technique to convert system generated errors in to user friendly messages.

There are two types of exceptions in java

1.**checked exceptions:**

These exceptions are checked at compile time,

Compile time errors occurs when the java programmer not followed syntaxes

Examples: IOException, SQLException

2. **unchecked exceptions:**

These exceptions checked at run time

Run time error occurs when user enters invalid input

Run time errors are called exceptions

Examples:

- ArithmeticException
- ArrayIndexOutOfBoundsException
- NullPointerException
- NumberFormatException
- StringIndexOutOfBoundsException
- FileNotFoundException

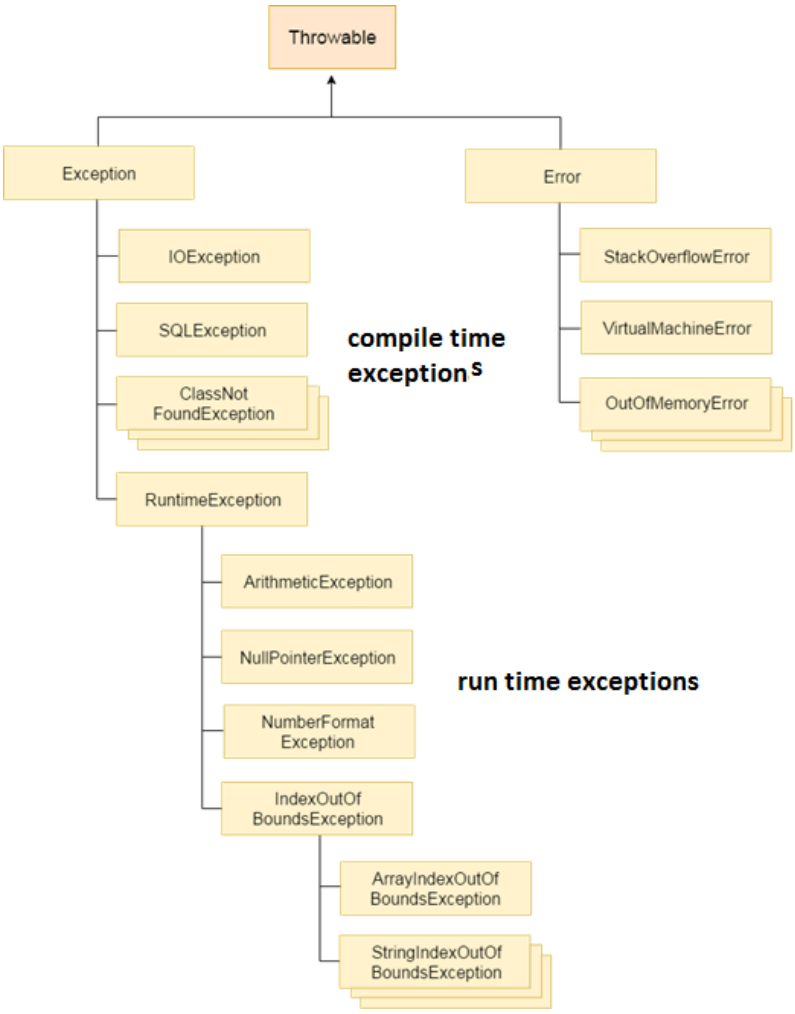
Checked Exception	Unchecked Exception
Checked exceptions occur at compile time.	Unchecked exceptions occur at runtime.
The compiler checks a checked exception.	The compiler does not check these types of exceptions.
These types of exceptions can be handled at the time of compilation.	These types of exceptions cannot be a catch or handle at the time of compilation these are run time exceptions
They are the sub-class of the exception class.	These are the subclasses of RuntimeException class
Examples of Checked exceptions:	Examples of Unchecked Exceptions:

- File Not Found Exception
- No Such Field Exception
- Interrupted Exception
- No Such Method Exception
- Class Not Found Exception
- No Such Element Exception
- Undeclared Throwable Exception
- Empty Stack Exception
- Arithmetic Exception
- Null Pointer Exception
- Array Index Out of Bounds Exception
- Security Exception

ExceptionHandling Heirarchy Chart

Throwable class

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Scanner Class in java

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

How to get Java Scanner

To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

```
Scanner in = new Scanner(System.in);
```

1)**double nextDouble():**It scans the next token of the input as a double.

2)**float nextFloat():**It scans the next token of the input as a float.

2)**int nextInt():**It scans the next token of the input as an Int.

3)**String nextLine():**It is used to get the input string that was skipped of the Scanner object.

4)**long nextLong():**It scans the next token of the input as a long.

5)**short nextShort():**It scans the next token of the input as a short.

```
import java.util.*;

public class ScannerExample {

public static void main(String args[]){

        Scanner in = new Scanner(System.in);

        System.out.print("Enter your name: ");

        String name = in.nextLine();

        System.out.println("Name is: " + name);

        in.close();

        }

}
```

output

```
Enter your name:vicky
```

```
Name is: vicky
```

(B)Contrast Error and Exception. Check with JVM error and ArrayIndexOutOfBounds Exception.

Difference between Error and Exception

Sr. No.	Error	Exception
1	Error is Classified as an unchecked type	Exception is Classified as checked and unchecked
2	It belongs to java.lang.Error	It belongs to java.lang.Exception
3	Error cannot be handled It is irrecoverable	Exception can be handled gy using exception handling concepts.It is recoverable
4	It occurs at runtime	It can occur at run time and compile time
5	OutOfMemoryError ,IOError	NullPointerException , SqlException

Check with JVM Error and Arrayindexoutofboundexception

class ArrayIndexOutOfBound_Demo {

```
public static void main(String args[])
	{
		try {
			int a[] = new int[5];
			a[6] = 9; // accessing 7th element in an array of
			// size 5
		}
		catch (ArrayIndexOutOfBoundsException e) {
			System.out.println("Array Index is Out Of Bounds");
		}
		catch (Error e) {
			System.out.println("JVM Error");
		}
	}
}
```

There are two types of exceptions in java related to libraries

1.built in exception/pre-defined exception

2. user defined exception/custom defined exception

Built-in exceptions are pre-defined exceptions which are available in Java libraries
Examples

- ArithmeticException
- ArrayIndexOutOfBoundsException
- NullPointerException
- NumberFormatException
- StringIndexOutOfBoundsException

User defined(Custom exception)

exception are these are defined by the programmer

Example

Invalid salary of employee

Invalid age of human beign

To display user friendly messages instead of system generated messages

Exception Handlers

We need the following five keywords

1.try

2.catch

3.finally

4.throw

5.throws

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
Try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
Catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to close the files etc
Throw	The "throw" keyword is used to throw an custom defined exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Built in exceptions examples
Arithmetic exception :It is thrown when an exceptional condition has occurred in an arithmetic operation
// Java program to demonstrate
// ArithmeticException

```
class ArithmeticException_Demo {
public static void main(String args[])
	{
		try {
			int a = 30, b = 0;
			int c = a / b; // cannot divide by zero
			System.out.println("Result = " + c);
		}
		catch (ArithmeticException e) {
			System.out.println("Can't divide a number by 0");
		}
	}
}
```

Output
Can't divide a number by 0

ArrayIndexOutOfBoundsException :It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

```
// Java program to demonstrate
// ArrayIndexOutOfBoundException
class ArrayIndexOutOfBound_Demo {
public static void main(String args[])
	{
		try {
			int a[] = new int[5];
			a[6] = 9; // accessing 7th element in an array of
			// size 5
		}
		catch (ArrayIndexOutOfBoundsException e) {
			System.out.println("Array Index is Out Of Bounds");
		}
	}
}
```

Output
Array Index is Out Of Bounds

FileNotFoundException : This Exception is raised when a file is not accessible or does not open.

```
// Java program to demonstrate
// FileNotFoundException
import java.io.*;
class File_notFound_Demo {

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

```
		try {


			FileReader fr = new FileReader(“E:\\cse.txt”);
		}
		catch (FileNotFoundException e) {
			System.out.println("File does not exist");
		}
	}
}
```

Output

File does not exist

NullPointerException :This exception is raised when referring to the members of a null object. Null represents nothing

```
// Java program to demonstrate NullPointerException
class NullPointer_Demo {
public static void main(String args[])
	{
		try {
			String a = null; // null value
			System.out.println(a.charAt(0));
		}
		catch (NullPointerException e) {
			System.out.println("NullPointerException..");
		}
	}
}
```

NullPointerException..

NumberFormatException :This exception is raised when a method could not convert a string into a numeric format.

```
// Java program to demonstrate
// NumberFormatException
class NumberFormat_Demo {
public static void main(String args[])
	{
		try {
			// "akki" is not a number
			int num = Integer.parseInt("akki");

			System.out.println(num);
		}
		catch (NumberFormatException e) {
			System.out.println("Number format exception");
		}
```

```
        }
}
```

StringIndexOutOfBoundsException : It is thrown by String class methods to indicate that an index is less than the size of the string.

```
class StringIndexOutOfBound_Demo {
public static void main(String args[])
   {
      try{
         String a = "This is like chipping "; // length is 22
         char c = a.charAt(24); // accessing 25th element
         System.out.println(c);
      }
      catch(StringIndexOutOfBoundsException e) {
         System.out.println("StringIndexOutOfBoundsException");
      }
   }
}
```

Output

StringIndexOutOfBoundsException

Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

MultipleCatchBlock1.java

```
   public class MultipleCatchBlock1 {


      public static void main(String[] args) {


            try{

                int a[]=new int[5];

                a[5]=30/0;

               }

               catch(ArithmeticException e)
```

```
            {
             System.out.println("Arithmetic Exception occurs");
            }
           catch(ArrayIndexOutOfBoundsException e)
            {
             System.out.println("ArrayIndexOutOfBounds Exception occurs");
            }
           catch(Exception e)
            {
             System.out.println("Parent Exception occurs");
            }
finally{}
           System.out.println("rest of the code");
    }
  }
```

Explain about finally exception handler with java code.

Finally block

Th finally block in java is used to put important codes such as clean up code e.g. closing the file or closing the connection. The finally block executes whether exception rise or not and whether exception handled or not. A finally contains all the crucial statements regardless of the exception occurs or not.

```
// Java program to demonstrate

// finally block in java When
// exception does not rise

import java.io.*;

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

```
            System.out.println("inside try block");

            // Not throw any exception
            System.out.println(34 / 2);
        }

        // Not execute in this case
        catch (ArithmeticException e) {

            System.out.println("Arithmetic Exception");

        }
        // Always execute
        finally {

            System.out.println(
                "finally : i execute always.");

        }
    }
}
```

Throw Keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

Syntax:

throw objectName;

or

throw new ClassName()

ValidAge.java

```
class ValidAge
{
public static void main(String args[])
{
int voterAge=17;
if(voterAge<18)

throw new ArithmeticException("Not Valid Age");
else
System.out.println("valid voter age");
```

```
}
}
```
javac ValidAge.java

java ValidAge

Exception in thread "main" java.lang.ArithmeticException: Not Valid Age

at ValidAge.main(ValidAge.java:11)

Throws Keyword: The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception

Throws keyword is used along with the method signature

Syntax:

Returntype methodname(arg1,arg2,arg3,arg4…..) throws Exception1,Exception2,,,,,,,,,

{

}

Example

void fileRead()throws FileNotFoundException,IOException

{

}

ThrowsExample.java

```
import java.io.*;
class ThrowsExample
{
public static void main(String args[])throws FileNotFoundException,IOException

{
FileReader fr=new FileReader("E:\\cse.txt");
FileWriter fw=new FileWriter("E:\\ece.txt");
int ch;
```

```
while((ch=fr.read())!=-1)
{
fw.write((char)ch);
}
fw.close();
fr.close();
System.out.println("FileReading and Writing Successfull");
}
}
```

Output

javac ThrowsExample.java

java ThrowsExample

FileReading and Writing Successfull

Difference between throw and throws in Java

	Throw	Throws
1	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2	Throw is followed by an instance.	Throws is followed by class.
3	Throw is used within the method.	Throws is used with the method signature.
4	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.
5	At a time we can throw one object at a time	At a time we can declare any no of exceptions using Throws keyword
6	throw objectname; (or) throw new Classname()	return_type method_name() throws exception_class_name{ //method code }

User defined exceptions

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, user can also create exceptions which are called ‘user-defined Exceptions’.

How to create user defined exception:

- Step 1:The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:

  classMyException extends Exception

- Step 2:We can write a default constructor in his own exception class.

  MyException(){}

We can also create a parameterized constructor with a string as a parameter.
We can use this to store exception details.

Step 3:We can call super class(Exception) constructor from this and send the string there.

MyException(String str)

{

super(str);

}

Step 4:To raise exception of user-defined type, we need to create an object to this exception class and throw it using throw clause, as:

MyException me = new MyException(“Exception details”);

throw me;

User defined exception example

VoterAgeLessException.java

```
import java.util.Scanner;
class VoterAgeLessException extends Exception
{
VoterAgeLessException(String str)
{
super(str);
}

public static void main(String args[])throws VoterAgeLessException
{
Scanner sc=new Scanner(System.in);
System.out.println("enter voter age");
try{
int age=sc.nextInt();
        if(age<18)
        {
        throw new VoterAgeLessException("voter age less");
        }
        else
        System.out.println("valid age");
}
catch(ArithmeticException a)
{
System.out.println(a);
}
finally
{

}
}
}
```

Output-1

```
javac VoterAgeLessException.java
java VoterAgeLessException
enter voter age
12
Exception in thread "main" VoterAgeLessException: voter age less
      at VoterAgeLessException.main(VoterAgeLessException.java:17) Output-2
```

Output2

```
javac VoterAgeLessException.java
```

```
java VoterAgeLessException

enter voter age
18
valid age
```

ExceptionPropagation:

when an exception happens, Propagation is a process in which the exception is being dropped from to the top to the bottom of the stack. If not caught once, the exception again drops down to the previous method and so on until it gets caught or until it reach the very bottom of the call stack. This is called exception propagation and this happens in case of Unchecked Exceptions.

How to propagate an unchecked exception

ExceptionPropagation.java

```
class ExceptionPropagation
{
	public void method1()
	{
	int a=30/0;
	System.out.println(a);
	}

	public void method2()
	{
	method1();
	}

	public void method3()
	{

	try{
	method2();
	}
	catch(ArithmeticException ae)
	{
	System.out.println("Arithmetic Exception occured");
	}

	}

public static void main(String args[])
{
ExceptionPropagation ep=new ExceptionPropagation();
ep.method3();
}
```

}

What is the difference between ClassNotFoundException and NoClassDefFoundError with suitable example.

ClassNotFoundException occurs when you try to load a class at runtime using **Class.forName()** methods and requested classes are not found in classpath. Most of the time this exception will occur when you try to run application without updating classpath with JAR files.

```
// Java program to illustrate
// ClassNotFoundException
public class Example {

    public static void main(String args[]) {
        try
        {
            Class.forName("GeeksForGeeks");
        }
        catch (ClassNotFoundException ex)
        {
            System.out.println("Class not found exception");
        }
    }
}
```

Output

```
java.lang.ClassNotFoundException: GeeksForGeeks
```

NoClassDefFoundError occurs when class was present during compile time and program was compiled and linked successfully but class was **not** present during runtime.

```
// Java program to illustrate
// NoClassDefFoundError
class GeeksForGeeks
{
    void greeting()
    {
        System.out.println("hello!");
    }
}

class G4G {
    public static void main(String args[])
    {
        GeeksForGeeks geeks = new geeksForGeeks();
        geeks.greeting();
    }
}
```

Above program will be successfully compiled and generate two classes GeeksForGeeks.class and G4G.class .

Now remove GeeksForGeeks.class file and run G4G.class.
At Java runtime **NoClassDefFoundError** will be thrown.

Serialization and Deserialization in java

Serialization is a mechanism of converting the state of an object into a byte stream.
The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

To make a Java object serializable we implement the **java.io.Serializable** interface.
The ObjectOutputStream class contains **writeObject()** method for serializing an Object.

```
public final void writeObject(Object obj)
                        throws IOException
```

Advantages of Serialization

1. To save/persist state of an object permenantly

Only the objects of those classes can be serialized which are implementing **java.io.Serializable** interface. Serializable is a **marker interface** (has no data member and method). It is used to "mark" java classes so that objects of these classes may get certain capability.

SerializableEx.java

```
import java.io.*;
class Student implements Serializable
{
private String sname;
 private int sno;
public void setName(String sname)
{
this.sname=sname;
}
public void setNo(int sno)
{
this.sno=sno;
}
public String getName()
{
return this.sname;
}
public int getSno()
{
return this.sno;
}

}
class SerializableEx
```

```
{
public static void main(String args[])
{
Student s=new Student();
s.setName("peter");
s.setNo(100);
try{
FileOutputStream file=new FileOutputStream(\\E:Student.ser, true);
ObjectOutputStream out=new ObjectOutputStream(file);
out.writeObject(s);
out.close();
file.close();
System.out.println("object serialized in student.ser");
}
catch(IOException io)
{
System.out.println(io);
}

}

}
```

```
javac SerializableEx.java
java SerializableEx
object serialized in student.ser
```

Deserialization in java

Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

The ObjectInputStream class contains **readObject()** method for deserializing an object.

```
public final Object readObject()
                  throws IOException,
               ClassNotFoundException
```

DeserializationEx.java

```
import java.io.*;
class DeserializableEx
```

```
{

public static void main(String args[])
{
Student s=null;
try{
FileInputStream file=new FileInputStream("\\E:Student.ser");
ObjectInputStream in=new ObjectInputStream(file);
s=(Student)in.readObject();
System.out.println("Student name=>"+s.getName());
System.out.println("Student no=>"+s.getSno());
in.close();
file.close();
}
catch(IOException io)
{
}
catch(ClassNotFoundException io)
{
}

}

}
javac DeserializationEx.java
java DeserializationEx
Student name=>peter
Student no=>100
```

Java Nested try block

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, the **inner try block** can be used to handle **ArrayIndexOutOfBoundsException** while the **outer try block** can handle the **ArithemeticException** (division by zero).

Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

NestedTryBlock.java

```
public class NestedTryBlock {

    public static void main(String args[])
    {
        // outer (main) try block
        try {


            // inner try block 2
            try {
                int arr[] = { 1, 2, 3, 4 };

                //printing the array element out of its bounds
                System.out.println(arr[10]);
            }

            // to handles ArithmeticException
            catch (ArithmeticException e) {
                System.out.println("Arithmetic exception");
                System.out.println(" inner try block 2");
            }
        }
        // to handle ArrayIndexOutOfBoundsException
        catch (ArrayIndexOutOfBoundsException e4) {
            System.out.print(e4);
            System.out.println(" outer (main) try block");
        }


        }

    }
```

Output java.lang.ArrayIndexOutOfBoundsException: 10 outer (main) try block

UNIT-5

Multi threading Introduction :

Multi taking:

Executing several tasks simultaneously is called multi-tasking

Two types of multi-tasking:

- **Process based multi-tasking:** Executing several tasks simultaneously where each task is separate independent process known as **Multiprocessing**.
  Example: typing a java program in editor, listening songs, downloading songs all these tasks done simultaneously each of this tasks having separate process
- **Thread based multi-tasking:** Executing several tasks simultaneously where each task is separate independent thread of a same program known as **Multithreading**. Thread based multi- tasking is related to programming
- Threads share the same address space.
- A thread is lightweight because they use less resource
- Cost of communication between the thread is low.

Threads are light weight because :

Normally a user thread, that can share same address space and resources with other threads, reducing context switching time during execution.

Difference between thread and process :

Thread	Process
Thread is a part of process or a part of a program	The Program under execution is process
Threads are **light weight process** because they uses less resources to communicate	process is **heavy weight process** because they uses more resources to communicate
Multiple threads shares the same address space of process	Process has its own address spaces
Context switching of thread is low cost	Context switching of process is costly
Inter thread communication is cost is low	Inter process communication is costly
Thread based multi-tasking is multi-threading	Process based multi-tasking is multi processing

Explain thread life cycle?

Thread Life Cycle:

A thread in Java at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:

Different States of Thread :

1. New
2. runnable
3. running
4. Blocked(non-runnable)
5. Terminate

1) New :

The thread is in new state if you create an instance of Thread class but before the invocation of start() method. When you call a start method it implicitly call run() method.

2) Runnable :

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running :

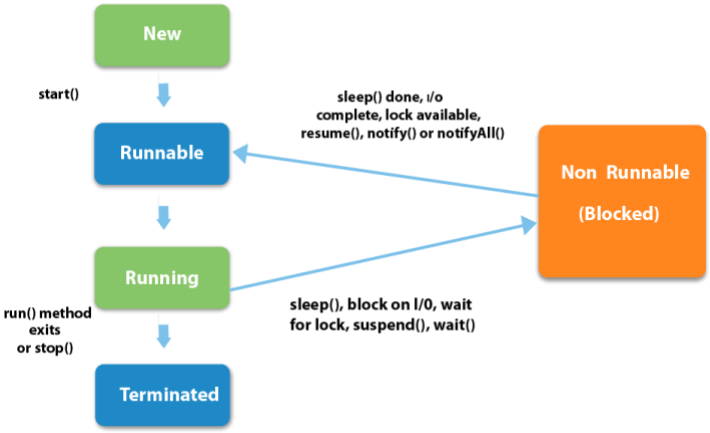
The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked) :

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated :

A thread is in terminated or dead state when its run() method exits.



Java provides a thread class that has various method calls in order to manage the behaviour of threads.

Thread class methods:

start(): Causes this thread to begin execution;

run(): When the **run**() **method** calls, the code specified in the **run**() **method** is executed. You can call the **run**() **method** multiple times.

sleep(long millis): Causes the currently executing thread to suspended for the specified number of milliseconds. Sleep method is a static method so it can be called by static method

setPriority(int newPriority): this method is used to set priority and Changes the current priority of the thread

1 to 10

1 MINIMUM_PROIRITY

10 MAXIMUM_PRORITY

5 NORMAL_PRIORITY

setName(String name): Changes the name of this thread to be equal to the argument name.

currentThread(): Returns a reference to the currently executing thread object

getId(): returns the id of a Thread

getName(): returns this thread's name

getPriority(): Returns this thread's priority. Priority ranges from 1to 10

MIN_PRIORITY =1

NORMAL_PRIORITY=5(default priority)

MAX-PRIORITY=10

interrupt(): Interrupts this thread

isAlive(): Tests if this thread is alive. It returns true is thread is running, it return false f thread is not running

isInterrupted(): Tests whether this thread has been interrupted
join(): Waits for the thread until it completes and executes the new thread
yield():A **yield()** method is a **static** method of **Thread** class and it can stop the currently executing thread and will give a chance to **other waiting threads of the same priority.** If in case there are no waiting threads or if all the waiting threads have **low priority** then the same thread will continue its execution.

How many ways we can create a Thread. Explain in detail
How threads are created:

Threads can be created by using two mechanisms :
1. Extending the Thread class :
Syntax :
Class classname extends Thread
{
Statements;
}
2. Implementing the Runnable Interface :
Syntax :
Class classname implements Runnable
{
Statements;
}

Thread is a part of a program. Each program can have multiple threads. Each thread has a priority which is used by thread scheduler to determine which thread must run first.

Creating Thread using Thread class :
MyThread.java :
Step1: Create class MyThread
Step 2:Extending Thread class

```
import java.lang.Thread;
class MyThread extends Thread
{

        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                System.out.println(i);
                }

        }
public static void main(String args[])
{
MyThread t1=new MyThread();
t1.start();
```

```
}
}
```
Output :
```
javac MyThread.java
java MyThread

1
2
3
4
5
```
Creating Thread using Runnable Interface :

MyThread.java :
```
import java.lang.Runnable;
class MyThread implements Runnable
{

        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                System.out.println(i);
                }

        }
public static void main(String args[])
{
MyThread mt=new MyThread();
Thread t=new Thread(mt);
t.start();

}
}
javac MyThread.java
java MyThread
1
2
3
4
5
```

Multithreading:

Multithreading in Java is a process of executing multiple threads simultaneously for maximum utilization of CPU.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation,server side applications etc.

Java provides a thread class that has various method calls in order to manage the behaviour of threads.

Thread class methods:

start(): Causes this thread to begin execution;

run(): When the **run**() **method** calls, the code specified in the **run**() **method** is executed. You can call the **run**() **method** multiple times.

sleep(long millis): Causes the currently executing thread to suspended for the specified number of milliseconds. Sleep method is a static method so it can be called by static method

setPriority(int newPriority): this method is used to set priority and Changes the current priority of the thread

setName(String name): Changes the name of this thread to be equal to the argument name.

currentThread(): Returns a reference to the currently executing thread object.It is a static method it can be called by classname

getId(): returns the id of a Thread

getName(): returns this thread's name

getPriority(): Returns this thread's priority. Priority ranges from 1to 10

MIN_PRIORITY =1

NORMAL_PRIORITY=5(default priority)

MAX-PRIORITY=10

interrupt(): Interrupts this thread

isAlive(): Tests if this thread is alive. It returns true is thread is running, it return false f thread is not running

isInterrupted(): Tests whether this thread has been interrupted

join(): Waits for the thread until it completes and executes the new thread.join method is a static method so it can be called by class name.

yield():itstops thecurrently executing thread and executes the threads of same priority .if there are no threads of same priority then it executes the currently executing thread

How to assign a priority to a thread? Can two threads have same priority?

Priority of the threads can be assigned by setPriority(int) method

When two threads have same priority we cant predict which thread will execute first

ThreadMethods.java :

```
import java.lang.Thread;
class ThreadMethods extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
		try
		{
		Thread.sleep(1000);
		}
		catch(InterruptedException ie)
		{
		}
		System.out.println(i);
		}
	}
public static void main(String args[])
{
ThreadMethods t1=new ThreadMethods();
t1.start();
System.out.println("Thread Id="+t1.getId());
System.out.println("Thread Name="+t1.getName());
t1.setName("ECE");
System.out.println("Thread Name="+t1.getName());
```

```
System.out.println("Thread Priority="+t1.getPriority());
t1.setPriority(10);
System.out.println("Thread Priority="+t1.getPriority());
try
{
t1.join();//after executing t1 thread it allows other threads to execute
}
catch(InterruptedException ie)
{
}
System.out.println("ThreadStatus="+t1.isAlive());// thread status running or not
ThreadMethods t2=new ThreadMethods();
t2.start();
}
}
```

Output :

javac ThreadMethods.java

```
 java ThreadMethods
Thread Id=14
Thread Name=Thread-0
Thread Name=ECE
Thread Priority=5
Thread Priority=10
1
2
3
4
5
ThreadStatus=false
1
2
3
4
5
```

write a program to Create two threads? Print one thread with even numbers and another thread with odd numbers?

EvenOddThread.java

```
import java.lang.Thread;
class EvenThread extends Thread
{

        public void run()
        {
                for(int i=0;i<=10;i=i+2)
                {
                System.out.println(i);
                }
        }
}

class OddThread extends Thread
{
        public void run()
        {
                for(int i=1;i<=10;i=i+2)
                {
                System.out.println(i);
                }
        }

}

class EvenOddThread
{
public static void main(String args[])
{
EvenThread t1=new EvenThread();
System.out.println("First Thread:");
t1.start();

try
{
t1.join();
}
catch(InterruptedException ie)
{
System.out.println("Interrupted exception");
}
System.out.println("Second Thread:");
OddThread t2=new OddThread();
t2.start();
```

}

}

First Thread:
0
2
4
6
8
10
Second Thread:
1
3
5
7
9

Synchronization:
Synchronization in java is the capability ***to control the access of multiple threads to any shared resource***.
Java Synchronization is better option where we want to allow only one thread to access the shared resource.
Uses

1. To prevent thread interference.
2. To prevent consistency problem.

   Concept of Lock in Java

   Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Thread Synchronization: Thread synchronization can be done in three ways

1. Synchronized method.
2. Synchronized block.
3. Static synchronization.

1.Synchronized method: If you declare any method as synchronized, it is known as synchronized method.
Synchronized method is used to lock an object for any shared resource.
Syntax:

```
synchronized methodname(arg1,arg2,agr3,,,,,,,,)
{
}
```

2.**Synchronized Block:**

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

Syntax to use synchronized block :

```
1. synchronized (object reference expression) {
2.   //code block
3. }
```

3.Static synchronization: If you make any static method as synchronized, the lock will be on the class not on object.

Syntax:

```
 synchronized  static methodname(arg1,arg2,arg3,,,,,)
{

}
```

Explain the synchronization of multiple threads in Java with an example :

Synchronization.java :-*

```
class MovieReservation implements Runnable
{
int ticket=2;
int t;
        MovieReservation(int t)
        {
        this.t=t;
        }

public  void run()
{
String name=Thread.currentThread().getName();

synchronized(this){

        if(t<=ticket)
        {
        System.out.println(name+"Ticket booked");
        ticket=ticket-1;
        }
        else
        {
        System.out.println(name+" ticket not booked");
        }
```

```
try{
Thread.sleep(1500);
}
catch(Exception e)
{
System.out.println(e);
}
}

}

}
class Synchronization
{
public static void main(String args[])
{
MovieReservation m=new MovieReservation(1);
Thread t1=new Thread(m);
t1.setName("ravi");
Thread t2=new Thread(m);
t2.setName("kalyan");
Thread t3=new Thread(m);
t3.setName("nina");
t1.start();
t2.start();
t3.start();
}
}
```

Output :

javac Synchronization.java

java Synchronization

raviTicket booked

kalyanTicket booked

nina ticket not booked

Communication between Thread :

Inter-thread communication in Java :

Inter-thread communication is all about allowing synchronized threads to communicate with each other.

Inter-thread communication is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed .It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

---

1) wait() method :

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

2) notify() method :
Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

3) notifyAll() method :
Wakes up all threads that are waiting on this object's monitor. Syntax:
Difference between wait and sleep?
Let's see the important differences between wait and sleep methods.

wait()	sleep()
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	is the method of Thread class
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

InterThreadCommunication.java :

```
class Accounts extends Thread
{
volatile int balance_amount=10000;

        public  synchronized void withdraw(int amount)
        {
                if(amount>balance_amount)
                {
                System.out.println("waiting for deposit");
                        try
                        {
                        this.wait();
                        }
                        catch(InterruptedException ie)
```

```
                {
                ie.printStackTrace();
                }

            }

            balance_amount = balance_amount -amount;
            System.out.println(amount +" Rs/- Amount is withdrawn");
            System.out.println("Balance Amount is :" + balance_amount);


        }

        public   void deposit(int amount)
        {
        synchronized(this){
        balance_amount=balance_amount+amount;
        System.out.println(amount+" deposited");
        System.out.println("balance amount="+balance_amount);
        this.notifyAll();
        }

}

}
class InterThreadCommunication
{
public static void main(String args[])
{
final Accounts acc=new Accounts();

new Thread(){public void run(){
acc.deposit(10000);
}
}.start();

        new Thread(){
        public void run()
        {
        acc.withdraw(1000);
        }
            }.start();


        new Thread(){
        public void run()
```

```
        {
        acc.deposit(30000);
        }
                }.start();


        new Thread(){
        public void run()
        {
        acc.withdraw(9000);
        }
                }.start();


}
}
```

javac InterThreadCommuniation.java
java InterThreadCommunication

10000 deposited
balance amount=20000
9000 Rs/- Amount is withdrawn
Balance Amount is :11000
30000 deposited
balance amount=41000
1000 Rs/- Amount is withdrawn
Balance Amount is :40000

File reading and writing in java :

Java FileWriter and FileReader classes are used to write and read data from text files

FileReader:

FileReader is a class which is useful to read data in the form of characters from a 'text' file.

FileReader fd=new FileReader(file): Creates a FileReader object of the given File and read the file

File reader class pre defined methods

read(): this method reads a single character. This method will read data until data available

FileWriter :

FileWriter is a class which is useful to create a file writing characters into the file.

Constructor:

FileWriter fw=new FileWriter(file): It Constructs a FileWriter object given a File object.

FileWriter class methods

write(String str) – this method Writes a string into a file.

Write a program to read the data and write the data from a file?

//copy data from one file to another file

// read data from one file and write in another file

FileReaderWriter.java :

```
import java.io.*;
class FileReaderWriter
{
public static void main(String args[])
{
try{
FileReader fr=new FileReader("E:\\cse.txt");//FileReader class
FileWriter fw=new FileWriter("E:\\ece.txt");//FileWriter class
        int ch;

        while((ch=fr.read())!=-1)
        {//reading and writing int to another file
        fw.write((char)ch);
        }
        System.out.println("file reading and writing success full");
        fr.close();
        fw.close();
        }
        catch(FileNotFoundException f)
        {
        System.out.println("file not found exception");
        }
        catch(IOException io)
        {
        System.out.println("IO Exception");
        }
}
}
```

```
javac FileReaderWriter.java
java FileReaderWriter
file reading and writing success full
```

Random access of file in java

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If the end of file reached before reading the data from than IOException is thrown

Constructors

RandomAccessFile(File file, String mode):	Creates a random access file stream to read /write the file .

Methods

length():It returns the length of the file
seek(long position):It sets the filepointer to the beginning of the file
close():It closes this random access file stream
write(int b):It writes the specified bytes to the file
read(): It reads the byte of data to a file

RandomAccessFileDemo.java

```
import java.io.*;

class RandomAccessFileDemo

{

public static void main(String args[])

{

try{

RandomAccessFile r1=new RandomAccessFile("E:\\cse.txt","r");

r1.seek(3);//set the location of the pointer

byte b[]=new byte[40];//creating 40 bytes of memory

r1.read(b);//read method() to read file

System.out.println(new String(b));

RandomAccessFile r2=new RandomAccessFile("E:\\ece.txt","rw");

//reading data from cse.txt and writing it into ece.txt

r2.write(b);// writing into ece.txt file

r1.close();
```

```
r2.close();

}

catch(FileNotFoundException f)

{

System.out.println("File not found Exception");

}

catch(IOException i)

{

System.out.println("IO Exception");

}

}
}
```

```
javac RandomAccessFileDemo.java
java RandomAccessFileDemo
name is peter joseph
```

AWT INTRODUCTION

There are two ways of giving inputting to java program

1. **CommandUser Interface(CUI) Inputting:** If any user interact with java program by passing same commands through command prompt is known as **CUI** inputting

2. **Graphical user interface(GUI)Inputting:** if any user interact with java program through a graphical window known as **GUI inputting**.

Graphical user interface

Graphical user interface(GUI) Offers user interaction through some graphical components.

Some of the graphical Components are Buttons ,TextField ,Radio buttons, List box ,ScrollBar etc

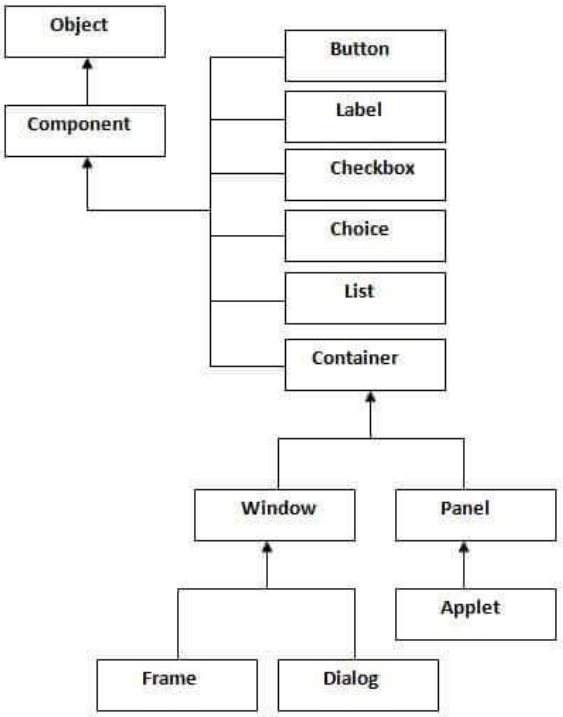
Abstract Window ToolKit(AWT)Controls

Java AWT (Abstract Window Toolkit) is ***an API to develop GUI or window-based applications*** in java.All these classes are defined in **java.awt** package.The java.awt package provides classes for AWT api such as TextField,Button,Label,TextArea,RadioButton,CheckBox,Choice,List etc**.**

Awt components are platform dependent and heavyweightt:

Awt components are **platform dependent** because program written in awt behaves differently in different operating systems and awt components are displayed with different sizes in different operating systems. **AWT components** are considered **heavy weight** because they are being generated by underlying operating system (OS). For example if you are create a button in **AWT** that means you are actually asking OS to create a text box

JAVA AWT HIERARCHY



Component class in java

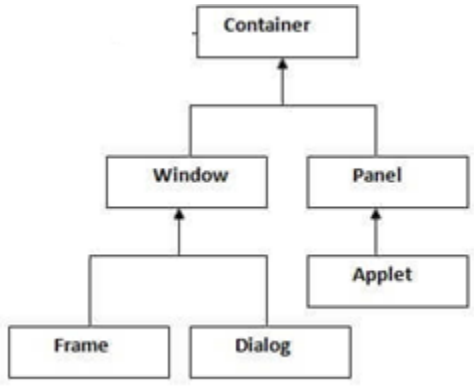
Component	Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Explain the AWT Container hierarchy and explain them?

Container Class in java



Container**:** The class **Container** is the super class for the containers of AWT. Container object can contain other AWT components. The Container is a component in AWT that can contain another component like buttons, textfields, labels etc. The classes that extends Container class are such as Window,Panel and Frame

Window

The window is the container that has no borders and menu bars.window is extended by Frame,Dialog

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.Panel is extended by Applet

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Container class Example

MyFrame.java

```
import java.awt.*;
class MyFrame extends Frame
{
MyFrame()
{
        setSize(500,500);
```

```
        setVisible(true);
        setLayout(null);
        setTitle("AWT Example");

}
public static void main(String args[])
{
MyFrame m=new MyFrame();
}
}
```

The Components of AWT(AWT Controls)

Label : It can be any user defined name used to identify input field like textbox, textarea etc.

```
Label l1=new Label("uname");
Label l2=new Label("password");
```

Button

This class can be used to create a push button.

Example

```
Button b1=new Button("submit");
Button b2=new Button("cancel");
```

Checkbox

This class can be used to design multi selection checkbox.

Example

```
Checkbox cb1=new Checkbox(".net");
Checkbox cb2=new Checkbox("Java");
Checkbox cb3=new Checkbox("html");
Checkbox cb4=new Checkbox("php");
```

Choice

This class can be used to design a drop down box with more options and supports single selection.

Example

```
Choice c=new Choice()
```

```
c.add("Andhra Pradesh");
c.add("Telangana");
c.add("Madhya Pradesh");
c.add("Maharastra");
```

List

This class can be used to design a list box with multiple options and support multi selection.

Example

```
List l=new List();
l.add("goa");
l.add("delhi");
l.add("pune");
```

Note: By holding clt button multiple options can be selected.

TextArea

TextArea class is a multi line region that displays text.Textfield is used for single line but TextArea class is used to display multiple line. It allows the editing of multiple line text. It inherits TextComponent class.

TextArea ta=new TextArea();

Student Registration program using AWT Controls or Awt Components

Registration.java

```
import java.awt.*;
class RegistrationDemo extends Frame
{
        RegistrationDemo()
        {
        setSize(1200,700);//setting size of frame
        setTitle("Registration");


        setBackground(Color.yellow);
        setForeground(Color.red);
        setLayout(null);
        setVisible(true);
```

```
Label l1=new Label("Student Registration");//heading Label
Font myFont = new Font("Arial",  Font.BOLD, 32);// heading font
l1.setFont(myFont);// //setting heading label font
add(l1);
l1.setBounds(420,45,320,35);


Label l2=new Label("Student Name:");
add(l2);
l2.setBounds(420,105,90,20);

//text field for giving input
TextField t=new TextField();
add(t);
t.setBounds(520,105,140,20);

Label l3=new Label("Gender");
add(l3);
l3.setBounds(420,165,50,20);

//radio buttons creation
CheckboxGroup gender=new CheckboxGroup();// for radio buttons
Checkbox male=new Checkbox("male",gender,false);
Checkbox female=new Checkbox("female",gender,false);
add(male);
add(female);

male.setBounds(520,150,90,50);
female.setBounds(620,150,140,50);


Label l4=new Label("Department");
add(l4);l4.setBounds(420,220,100,20);

Choice dept=new Choice();// choi
//drop down box for single selection
dept.add("SELECT");
dept.add("CSE");
dept.add("EEE");
```

```
dept.add("ECE");
dept.add("MECH");
add(dept);
dept.setBounds(520,220,150,20);

Label l5=new Label("Course");
add(l5);l5.setBounds(420,275,90,20);

//drop downbox for multiple selection
List li=new List();
li.add("Select");
li.add("Java");
li.add("C");
li.add("C++");
li.add("DotNet");
add(li);
li.setBounds(520,270,150,20);


Label l6=new Label("Qualification");
add(l6);
l6.setBounds(420,325,90,20);
//check boxes for multiple selection
Checkbox ssc=new Checkbox("SSC");// checkbox
Checkbox inter=new Checkbox("Inter");
Checkbox degree=new Checkbox("Degree");
add(ssc);
add(inter);
add(degree);

ssc.setBounds(520,320,50,30);
inter.setBounds(600,320,50,30);
degree.setBounds(680,320,60,30);


Label l7=new Label("Address");
add(l7);
l7.setBounds(420,350,90,20);
```

```
        //text area for address

TextArea t1=new TextArea();
        add(t1);
        t1.setBounds(520,350,200,100);



        Button b=new Button("Register");
        add(b);
        b.setBackground(Color.CYAN);
        b.setBounds(520,450,50,30);
        }
public static void main(String args[])
{
RegistrationDemo r=new RegistrationDemo();//calling constructor
}


}javac Registration.java
java Registration
```

MenuBar:

MenuBar:A Menu bar is created to the frame ;

Menubar created by

MenuBar mb=new MenuBar();

Creating menu

Menu m=new Menu("File");

Adding items to menu

m.add("cut");

m.add("copy");

m.add("paste");

adding menu to menubar

mb.add(m)

Illustrate a program to create EDIT Menu (Cut, Copy, and Paste) of NOTEPAD.

MenuBarExample.java

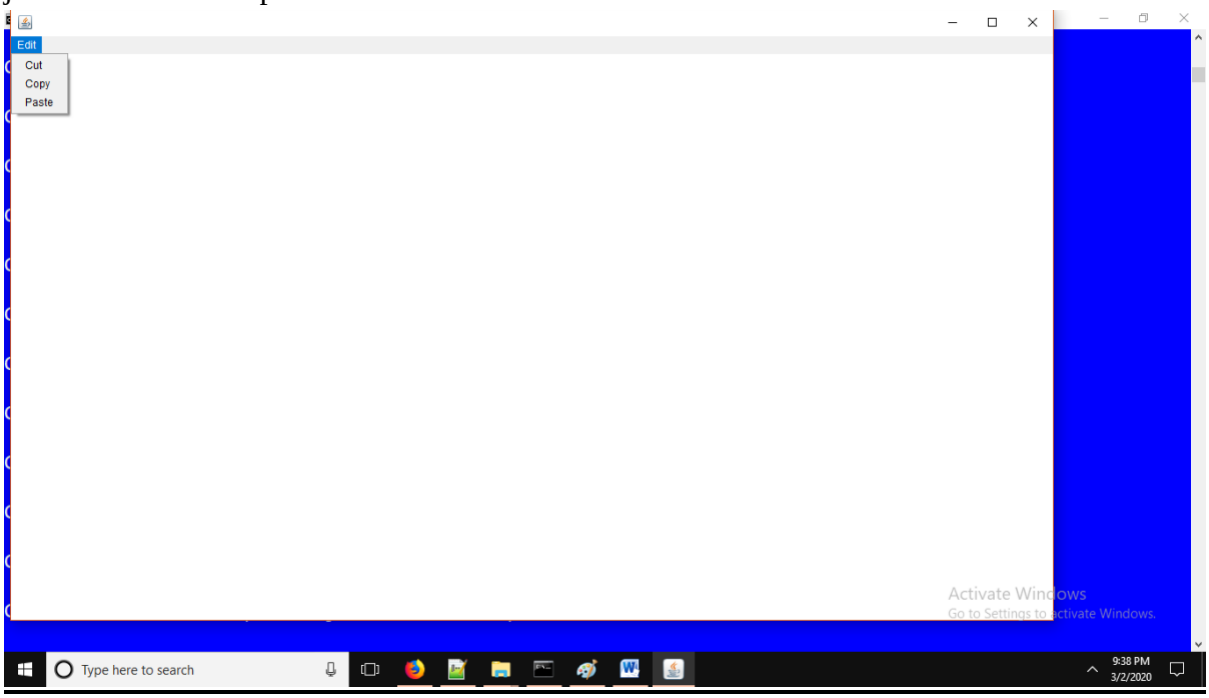
```
import java.awt.*;
class MenuBarExample extends Frame
{
MenuBarExample()
{
setSize(1200,700);
        setLayout(null);
        setTitle("Registration");
        setVisible(true);
        MenuBar mb=new MenuBar();
        Menu m=new Menu("Edit");

        m.add("cut");
        m.add("copy");
        m.add("paste");


        mb.add(m);
```

```
        setMenuBar(mb);


}
public static  void main(String args[])
{
MenuBarExample m=new MenuBarExample();
        }
}
```

Output

javac MenuBarExample.java

java MenuBarExample



Layouts

Layout means the arrangement of components within the container in a particular order.

The java.awt package provides 5 basic layouts. Each layout has its own significance and all of them are completely different.

The 5 layouts available in the java.awt library are:

1. Border Layout
2. Card Layout
3. Flow Layout
4. Grid Layout

5. GridBag Layout

The BorderLayout is a layout which organizes components in terms of direction. A border layout divides the frame or panel into 5 sections – North, South, East, West and Centre.

Constructor of border layout

BorderLayout(): creates a border layout with the given horizontal and vertical gaps between the components.

fields of the BorderLayout class

- BorderLayout.NORTH
- BorderLayout.SOUTH
- BorderLayout..EAST
- BorderLayout.WEST
- BorderLayout.CENTER

Illustrate a Java program to implement Border Layout Manager?

Border.java

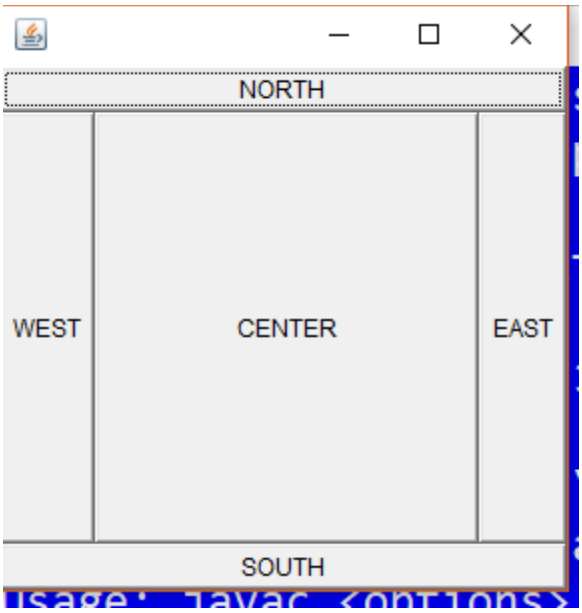
```
import java.awt.*;

public class Border extends Frame{

Border(){

   Button b1=new Button("NORTH");;

   Button b2=new Button("SOUTH");;

   Button b3=new Button("EAST");;

   Button b4=new Button("WEST");;

   Button b5=new Button("CENTER");;


   add(b1,BorderLayout.NORTH);

   add(b2,BorderLayout.SOUTH);

   add(b3,BorderLayout.EAST);
```

```
    add(b4,BorderLayout.WEST);
    add(b5,BorderLayout.CENTER);
    setSize(300,300);
    setVisible(true);
    setLayout(new BorderLayout());
}
public static void main(String[] args) {
    new Border();
}
}
```

output

javac Border.java

java Border



Card Layout

The CardLayout class manages the components in such a way that only one component is visible at a time. It treats each component as a card in the container. Only one card is visible at a time, and the container acts as a stack of cards.

Constructors

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.

   Methods of card layout

- **next(Container parent):** is used to flip to the next card of the given container.
- **previous(Container parent):** is used to flip to the previous card of the given container.
- **first(Container parent):** is used to flip to the first card of the given container.
- **last(Container parent):** is used to flip to the last card of the given container.
- **show(Container parent, String name):** is used to flip to the specified card with the given name.

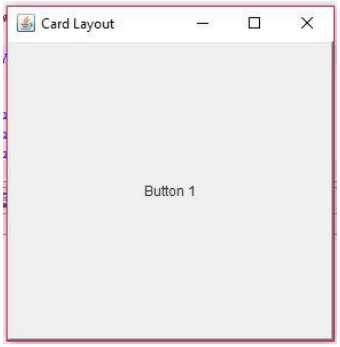
MyCardLayout.java

```
import java.awt.*;

public class MyCardLayout extends Frame {

MyCardLayout(){


   Button b1=new Button("1");

   Button b2=new Button("2");

   Button b3=new Button("3");

   Button b4=new Button("4");

   Button b5=new Button("5");

   Button b6=new Button("6");

   Button b7=new Button("7");

   Button b8=new Button("8");
```

```
    Button b9=new Button("9");

    add(b1);add(b2);add(b3);add(b4);add(b5);
    add(b6);add(b7);add(b8);add(b9);

   setLayout(new CardLayout());
    //setting grid layout of 3 rows and 3 columns

    setSize(300,300);
    setVisible(true);
}

public static void main(String[] args) {
    new MyCardLayout();
}
}
```

Output

```
javac MyCardLayout.java
java MyCardLayout
```



FlowLayout: The FlowLayout is used to arrange the components in a line, one after another (in a flow).

Constructor

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

Following are the fields in FlowLayout class.

- FlowLayout.LEFT
- FlowLayout.RIGHT
- FlowLayout.CENTER

MyFlowLayout.java

```
import java.awt.*;

public class MyFlowLayout extends Frame{

MyFlowLayout(){


   Button b1=new Button("1");

   Button b2=new Button("2");
```

```
    Button b3=new Button("3");
    Button b4=new Button("4");
    Button b5=new Button("5");

    add(b1);add(b2);add(b3);add(b4);add(b5);

    setLayout(new FlowLayout(FlowLayout.RIGHT));
    //setting flow layout of right alignment

    setSize(300,300);
    setVisible(true);
}
public static void main(String[] args) {
    new MyFlowLayout();
}
}
```

Grid Layout

The GridLayout manager is used to arrange the components in the two-dimensional grid. Each component is displayed in a rectangle.

Constructors of grid layout

1. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.

Example

GridLayout g=new GridLayout(2,3)

2 rows and 3 columns

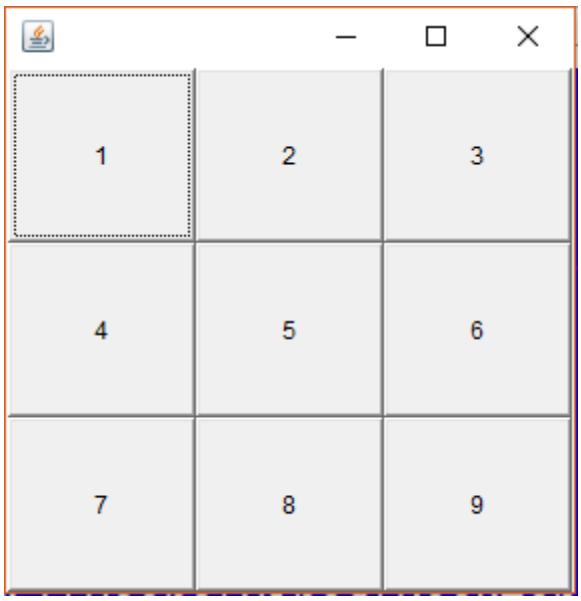
//**MyGridLayout.java**

```
import java.awt.*;

public class MyGridLayout extends Frame{

MyGridLayout(){

   Button b1=new Button("1");

   Button b2=new Button("2");

   Button b3=new Button("3");

   Button b4=new Button("4");
```

```
    Button b5=new Button("5");
        Button b6=new Button("6");
        Button b7=new Button("7");
    Button b8=new Button("8");
        Button b9=new Button("9");

    add(b1);add(b2);add(b3);add(b4);add(b5);
    add(b6);add(b7);add(b8);add(b9);

    setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns

    setSize(300,300);
    setVisible(true);
}
public static void main(String[] args) {
    new MyGridLayout();
}
}
```

Output

```
javac MyGridLayout.java
java MyGridLayout
```



GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells.

constructor

- GridBagLayoutgrid = new GridBagLayout();

Event:

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.

Sources of Event

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Handling in AWT

In general you cannot perform any action on dummy GUI screen even any button click or select any item. To perform some actions on these dummy GUI screen you need some predefined classes and interfaces. All these type of classes and interfaces are available in **java.awt.event** package.

Changing the state of an object is known as an **event**.

The process of handling the request in GUI screen is known as **event handling** (event represent an action). It will be changes component to component.

Listener interfaces consists of methods

Listener Interfaces	**Methods**
ActionListener	actionPerformed(ActionEvent ae)
MouseListener	mouseEntered(MouseEvent me) mouseReleased(MouseEvent me) mouseClicked(MouseEvent me) mousePressed(MouseEvent me) mouseExited(MouseEvent me)
KeyListener	keyTyped(KeyEvent ke) keyReleased(KeyEvent ke) keyPressed(KeyEvent ke)

ItemListener	itemStateChanged(ItemEvent ie)
WindowListener	windowClosing(WindowEvent we) windowClosed(WindowEvent we) windowOpened(WindowEvent we) windowActivated(WindowEvent we) windowDeactivated(WindowEvent we) windowIconified(WindowEvent we) windowDeconified(WindowEvent we)

Event Delegation Model:

When we create a component(buttons,radio buttons) generally the component is display on the screen but is not capable of performing any actions for example a submit button which can be display but cannot perform any action. But user want to perform some action. Hence when he clicks on the button clicking like this is called event.

An event represents some action done on a component

Event delegation model represents that when an event is generated by the user on a component it delegates to listener interface and listener call a method in response to event finally the event is handled by the method

The event-delegation model has two advantages

1.It enables event handling to be handled by objects . This allows a clean separation between a component's design and its use.

2.The other advantage of the event-delegation model is that it performs much better in applications where many events are generated.

Syntax to Handle the Event

Example

```
class className implements XXXListener
{
.......
.......
}
addcomponentobject.addXXXListener(this);
.......
// override abstract method of given interface and write proper logic
public void methodName(XXXEvent e)
{
.......
.......
}
.......
}
```

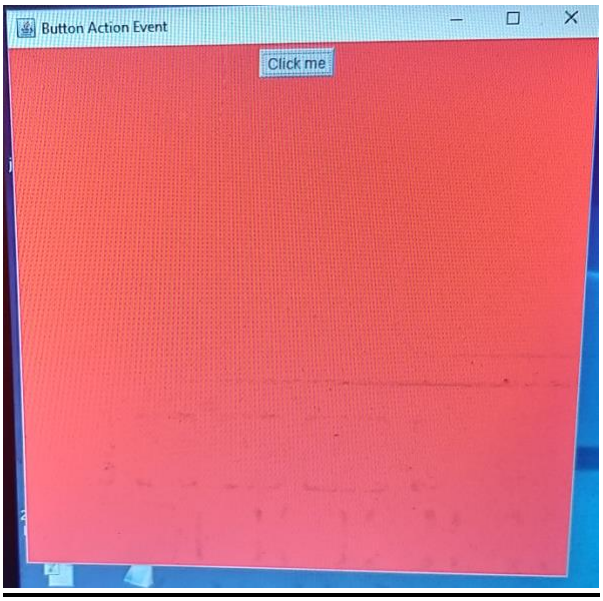
Implement Button Action event or ActionListener interface in java

ActionListernerDemo.java

```
import java.awt.*;

import java.awt.event.*;


class ActionListernerDemo extends Frame implements ActionListener

{

Button b;

ActionListernerDemo()

{


setSize(500,500);
```

```
setLayout(new FlowLayout());
setVisible(true);
setTitle("Button Action Event");
setBackground(Color.red);
 b=new Button("Click me");
add(b);
b.setBounds(100,100,40,50);
b.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
if(ae.getSource()==b)
setBackground(Color.blue);
}
public static void main(String args[])
{
ActionListernerDemo a=new ActionListernerDemo();
}
}
```

Output



Implement a program on Mouse Event using awt

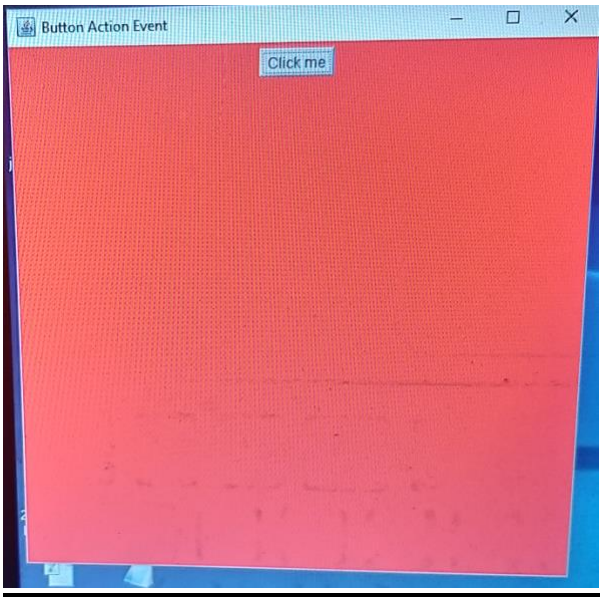
```
import java.awt.*;
import java.awt.event.*;
class MouseListenerDemo extends Frame implements MouseListener
{
Button b;
MouseListenerDemo()
{
setSize(1000,700);
setVisible(true);
setLayout(null);
setBackground(Color.red);
setTitle("ButtonEvent");
 b=new Button("Click me");
b.setBounds(100,100,100,50);
```

```
add(b);
b.addMouseListener(this);
}
public void mouseClicked(MouseEvent ae)
{
if(ae.getSource()==b)
setBackground(Color.green);
}
public void mousePressed(MouseEvent ae)
{
}
public void mouseExited(MouseEvent ae)
{
}
public void mouseReleased(MouseEvent ae)
{
}
public void mouseEntered(MouseEvent ae)
{
}



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

```
MouseListenerDemo a=new MouseListenerDemo();
}

}
```

javac MouseListenerDemo.java

java MouseListenerDemo



APPLETS

Java programs can be divided into two categories :-

i) Applications and

ii) Applets.

Applications are the programs that contain main() method and applets are the programs that do not contain main() method.

Applet: Applet is a Java program that runs on a browser. **Applet** is a predefined class in **java.applet** package used to design distributed application. It is a client side technology. Applets are run on web browser.

Difference between an Application program and Applet program:

S.No.	Property	Application	Applet
1.	main() method	Exists	does not exist
2	Execution	needs JVM	needs a browser like Netscape, chrome etc.

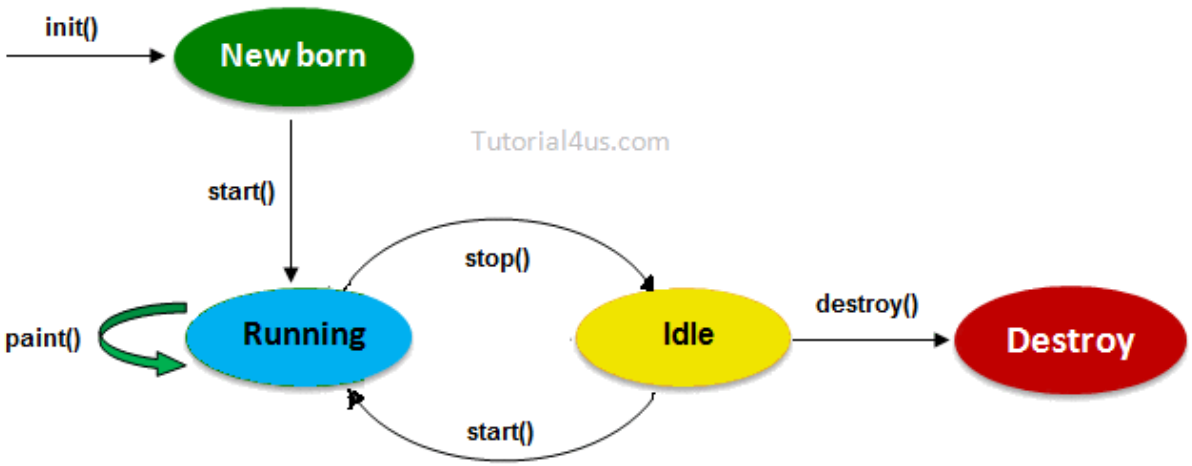
Advantage of Applets

- Applets are supported by most web browsers.
- Applets works on client side so less response time.
- Easy to develop applet, just extends applet class.

Just like threads,applets too got a life cycle

Explain the Applet Life Cycle.

Life Cycle of an Applet:



Applet goes through different stages of execution between its birth and death. **init():** Which will be executed whenever an applet program start loading, it contains the logic to initiate the applet properties.

start(): It will be executed whenever the applet program starts running.

stop(): Which will be executed whenever the applet window or browser is minimized.

destroy(): It will be executed whenever the applet window or browser is going to be closed (at the time of destroying the applet program permanently).

paint(): It will be executed whenever action performed

Creation of Applet

Syntax:

```
class className extends Applet
{
.....
// override lifecycle methods
.....
}
```

Running of applet programs

Applet program can run in two ways.

- Using html (in the web browser)
- Using appletviewer tool (in applet window)

Running an applet program using html

appletviewer Filename.html

Running of applet using appletviewer

appleviewer Filename.java

Some browsers does not support **<applet>** tag so that Sun MicroSystem was introduced a special tool called **appletviewer** to run the applet program.

In this Scenario Java program should contain <applet> tag in the commented lines so that appletviewer tools can run the current applet program.

Applet Class

Applet class contains
init(),paint,start(),stop(),destroy() **methods**

Structure of Applet Program:

The structure of Applet program is

- package section
- Html section
- Java App is a sub class which extends from Applets class
- Applet class contains init(),paint(),stop(),destroy() methods

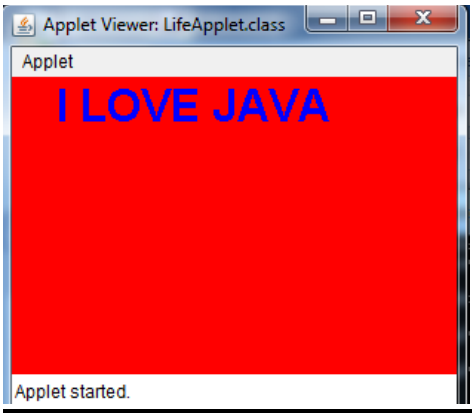
```
import java.applet.*;

import java.awt.*;

/*

<applet code="FirstApplet.class" width="300" height="200"></applet>

*/

public class FirstApplet  extends  Applet

{

public void init()

{

}

public void stop()

{

}

public void destory()

{

}

public void paint(Graphics g)
```

```
{
Font f=new Font("Arial",Font.BOLD,30);
g.setFont(f);
setBackground(Color.RED);
setForeground(Color.BLUE);

g.drawString("I LOVE JAVA",30,30);
}
}
```

Output

javac FirstApplet.java

appletviewer FirstApplet.java



Create an Applet that changes the Font and background Color depending upon the user selection from the input? .

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class HomeApplet extends Applet implements ActionListener,ItemListener{
```

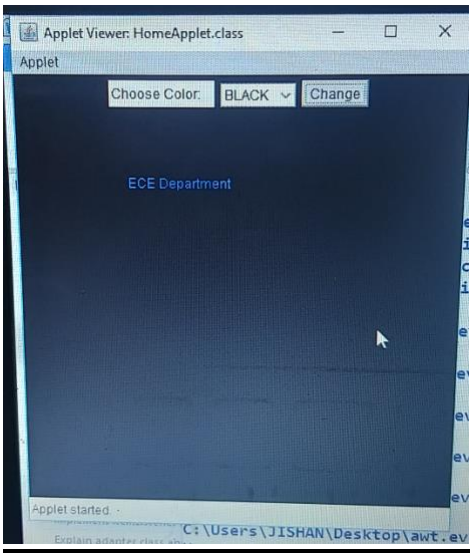
```
/* <applet code="HomeApplet.class" height="400" width="400">
                </applet>*/
String name;
Label l;
Choice colors; //List colors
Button b;
String col="";
public void init() {
        l = new Label("Choose Color:");
        colors = new Choice();
        colors.add("BLUE");
        colors.add("GRAY");
        colors.add("BLACK");
        b = new Button("Change");
        add(l);add(colors);add(b);
        b.addActionListener(this);
        colors.addItemListener(this);

        setBackground(Color.CYAN);
        setForeground(Color.BLUE);
}
public void start() {

}
```

```
    public void paint(Graphics g) {

        if(col.equals("BLUE"))
            setBackground(Color.BLUE);
        else if(col.equals("BLACK"))
            setBackground(Color.BLACK);
        else
            setBackground(Color.GRAY);

    }
    public void stop() {

    }
    public void destroy() {
    }
    public void actionPerformed(ActionEvent ae) {
        col = colors.getSelectedItem();
        repaint();
    }
    public void itemStateChanged(ItemEvent ie) {
        col = colors.getSelectedItem();
        repaint();
    }
}
```

Output

javac HomeApplet.java

appletviewer HomeApplet.java



How to assign a priority to a thread? Can two threads have same priority?

We can assign priority to the thread by setPriority(int) method

Yes Two Threads can have same priority.

There are 3 static variables defined in Thread class for priority.
public static int MIN_PRIORITY: This is minimum priority that a thread can have. Value for this is 1.
public static int NORM_PRIORITY: This is default priority of a thread if do not explicitly define it. Value for this is 5.
public static int MAX_PRIORITY: This is maximum priority of a thread. Value for this is 10.
Get and Set Thread Priority:

1. **public final int getPriority():** java.lang.Thread.getPriority() method returns priority of given thread.
2. **public final void setPriority(int newPriority):** java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.

import java.lang.*;

```
class ThreadDemo extends Thread {
    public void run()
    {
        System.out.println("Inside run method");
    }

    public static void main(String[] args)
    {
        ThreadDemo t1 = new ThreadDemo();
        ThreadDemo t2 = new ThreadDemo();

        t1.setPriority(2);
        t2.setPriority(5);
        t1.start();
        t2.start();
    }
}
```

Output

```
Inside run method
Inside run method
```