3.Class & object

3.1.Introduction to class, class members,Creating instances of class

3.2.Static and final

3.3.Array :1D & 2D

3.4.Command line arguments

3.5.Input stream reader, scanner class

3.6.Constructors, overloading of methods and constructors

3.7.Inner Class

3.1.Introduction to class, class members,Creating instances of class

An entity that has state and behaviour is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

state: represents data (value) of an object.

Behaviour: represents the behaviour (functionality) of an object such as deposit, withdraw etc.

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

For Example: Pen is an object. Its name is Reynolds, colour is white etc. known as its state. It is used to write, so writing is its behaviour.

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

Class Definition

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

data members

methods

constructors

and block

class and interface

Syntax to declare a class:

```
class <class_name>{
    data member;  method;
}
```

Instance variable:A variable that is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when object(instance) is created. That is why, it is known as instance variable.

Method

In java, a method is like function i.e. used to expose behaviour of an object.

Advantage of Method

1.Code Reusability

2.Code Optimization

new keyword:

The new keyword is used to allocate memory to an object at runtime.

Java Naming conventions:Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.But, it is not forced to follow. So, it is known as convention not rule.All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Advantage of naming conventions in java:

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY e

Simple Example of Object and Class

```
class Student1{
 int id;//data member (also instance variable)
  String name;//data member(also instance variable)

 public static void main(String args[]){
  Student1 s1=new Student1();//creating an object of Student
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

output:

```
0
null
```

Example of Object and class that maintains the records of students

```
class Student2{
 int rollno;
String name;
 void insertRecord(int r, String n){ //method
  rollno=r;
  name=n;
    }
 void displayInformation(){//method
 System.out.println(rollno+" "+name);
     }
   public static void main(String args[]){
  Student2 s1=new Student2();
   Student2 s2=new Student2();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
   s2.displayInformation();
     }
   }
```

output:

```
111 Karan
222 Aryan
```

Another Example of Object and Class

```
class Rectangle{
int length,width;
void insert(int l, int w){
  length=l;
 width=w;
}
  void calculateArea(){
  System.out.println(length*width);
  }
  public static void main(String args[]){
 Rectangle r1=new Rectangle();
 Rectangle r2=new Rectangle();
 r1.insert(11,5);
 r2.insert(3,15);
 r1.calculateArea();
 r2.calculateArea();
  }
 }
```

output:

```
55
45
```

Anonymous object:

Anonymous simply means nameless. An object that have no reference is known as anonymous object.If you have to use an object only once, anonymous object is a good approach.

```
  class Calculation{
   void fact(int n){
    int fact=1;
  for(int i=1;i<=n;i++){
    fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}
  public static void main(String args[]){
 new Calculation().fact(5);//calling method with anonymous object
}
}
```

output:

```
 120
```

Creating multiple objects by one type only.We can create multiple objects by one type only as we do in case of primitives.

Example:

```
class Rectangle{
 int length,width;
  void insert(int l,int w){
 length=l;
 width=w;
 }
 void calculateArea(){
 System.out.println(length*width);
 }
  public static void main(String args[]){
 Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
  r1.insert(11,5);
 r2.insert(3,15);
  r1.calculateArea();
 r2.calculateArea();
}
}
```

output:

```
55
45
```

Advantage of OOPs over Procedure-oriented programming language

1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.

2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.

3) OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

What is difference between object-oriented programming language and object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

3.2 static and Final keyword

The **static keyword** in java is used for memory management. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance/object of the class.

The static can be:

- variable (also known as class variable)
- method (also known as class method)
- block
- nested class

static variable

- If you declare any variable as static, it is known static variable.
- The static variable can be used to refer the common property of all objects (that is not unique for each object)

  e.g. company name of employees,college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable:

```
class Student{
    int rollno;
    String name;
    String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created.

All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once.

Java static property is shared to all objects.

```
//Program of static variable
 class Student{
  int rollno;
  String name;
  static String college ="ITS";
  Student8(int r,String n){
  rollno = r;
  name = n;
  }
 void display (){
 System.out.println(rollno+" "+name+" "+college);
 }
  public static void main(String args[]){
  Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 s1.display();
 s2.display();
 }
}
```

output:

```
111 Karan ITS
222 Aryan ITS
```

Program of counter without static variable

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

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

output:

```
1 1 1
```

Program of counter by static variable

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

```
class Counter2{
static int count=0;//will get memory only once and retain its value
  Counter2(){
  count++;
System.out.println(count);
}
  public static void main(String args[]){
  Counter2 c1=new Counter2();
  Counter2 c2=new Counter2();
   Counter2 c3=new Counter2();
   }
}
```

Output :

```
1 2 3
```

static method

- If you apply static keyword with any method, it is known as static method. A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Example:

```
//Program to get cube of a given number by static method  class
  Calculate{
  static int cube(int x){
  return x*x*x;
  }
  public static void main(String args[]){
  int result=Calculate.cube(5);  System.out.println(result);
  }
}
```

Output:

```
125
```

Example://Program of changing the common property of all objects(static field).

```
class Student9{
    int rollno;
    String name;
    static String college = "ITS";
    static void change(){
    college = "BBDIT";
    }
    Student9(int r, String n){
    rollno = r;
    name = n;
    }
    void display (){
    System.out.println(rollno+""+name+"”+college);
    }
      public static void main(String args[]){
   Student9.change();
   Student9 s1 = new Student9 (111,"Karan");
   Student9 s2 = new Student9 (222,"Aryan");
 s1.display();
   s2.display();
   }
}
```

Output:

```
111 Karan BBDIT
 222 Aryan BBDIT
```

Restrictions for static method

There are two main restrictions for the static method. They are:

- The static method can not use non static data member or call non-static method directly.
- this and super cannot be used in static context.

```
class A{
 int a=40;//non static

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

Output:

```
Compile Time Error
```

Q) why java main method is static?

Because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.

Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

Example of static block

```
class A2{
  static{
  System.out.println("static block is invoked");
  }
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Output:

```
static block is invoked    Hello main
```

Q) Can we execute a program without main() method?

Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

```
class A3{
static{
   System.out.println("static block is invoked");
    System.exit(0);
   }
}
```

Output:

static block is invoked (if not JDK7)

In JDK7 and above, output will be

```
Error: Main method not found in class A3, please define the main method as:
public static void main(String[] args)
```

Final Keyword

The final keyword in java is used to restrict the user.

Final can be:

- variable
- method
- class

final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example:

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike{
  final int speedlimit=90;//final  variable
  void run(){
  speedlimit=400;
  }
  public static void main(String args[]){
  Bike obj=new Bike();
  obj.run();
  }
}//end of class
```

Output:Compile Time Error

final method

- If you make any method as final, you cannot override it.

Example:

```
class Bike{
  final void run(){
  System.out.println("running");
  }
}
 class Honda extends Bike{
   void run(){
   System.out.println("running safely with 100kmph");
   }
   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}   Output:Compile Time Error
```

final class

If you make any class as final, you cannot extend it

Example:

```
final class Bike{}
  class Honda1 extends Bike{
  void run(){
  System.out.println("running safely with 100kmph");
  }
   public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

Output:Compile Time Error

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it.

Example:

```
class Bike{
  final void run(){
  System.out.println("running...");
  }
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

Output:running...

Q) What is blank or uninitialized final variable?

- A final variable that is not initialized at the time of declaration is known as blank final variable.
- If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.
- It can be initialized only in constructor.
- Example of blank final variable

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

Q) Can we initialize blank final variable?

Yes, but only in constructor.

Example:

```
class Bike10{
  final int speedlimit;//blank final variable
   Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }
    public static void main(String args[]){
    new Bike10();
 }
}
```

output:

```
70
```

static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example

```
class A{
   static final int data;//static blank final  variable
   static{
    data=50;
   }
   public static void main(String args[]){
    System.out.println(A.data);
  }
}
```

output:

```
50
```

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
class Bike11{
  int cube(final int n){

   n=n+2;//can't be changed as n is final  n*n*n;
  }

  public static void main(String args[]){

  Bike11 b=new Bike11();
    b.cube(5);
 }
}
```

Output:Compile Time Error

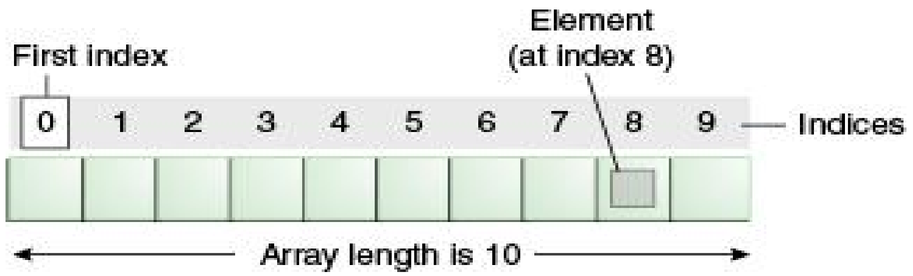
Q) Can we declare a constructor final?

- No, because constructor is never inherited.

3.3.Array :1D & 2D

An array is a collection of similar type of elements that have contiguous memory location.

- In Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.
- Array in java is index based, first element of the array is stored at 0 index.



Advantage of Java Array

Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.

Random access: We can get any data located at any index position. Disadvantage of Java Array

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array

Syntax

1.dataType[] arr; (or)

2.dataType []arr; (or)

3.dataType arr[];

Instantiation of an Array

dataType []arrayRefVar=**new** datatype[size];

Example

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
 //printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.print(a[i]);
    }
 }
```

output:

```
10 20 70 40 50
```

Declaration, Instantiation and Initialization of Java Array

int a[]={33,3,4,5};//declaration, instantiation and initialization

Example:

```
class Testarray1{
public static void main(String args[]){
 int a[]={33,3,4,5};//declaration, instantiation and initialization
 //printing array
for(int i=0;i<a.length;i++)//length is the property of array
 System.out.print(a[i]);
   }
   }
```

Output:

```
33 3 4 5
```

Passing Array to method

We can pass the java array to method so that we can reuse the same logic on any array.

Example:

```
class Testarray2{
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
 min=arr[i];
  System.out.println(min);
}
 public static void main(String args[]){
 int a[]={33,3,4,5};
min(a);//passing array to method
}
}
```

output: 3

Multidimensional array

Data is stored in row and column based index (also known as matrix form) in Multidimensional array.

Syntax:

```
 dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

Example

```
  int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array

```
   arr[0][0]=1;        arr[0][1]=2;      arr[0][2]=3;

   arr[1][0]=4;        arr[1][1]=5;      arr[1][2]=6;

   arr[2][0]=7;        arr[2][1]=8;      arr[2][2]=9;
```

Example

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

   //declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

   //printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
    System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}
}
```

Output:

```
1 2 3
2 4 5
4 4 5
```

Copying an array

We can copy an array to another by the arraycopy method of System class.

Syntax of arraycopy method

```
public static void arraycopy(
Object src, int srcPos,Object dest, int destPos, int length
)
```

Example

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

        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e','i', 'n', 'a', 't', 'e', 'd' };

        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

output:caffein

Example on 2D array to add 2 matrices

```
class Testarray5{
public static void main(String args[]){
     //creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
    //creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];
    //adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
          c[i][j]=a[i][j]+b[i][j];
          System.out.print(c[i][j]+" ");
          }
    System.out.println();
  }
}
}
```

Output:

```
2 6 8
6 8 10
```

Command Line arguments in Java

What is Command Line Argument?

Command Line Argument is information passed to the program when you run the program. The passed information is stored as a string array in the main method. Later, you can use the command line arguments in your program.

Example While running a class Demo, you can specify command line arguments as

java Demo arg1 arg2 arg3 …

Important Points

There is no restriction on the number of java command line arguments.You can specify any number of arguments

Information is passed as Strings.

They are captured into the String args of your main method

If we run a Java Program by writing the command “java Hello Geeks At GeeksForGeeks” where the name of the class is “Hello”, then it will run upto Hello. It is command upto “Hello” and after that i.e “Geeks At GeeksForGeeks”, these are command line arguments.

When command line arguments are supplied to JVM, JVM wraps these and supply to args[]. It can be confirmed that they are actually wrapped up in args array by checking the length of args using args.length

```
// Program to check for command line arguments
class Hello
{
    public static void main(String[] args)
    {
        // check if length of args array is
        // greater than 0
        if (args.length > 0)
        {
            System.out.println("The command line"+
                               " arguments are:");

            // iterating the args array and printing
            // the command line arguments  for
            (String val:args)
                System.out.println(val);
        }
        else
            System.out.println("No command line "+
                               "arguments found.");   }
}
```

```
class Demo{
     public static void main(String
         b[]){  System.out.println("Argument one =
         "+b[0]);  System.out.println("Argument two =
         "+b[1]);
    }
}
```

```
javac Demo.java

java Demo apple orange  output:
Argument one= apple  Argument two=orange
```

Java program to find the sum of command line integer arguments

```
public class Demo{
public static void main(String[] args) {
 int sum=0,x;
   for(int i=0;i<args.length;i++)
       {
       x=Integer.parseInt(args[i]);
       sum=sum+x;
        }
             System.out.println("sum i="+sum);
    }
}
```

```
javac  Demo.java


java Demo 1 2 3 4 5
```

output:

```
sum =15
```

3.5. scanner class and Input Stream Reader

Java Scanner class

- There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- The **Scanner** class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.
- Java Scanner class is widely used to parse text for string and primitive types using regular expression.
- Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Example to get input from console

Example of the Java Scanner class which reads the int, string and double value as an input:

```
import java.util.Scanner;
class ScannerTest{
 public static void main(String args[]){
 Scanner sc=new Scanner(System.in);
  System.out.println("Enter your rollno");
  int rollno=sc.nextInt();
 System.out.println("Enter your name");
  String name=sc.next();
   System.out.println("Enter your fee");
 double fee=sc.nextDouble();
    System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
    sc.close();
 }
}
```

Output:

```
Enter your rollno  111

Enter your name  Ratan

Enter your fee

450000

Rollno:111  name:Ratan   fee:450000
```

Java Scanner Example with delimiter

Let's see the example of Scanner class with delimiter. The \s represents whitespace.

```
import java.util.*;
public class ScannerTest{
public static void main(String args[]){
      String input = "10 tea 20 coffee 30 tea buiscuits";

      Scanner s = new Scanner(input).useDelimiter("\\s");

      System.out.println(s.nextInt());
      System.out.println(s.next());
      System.out.println(s.nextInt());
      System.out.println(s.next());
      s.close();
}
}
```

Output:

```
10
tea
20
coffee
```

InputStreamReader Class

The InputStreamReader class reads characters from a byte input stream. It reads bytes and decodes them into characters using a specified charset. The decoding layer transforms bytes to chars according to an encoding standard . There are many available encodings to choose from.

InputStreamReader class performs two tasks:

Read input stream of keyboard.
Convert byte streams to character streams.

The Java.io.InputStreamReader class is a bridge from byte streams to character streams.It reads bytes and decodes them into characters using a specified charset.

InputStreamReader Class Syntax:
public class InputStreamReader extends Reader

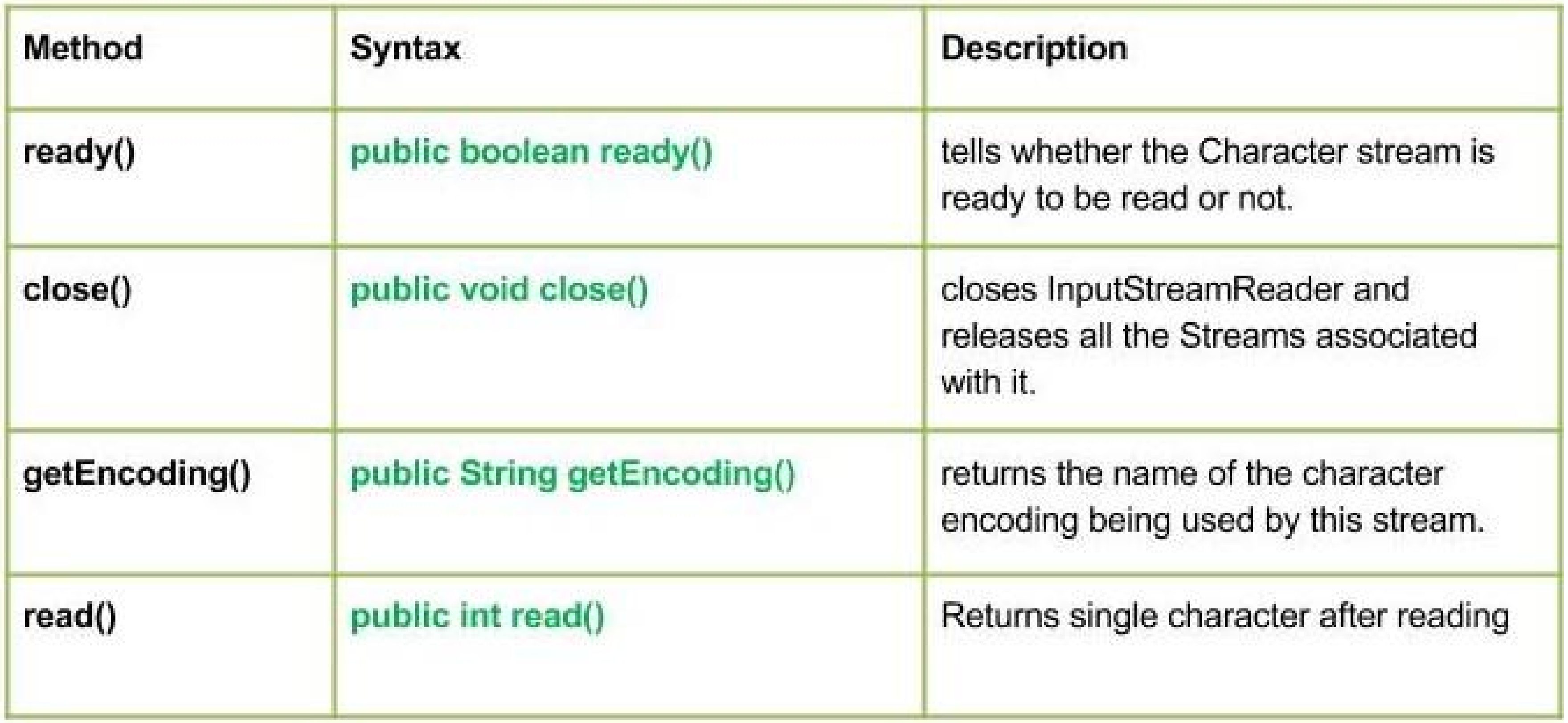
Example:

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

 InputStreamReader isr=new InputStreamReader(System.in);
  BufferedReader br=new BufferedReader(isr);

 System.out.println("enter your name");
   String name=br.readLine();

  System.out.println("enter your rollno");
       int roll=Integer.parseInt(br.readLine());

  System.out.println("enter your fee");
       double fee =Double.parseDouble(br.readLine());

   System.out.println("name="+name);
       System.out.println("rollno="+roll);
       System.out.println("Fee="+fee);

     }
}
```

Input Stream Reader Class constructors

Constructor	Description
InputStreamReader(InputStream in)	This creates an InputStreamReader that uses the default charset.
InputStreamReader(InputStream in, Charset cs)	This creates an InputStreamReader that uses the given charset.
InputStreamReader(InputStream in, CharsetDecoder dec)	This creates an InputStreamReader that uses the given charset decoder.
InputStreamReader(InputStream in, String charsetName)	This creates an InputStreamReader that uses the named charset.

Input Stream Reader Class Methods



Method	Syntax	Description
ready()	public boolean ready()	tells whether the Character stream is ready to be read or not.
close()	public void close()	closes InputStreamReader and releases all the Streams associated with it.
getEncoding()	public String getEncoding()	returns the name of the character encoding being used by this stream.
read()	public int read()	Returns single character after reading

An example with read() and close() method

The read() method of an InputStreamReader returns an int which contains the char value of the char read. Here is a Java InputStreamReader read()

```
int data = inputStreamReader.read();
```

You can cast the returned int to a char like this:

```
char aChar = (char) data;
```

Closing an InputStreamReader

When you are finished reading characters from the InputStreamReader you should remember to close it. Closing an InputStreamReader will also close the InputStream instance from which the InputStreamReader is reading.

Closing an InputStreamReader is done by calling its close() method. Here is how closing an InputStreamReader looks:

```
inputStreamReader.close();
```

3.6.Constructors, overloading of methods and constructors

Method Overloading

- If a class have multiple methods by same name but different parameters, it is known as Method Overloading.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantage of method overloading?

- Method overloading increases the readability of the program.

Different ways to overload the method

By changing number of arguments in function

By changing the data type in function

Example of Method Overloading by changing the no. of arguments

```
.class Calculation{
  void sum(int a,int b){
  System.out.println(a+b);
  }
  void sum(int a,int b,int c){
  System.out.println(a+b+c);
  }

  public static void main(String args[]){
   Calculation obj=new Calculation();
   obj.sum(10,10,10);
  obj.sum(20,20);


  }
  }
```

output:

```
30
40
```

Example of Method Overloading by changing data type of argument

```
class Calculation2{
   void sum(int a,int b){
   System.out.println(a+b);
   }
   void sum(double a,double b){
   System.out.println(a+b);
   }
public static void main(String args[]){

   Calculation2 obj=new Calculation2();
   obj.sum(10.5,10.5);

   obj.sum(20,20);
    }
   }
```

output:

```
21
40
```

Why Method Overloaing is not possible by changing the return type of method?

Example:

```
class Calculation3{
  int sum(int a,int b){
  System.out.println(a+b);
  }
  double sum(int a,int b){
  System.out.println(a+b);
  }

  public static void main(String args[]){
  Calculation3 obj=new Calculation3();
  int result=obj.sum(20,20); //Compile Time Error
    }
}
```

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

Can we overload main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading.

Example:

```
class Overloading1{
   public static void main(int a){
   System.out.println(a);
   }

   public static void main(String args[]){
    System.out.println("main() method invoked");
   main(10);
   }
}
```

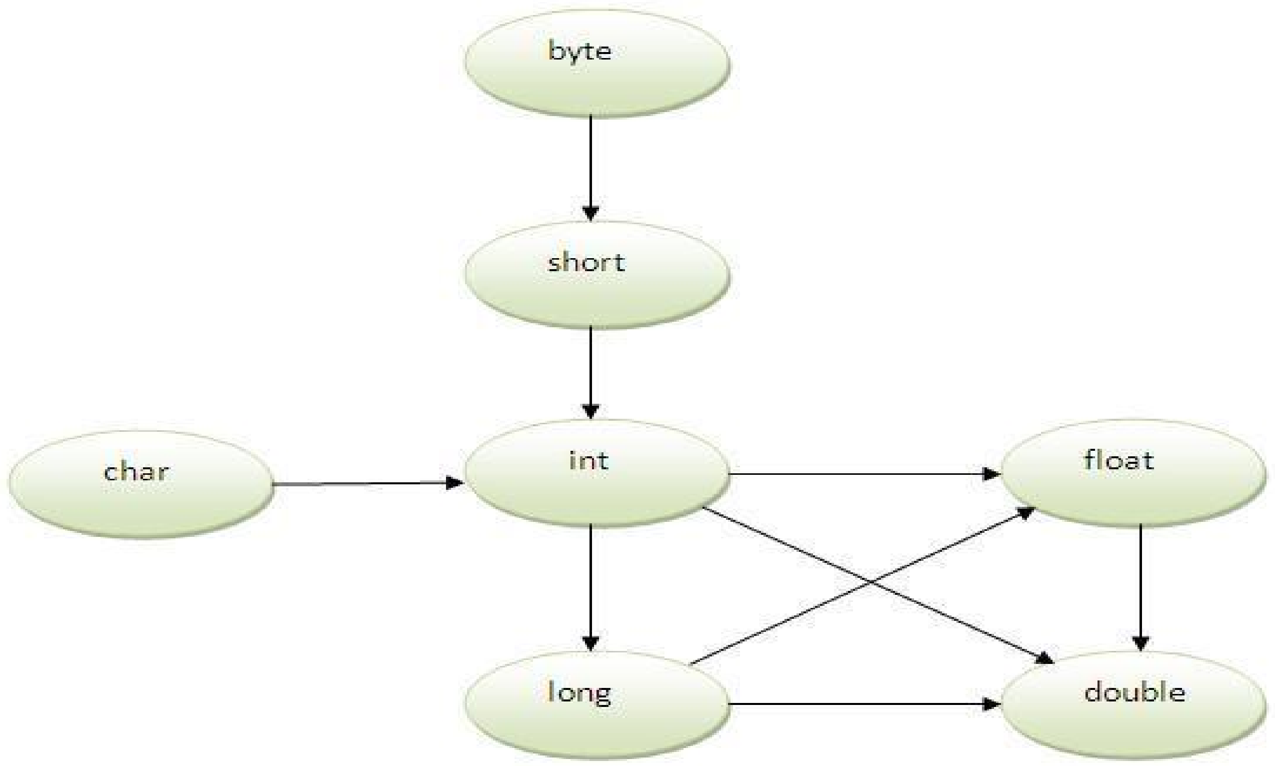
Output:

```
main() method invoked  10
```

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:

As displayed in the below diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int,long,float or double. The char datatype can be promoted to int,long,float or double and so on.



Example of Method Overloading with TypePromotion

```
class OverloadingCalculation1{
  void sum(int a,long b){
  System.out.println(a+b);
  }
  void sum(int a,int b,int c){
  System.out.println(a+b+c);
  }

  public static void main(String args[]){
   OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,20);//now second int literal will be promoted to long
  obj.sum(20,20,20);

  }
}
```

Output:

```
40
60
```

Example of Method Overloading with TypePromotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

```
class OverloadingCalculation2{
  void sum(int a,int b){
  System.out.println("int arg method  invoked");
  }
  void sum(long a,long b){
  System.out.println("long arg method  invoked");
  }
  public static void main(String args[]){
  OverloadingCalculation2 obj=new OverloadingCalculation2();
  obj.sum(20,20);//now int arg sum() method gets invoked
  }
}
```

Output:

```
int arg method invoked
```

Example of Method Overloading with Type Promotion in case ambiguity

```java
class OverloadingCalculation3{
  void sum(int a,long b){
  System.out.println("a method invoked");
  }
  void sum(long a,int b){
  System.out.println("b method invoked");
  }

  public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCalculation3();
  obj.sum(20,20);//now ambiguity
  }
}
```

Output:

```
Compile Time Error
```

Constructor

Constructor is a special type of method that is used to initialize the object.

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

Rules for creating java constructor

- Constructor name must be same as its class name
- Constructor must have no explicit return type

There are two types of constructors:

- Default constructor (no-arg constructor)
- Parameterized constructor

Default Constructor

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

Syntax of default constructor:

<class_name>(){ }

Example of default constructor

```
class Bike1{
Bike1(){
System.out.println("Bike is created");
}
public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

Output:

```
Bike is created
```

Q) **What is the purpose of default constructor?**

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example:

```
class Student3{
int id;
String name;
  void display(){
   System.out.println(id+" "+name);
  }
  public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display();
   } }
```

output:

```
0 null  0 null
```

parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example:

```
class Student4{
    int id;
    String name;
    Student4(int i,String n){
     id = i;
    name = n;
    }
    void display(){
    System.out.println(id+" "+name);
    }
    public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
      s1.display();
      s2.display();
     }
}
```

output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.

Example

```
class Student5{
  int id;
   String name;
  int age;
  Student5(int i,String n){
  id = i;
  name = n;
  }
  Student5(int i,String n,int a){
  id = i;
  name = n;
  age=a;
  }
  void display(){
  System.out.println(id+" "+name+" "+age);
  }
```

```
      public static void main(String args[]){
      Student5 s1 = new Student5(111,"Karan");
      Student5 s2 = new Student5(222,"Aryan",25);
      s1.display();
       s2.display();
   }
    }
```

output:

```
111 Karan 0
222 Aryan 25
```

Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

Example:

```
class Student6{
    int id;
    String name;
    Student6(int i,String n){  id = i;
    name = n;
    }
    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){
    System.out.println(id+" "+name);
    }

    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
     Student6 s2 = new Student6(s1);
            s1.display();
            s2.display();
        }
}
```

Output:

```
111 Karan
111 Karan
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

Example:

```
class Student7{
    int id;
    String name;
    Student7(int i,String n){
    id = i;
    name = n;
    }
    Student7(){}//default constructor
    void display(){
    System.out.println(id+" "+name);
    }
    public static void main(String args[]){
     Student7 s1 = new Student7(111,"Karan");
    Student7 s2 = new Student7();
      s2.id=s1.id;
      s2.name=s1.name;  s1.display();
      s2.display();
     }
}
```

output:

```
 111 Karan
111 Karan
```

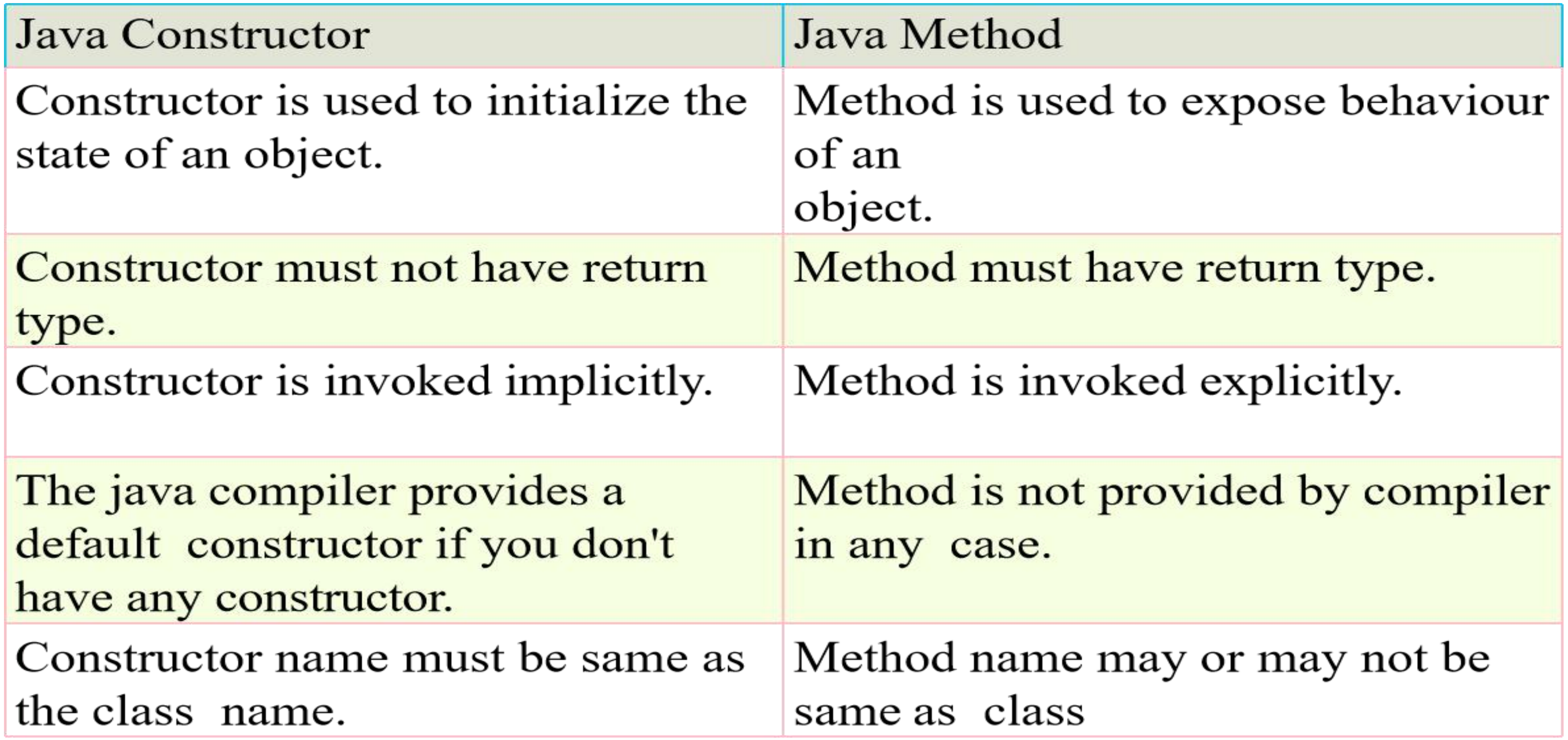
Q) Does constructor return any value?

Ans: yes, that is current class instance (You cannot use return type yet it returns a value)

Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

Difference between constructor and method in java



Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class

Destructor

- Destructor is a method called when the destruction of an object takes place. “ The main goal of the destructor is to free up the allocated memory and also to clean up resources like the closing of open files, closing of database connections, closing network resources, etc.
- Destructors in Java can be learned with the finalize method.
- The concept is as same as the finalize method. Java works for all except the destructor with the help of Garbage collection.
- Therefore in case, there is a need for calling the destructor, it can be done with the help of the finalize method.This method is not independent as it relies on Garbage Collection.
- The garbage collector is a thread that deletes or destroyed the unused object in the heap area. Say if the object is connected to a file or say some database application or network connections, before deleting or destroying the object it has to close all the connections related to these resources before the garbage collection takes place. These closing of the functions is done by calling the finalize method.

Destructor

Syntax:

```
Class Object
{
protected void finalize()
{
//statements like closure of database connection
}
}
```

- The destructor has a finalize() method in java which is similar to the destructor in C++. When the objects are created they are stored in the heap memory. These are accessible by main or child threads. So when these objects are no more used by the main thread or its child threads they become eligible for garbage collection and the memory which was acquired now becomes available by new objects being created.
- Before an object is a garbage collected by the garbage collector the JRE (Java Runtime Environment) calls the finalize() method to close the input-output streams, the database connections, network connections, etc.

Destructor

Garbage collection is mainly the process of marking or identifying the unreachable objects and deleting them to free the memory. The implementation lives in the JVM, the only requirement is that it should meet the JVM specifications.

Example:

```
public class A {
public void finalize() throws Throwable{
System.out.println("Object is destroyed by the Garbage Collector");
}
public static void main(String[] args) {
A test = new A();
test = null;
System.gc();
}
}
```

Advantages of garbage collection in Java:

- It automatically deletes the unused objects that are unreachable to free up the memory
- Garbage collection makes Java memory efficient
- It need not be explicitly called since the implementation lives in the JVM
- Garbage collection has become an important and standard component of many programming languages

3.7.Inner Class

Nested Classes

- In Java, just like methods, variables of a class too can have another class as its member.
- The class written within another class is called the **nested class**, and the class that holds the inner class is called the **outer class**.

Syntax

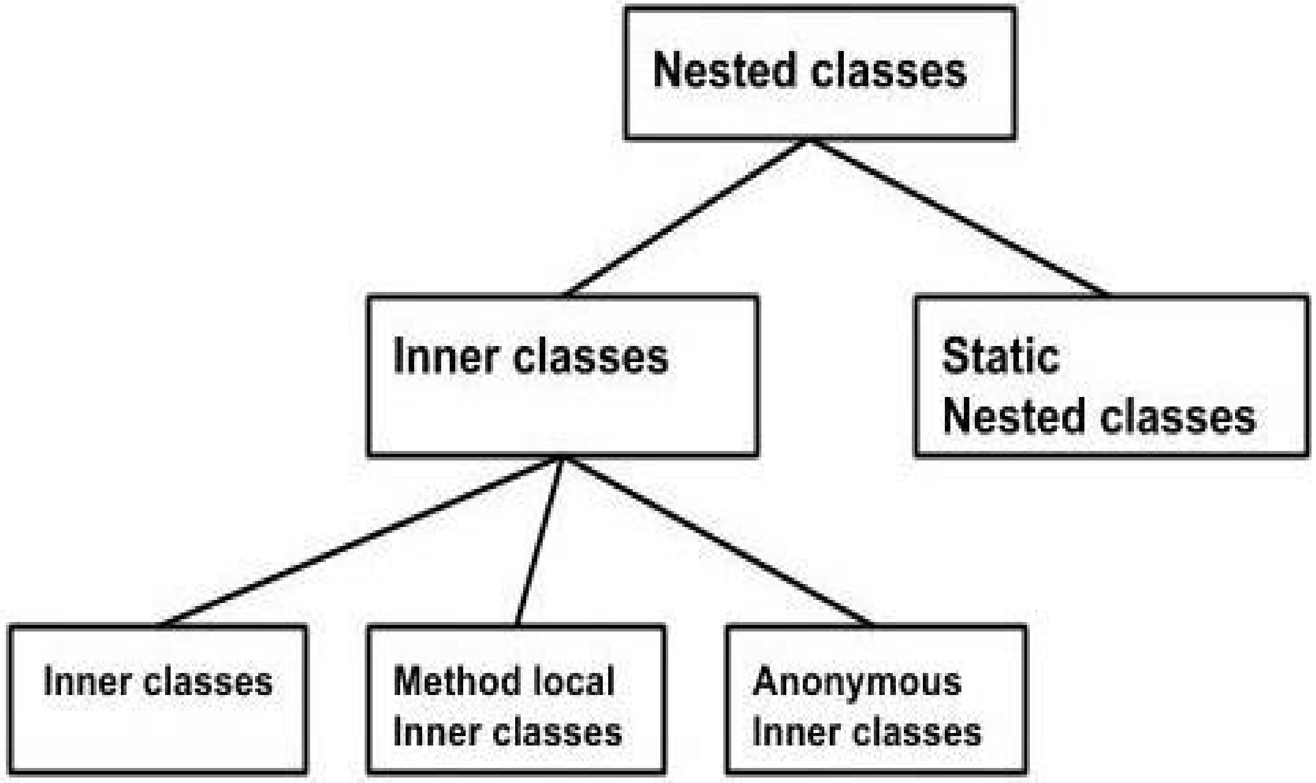
Following is the syntax to write a nested class.

Here, the class **Outer_Demo** is the outer class and the class **Inner_Demo** is the nested class.

```
class Outer_Demo {
class Nested_Demo {
   }
}
```

Nested classes are divided into two types

- **Non-static nested classes** – These are the non-static members of a class.
- **Static nested classes** – These are the static members of a class.



Inner Classes (Non-static Nested Classes)

Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier **private**, but if we have the class as a member of other class, then the inner class can be made private. And this is also used to access the private members of a class.

Inner classes are of three types depending on how and where you define them.

They are

- Inner Class
- Method-local Inner Class
- Anonymous Inner Class Inner Class

Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

Example:

```
class Outer_Demo {
   int num;
   // inner class
   private class Inner_Demo {
      public void print() {
         System.out.println("This is an inner class");
      }
   }
   // Accessing he inner class from the method
     void display_Inner() {
      Inner_Demo inner = new Inner_Demo();
      inner.print();
   }
}
public class My_class {
   public static void main(String args[]) {
      // Instantiating the outer class
      Outer_Demo outer = new Outer_Demo();
      // Accessing the display_Inner()
      //method.
      outer.display_Inner();
   }
}
```

Output

```
This is an inner class.
```

Accessing the Private Members

As mentioned earlier, inner classes are also used to access the private members of a class.

Suppose, a class is having private members to access them. Write an inner class in it, return the private members from a method within the inner class, say, getValue(), and finally from another class (from which you want to access the private members) call the getValue() method of the inner class.

To instantiate the inner class, initially you have to instantiate the outer class. Thereafter, using the object of the outer class, following is the way in which you can instantiate the inner class.

```
Outer_Demo outer = new Outer_Demo();
Outer_Demo.Inner_Demo inner = outer.new Inner_Demo();
```

Example:

```
class Outer_Demo {
   // private variable of the outer class
   private int num = 175;

   // inner class
   public class Inner_Demo {
   public int getNum() {
         System.out.println("This is the getnum method of the inner class");
         return num;
      }
   }
}
```

```
public class My_class2 {

  public static void main(String args[]) {
    // Instantiating the outer class
    Outer_Demo outer = new Outer_Demo();

    // Instantiating the inner class
    Outer_Demo.Inner_Demo inner = outer.new Inner_Demo();
    System.out.println(inner.getNum());
  }
}
```

Output:

```
The value of num in the class Test is: 175
```

Method-local Inner Class

In Java, we can write a class within a method and this will be a local type. Like local variables, the scope of the inner class is restricted within the method.

A method-local inner class can be instantiated only within the method where the inner class is defined. The following program shows how to use a method-local inner class.

Example:

```
public class Outerclass {
  // instance method of the outer class
  void my_Method() {
    int num = 23;
    // method-local inner class
    class MethodInner_Demo {
      public void print() {
        System.out.println("This is method inner class "+num);
      }
    } // end of inner class
    // Accessing the inner class
    MethodInner_Demo inner = new MethodInner_Demo();
    inner.print();
  }
  public static void main(String args[]) {
  Outerclass outer = new Outerclass();
  outer.my_Method();
  }
}
```

Output:

```
This is method inner class 23
```

Anonymous Inner Class

An inner class declared without a class name is known as an anonymous inner class.

In case of anonymous inner classes, we declare and instantiate them at the same time.

Generally, they are used whenever you need to override the method of a class or an interface.

Syntax:

```
AnonymousInner an_inner = new AnonymousInner() {
  public void my_method() {
      ........
      ........
  }
};
```

Example:

```
abstract class AnonymousInner {
   public abstract void mymethod();
}
public class Outer_class {
   public static void main(String args[]) {
      AnonymousInner inner = new AnonymousInner() {
         public void mymethod() {
            System.out.println("This is an example of anonymous inner class");
         }
      };
      inner.mymethod();
   }
}
```

Output:

```
This is an example of anonymous inner class
```

In the same way, you can override the methods of the concrete class as well as the interface using an anonymous inner class.

Anonymous Inner Class as Argument

Generally, if a method accepts an object of an interface, an abstract class, or a concrete class, then we can implement the interface, extend the abstract class, and pass the object to the method. If it is a class, then we can directly pass it to the method.But in all the three cases, you can pass an anonymous inner class to the method.

syntax:

```
obj.my_Method(new My_Class() {
  public void Do() {
     .....
     ....
     }
     });
```

Example:

```
interface Message {
  String greet();
}
public class My_class {
  // method which accepts the object of interface Message
  public void displayMessage(Message m) {
    System.out.println(m.greet() +", This is an example of anonymous inner class as an
    argument");
  }
  public static void main(String args[]) {
    // Instantiating the class
    My_class obj = new My_class();
    // Passing an anonymous inner class as an argument
    obj.displayMessage(new Message() {public String greet() { return "Hello";}});
        }
      }
```

Output

```
Hello This is an example of anonymous inner class as an argument
```

Static Nested Class

- A static inner class is a nested class which is a static member of the outer class.
- It can be accessed without instantiating the outer class, using other static members.
- Just like static members, a static nested class does not have access to the instance variables and methods of the outer class.

Syntax

```
class MyOuter {
  static class Nested_Demo {
  }
}
```

Example:

```
public class Outer {
   static class Nested_Demo {
   public void my_method() {
         System.out.println("This is my nested class");
      }
   }
      public static void main(String args[]) {
        Outer.Nested_Demo nested = new Outer.Nested_Demo();
      nested.my_method();
   }
}
```

Output:

```
This is my nested class
```