3. Objects And Classes

3.1 Classes:

- A Class is a user defined data type.
- A class encapsulates a group of logically related data items and functions that work on them. In Java, the data items are called fields and the functions are called methods.
- Each and every concept in java program is encapsulated in a class.
- Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and its misuse. This task can be accomplished by marking each method or variable as private or public.
- A class defines the shape and behaviour of an object and is a template for multiple objects with similar features.

**Defining a Class**

The general form of class definition is:

```java
Class classname
{

type variablename;


type methodname(parameter - list)
{
    // method body
}
}
```

- A class is defined using the class keyword where the c of *class* is in small case.
- Once a class is defined, instances(objects) of the class can be created and each instance can have different copy of instance variables(attributes fields).
- The data fields inside the body of the class are called as instance variables as they are created when ever an object of the class is instantiated.
- Methods declaration has four parts;
  1. Name of the method
  2. Type of the value the method returns.
  3. List of parameters

**Example:**
Class Book
{
    String name;
    String Authorname;
    int nopages;
    float price;
    String DisplayName()
    {
        System.out.println("Name of the book is"+name);
    }
}

• In the above example, a Book class is created and now any number of instances of this class can be created and each instance can have different attributes. For example, one object of the Book class can have the attribute name as “Tale of Two Cities” while another may have ‘Learn Java’.
• The methods in the class operate on individual objects of the class. Suppose we have two instances obj1 and obj2 of the Book class. Then obj1.DisplayName() displays the name of the obj1, while obj2.DisplayName() displays that of obj2

Scope of Variables:
Most of the times, we may have many methods and variables within a class. Instance variables in the class are accessible by all the methods in the class but a method can not access the variables declared in other methods.

Example:
Class Access
{
    int x;
    void method1 ( )
    {
        int y=10;       //legal
        x=y;             //legal as x is an instance variable of the class access.
    }
    void method2( )
    {
int z=2; //legal
x=5;     //legal as x is an instance variable of the class access
y=1;     //illegal because y is the variable of method1.
}
}

Accessing Instance Variables and Methods

Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path –

/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();

Example

This example explains how to access instance variables and methods of a class.

class Puppy {
    int puppyAge;

    public Puppy(String name) {
        // This constructor has one parameter, name.
        System.out.println("Name chosen is:" + name);
    }

    public void setAge( int age ) {
        puppyAge = age;
    }

    public int getAge( ) {

System.out.println("Puppy's age is :" + puppyAge );
return puppyAge;
}

public static void main(String []args) {
    /* Object creation */
    Puppy myPuppy = new Puppy("tommy");

    /* Call class method to set puppy's age */
    myPuppy.setAge(2);

    /* Call another class method to get puppy's age */
    myPuppy.getAge();

    /* You can access instance variable as follows as well */
    System.out.println("Variable Value :" + myPuppy.puppyAge);
}

If we compile and run the above program, then it will produce the following result –

**Output**
Name chosen is :tommy
Puppy's age is :2
Variable Value :2

### 3.2 Access Specifiers(public, protected, private, default)
There are two types of modifiers in Java: access modifiers and non-access modifiers.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

**There are four types of Java access modifiers:**

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **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.

3. **Protected**: 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. **Public**: 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.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

**Let's understand the access modifiers in Java by a simple table.**

<table>
<thead>
<tr>
<th>Access Modifier</th>
<th>within class</th>
<th>within package</th>
<th>outside package by subclass only</th>
<th>outside package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Default</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Protected</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Public</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

1) **Private**

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
public static void main(String args[]){
    A obj=new A();
    System.out.println(obj.data);//Compile Time Error
    obj.msg();//Compile Time Error
}
}

**Role of Private Constructor**

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

class A{
private A(){}/private constructor
void msg(){System.out.println("Hello java");}
}

public class Simple{
    public static void main(String args[]){
        A obj=new A(); //Compile Time Error
    }
}
2) Default

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

//save by A.java

package pack;

class A{
    void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;

import pack.*;

class B{
    public static void main(String args[]){
        A obj = new A();//Compile Time Error
        obj.msg();//Compile Time Error
    }
}
In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifier.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

//save by A.java

package pack;

public class A{
    protected void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;

import pack.*;

class B extends A{
    public static void main(String args[]){

B obj = new B();

    obj.msg();

    }

    }

Output:Hello

4) Public

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

//save by A.java

package pack;

public class A{

    public void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.*;

class B{

    public static void main(String args[]){
        A obj = new A();
        obj.msg();

}
Java Access Modifiers with Method Overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

class A{
    protected void msg(){System.out.println("Hello java");}
}

class Simple extends A{
    void msg(){System.out.println("Hello java");}//C.T.Error

public static void main(String args[]){
    Simple obj=new Simple();
    obj.msg();
}

The default modifier is more restrictive than protected. That is why, there is a compile-time error.

3.3 Array of Objects

JAVA ARRAY OF OBJECT, as defined by its name, stores an array of objects. Unlike a traditional array that store values like string, integer, Boolean, etc an array of objects stores OBJECTS. The array elements store the location of the reference variables of the object.

Syntax:
Class obj[] = new Class[array_length]

Example:

class ObjectArray{
    public static void main(String args[]){
        Account obj[] = new Account[2];
        //obj[0] = new Account();
        //obj[1] = new Account();
        obj[0].setData(1,2);
        obj[1].setData(3,4);
        System.out.println("For Array Element 0");
        obj[0].showData();
        System.out.println("For Array Element 1");
        obj[1].showData();
    }
}

class Account{
    int a;
    int b;
    public void setData(int c, int d){
        a = c;
        b = d;
    }
    public void showData();
}
System.out.println("Value of a ="+a);
System.out.println("Value of b ="+b);
}
}

Output:
For Array Element 0
Value of a =1
Value of b =2
For Array Element 1
Value of a =3
Value of b =4

3.4 Constructor:

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.
Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

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

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. `<class_name>(){}`

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```java
//Java Program to create and call a default constructor
class Bike1{
    //creating a default constructor
    Bike1(){System.out.println("Bike is created");}
    //main method
```
public static void main(String args[]){
    //calling a default constructor
    Bike1 b=new Bike1();
}

Output:
Bike is created

Java Parameterized Constructor
A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?
The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor
In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
Student4(int i,String n){
    id = i;
    name = n;
}

//method to display the values
void display(){System.out.println(id + " " + name);}

public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
}

Output:
111 Karan
222 Aryan

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.
Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

**Example of Constructor Overloading**

//Java program to overload constructors

class Student5{
    int id;
    String name;
    int age;

    //creating two arg constructor
    Student5(int i,String n){
        id = i;
        name = n;
    }

    //creating three arg constructor
    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

**this keyword in java**

There can be a lot of usage of Java **this keyword**. In java, this is a **reference variable** that refers to the current object.

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

**Example:**

class Student{
    int rollno;
}
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}

Output:
111 ankit 5000
112 sumit 6000

3.5 static block, static fields and method
The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

1) Java static variable

- If you declare any variable as static, it is known as a static variable.
- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

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

Understanding the problem without static variable

```java
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 the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

2) Java 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 the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

**Example of static method**

//Java Program to demonstrate the use of a static method.

class Student{
    int rollno;
    String name;

    static String college = "ITS";

    //static method to change the value of static variable
    static void change(){
        college = "BBDIT";
    }

    //constructor to initialize the variable
    Student(int r, String n){
        rollno = r;
        name = n;
    }

    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}

//Test class to create and display the values of object
public class TestStaticMethod{
    public static void main(String args[]){
        Student.change(); // calling change method
        // creating objects
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
        Student s3 = new Student(333,"Sonoo");
        // calling display method
        s1.display();
        s2.display();
        s3.display();
    }
}

Output: 111 Karan BBDIT
        222 Aryan BBDIT
        333 Sonoo BBDIT

3) Java static block

- Is used to initialize the static data member.
- It is executed before the 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

3.6 Inner Class:
Java inner class or nested class is a class which is declared inside the class or interface.

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

Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

class Java_Outer_class{
    //code
    class Java_Inner_class{
        //code
    }
}

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
3) **Code Optimization**: It requires less code to write.

### 3.7 Creating Accessing and using Packages:

A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

There may be hierarchy of package that means one package can be defined in the other package.

For example: Java.util.jar

In this jar is class in the package util and that util is a package within the package java.

Defining a package is as foloows:

Package packagename;

E.g. package sports;

To define the name of the package it is keep in mind that the name of package is case sensitive.

We can define hierarchy of packages as follows;

Package package1.package2.package3…..packageN;

Package and member access:

<table>
<thead>
<tr>
<th>Access Modifier---&gt;</th>
<th>Public</th>
<th>Protected</th>
<th>Friendly</th>
<th>Private Protected</th>
<th>Private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Location</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>↓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same class</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Subclass in same package</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Other classes in same package</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
### Example:

Import mypackage.io.AcceptInput;

Public class check

{
    Public static void main(String args[])
    {
        System.out.println("Enter a number");
        int a=AcceptInput.readInt();
        System.out.println("We accept"+a);
    }
}

**Output:**

Enter a number:10

We accept 10

### 3.8 Wrapper Class:

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

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

**Use of Wrapper classes in Java**
Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

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

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>Char</td>
<td>Character</td>
</tr>
<tr>
<td>Byte</td>
<td>Byte</td>
</tr>
<tr>
<td>Short</td>
<td>Short</td>
</tr>
<tr>
<td>Int</td>
<td>Integer</td>
</tr>
<tr>
<td>Long</td>
<td>Long</td>
</tr>
<tr>
<td>Float</td>
<td>Float</td>
</tr>
<tr>
<td>Double</td>
<td>Double</td>
</tr>
</tbody>
</table>

**3.9 Garbage Collection:**
In java, garbage means unreferenced objects.

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

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

**Advantage of Garbage Collection**

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

**Finalize() Method:**

- Finalize method works just opposite to Constructors.
- Finalize method is used for removing an object.
- The Finalize method is called explicitly to define the task to be performed
- Java run time is an automatic garbage collecting system. It automatically frees up the memory resources used by the objects. But objects may hold other non object resources such as window system fonts and so. The garbage collector can not free these resources. In order to free these resources, the finalize() method is used.