

Unit I Part B Object and Classes

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

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.



Class

Collection of objects is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use method overloading and method overriding to achieve polymorphism.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing. In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines. A java class is the example of encapsulation.

What is an object in Java

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

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

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Example 1

```

class Student
{
    int id=100;//field or data member or instance variable
    String name="ravi";

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

```

Example 2

```

class Student
{
    int id=101;
    String name="ravi";
}
class TestStudent1
{
    public static void main(String args[])
    {
        Student s1=new Student();
        System.out.println(s1.id);
        System.out.println(s1.name);
    }
}

```

Example 3

```

class Student
{
    int rollno;
    String name;
    void insertRecord(int r, String n)
    {
        rollno=r;
        name=n;
    }
    void displayInformation()
    {

```

```

        System.out.println(rollno+" "+name);
    }
}
class TestStudent4
{
    public static void main(String args[])
    {
        Student s1=new Student();
        Student s2=new Student();
        s1.insertRecord(111,"Karan");
        s2.insertRecord(222,"Aryan");
        s1.displayInformation();
        s2.displayInformation();
    }
}

```

Constructors

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.

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

Default Constructor

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

Example 4

```

class Bike1
{
    //creating a default constructor
    Bike1()
    {
        System.out.println("Class Bike is created");
    }
    public static void main(String args[])

```

```

    {
        //calling a default constructor
        Bike1 b=new Bike1();
    }
}

```

Parameterized Constructor

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

Example 5

```

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

```

Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

Example 6

```

class Student
{

```

```

int id;
String name;
//constructor to initialize integer and string
Student(int i,String n)
{
    id = i;
    name = n;
}
//constructor to initialize another object
Student(Student s)
{
    id = s.id;
    name =s.name;
}
void display()
{
    System.out.println(id+" "+name);
}
public static void main(String args[])
{
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(s1);
    s1.display();
    s2.display();
}
}

```

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.

Example 7

```

class Student
{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student(int i,String n)
    {

```

```

        id = i;
        name = n;
    }
    //creating three arg constructor
    Student(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[])
    {
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan",25);
        s1.display();
        s2.display();
    }
}

```

Java static keyword

The **static keyword** in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. 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)

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).

Example 8

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

Example 9

```
class Counter
{
    static int count=0;
    Counter()
    {
        count++;
        System.out.println(count);
    }
}
```



```

    }
    public static void main(String args[])
    {
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}

```

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 10

```

class Student
{
    int rollno;
    String name;
    static String college = "CCSIT";
    static void change()
    {
        college = "COE";
    }
    Student(int r, String n)
    {
        rollno = r;
        name = n;
    }
    void display()
    {
        System.out.println(rollno+" "+name+" "+college);
    }
}

public class TestStaticMethod
{
    public static void main(String args[])
    {
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
        Student.change();
        Student s3 = new Student(333,"Sonu");
    }
}

```

```
        s1.display();
        s2.display();
        s3.display();
    }
}
```

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

Example 11

```
class Adder
{
    static int add(int a,int b)
    {
        return a+b;
    }
    static int add(int a,int b,int c)
    {
        return a+b+c;
    }
}
class TestOverloading1
{
    public static void main(String[] args)
    {
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(11,11,11));
    }
}
```

2) Method Overloading: changing data type of arguments

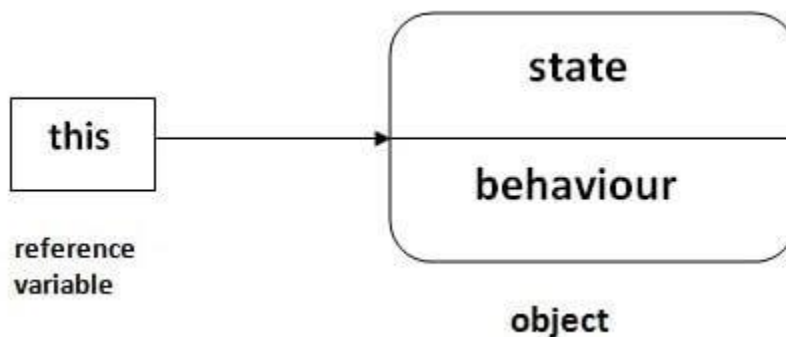
In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

Example 12

```
class Adder
{
    static int add(int a, int b)
    {
        return a+b;
    }
    static double add(double a, double b)
    {
        return a+b;
    }
}
class TestOverloading2
{
    public static void main(String[] args)
    {
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(12.3,12.6));
    }
}
```

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.



Usage of java this keyword

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.

this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
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 TestThis
{
    public static void main(String args[])
    {
        Student s1=new Student(111,"ankit",5000);
        Student s2=new Student(112,"sumit",6000);
        s1.display();
        s2.display();
    }
}
```