OOP-Unit-5 Notes

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. It can't be physical.

A class in Java can contain:

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

Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

What is an object in Java

An entity that has state and behavior 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 an intangible object is the banking system.

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.

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.

Example:

```
class A
{
        int s=2;
        public static void main(String args[])
        {
                A a=new A();  //object creation
                System.out.println(a.s)
        }
}
```

Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

```
class Student{

 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=05;
  s1.name="Ayushman";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
   }
}
```

Method in Java

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

The most important method in Java is the **main()** method.

Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point

User defined Method

Once we have defined a method, it should be called. The calling of a method in a program is simple. When we call or invoke a user-defined method, the program control transfer to the called method.

Example:

```
import java.util.*;
public class EvenOdd
{
public static void main (String args[])
{
//creating Scanner class object
Scanner sc=new Scanner(System.in);
System.out.print("Enter the number: ");
//reading value from user
int n=sc.nextInt();
//method calling
checkEvenOdd(n);
}
//user defined method
public static void checkEvenOdd(int n)
{
//method body
if(n%2==0)
System.out.println(num+" is even");
else
System.out.println(num+" is odd");
}
}
```

Parameter Passing/Call by value

If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example:

```
class Operation{
 int data=50;
 void change(int data){
 data=data+100;//changes will be in the local variable only
 }
   public static void main(String args[]){
       Operation op=new Operation();

   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);
   }
}
```

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

Java Recursion Example: Factorial Number

```
Import java.util.*;
public class RecursionExample3 {
    static int factorial(int n){
          if (n == 1)
          return   1;
          else
            return(n * factorial(n-1));
    }

public static void main(String[] args) {
Scanner sc=new Scanner(System.in);
int n=sc.nextInt();
System.out.println("Factorial="+factorial(n));
}
}
```

Method Overloading in Java

If a class has multiple methods having same name but different in 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.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading increases the readability of the program.

Different ways to overload the method

Method Overloading:

```
// Java program to demonstrate working of method
// overloading in Java

public class Sum {

    // Overloaded sum(). This sum takes two int parameters
    public int sum(int x, int y) { return (x + y); }

    // Overloaded sum(). This sum takes three int parameters
    public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }

    // Overloaded sum(). This sum takes two double
    // parameters
    public double sum(double x, double y)
    {
        return (x + y);
    }

    // Driver code
    public static void main(String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}
```

Java 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.

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

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:

<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 Program to create and call a default constructor
class Car{
//creating a default constructor
Bike1()
{
System.out.println("Car is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Car();
}
}
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){
System.out.println(id+" "+name);}
public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display();
}
}
```

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(123,"Anand");
    Student4 s2 = new Student4(345,"Pradeep");
    //calling method to display the values of object
    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.

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(123,"Anand");
    Student5 s2 = new Student5(345,"Pradeep",25);
    s1.display();
    s2.display();
    } }
```

Access Control/Modifiers in Java

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.

Java static keyword

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).

Example of static variable

```
//Java Program to demonstrate the use of static variable
class Student{
   int rollno;//instance variable
   String name;
   static String college ="PIET";//static variable
   //constructor
   Student(int r, String n){
   rollno = r;
   name = n;
   }
   //method to display the values
   void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(110,"Aman");
 Student s2 = new Student(222,"Mahesh");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();
 }
}
```

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 = "PIET";
  //static method to change the value of static variable
     static void change(){
     college = "PIT";
     }
     //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,"Pavan");
    //calling display method
    s1.display();
    s2.display();
    s3.display();
    }
}
```

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

1) Java final variable

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

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;   //Compile time error
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

2) Java final method

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

3) Java final class

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

Java Inner Classes (Nested Classes)

Java inner class or nested class is a class that is declared inside the class or interface.

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

Additionally, it can access all the members of the 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 three advantages of inner classes in Java. They are as follows:

1. Nested classes represent a particular type of relationship that is **it can access all the members (data members and methods) of the 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.

Example:

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  TestMemberOuter1.Inner in=obj.new Inner();
  in.msg();
 }
}
```