Objects and Classes in Java

Java Classes

A class in Java is a set of objects which shares common characteristics/ behavior and common properties/ attributes. It is a user-defined blueprint or prototype from which objects are created. For example, Student is a class while a particular student named Ravi is an object.

Properties of Java Classes

- Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
- Class does not occupy memory.
- Class is a group of variables of different data types and a group of methods.

A Class in Java can contain:

- Data member
- Method
- Constructor
- Nested Class
- Interface

Class Declaration in Java

```
access_modifier class <class_name>
{
   data member;
   method;
   constructor;
   nested class;
   interface;
}
```

Java Objects

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.

Objects are the instances of a class that are created to use the attributes and methods of a class.

Object and Class Example: main within the class

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

Save File: Student.java

//Java Program to illustrate how to define a class and fields

//Defining a Student class.

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

Object and Class Example: main outside the class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

Save File: TestStudent1.java

//Java Program to demonstrate having the main method in

//another class

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

1) **Object and Class Example: Initialization through reference**

Save File: TestStudent2.java

```
class Student
{
 int id;
 String name;
}
class TestStudent2
{
 public static void main(String args[])
{
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonu";
  System.out.println(s1.id+" "+s1.name);
 }
}
```

2) Object and Class Example: Initialization through method
Save File: TestStudent4.java

```
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();
 }
}
```

3) Object and Class Example: Initialization through a constructor

Save File: TestEmployee.java

```
class Employee
{
   int id;
   String name;
   float salary;
   void insert(int i, String n, float s)
 {
      id=i;
      name=n;
      salary=s;
   }
   void display()
{
System.out.println(id+" "+name+" "+salary);
}
}
public class TestEmployee
 {
public static void main(String[] args)
 {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
    e3.insert(103,"nakul",55000);
    e1.display();
    e2.display();
    e3.display();
}
}
```