UNIT III

Basics of objects and classes in java

Java classes is a blueprint or template used to create object.

It serves as a fundamental building block in Java programming, encapsulating data (fields) and behaviors (methods) into a single unit. you specify the attributes and behaviors that objects of that class will possess.
The attributes, also known as fields or instance variables, represent the state or characteristics of objects.
The behaviors, represented by methods, define the actions that objects can perform.

Components of Java Class

In Java, a class serves as a blueprint for creating objects. It encapsulates data and behavior into a single unit. Here are the main components of a Java class:

- **Class Declaration**: The class declaration defines the name of the class and any inheritance or interfaces it implements.

```
public class MyClass {
   // class body
```

}

- **Fields (Instance Variables): Fields represent the state or attributes of objects created from the class.**

private int age;

- **Methods: Methods define the behavior or actions that objects of the class can perform.**

```
public void display() {
   System.out.println("Age: " + age);
}
```

Rules for creating a class

In order to create a class, these rules must be followed-

- The “class” keyword should be used.
- The name of the class should start with an uppercase letter.
- The Java file can contain any number of classes but should not have more than one public class. The file name should be named after the public class followed by the “.java” extension.
- One class should only inherit another single class.

Java Objects

A Java object is an instance of a class. It represents a specific realization of the class blueprint, with its own unique set of data values for the fields defined in the class.
Objects are created using the new keyword followed by the class name, along with any required arguments to initialize the object's state. Each object created from a class has its own separate memory space allocated for its fields, allowing it to maintain its own state independent of other objects created from the same class.
Objects encapsulate both data (fields) and behavior (methods) into a single unit. They can interact with each other by invoking methods and accessing fields. In essence, objects are the building blocks of object-oriented programming in Java, allowing developers to model real-world entities and create modular, reusable, and maintainable software components.
An object consists of :

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by the methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

Syntax of an object

The syntax for creating an object in Java is:

```
ClassName objectName = new ClassName();
```

For example:

```
Car myCar = new Car();
```

Here, **Car** is the class name, **myCar** is the object name, and **new Car()** instantiates a new object of the **Car** class.

Difference Between Java Classes and Object

There are many differences between object and class. A list of differences between object and class are given below:

No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}
5)	Object is created **many times** as per requirement.	Class is declared **once.**
6)	Object **allocates memory when it is created.**	Class **doesn't allocated memory when it is created.**
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class keyword.

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

1) 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=101;

s1.name="Sonoo";

System.out.println(s1.id+" "+s1.name);//printing members with a white space
}
```

```
}
```

Output:

```
101 Sonoo
```

We can also create multiple objects and store information in it through reference variable.

File: TestStudent3.java

```
class Student{
int id;
String name;
}
class TestStudent3{
public static void main(String args[]){
//Creating objects
Student s1=new Student();

Student s2=new Student();
//Initializing
objects s1.id=101;
s1.name="Sonoo";
s2.id=102;
s2.name="Amit";
```

```
//Printing data System.out.println(s1.id+" "+s1.name);

System.out.println(s2.id+" "+s2.name);

}
}
```

Output:

```
101 Sonoo
102 Amit
```

1) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

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();
}
}
```

Output:

2) Object and Class Example: Initialization through a constructor

We will learn about constructors in Java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

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();
}
}
```

Output:

Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

File: TestRectangle1.java

```
class Rectangle{
int length;
int width;
void insert(int l,
int w){length=l;
width=w;
}
```

```
void calculateArea(){System.out.println(length*width);}
}
class TestRectangle1{
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
```