OOPs in Java (IT2C01)

Dr. Vijaypal Singh Rathor, Assistant Professor

CSE Discipline, PDPM IIITDM Jabalpur

Outline

1. Class
2. Object
3. Variable
4. Method
5. Constructor

Class

- A class is mainly a user defined blueprint to create any object.
- A Java program consists of one or more classes.
- It is the abstract descriptions of objects.
- Declaration of a Class includes some components:
  - Modifiers
  - Class name
  - Superclass (optional)
  - Interfaces (optional)
  - Body

Class: Example

```
public class Cat {
        String name;
        int age;
        String color;
        void barking() {
        }
        void sleeping() {
        }
        void hungry() {
        }
}
```

Object

- An object is an instance of a class.
- It is a basic unit of object oriented programming and represents the real life entities.
- Object can be tangible and intangible.
- Object has three characteristics:
  - **State:** It represents the data (value) or attributes of an object.
  - **Behavior:** It represents the behavior (functionality) of an object. It also reflects the response of an object with other objects.
  - **Identity:** An object identity is typically implemented by an unique name to interact with other objects.

Object (Cont…)

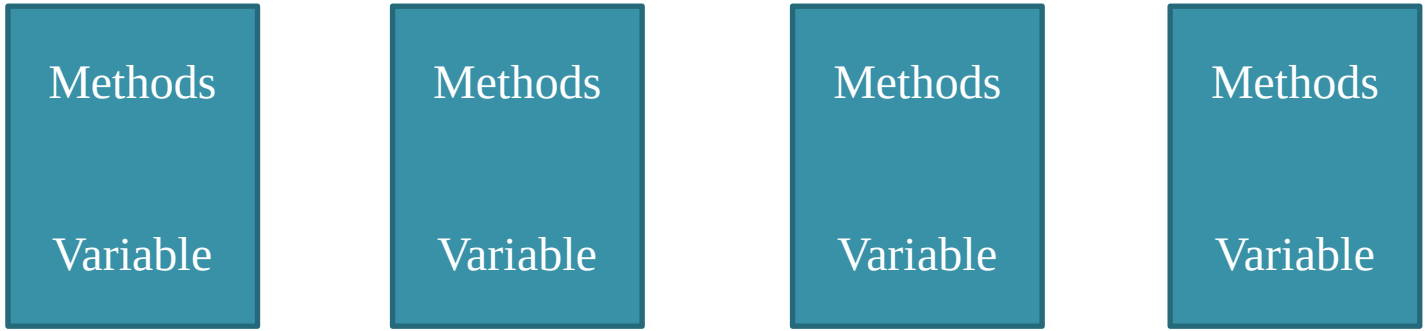
- Let us consider that Bike is an object and its name is Pulsar.
- The color of the Pulsar can be Black, which is known as its state.
- Pulsar is used to ride from one place to another, so riding is its behavior.

Object (Cont…)

- When an object is created, the class of the object is instantiated.
- "new" keyword instantiates a class and allocates new memory for objects.
- All the instances share the attribute and behavior of the class. But the values of those attributes are unique for each object.
- A class can have many instances:
  - Pulsar Obj1_Pulsar1 = new Pulsar()
  - Pulsar Obj2_Pulsar2 = new Pulsar()
  - Pulsar Obj3_Pulsar3 = new Pulsar()

Object (Cont…)

- Heap Memory



Object : Example-1

```
class Object
{
  int ID=101;
  String Name="Suyel";
  public static void main(String args[])
  {
     Object Obj=new Object();
     System.out.println(Obj.ID);
     System.out.println(Obj.Name);
 }
}
```

- Output

  101
  Suyel

Object: Example-2

```
class Abc
{
    void init()
    {
        int ID=101;
        String Name=“Ajay";
        System.out.println(ID+Name);
    }
    public static void main(String args[])
    {
        Abc Obj=new Abc();
        Obj.init();
    }
}
```

- Output

  101Ajay

Creating Object in Other Class

```
//Object_Par.java
class Object_Par
{
    int ID=101;
    String Name="Suyel";
}
```

```
//Object_Child.java
class Object_Child{
    public static void main(String args[]){
        Object_Par Obj=new Object_Par();
        System.out.println(Obj.ID);
        System.out.println(Obj.Name);
    }
}
```

- Compile both the class i.e. Object_Par and Object_Child
- Run Object_Child
- **Output:**

  101

  Suyel

Variable

- There can be three variable types in a class:
  - **Local variables:** Local variables are declared within the method. These variables are destroyed, when method is completed.
  - **Instance variables:** Instance variables are declared within a class, but outside any method. These type of variables can be accessed from inside any method.
  - **Class variables:** Class variables are declared within a class with the static keyword, but outside any method. If changes are made to the class variable, all other instances can see the effect of the changes.

Variable: Example of local variables

```
class Abc
{
    void init()
   {
       int ID=101;
       String Name=“Vijay";
   }
}
```

```
class Main
{
     public static void main(String args[])
    {
        Abc Obj=new Abc();
        System.out.println(Obj.id);
    }
}
```

Variable (Cont…)

- **Example of instance variables**

```
class Counter
{
      int count=0;          //instance variable
      void Counter1()
     {
           count++;
           System.out.println(count);
     }
     public static void main(String args[])
    {
         Counter C1=new Counter();
         Counter C2=new Counter();
         Counter C3=new Counter();
         C1.Counter1();
         C2.Counter1();
         C3.Counter1();
    }
}
```

❖**Output**

1

1

1

Variable (Cont…)

- **Example of class variables**

```
class Main
{
     static int count=0;          //will get memory only once and retain its value
     void Co2()
     {
         count++;                   //incrementing the value of static variable
         System.out.println(count);
     }
     public static void main(String args[])
     {
         Main c1=new Main();
         Main c2=new Main();
         Main c3=new Main();
         c1.Co2();
         c2.Co2();
         c3.Co2();
     }
}
```

❖**Output**

1
2
3

Variable: Combination

Student.java

```
class Student{
    int rollno;     //instance variable
    String name;
    static String college ="ITS";     //static variable
    void  Student1(int r, String n) {
        rollno = r;
        name = n;
    }
    void display () {
   System.out.println(rollno+" "+name+" "+college);
    }
}
```

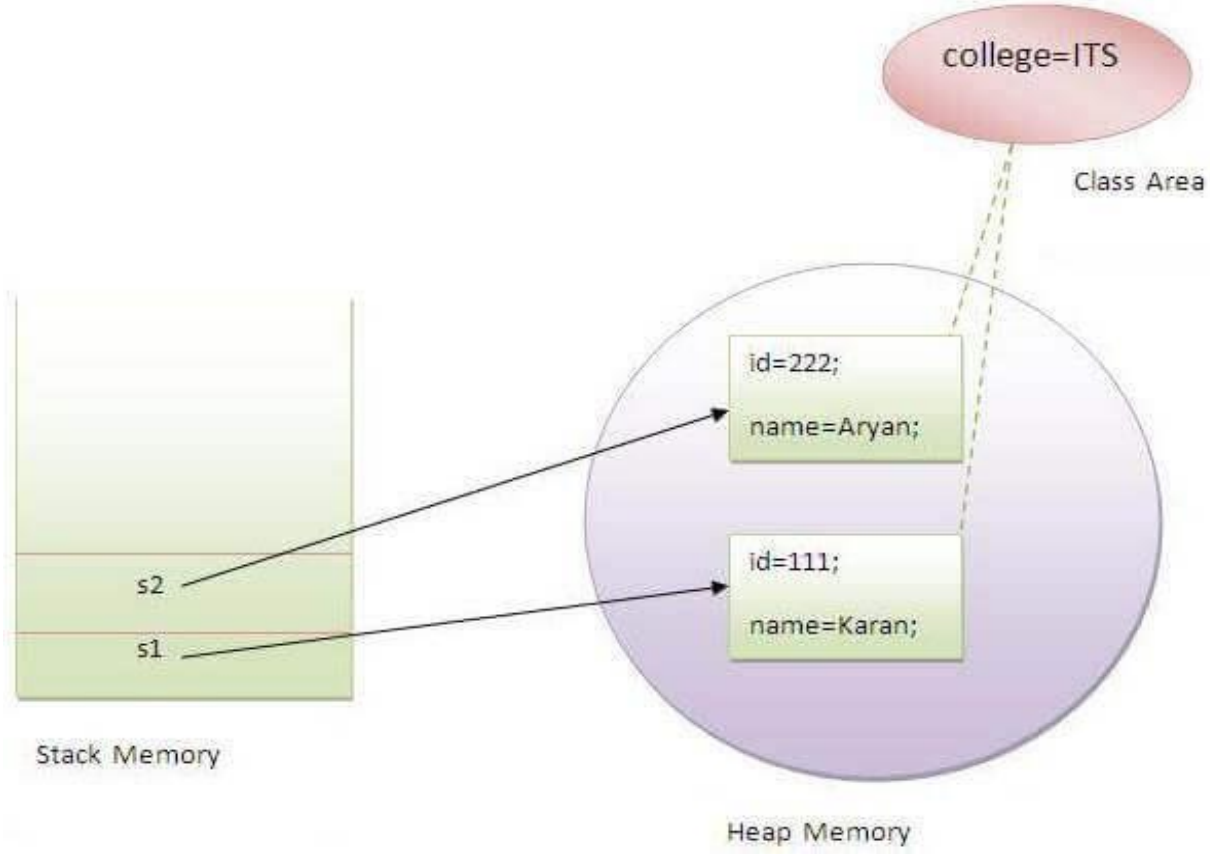
❖**Output**

111 Karan ITS
222 Aryan ITS

TestStaticVariable1.java

```
public class TestStaticVariable1
{
    public static void main(String args[])
   {
        Student s1 = new Student();
        Student s2 = new Student();
        s1.Student1(111, Karan);
        s2.Student1(222, Aryan);
        s1.display();
        s2.display();
    }
}
```

Variable (Cont…)



Creating Object in Class

- **Creating multiple objects by one type**
  - It is most popular.
  - Normally, different objects of a class is required in many methods. So, we create a reference and use it as per our requirements.
  - Memory wastage is less.

```
Hello multiple;   //reference variable
multiple = new Hello();   //object creation
```

Initialization of an Object

- There are three ways to initialize an object:
  - Initialization through reference
  - Initialization through method
  - Initialization through **a constructor**

Initialization Through Reference

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

```
class Employee
{
  int ID;
  String Name;
}
class Object_Initialization
{
   public static void main(String args[])
   {

     //Creating objects
     Employee Emp1=new Employee();
     Employee Emp2=new Employee();

     //Initializing objects
     Emp1.ID=2019;
     Emp1.Name="Suyel";
     Emp2.ID=2016;
     Emp2.Name="Arpit";

     //Printing data
     System.out.println(Emp1.ID+" "+Emp1.Name);
     System.out.println(Emp2.ID+" "+Emp2.Name);
   }
}
```

- Output

  2019 Suyel
  2016 Arpit

Method in Java

- **A method** is a block of code, which only runs when it is called.
- We can pass data known as parameters/arguments into a method.
- Method is used to perform certain actions.
- A method defines the behavior of any object.
- It defines an interface to expose the state of an object.

Method in Java (Cont…)

- Why we use method?
  - **To reuse code:** It defines the code once and we can use the same code (method) many times.
- All the methods are defined within the class.
- Java provides some pre-defined methods, such as println(). We can create our own method to perform certain actions.
- **Example:**

```
<Access Specifier> <Return Type> <Method Name> (arguments)
    {
       Statements or behavor;
        return <Variable Name>    //return is used, when there is a return
    }
```

Method in Java (Cont…)

- **Method Name**: A method has a unique name within the class in which it is defined. However, a method might have the same name as other method names within the same class because Java supports method overloading.

- **Parameter List:** Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, we must use empty parentheses ().

- **Method Body:** The code or operation that we need to execute.

Method in Java (Cont…)

- **Modifier:** It defines access type of the method i.e. from where it can be accessed in the application. There are 4 types of access specifiers in Java:
  - **public:** accessible in all the class in the application.
  - **protected:** accessible within the class in which it is defined and in its subclass.
  - **private:** Hide accessible only within the class in which it is defined.
  - **default** (declared/defined without using any modifier): accessible within same class and package within which its class is defined.
- **The return type** : The data type of the value returned by the method or void, if does not return a value.

Method in Java (Cont…)

- **Access Specifier**

Accessible to	public	protected	default	private
Same class	Yes	Yes	Yes	Yes
All classes in the same package	Yes	Yes	Yes	No
All sub classes in the different package	Yes	Yes	No	No
All classes in the different package	Yes	No	No	No

Example: Initializing Object Through Method

```
class Object_Initialization1
{
  int ID;
  String Name;
  void Record(int Identity, String Full_Name)
  {
    ID=Identity;
    Name=Full_Name;
  }
  void Display()
  {
    System.out.println("ID: "+ ID + ", " + "Name: " + Name);
  }
  public static void main(String args[])
  {
     Object_Initialization1 Emp1=new Object_Initialization1();
     Object_Initialization1 Emp2=new Object_Initialization1();
     Emp1.Record(2019, "Suyel");
     Emp2.Record(2016, "Arpit");
     Emp1.Display();
     Emp2.Display();
  }
}
```

❑**Output**

ID: 2019, Name: Suyel
ID: 2016, Name: Arpit

Examples of Method

Return from Method

```
class Method
{
   public static int square()
   {
      int i=45;
      return i*i;    // return
   }
   public static void main(String[] args)
   {
      int result;
      result = square();
      System.out.println("Squared of 45 is: " + result);
   }
}
```

❖**Output**

Squared of 45 is: 2025

Examples of Method (Cont…)

❑ Return with Arguments

```
public class Method1
{
   public int addition(int x, int y)     //Arguments
   {
      int z = x+y;
      return z;     //Return
   }
   public static void main(String args[])
   {
      Method1 return_argument = new Method1();
      int addition = return_argument.addition(4, 5);
      System.out.println("The sum of x and y is: " + addition);
   }
}
```

❖Output

The sum of x and y is: 9

Examples of Method (Cont…)

- **No Return from Method**

```
class Method2
{
  public void display()
  {
     System.out.println("Hello Friends");    //No Return
  }
  public static void main(String args[])
  {
    Method2 No_Return = new Method2();
    No_Return.display();
  }
}
```

❖Output

Hello Friends

Examples of Method (Cont…)

- **No Return with Arguments**

```
public class Method3
{
  public static void details(String Name, int Age)
  {
    System.out.println("Age of "+ Name +" is " + Age);
  }
  public static void main(String[] args)
  {
    details("Deep", 19);
    details("Karan", 19);
    details("Vani", 19);
  }
}
```

❖Output

Age of Deep is 19
Age of Karan is 19
Age of Vani is 19

Examples of Method (Cont…)

Static Method

- If we 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 only access static data member and can change the value of it.
- “this” and “super” cannot be used in static context.

Examples of Static Method

```
public class Method4
{
    private static int square(int x)
    {
        return x * x;
    }
    public static void main(String[] args)
    {
        for (int i = 0; i <= 3; i++)
        {
           int result = square(i);
           System.out.println("Square of the number i.e. " + i + " is : " + result);
        }
    }
}
```

❑**Output**

Square of the number i.e. 0 is : 0
Square of the number i.e. 1 is : 1
Square of the number i.e. 2 is : 4
Square of the number i.e. 3 is : 9

Examples of Static Method (Cont…)

```
class Method5
{
    public static int sum(int a, int b)
    {
        System.out.println("a = " + a + " b = " + b);
        int c = 0;
        c = a + b;
        System.out.println("c = " + c);
        return c;
    }
    public static void main(String args[])
    {
        int a = 34;
        int b = 56;
        int c = 0;
        sum(a, b);
        System.out.println("c = " + c);
    }
}
```

- Output

  a = 34 b = 56

  c = 90

  c = 0

Method Overloading

```
class Adder{
  int add(int a,int b)
  {
    return a+b;
  }

  int add(int a,int b,int c)
  {
    return a+b+c;
   }
}
```

```
class TestOverloading1{
      public static void main(String[] args)
      {
           Adder x=new Adder();
           System.out.println(x.add(1,2));
           System.out.println(x.add(1,2,3));
      }
}
```

Initialization Through Constructor

- Every class has a constructor.
- If we don’t define a constructor in a class, compiler builds a default constructor.
- **Constructor** is used to initialize any object.
- The name of the constructor is the same name as the class.
- A class can have many constructor.
- It does not have return type.

Constructor (Cont..)

- Here, we have shown the instance variable name of the object.
- Output is what we passed to the name during initialization in constructor.
- When the object is created, the constructor is invoked.
- “this” is a keyword that points to the current object.

```
public class Constructor
{
   String name;
   Constructor()       //Constructor
   {
      this.name = "Give me a party";
   }
   public static void main(String[] args)
   {
      Constructor obj = new Constructor();
      System.out.println(obj.name);
   }
}
```

Constructor (Cont..)

- There are three types of constructor:
  - **Default:** If we do not implement any constructor in a class, compiler uses a default constructor.
  - **No argument:** It is a constructor with no arguments.
  - **Parameterized:** This type of constructor has parameters or arguments.

Default Constructor

```
class Constructor
{
    public static void main (String args[])
   {
      Constructor obj=new Constructor()
    }
}
```

```
class Constructor
{
   Constructor()
   {
   }
   public static void main (String args[])
  {
      Constructor obj=new Constructor()
   }
}
```

No Argument Constructor

```
class Constructor1
{
      public Constructor1()
      {
          System.out.println("This is what we call no argument constructor");
      }
      public static void main(String args[])
      {
          Constructor1 java= new Constructor1();
      }
}
```

❖**Output**

This is what we call no argument constructor

Parameterized Constructor

```
class Constructor2
{
    int EmpID;
    String EmpName;
    Constructor2(int ID, String Name)
    {
        this.EmpID = ID;
        this.EmpName = Name;
    }
    void Bennett(){
        System.out.println("ID: " + EmpID + ", Name: " + EmpName);
    }
    public static void main(String args[])
    {
        Constructor2 Obj_Emp1 = new Constructor2(2019,"Suyel");
        Constructor2 Obj_Emp2 = new Constructor2(2016,"Arpit");
        Obj_Emp1.Bennett();
        Obj_Emp2.Bennett();
    }
}
```

❖**Output**

ID: 2019, Name: Suyel
ID: 2016, Name: Arpit

Parameterized Constructor (Cont…)

```
class Constructor3
{
   private int EmpID;
   public Constructor3()
   {
       this.EmpID = 2019;
   }
   public Constructor3(int ID)   //Parameterized constructor
   {
       this.EmpID = ID;
   }
   public int Bennett()
   {
       System.out.println("ID: " + EmpID);
       return EmpID;
   }
   public static void main(String args[])
   {
       Constructor3 Obj_Emp1 = new Constructor3();
       Constructor3 Obj_Emp2 = new Constructor3(2016);
       Obj_Emp1.Bennett();
       Obj_Emp2.Bennett();
   }
}
```

❖**Output**

ID: 2019
ID: 2016

Comparisons between Constructor and Method

Constructor	Method
Constructor is used to initialize an object.	Method is used to define functionality of an object.
Constructors are invoked implicitly.	Methods are invoked explicitly.
If constructor is not present, a default constructor is invoked by the compiler.	Here, no default method is invoked.
Constructor does not return any value.	Method may/may not return a value.
Constructor must have the same name as the class.	Method must not have the same name as the class.

Thank You