Day-3 JA-111 Batch: Object Oriented Programming

The ultimate objective of OOP is to represent the things in computer as they are observed in the real world. OOP is an approach to organize the code, it is giving us syntax to write the code, syntax is provided by programming languages like java, C++, python etc.

The OOP has following to study

1. class
2. object
3. Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism

Encapsulation, Inheritance and Polymorphism are considered as the key features of OOP.

Let us start discussion with the class and object, first go through the analogy first.

Let us consider an example of a ceiling fan that might be available over your head. Take a look at it and note down some attribute that describe about it like.

1. No. of blades
2. Company name
3. Colour
4. On/Off
5. Weight
6. Rotation Per Minute

And talking about operations, that changes the description of these components.

1. Turning it on from off and vice-versa
2. Changing its colour
3. Making it light in weight
4. Increasing/Decreasing rotation per minute

In the example of fan, that we have discussed above, we are using some attribute to describe a real world entity these attributes are called fields. The values in the fields define the **state**.

Apart from this, we are also discussing operations that make changes in the state of the real world entity these operations are known as **behaviours** i.e. operations are used to make changes in the state.

Now again coming back to our example and observe following scenario. Fan that you were observing is same as all other fan that have same model no and manufactured by same company in the terms of state and behaviour?

Not surprisingly, Answer to observations is yes.

Means we can say that before manufacturing the fans, someone have already define their states and behaviour, say on a document, and that document serve as a blueprint for all fans of same model.

Also that document itself is a passive entity; i.e. just defining states and behaviours in a document does not create a fan, but company made a real world entity by following that blueprint. That real world entity is called **object** while the document that serve as a blueprint for that real world object is called **class**.

Class

- A Class is the blueprint from which individual objects are created.
- It is a passive entity.
- It is a user defined data type.
- It is a template that unites data and code together into a single entity i.e. this is called basic unit of encapsulation.

Object

- An object is an instance of a class that contains fields defined in the class and exhibit behaviour.
- It is a real world entity.
- An object stores its state in *fields* (variables in some programming languages) and exposes its behaviour through *methods*.
- An object for class is what a variable for primitive type is.

Abstraction

Now! Again come back to the example of fan. In the season of summer you are coming to home from office/college and feeling tired then you go to your drawing room, switching on fan, sitting comfortably on chair and start enjoying cool air.

In the situations that we have discussed above, do you need to bother about how a fan rotates?

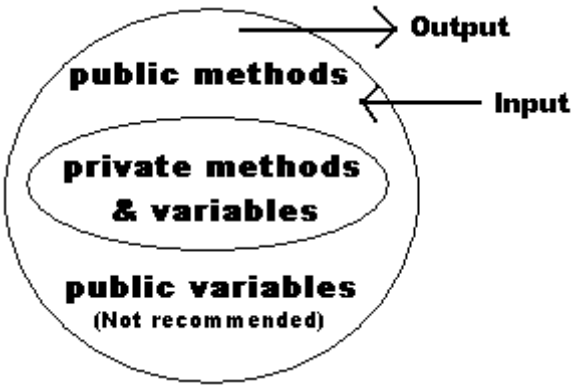
Simple answer: No! You are able to use an object even you don't have to bother about its internal working. You just need to have operating knowledge only so you are using an object without knowing its internal mechanism and this is what abstraction is.

Encapsulation

General meaning of encapsulation is

> "To completely cover something especially so that it will not touch anything else."

The wrapping up of data and its associated function together in a single unit (called class) and keeps both safe from outside interference and misuse is known as encapsulation. Encapsulation works as a wrapper that protect data and code from arbitrary access by other code defined outside the wrapper.



private variable and methods are covered by public methods & variables, but creating public variable is not recommended

To protect the data from external interference concept of data hiding is used i.e. data is hidden and cannot be accessed by external world. This is achieved with the help of access specifiers. Data hiding is an integral part of encapsulation.

Rest of the concept of OOP, we will cover later

Let us take a look how to create class and object in java

Defining Class & object in java

General form a class in java is:

```
[access-modifier] class class-name{
    //field,constructor,and method declarations
}
```

access-modifier can be public only yet you can skip the access modifier. class is a keyword in java and class-name can be any valid identifier; best to stick to naming convention.

Creating an object is a two-step process-

Step 1: Creating a reference of class

```
class-name obj-name;
```

Actually a reference points to an object it is not an object actually.

Step 2: Allocating memory for object and assign it to reference.

```
obj-name = new class-name();
```

new is a keyword provided by java that is responsible for allocating memory dynamically, new operator take data type as an operand and allocate space equivalent to the size of data type and return a reference to memory allocated. One thing to note about new is, it allocate memory at run time.

For the sake of convenience, java allow us to merge both steps, just write

```
class-name obj-name = new class-name();
```

All variable that are defined within a class are called instance variable. To access members of class (i.e. variables & methods) we have to use dot(.) operator, that links object name with member. General form for accessing instance variable using object is:

```
obj-name.instance-variable-name
```

An Example:

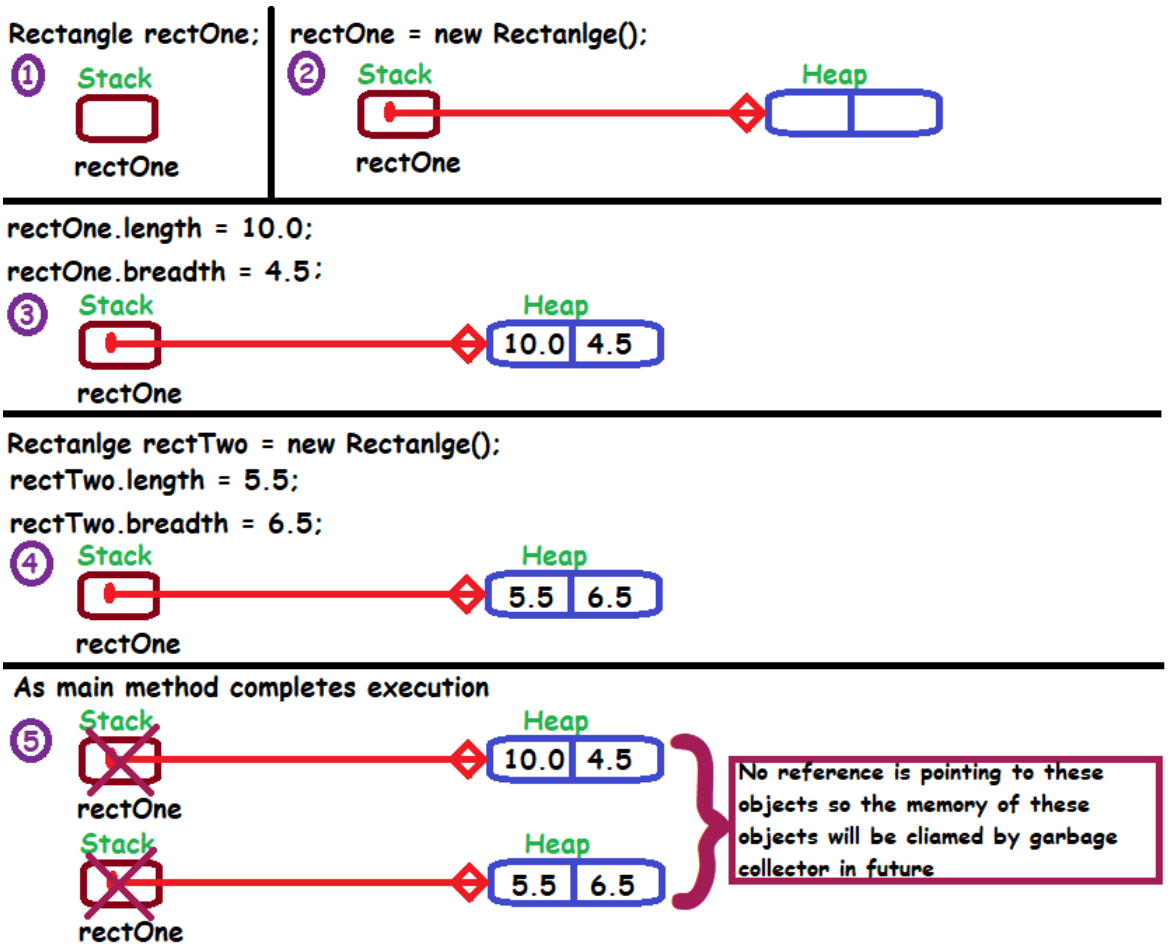
```
//filename: Rectangle.java
package com.masai.p1_class_obj_demo;
public class Rectangle {
   double length;
   double breadth;
}

//filename: RectangleDemo.java
package com.masai.p1_class_obj_demo;
public class RectangleDemo{
   public static void main(String[] args) {
      Rectangle rectOne;
      rectOne = new Rectangle();
      rectOne.length = 10.0;
      rectOne.breadth = 4.5;

      Rectangle rectTwo = new Rectangle();
      rectTwo.length = 5.5;
      rectTwo.breadth = 6.5;
```

```
        System.out.println("Area of first rectangle is " + (rectOne.length * rectOne.breadth));
        System.out.println("Area of second rectangle is " + (rectTwo.length * rectTwo.breadth));
    }
}
```

Output

Area of first rectangle is 45.0

Area of second rectangle is 35.75



Think: Why primitive data-types are not dynamic?

Java does dynamic memory allocation for arrays and object not for primitive data types. Reason behind implementing primitive data type not as an object is interest of efficiency, because creating an object at run time generates a significant overhead and java required to maintain attributes of each object separately. Java implements primitive data types simply to avoid overhead and achieve speed.

Whenever one reference variable is assigned to another reference variable then java does not make bit-wise copy of object being pointed by assigning reference instead it just create a copy of assigning reference, means after assignment we have another reference variable that points to similar object currently pointed by assigning reference. Hence, if we make changes in values of instance variable using one reference variable then changes will be effective for another reference because ultimately they are pointing to same object.

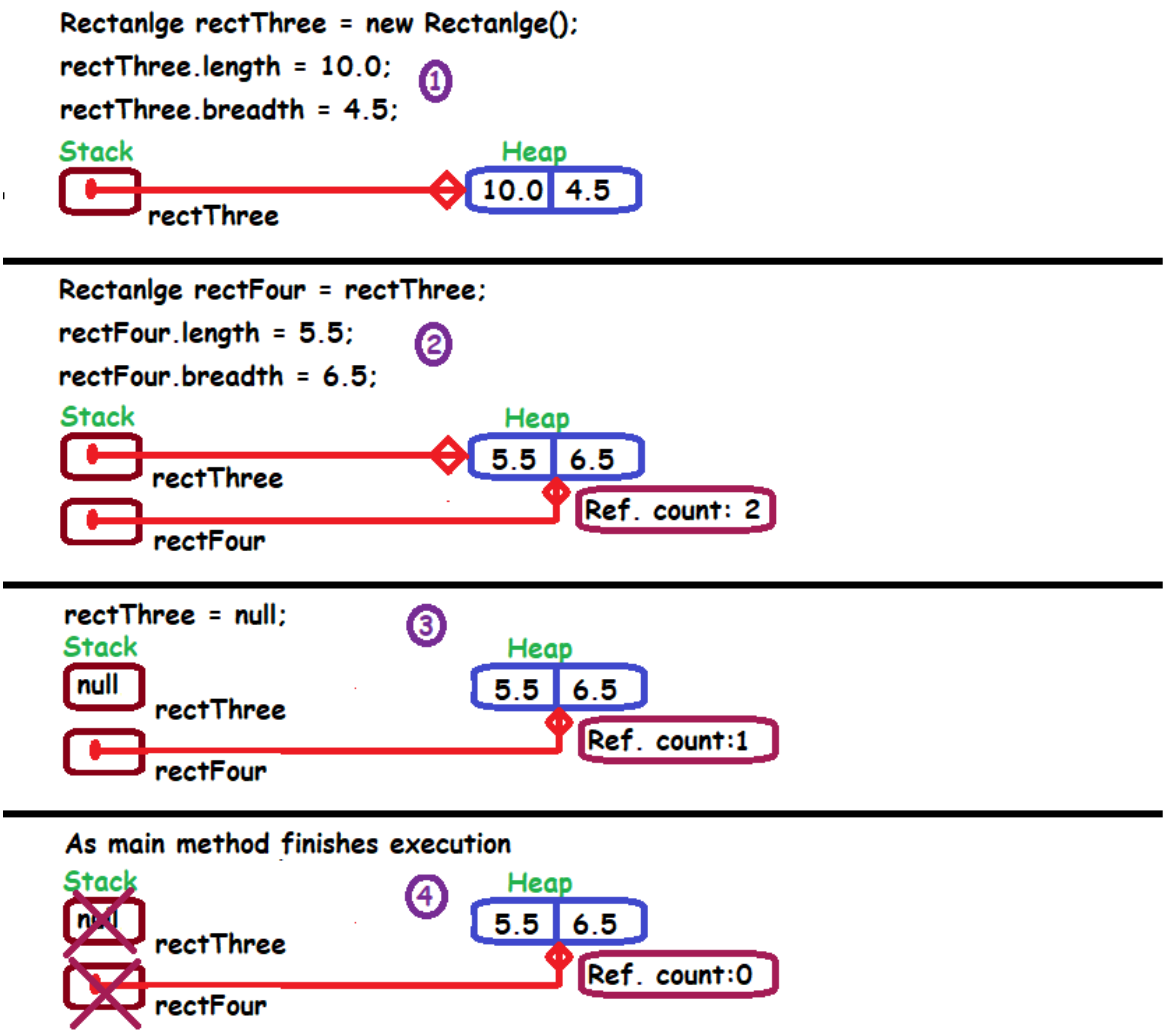
An Example

Keep the file Rectangle.java as it is

```
package com.masai.p1_class_obj_demo;
```

```
public class RectangleDemo {
    public static void main(String[] args) {
        Rectangle rectThree = new Rectangle();
        rectThree.length = 10.0;
        rectThree.breadth = 4.5;

        Rectangle rectFour = rectThree;
        rectFour.length = 5.5;
        rectFour.breadth = 6.5;

        System.out.println("Area for rectThree is " + (rectThree.length * rectThree.breadth));
        System.out.println("Area for rectFour is " + (rectFour.length * rectFour.breadth));

        rectThree = null;
        System.out.println("Bye Bye");
    }
}
```

Output
Area for rectThree is 35.75
Area for rectFour is 35.75
Bye Bye



Tip: One object can be pointed by any number of references but a reference can point to one object only.

Method

Method refers to a block of code that is used to perform a specific operation. Method increases modularity of the code.

General form of declaring a method is

```
[access-specifier] return-type method-name(parameter list) {
        //body of method
}
```

- The access-specifier can public, protected or private yet it can be skipped.
- return-type defines type of value to be returned by method. if a method does not return a value then return-type must be void.
- Method-name is simply name of method, must be a valid identifier and should be according to the naming convention standards.
- parameter-list is simply sequence of type and identifiers separated by comma. If a method do not have any parameter, then simply place a blank pair of parenthesis after method-name.

To call a method outside the class we have to use dot(.) operator. to call a method simply write

```
object-name.method-name(argument-list);
```

- A parameter is a variable defined by a method and receives a value when a method is called and an argument is a value that is passed when a method is called.

- A method of a class is always called relative to an object; and once invocation took place, the calling object is known then instance variable of this object can be accessed directly.

An Example

```
//filename: Rectangle.java
package com.masai.p1_class_obj_demo;

public class Rectangle {
    double length;
    double breadth;

    void setDimension(double l, double b) {
        length = l;
        breadth = b;
    }

    double getArea() {
        double area = length * breadth;
        return area;
    }
}

//filename: RectangleDemo.java
package com.masai.p1_class_obj_demo;

public class RectangleDemo {
    public static void main(String[] args) {
```

```
        Rectangle rectOne = new Rectangle();

        rectOne.setDimension(10.5, 9.5);
        double area = rectOne.getArea();
        System.out.println("Area of rectOne is " + area);

        Rectangle rectTwo = new Rectangle();
        rectTwo.setDimension(4.5, 5.5);
        System.out.println("Area of rectOne is " + rectTwo.getArea());
    }
}
```

Output
Area of rectOne is 99.75
Area of rectOne is 24.75

Parameter passing mechanism in java

- Primitive data types are always call by value hence changes made in parameter does not affect arguments.
- Array, objects are passed by reference hence changes made in parameter affects arguments.

An example of passing values of primitive data type

```
class A{
    void show(int a){
        a = a + 10;
    }

    public static void main(String args[]){
        A a = new A();
        int i = 20;
        System.out.print(i + " ");
        a.show(i);
        System.out.print(i);
    }
}
```

Output: 20 20

An example of passing object

```
class P{
    int i, j;

    void display(){
        System.out.println(i + " " + j);
    }
}

class Q{
    void change(P temp){
        temp.i = temp.i + 10;
        temp.j = temp.j + 20;
    }
```

```
    public static void main(String args[]){
        P p = new P();
        p.i = 10;
        p.j = 20;
        p.display();

        Q q = new Q();
        q.change(p);
        p.display();
    }
}
```

Output
10 20
20 40

Polymorphism

Taking example of your mobile phone again, Each mobile have call button (having green icon of phone) with it now consider observe reaction of pressing call button in following situations

1. Let say you have an incoming call on your phone. You are pressing call button, now you can talk to person who have make call to you.
2. Let say you have received a message and you are reading it, at the time of reading press call button you will get list of all numbers that are in message as well as mobile number of sender.
3. Let say you are browsing your dialled number list and on when your control is on a particular number again press call button, now you are calling on that particular number.

From the above discussion we can make a simple observation that applying same behaviour (that is, pressing call button) on an object (that is, mobile) react differently in different situation. That feature is known as polymorphism.

Simple we can say polymorphism means many forms. It is a feature which allows one interface to be used for general class of action and specific action is determined by nature of situation. More specifically it can be defined as “single interface, multiple methods”

It is of two types

Compile time polymorphism/Early binding/Static Binding

The method to be called will be decided by the compiler i.e. which method will be executed is decided at the compile time that’s why it is Compile time polymorphism
Method Overloading

- More than one method has same name
- There parameters must be different (may be by type or quantity or order)
- Return type does not play any role in method overloading
- Tip: Use method overloading when more than one method is performing same type of task

An Example

```
package com.masai.p1_class_obj_demo;

class Triangle{
    double calculateArea(double base, double height) {
        double area = 0.5 * base * height;
        return area;
    }
```

```
   double calculateArea(double a, double b, double c) {
      double s = (a + b + c)/2;
      double area = Math.sqrt(s * (s - a) * (s - b) * (s - c));
      return area;
   }

   public static void main(String args[]) {
      Triangle t = new Triangle();
      System.out.println("Area of traingle for base 12 cm and perpendicular 5 cm is " + t.calculateArea(5, 12));
      System.out.println("Area of traingle for sides 7, 49 and 50 cm is " + t.calculateArea(7, 49, 50));
   }
}
```

Output
Area of traingle for base 12 cm and perpendicular 5 cm is 30.0
Area of traingle for sides 7, 49 and 50 cm is 171.0438540258024

Runtime polymorphism/Late binding/Dynamic Binding
The method to be called will be decided by the run time system i.e. which method will be executed is decided at the run time that's why it is run-time polymorphism. Its implementation is provided using method overriding that we will discuss with inheritance.

You Activity

Create class with named SimpleInterest with three variables for principle amount, time period and interest rate of double data type.

Create two methods with following proto-type

```
void setValues(double pa, double tp, double ir){
     //write code to assign value here
}
```

and

```
double getInterestAmount(){
     //write code to computer and return the interest amount rounded to two decimal places
}
```

//Use following code to round up to two decimal places (Math.round(value * 100))/100.0
//Tip: Say the interest amount is 253.2659874 then (Math.round(253.2659874 * 100))/100.0 so it should return 253.27

```
class SimpleInterestTester{
     public static void main(String args[]){
          SimpleInterest siOne = new SimpleInterest();
          SimpleInterest siTwo = new SimpleInterest();

          siOne.setValues(1005, 2, 7.5);
          siTwo.setValues(1235.50, 2.5, 8.25);
     }
}
```

Sample Output

Simple interest amount for siOne is 150.75
Simple interest amount for siTwo is 254.82