

## Shallow Copy and Deep copy :

### Shallow Copy :

- \* In Shallow, We will create **only one object** and same existing object will be referred by multiple reference variable.
- \* In this shallow copy, If we modify the content of one object by any of the reference variable then original object will be modified.

### Diagram for the Code :

```
Laptop lap1 = new Laptop("HP", 80000);
System.out.println(lap1);

Laptop lap2 = lap1;
lap2.setBrand("HP IRIS");
lap2.setPrice(90000);
```



### Deep Copy :

- \* In Deep copy we will create two different objects, Here one object will copy the data of another object.
- \* Here If we modify the content of one object then another objec content will not be modified.

### package com.ravi.deep\_copy;

```
public class Employee
{
    private int id;
    private String name;

    public Employee() //User-defined NO Argument Constructor
    {
        id = 0;
        name = null;
    }

    public Employee(int id, String name)
    {
        super();
        this.id = id;
        this.name = name;
    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    @Override
    public String toString()
    {
        return "Employee [id=" + id + ", name=" + name + "]";
    }
}
```

### package com.ravi.deep\_copy;

```
public class EmployeeDemo
{
    public static void main(String[] args)
    {
        Employee e1 = new Employee("HP", 80000);
        System.out.println("Before Change :");
        System.out.println(e1);

        Employee e2 = e1;
        e2.setId(90000);
        e2.setName("Raj");

        System.out.println("After Change :");
        System.out.println(e1);
        System.out.println(e2);
    }
}
```

### Pass By Value :

- \* Java does not support Pointers so, In java we have only **Pass by Value**
- \* Pass By Value sending the copy of Original Data to the Method.

### package com.ravi.pass\_by\_value;

```
public class PassByValueDemo1
{
    public static void main(String[] args)
    {
        int y = 100;
        accept(y);
        System.out.println(y);
    }

    public static void accept(int x)
    {
        x = 200;
    }
}
```

### package com.ravi.pass\_by\_value;

```
public class PassByValueDemo2
{
    public static void main(String[] args)
    {
        int y = 100;
        y = accept(y);
        System.out.println(y);
    }

    public static int accept(int x)
    {
        x = 200;
        return x;
    }
}
```

### package com.ravi.pass\_by\_value;

```
class Customer
{
    private double bill = 24000;
```

```
    public double getBill()
    {
        return bill;
    }

    public void setBill(double bill)
    {
        this.bill = bill;
    }

    @Override
    public String toString()
    {
        return "Customer [bill=" + bill + "]";
    }
}
```

### public class PassByValueDemo3

```
{
    public static void main(String[] args)
    {
        Customer c1 = new Customer();
```

```
        System.out.println(c1.getBill()); //24000
        accept(c1);
        System.out.println(c1.getBill()); //30000
    }
}
```

### public static void accept(Customer cust)

```
{
    cust.setBill(30000);
}
```

### package com.ravi.pass\_by\_value;

```
class Product
{
    private double price = 29000;
```

```
    public double getPrice()
    {
        return price;
    }

    public void setPrice(double price)
    {
        this.price = price;
    }

    @Override
    public String toString()
    {
        return "Product [price=" + price + "]";
    }
}
```

### public class PassByValueDemo4

```
{
    public static void main(String[] args)
    {
        Product p1 = new Product();
        System.out.println(p1.getPrice()); //29000
        accept(p1);
        System.out.println(p1.getPrice()); //29000
    }
}
```

### public static void accept(Product prod)

```
{
    prod = new Product();
    prod.setPrice(34000);
}
```

### Java Memory Management :

- \* All the static fields are strored in a special memory area called CLASS AREA OR METHOD AREA.

- \* All the non static fields are strored in a special memory area called HEAP AREA.

- \* All the local and parameter variables are strored in a special memory area called STACK AREA.

### What is Garbage Collection in Java ?

### -----

