java.lang.Object

- Object class is defined in java.lang package.
- Object class is the super most class for all the classes in java.
- In Object class there are 11 non-static methods.

non-static methods m of java.lang.Object class :

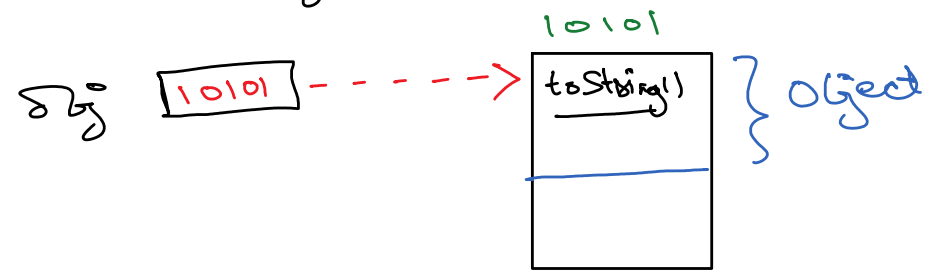
1	**public String toString()**
2	**public boolean equals(Object o)**
3.	**public int hashCode()**
4.	protected Object clone() throws CloneNotSupportedException
5.	protected void finalize() ---> deprecated
6.	final public void wait() throws InteruptedException
7	final public void wait(long) throws InteruptedException
8.	final public void wait(long , int) throws InteruptedException
9.	final public void notify() throws InteruptedException
10.	final public void notifyAll() throws InteruptedException
11.	final class getClass()

1. public String toString() :

- toString() returns String.
- toString() implementation of Object class returns reference of an object in String format.
- toString() of Object class is implicitly called every time, when a programmer tries to print the reference of an object.

```
class Fruit
{

}
```

```
Fruit obj = new Fruit();
```




```
S.o.pln(obj);   // 10101 ✗
```

// implicitly obj.toString() is called

```
className @ Hexadecimal-number
```

generated based on hashcode of object

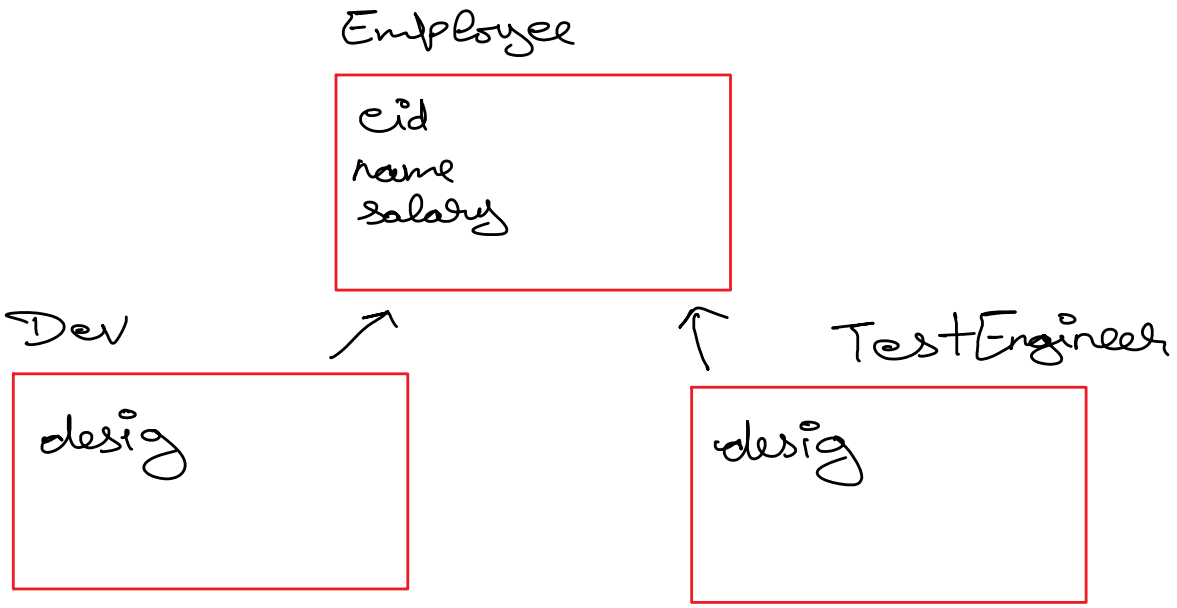
To Override toString() :

purpose:

- we override toString() to print state of an object instead of reference of the object.

for example refer, workspace/object_class/src/pack1/Demo3

Task1 :



User :

1.) Employee e1 ;

Case 1 : e1 = new Dev()
sop (e1);

Case 2 : e1 = new TE()
sop (e1);

2. public boolean equals(Object o) :

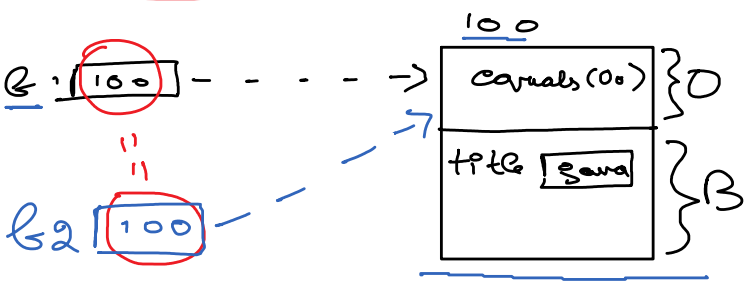
- the return type of equals method is boolean.
- For equals method we can pass reference of any object.
- The java.lang.Object class implementation of equals method is used to compare the reference of two objects. (it behaves exactly like == operator)

Example :

```
class Book
{
    String title;
     Book(String title) {
        this.title = title ;
    }
}
```

Case 1

```
Book b1 = new Book("Java");
Book b2 = b1;
```



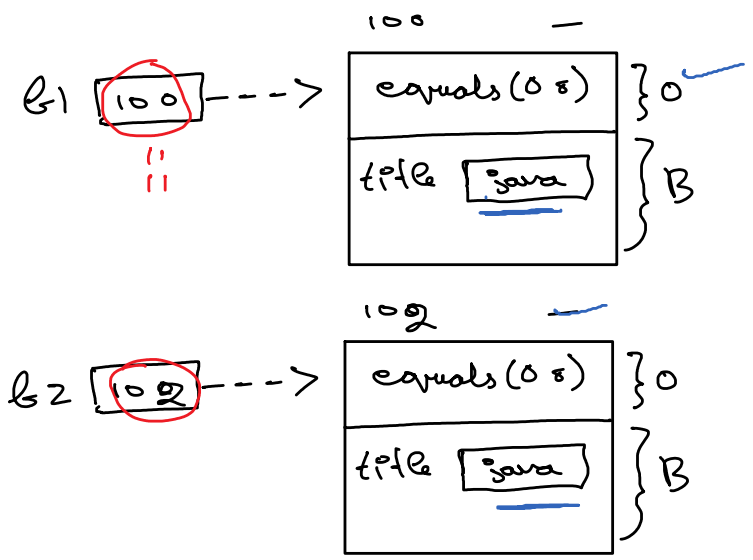

```
Sop(b1.title); // java
Sop(b2.title); // java
```

1) Sop(b1 == b2); //true

2) Sop(b1.equals(b2)); //true

Case 2

```
Book b1 = new Book("Java");
Book b2 = new Book("Java");
```




```
Sop(b1.title); // java
Sop(b2.title); // java
```

2.) sop (b1.equals(b2));
//true

refer,
workspace/object_class/pack2/ BookDriver1.java

sop (b2.title); // java

1.) sop (b1 == b2); false

2.) sop (b1.equals(b2)); false

refer,
workspace/object_class/pack2/ BookDriver2.java

Note:

- == operator compares reference,
  - if it returns true, it is reference of same object(1 object) **case1**
  - if it returns false, it is reference of different objects(2 objects) **case2**
- java.lang.Object class design of **equals(Object)** is very much similar to operator == (as seen in case1 & case2)

Pg4

Wednesday, April 29, 2020 8:29 AM

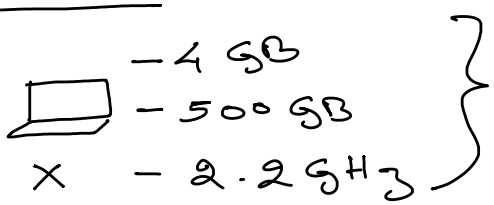
Case 1:

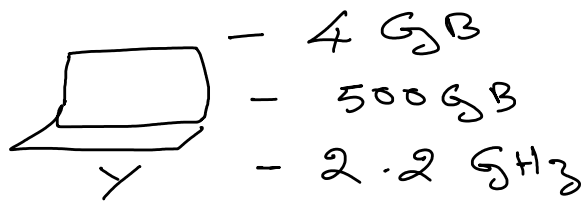
int i = 10 ; int j = 20

i 10 j 20

i == j → compare values

Case 2:



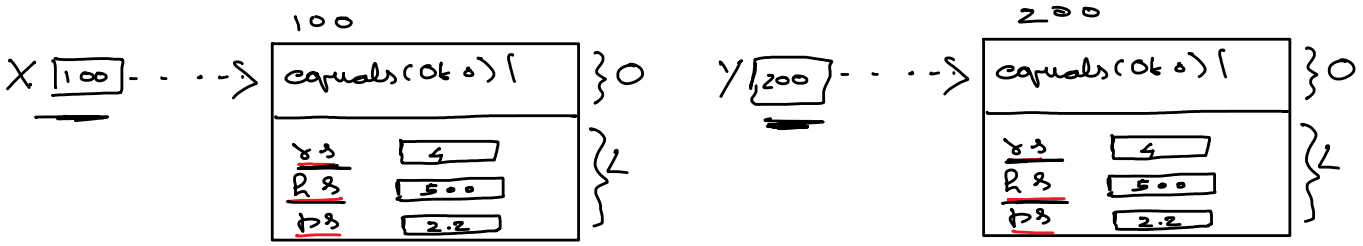



```
class Laptop
{
    int rs
    int hs
    double ps
    Laptop ( ){ }
    Laptop (int rs, int hs, double ps) {
        this.rs = rs ;
        this.hs = hs ;
        this.ps = ps ;
    }
    public String toString() {
```

```
        Sop("Ram_size:" + rs);
        Sop("Hard Disk:" + hs);
        Sop("processor speed:" + ps);
        return "";
    }
}
```

```
Laptop X = new Laptop(4, 500, 2.2);
Laptop Y = new Laptop(8, 500, 2.2);
Sop(X);   // toString() is called
Sop(Y);   //        "          "
```

// check whether config of X & Y is same or not



1.) `Sop(x == y);` // false

2.) `Sop(x.equals(y));` // false

current passed

How to change design of a method ?

→ override equals() in Laptop class

old design : was comparing reference of 2 laptop.

new design : should compare state of 2 laptops

```
class Laptop
{
    .
    .
    .
    // override equals method        this
    public boolean equals (Object o)
    {                                  o → 200
        // design
                    100
        return this.xs == ((Laptop) o).xs
                && this.hs == ((Laptop) o).hs
                && this.ps == ((Laptop) o).ps;
                \_____/      \_____/
                current      passed
    }
}
```

To Override equals(object) :

Purpose: We override equals method to compare the state of two objects instead of comparing reference of two objects.

design tip : (in equals method compare the state of current(this) object with passed object by downcasting it)

example refer, **workspace/object_class/src/pack2/Laptop.java**

Note :

1. if equals method is not overridden it compares the reference of 2 objects similar to == operator.
2. if equals method is overridden in a class, then it compares the state of objects and not the reference. in such case if we have to compare the reference it is possible only with the help of == operator.

Assignment1 :

Book

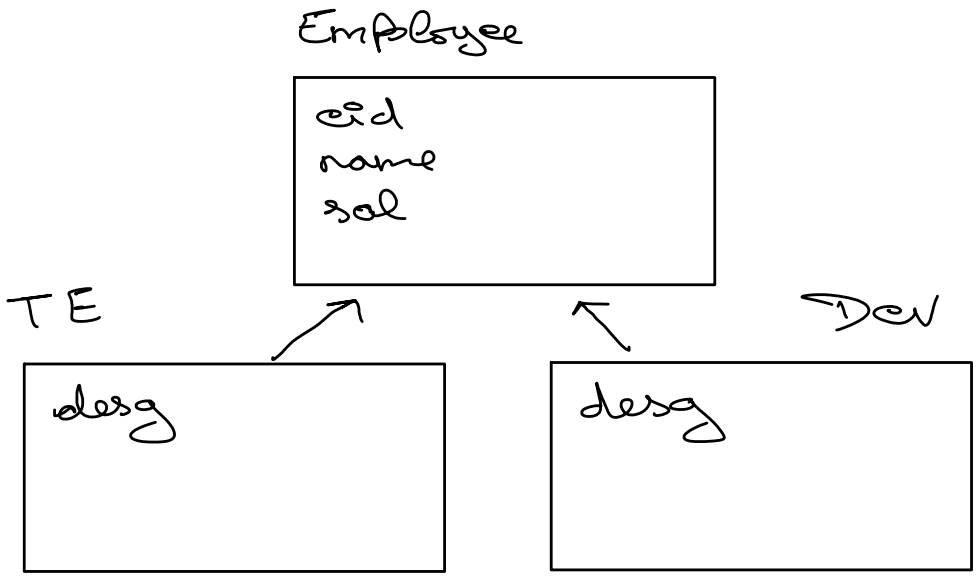
Bid Author_name title Price

1. design the class for Book,
2. user should be able to display the state easily
3. user should be able to compare 2 books easily
4. implement your design

Assignment2:

Employee

Assignment2:



1. design the class for employee hierarchy,
2. user should be able to display the state easily
3. user should be able to compare 2 employees easily
4. implement your design

3. public int hashCode():

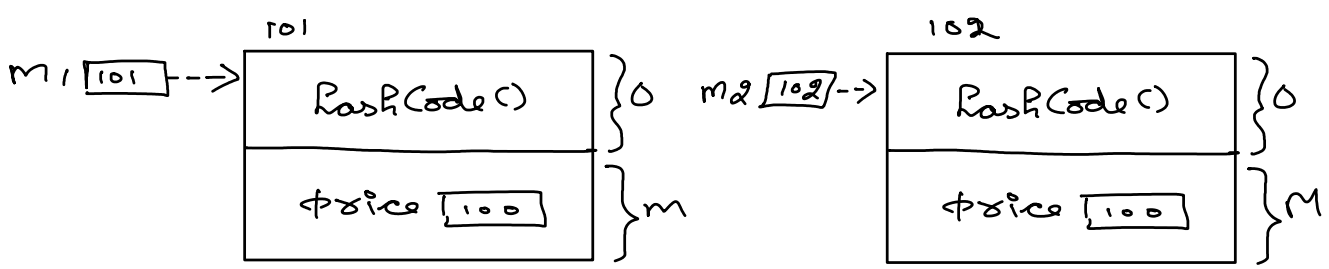
- hashCode() returns integer number
- java.lang.Object class implementation of hashCode() returns a unique integer number for every object created.
- The unique integer number is generated based on the reference of the object

example:

```
class Mango
{
    int price ;
    Mango(int price) {
        this.price = price ;
    }
}
```

Case 1:

```
Mango m1 = new Mango(100);
Mango m2 = new Mango(100);
```

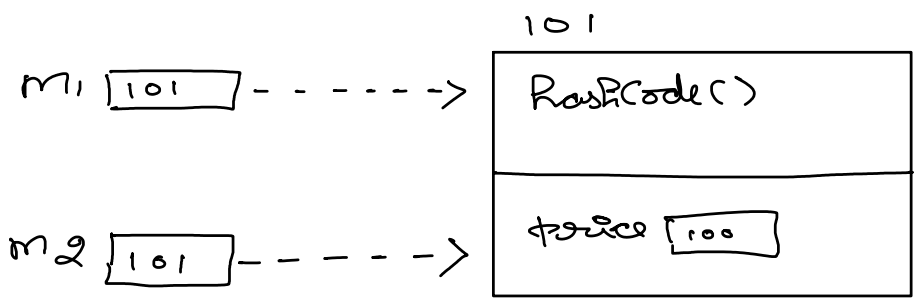



```
S.o.pln(m1.hashCode())
101

S.o.pln(m2.hashCode())
102

S.o.pln(m1.hashCode() == m2.hashCode())// false
S.o.pln(m1.equals(m2));// false
```

```
S.o.pln (m1.equals(m2)); // false
S.o.pln (m1 == m2); // false.
```

Case.2:

```
Mangs m1 = new Mangs(100);
Mangs m2 = m1;
```




```
S.o.pln ( m1 == m2); // true
S.o.pln ( m1.equals(m2)); // true
S.o.pln ( m1.hashCode() == m2.hashCode()); //true
```

Note :

1. by observing case1 & case2 it is clear that according to java.lang.Object class implementation of hashCode() and equals() :

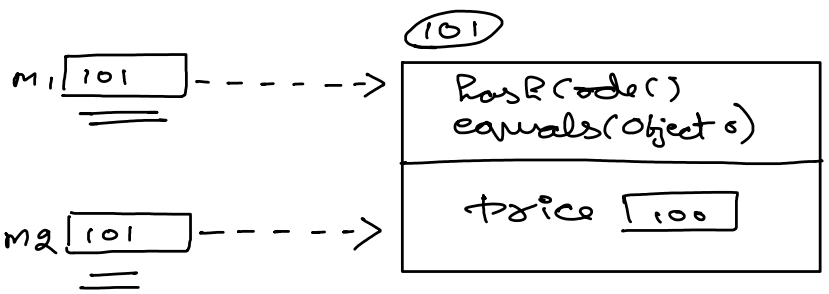
   - The hashCode() of two objects is different if equals() method returns false for them.
   - The hashCode() of two objects is same if equals() method returns true from them

example2 :

```
class Mango
{
    int price ;
    Mango( int price ) {
     this.price = price ;
    }
    // override equals method

    public boolean equals( Object obj ) {
        return this.price == ((Mango)obj).price ;
    }
}
```

Case 3 :

```
Mango m1 = new Mango(100);
Mango m2 = m1;
```

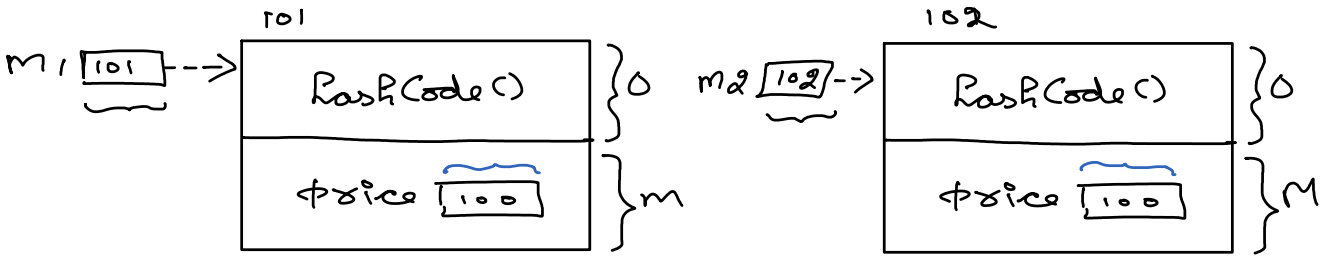



```
S.o.pln ( m1 == m2 ); // true
S.o.pln ( m1.equals(m2) ); // true
S.o.pln( m1.hashCode() == m2.hashCode() ); // true
```

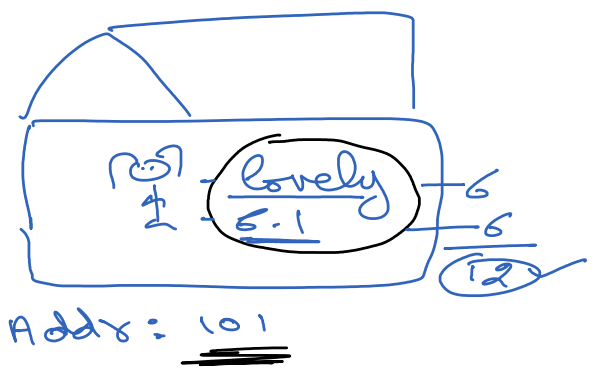
Case 4 :

```
Mango m1 = new Mango(100);
Mango m2 = new Mango(100);
```

Mango m2 = new Mango(100);




```
S.o.println(m1 == m2); // false
S.o.println(m1.equals(m2)); // true
S.o.println(m1.hashCode() == m2.hashCode()); // false
S.o.println(m1.hashCode()); 101
S.o.println(m2.hashCode()); 102
```



Assume equals is overriden

```
lovely.hashCode(); // ~~100~~ 12
lovely.hashCode(); // ~~101~~ 12
```

```
S.o.p(lovely.equals(lovely)); // true
```

Note:

- From the above observation it is clear that, if equals() method is overridden, it is strongly recommended to override hashCode() method also.
- If equals() method compares the state of an object, hashCode() should be designed such that it generates integer based on state of an object.

For examples refer, **workspace/object_class/src/pack3**