**OBJECT CLASS:**

There are 11 non static methods in the Object class, which is of package java.lang.

1. **clone():** Access specifier- protected, returns-object

2. **finalize():** Access specifier- protected, returns-void

3. **toString()**: Access specifier- public, returns-String

4. **hashCode():** Access specifier- public, returns-int

5. **equals():** Access specifier- public, returns-Boolean

6. **getClass():** Access specifier- public, returns-class

7. **notify():** Access specifier- public, returns-void

8. **notifyAll():** Access specifier- public, returns-void

**toString() method:**

This method takes the object as input and returns String;

Program:

class Orange

{

}

Class Demo1{

public static void main(String...args)

{

Orange o1=new Orange();

System.out.println(o1);//here, address is printing

System.out.println(o1.toString());//here, also address is printing

String s1=new String("Hello");

System.out.println(s1);//here, hello is printing

System.out.println(s1.toString());//here, also hello is printing

}

}

**Note:**

toString() method, by default it will returns the address of an object, In order to make it return a value we need to over-ride it,

Observer,

String s1=new String(“Hello”);

S1

HELLO

String object is created with Hello String and S1 is pointing towards “Hello”.

String s2=“Hello”

Hello

S2

Here also, String object is created with Hello string and s2 is pointing towards it

String s1=new String(“Hello”);

String s2=“Hello” //Both are same

**Important:**

System.out.println(o1.toString());//here, address is printing

System.out.println(s1.toString());//here, hello is printing

basically s1,toString() should also print address but it returing String.toString() method in object class returns address, but in String class when we declare String s1=new String(“Hello”); (or)

String s2=“Hello”. toString() method is automatically over-ridden to return String that is the reason why its returning String.

THIS HAPPENS ONLY IN STRING CLASS, IN OTHER CLASS WE HAVE TO EXPLICITLY OVER RIDE THE toString() METHOD TO RETURN THE DESIRED VALUE

Program:

class A

{

int i=100;

public String toString()

{

return "i=" + i;

}

}

class Demo8

{

public static void main(String...args)

{

A a1=new A();

System.out.println(a1.toString());//it will return the i=100, as we have over ridden the toString() method

System.out.println(a1);//it will also return i=100, because as we have over ridden the toString()

System.out.println(new A());//it will also return i=100; because we have over ridden the toString()

}

}

O/P:

i=100;

i=100;

i=100;

Note:

Once if you override the toString() method in a class (Ex:class A) to return a desired value, then you can never able to print the address of that class(class A)

Program:

class B{

int i=100;

int j=100;

public String toString()

{

return "i= "+i+"j= "+j;

}

}

class Demo9{

public static void main(String...args)

{

B b1=new B();

System.out.println(b1);//prints value

System.out.println(b1.toString());// prints value

System.out.println(new B());//prints value

System.out.println(new Demo9());//prints address

}

}

O/P

G:\practice\_java\src\com\qsp\objectclassprograms>java Demo9

i= 100 j= 100

i= 100 j= 100

i= 100 j= 100

Demo9@2a139a55

**Note:**

System.out.println(new Demo9());//prints address because we have not over ridden toString() method in Demo9 class , we have only over ridden in B class.

**IMPORTANT POINTS**

1.toString() method by default it will returns address, if we want to return a value then we need to override it

2.Only in String class, we no need to override the toString() method explicitly, when we declare a string variable and when assign the value to it, then toString() method is automatically over ridden.

3.Once toString() method is over ridden in a class to return a value, then we will never able to print the address of that class

**equals() method:**

By default, equals() method takes two objects as input and compare the address.

Program:

class C{

}

class Demo10{

public static void main(String...args){

C c1=new C();

C c2=new C();

C c3=new C();

System.out.println(c1);

System.out.println(c2);

System.out.println(c3);

System.out.println(c1==c3);// it will compare the address

System.out.println(c1.equals(c3));// as equals method is not over ridden it will also compare the address

c1=c3;

System.out.println(c1==c3);

System.out.println(c1.equals(c3));

System.out.println(c1);

System.out.println(c2);

System.out.println(c3);

}

}

O/P:

G:\practice\_java\src\com\qsp\objectclassprograms>java Demo10

C@2a139a55

C@15db9742

C@6d06d69c

false

false

true

true

C@6d06d69c

C@15db9742

C@6d06d69c

equals() method in String class is automatically overridden to compare the values.

Program:

class Demo11{

public static void main(String...args){

String s1=new String("Hello");

String s2=new String("Hello");

System.out.println(s1==s2);//it will compare the address

System.out.println(s1.equals(s2));//as equals method is automatically over ridden, it will compare the values

String s3="java";

String s4="bava";

System.out.println(s3==s4); //it will compare the address

System.out.println(s3.equals(s4));// //as equals method is automatically over ridden, it will compare the values

}

}

O/P:

G:\practice\_java\src\com\qsp\objectclassprograms>java Demo11

false

true

false

false

Other than String class, In any other class if you want the equals method to compare the values, then you have to explicitly override the equals method.

Program:

class Orange

{

int wt;

Orange(int i)

{

wt=i;

}

public boolean equals(Object o)//OVER RIDING THE EQUALS METHOD

{

if(o==null)

return false;

if(this==o)

return true;

if(this.wt==((Orange)o).wt)

{

return true;

}

else

{

return false;

}

}

}

class Demo12{

public static void main(String...args){

Orange o1=new Orange(100);

Orange o2=new Orange(120);

Orange o3=new Orange(100);

System.out.println(o1==o2);// COMPARES THE ADDRESS

System.out.println(o1==o3); // COMPARES THE ADDRESS

System.out.println(o1.equals(o2));

o3=null;//PASSING NULL VALUE

o3=o1;//PASSING THE SAME VALUE

System.out.println(o1.equals(o3));

}

}

**NOTE:**

Assume o1.equals(o3) then current object is o1 and we are passing o3 to the method, Then it becomes Object o=o3

We know that,

Orange o3=new Orange();

i.e

o3

ORANGE

Now, when Object o=o3 that means,

Object o

ORANGE

We can assume that,

Object o=new Orange();

Here, up-casting is happening that means sub class object behaving like super class, so sub class object cannot access its own member.

Now,

if(this.wt==((Orange)o).wt)

this.wt is current object weight i.e., o1 it compares it passed object weight o3, hence o3 is up-casted to object o, it cannot access its own variable which is wt, so its again down-casted to Orange and it will compares

O/P:

G:\practice\_java\src\com\qsp\objectclassprograms>java Demo12

false

false

false

true

Q. Difference between == and .equals()?

A.== is a relational equality operator which is used to

1.Compare two primitive variables, where it compares the value like x==y orx==10

2.Compare two reference variables, where it compares the address of the object

Equals() is a non-static method of Object class which by default compares the address. If we over ride we can compare the values.

Ex. String class, Wrapper class

Q. How do you generate an address?

Program:

package com.qsp.pack2;

class A

{

}

public class Printaddress {

public static void main(String[] args) {

A a1=new A();

System.out.println(a1.getClass().getName()+"@"+Integer.toHexString(a1.hashCode()));

}

}

O/P:

com.qsp.pack2.A@2a139a55

Note:

getClass() method returns the class

reflector A class

getClass() method means, by seeing the original class it creates the another class with has Original class as reflector

getName() will(return the name of the entity) calls the reflector in the class returned by getClass()

a1.hashCode():

it generates random int value and returns it

Integer.toHexString();

This will convert the int value into Hexadecimal

544654894168411646 5145641651511616515

|  |  |
| --- | --- |
| ||    100    Object A  26516516513251651 | 100  Object B  565465465165165151 |
| 120  Object C | Object D  150 |

Object A and Object B has same Value then their hash code should be equal, but here their hash code is different

Now we need to override the hash code to make Hash code of Object A and Object B equal

Program:

package com.qsp.pack2;

class Orange{

int wt;

Orange(int wt)

{

this.wt=wt;

}

public String toString()

{

return "wt=" + wt;

}

public boolean equals(Object o)

{

if(o==null)

return false;

if(this==o)

return true;

if(this.wt==((Orange)o).wt)

return true;

return false;

}

public int hashCode()// overriding the hash code method

{

return wt\*7\*11\*13;

}

}

public class Demo3 {

public static void main(String[] args) {

// TODO Auto-generated method stub

Orange o1=new Orange(100);

Orange o2=new Orange(120);

Orange o3=new Orange(100);

System.out.println(o1==o3);

System.out.println(o1.equals(o3));

System.out.println(o1.hashCode());

System.out.println(o2.hashCode());

System.out.println(o3.hashCode());

}

}

O/P:

false

false

100100

120120

100100

**Note:**

Reference variable o1 & o3 will have same hash Code after over riding the hashcode(), that does not mean that both refer to same Object

1.WHEN TWO OBJECTS(OF SAME TYPE) ARE EQUAL THEN THEIR HASHCODE SHOULD BE SAME,BUT WHEN HASHCODE OF TWO OBJECTS(DIFFERENT) ARE SAME THEN THE OBJECTS NEED NOT TO BE EQUAL.

IN OTHER WORDS, TWO DIFFERENT OBJECTS CAN BE IN SAME BUCKET(CAN HAVE SAME HASHCODE)

I.E, WHEN YOU OVER RIDE EQUALS METHODS OF A CLASS, WE SHOULD ALSO OVER RIDE HASHCODE METHOD