**ObjectClass**

This class is the super class of all classes in Java Programming Language. If we are creating a class then in JDK compiler script written that compiler will add “extends Object” class automatically inside the class. Means the all class should be child of Object class. Inside Object class there total 12 methods are there. But among 12 methods only 9 methods we are still using in our day to day program. All these 9 methods are very important for Development. These 9 methods are:

1. getClass()
2. toString()
3. hashCode()
4. equals()
5. clone()
6. finalize()
7. wait()
8. notify()
9. notifyAll()

Here in normal program we are using toString(), equals() and hashCode() method many times. getClass() method we are using whenever we required to get the information about a particular class then we are using this method. clone() method we are using for cloning operation. If we are not using any class object and we are removing all references from this object and making it as abandoned object and inform to Garbage Collector for delete this abandoned object then we have to use this finalize() method. If we want to give instruction to Garbage Collector for deletion of Object then in that case we are using this finalize() method. wait() method is used for keep thread to wait. notify() method used for notifying thread to wakeup and start work. notifyAll() method used for notifying all thread which are in wait state to start work. We will explain all these method one by one with examples below.

toString():

Examples:

**package**com.lara.ObjectNotes;

**class** A

{

**int**i;

}

**public classONotes1**

{

**public static void** main(String args[])

{

A a = **new**A();

a.i = 10;

System.*out*.println(a);

}

}

/\*

\* Here in this program we are trying to print class A object reference. But it will print only memory location of class A not the value inside this object. Because in this class A there is no

\* toString() method got override. So for this reason its showing only memory location. Here is the output of this program.

\* com.lara.ObjectNotes.A@20eb607d

\*/

Examples:

**package**com.lara.ObjectNotes;

**import**java.lang.\*;

**class** A1

{

**int**i;

}

**public classONotes2**

{

**public static void** main(String args[])

{

A a1 = **new**A();

a1.i = 10;

String s1 = a1.toString();

System.*out*.println(a1);

System.*out*.println(s1);

}

}

/\*

Here the output of this program is same as previous program even if we are using toString() method with class A object references. But still same result

is coming. Here is the output of this program. Even if we are trying to print the value of String value s1 then also same result will come because inside class A no toString() method got

override.

com.lara.ObjectNotes.A@7e938b4a

com.lara.ObjectNotes.A@7e938b4a

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** A2

{

**int**i;

}

**public classONotes3**

{

**public static void** main(String args[])

{

A2 a = **new**A2();

System.*out*.println("Value of a before initialize: "+a);

a.i = 10;

System.*out*.println("Value of a after initialize: "+a);

}

}

/\*

\* Here in this program we have created one class name A2 where we have declared one class level variable or global variable i of int type. Inside ONotes3 class we have created object of class

\* A2 with reference a. Here in this ONotes class we are trying to print class A2 object reference to know about the value of i. But it will print only memory location not the value of i. After this

\* printing we have initialized i value using this reference as 10 and trying to print same object reference again but still it will print only same memory location because there is no toString()

\* method got override inside this class A2. So for this reason it will print only memory location. Here is the below output of this program.

\*

\* Value of a before initialize: com.lara.ObjectNotes.A2@7e938b4a

\* Value of a after initialize: com.lara.ObjectNotes.A2@7e938b4a

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** B

{

**int**i;

}

**public classONotes4**

{

**public static void** main(String args[])

{

B b1 = **new**B();

b1.i = 10;

B b2 = **new**B();

b2.i = 10;

System.*out*.println("Value of b1: "+b1);

System.*out*.println("Value of b2: "+b2);

}

}

/\*

\* Here in this program we have created a class B in which we have declared a class level variable i which is int type. Inside class ONotes4 we have created two objects of class B i.e b1 and b2

\* and also initializing value of i using this two object references. i.e 10.and trying to print both object reference to fetch the value of i inside this object. But it will not print the value of i. It

\* will print only memory location because inside class B there is no toStrinig() method either override or created. So for this reason its showing this memory location only. Here is the below

\* output of this program.

\*

\* Value of b1: com.lara.ObjectNotes.B@89de832

\* Value of b2: com.lara.ObjectNotes.B@36f72f09

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** C

{

**int**i;

}

**public classONotes5**

{

**public static void** main(String args[])

{

C c1 = **new**C();

C c2 = c1;

c1.i = 10;

c2.i = 20;

System.*out*.println("Value of c1: "+c1);

System.*out*.println("Value of c2: "+c2);

}

}

/\*

\* Here in this program we have created a class name C. Inside this class C we have declared one variable i.ei which is int type. Inside class ONotes5 we have created two objects of class c i.e

\* c1 and c2. Here both c1 and c2 having same memory location and both are pointing to same object. So when we are writing this line c1.i = 10 then its changing the value of object of class C

\* and when we are writing this line as c2.i = 20. Earlier value of class C object was 10 now its changes to 20. So here we are trying to print the value of class c1 and c2 but we are getting only

\* memory location. because inside this class C we have not override toString() method from object class for this reason we are unable to print c1 and c2. Its only printing memory location.

\* Here below is the output of this program.

\* Value of c1: com.lara.ObjectNotes.C@89de832

\* Value of c2: com.lara.ObjectNotes.C@89de832

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** D

{

**int**i;

D(**int**i)

{

**this**.i = i;

}

}

**public classONotes6**

{

**public static void** main(String args[])

{

D d1 = **new**D(10);

String s1 = d1.toString();

System.*out*.println("Value of d1: "+d1);

System.*out*.println("Value of s1: "+s1);

}

}

/\*

\* Here in this program we have created a class name D and inside this class we have declared a class level variable i.ei which is int type. and also created a constructor which is taking one

\* parameteri.eint type. which is used to initialize class level variable. So JVM will not create default constructor inside this class D. Inside class ONotes6 we have created one object d1 of class

\* D and one String object s1. While Creating object of class D then we are passing value 10 inside class D constructor which is initializing class level variable using class D constructor.

\* One statement we have mentioned that d1.toString() method calling and passing values to String class object i.e s1. Here in this line using class D object reference we are calling toString()

\* method to fetch the value of class d Object. and storing this value inside String class object s1. After all these we are trying to print both object reference to fetch value of inside object of

\* class D. Here below is the output of this program.

\*

\* Value of d1: com.lara.ObjectNotes.D@4e44ac6a

\* Value of s1: com.lara.ObjectNotes.D@4e44ac6a

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** E

{

}

**public class ONotes7**

{

**public static void** main(String args[])

{

E e = **null**;

System.*out*.println("Value of e: "+e);

}

}

/\*

\* Here in this program we have created a class name with E. Inside this class there is no variable or methods. Inside class ONotes7 we have created only reference of class E not object and

\* assign that reference value as null. Here we are trying to print this reference. So we will get null. Here is the output of this program.

\*

\* Value of e: null

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** E

{

}

**public class ONotes7**

{

**public static void** main(String args[])

{

E e = **null**;

System.*out*.println("Value of e: "+e);

}

}

/\*

\* Here in this program we have created a class name with E. Inside this class there is no variable or methods. Inside class ONotes7 we have created only reference of class E not object and

\* assign that reference value as null. Here we are trying to print this reference. So we will get null. Here is the output of this program.

\*

\* Value of e: null

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** F

{

**int**i, j;

}

**public class ONotes8**

{

**public static void** main(String args[])

{

F f1 = **new**F();

String s1 = "Hello to All"+f1; // Here f1 is derived referal operands or derived data types. Here f1 is automatically call toString() method due to auto boxing.

System.*out*.println(s1);

}

}

/\*

\* Here in this program we have created class name F and inside this class we have created two class level variable i.ei, j. Inside class ONotes8 we have created object of class F and also we

\* have created one String object s1. Here one String value is concatenate with class F object reference. Here inside main() method while creating String object that time JVM will read from

\* right hand side to left side. While reading from right to left then its find that user want to concatenate one String value with class F object reference. While JVM will concatenate then in that

\* case its using auto boxing process to convert object to String value. So here value of this is Hello to All with class name f @ hexadecimal number. This value is storing inside String object s1.

\* Here we are trying to print the value of String s1. As we know that inside String class already toString() method got override for this reason its showing actual value of s1. So here below is the

\* output of this program.

\*

\* Hello to Allcom.lara.ObjectNotes.F@36f72f09

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** G

{

}

**public class ONotes9**

{

**public static void** main(String args[])

{

G g1 = **new**G();

String s1 = g1+"end";

System.*out*.println("Value of s1: "+s1);

}

}

/\*

\* Here in this program we have created a class name G. Inside class ONotes9 we have created object of class G and one String class object s1. While creating String object that time we are

\* concatenate g1 object reference with one String value i.e "end" and Storing this value to String object reference. Here we are trying to print the value of String object reference. So the output

\* becomes like this package name with class name @ hexadecimal number with end. Here is the output of this program.

\*

\* Value of s1: com.lara.ObjectNotes.G@3d0bbf6dend

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** H

{

}

**public class ONotes10**

{

**public static void** main(String args[])

{

H h1 = **new**H();

// String s1 = h1+h1; //Here both operands will never be class object reference

// System.out.println(s1);

String s2 = " "+h1+h1;

System.*out*.println(s2);

}

}

/\*

\* Here in this program we have created one class name H which has no data inside it. Inside class ONotes10 we have created one object this class H and trying to concatenate these two

\* object reference and stored into a String object s1. But as we know that during concatenation operation one operand should be String class type or related to String class types. But here

\* in this case these two operands are belongs to class H type there is no relation of String class So for this reason its showing Syntax error.

\* For this reason we have commented these lines.

\*

\* Here in case of String object s2 we have added one operand as String value which is empty but its in String value. So while operation will start that time JVm will read from right hand side

\* to left hand side. So for this reason it will first check whether any String value of object is there or not. Its found that one value is there so JVM will first start concatenate opeeration for that

\* String value with h1 and then whatever result will come after that it will again concatenate with h2 and store into String object i.e s2. So here is the output of this program.

\*

\* com.lara.ObjectNotes.H@3d0bbf6dcom.lara.ObjectNotes.H@3d0bbf6d

\*

\* Here same output will come. Inside class H there is no toString() method got override. So for this reason while concatenate this class object into String value then also same package name

\* with class name @ hexadecimal number will come and also add this value to another same value of class H reference and then add with that String value. So for this reason this above

\* output is generate.

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** I

{

**int**i, j;

I(**int** j)

{

**this**.j = j; /\* Here we are just initialize only j value not i value of class I. So when we will generate object of this class then in that case also inside constructor of this class we have to pass

value of j only not the value of i. So i value will remain same as default value given by Java i.e "0"\*/

}

**public** String toString()

{

**return**"j: "+j;

}

}

**public class ONotes11**

{

**public static void** main(String args[])

{

/\*Here below line inside constructor of class I we have pass int value which will go to class I constructor there JVM will check that to whom this value is assigning there JVM will get this

value is assigning to j not i. So JVM will remove old default value of j and assigned new value as 20. But still i value will reamin same as default value.\*/

I i1 = **new**I(20);

/\*Here we are trying to print the value of i1 but inside class I there is no toString() method got override so for this reason its showing only package name with class name @ hexadecimal

\* number. So it will not print the value inside this i and j.\*/

System.*out*.println(i1);

}

}

/\*

\* Here in this program how toString() method is calling. Main important part is if we are not override toString() method from object class and we are trying to print object reference of this

\* class then we are getting package name with class name @ hexadecimal number. Actually this toString() method is calling inside valueOf() method of String class written inside the println()

\* method of PrintStream class. While we are trying to print of a class object reference using this line "System.out.println()" then inside println() method one statement written i.e

\* "String s1 =String.valueOf(x)" this x is the object reference value and inside this valueOf () method we are first checking that this x is null or not . If that value is nul then it will return null

\* value as String format otherwise it will again call obj.toString() method which is calling from object class directly. Please refer image name toString() functionality inside img folder of

\* webcontent.

\*

\* Here toString() method got override inside class I. So for this reason its showing actual value inside this class. Here below is the output of this program:

\* j: 20

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** K

{

**int**i;

**public** String toString()

{

**return**"I : "+i; //Here user passing value inside this variable. Whatever value we will pass it will assign here.

}

}

**public class ONotes12**

{

**public static void** main(String args[])

{

K k1 = **new**K();

System.*out*.println(k1);

String s1 = k1.toString();

String s2 = "Hello"+k1;

String s3 = k1+"Hello";

System.*out*.println("s1: "+s1);

System.*out*.println("s2: "+s2);

System.*out*.println("s3: "+s3);

}

}

/\*

\* Here in thi program we are trying to assign value to variable i of class K using three ways. One using toString() method , second process is first concatenate String value with operand as class

\* K object reference k1 and third process is first taking object reference k1 of class K concatenate with one String value. We are trying to print all these values by storing three String class

\* object i.e s1, s2 and s3.

\*

\* Here when we are printing s1 then JVM will check what user want to print using this s1 object. It will get that user calling toString() method using class K object reference k1 mean time JVM

\* will check whether toString() method got override inside this class K or notother wise it will call toString() method of Object class. Here user already toString() method got override inside

\* this class K. So JVM will call this method and check what implementation written inside this method. Its found that user wants to return value of i. But it will found that i value is already 0

\* by default. So it will return this value and print this value.

\*

\* Here in case of s2 object also happens same thing but first it will check value inside k1 using toString() method and then concatenate operation will happens.

\*

\* Here in case of s3 also same thing happens like s2 first check value of k1 and then concatenate operation will happens.

\*

\* Here below is the output of this program:

\* I : 0

\* s1: I : 0

\* s2: HelloI : 0

\* s3: I : 0Hello

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** L

{

**int**a, b;

L(**int** a, **int** b)

{

**this**.a = a;

**this**.b = b;

}

**public** String toString()

{

**return**"[a: "+a+", "+"b: "+b+"]";

}

}

**public class ONotes13**

{

**public static void** main(String args[])

{

L l1 = **new**L(10, 20);

L l2 = **new**L(20, 10);

System.*out*.println(l1);

System.*out*.println(l2);

}

}

/\*

\* Here inside this program we are trying to assign value to variables declared inside the class L i.e a, b and using toString() method we are trying to fetch that values from that class L object.

\*

\* While we are creating objects of this class L inside class ONotes13 then inside this class constructor we are passing two int types values. While JVM will read this statement

\* "L l1 = new L(10, 20) then it will check that inside class whether this types of constructor or not. If there then it will check whether that argumented constructor is taking to parameter as int

\* types or not. If taking then what implementation written inside this class L constructor. JVM found that its initializing these two a & b variables. So while trying to fetch that value then sop

\* statement calling println() method so that inside this method one statement written i.e " String s = String.valueOf(x)" Here x is object values. While calling this valueOf() method then inside

\* this class JVm is checking that whether object is null or is there value available. Here inside this object value available so it will not check null and it will call toString() method using this x

\* and then again JVM will check that whether toString() method is got override or not. JVM found that user already override this toString() method and trying to print these two values. So it

\* JVM will call this method not object class toString() method and print these values.

\*

\* Here below is the output of this program:

\* Please refer DFD1 diagram inside webcontent-->img folder.

\*

\* [a: 10, b: 20]

\* [a: 20, b: 10]

\*

\* Here inside class L if we are not override this toString() method then JVM will automatically call toString() method of Object Class and which will give output as class with package name

\* @ hexadecimal nuber

\*/

Examples:

**package**com.lara.ObjectNotes;

**import**java.util.ArrayList;

**public class ONotes14**

{

@SuppressWarnings({ "unchecked", "rawtypes" })

**public static void** main(String args[])

{

Integer i = **new**Integer(20);

String s1 = **new**String("abc");

StringBuildersb = **new**StringBuilder("hello");

ArrayList list = **new**ArrayList();

list.add("First Element");

list.add("2nd Element");

System.*out*.println(i);

System.*out*.println(s1);

System.*out*.println(sb);

System.*out*.println(list);

}

}

/\*

\* Here in this class we have created one Integer class object which is a wrapper class, one String class object, one StringBuffer class object and another one is ArrayList class object.

\* Here among all these class one thing is common them i.etoString() method. Here in all these class this method already override no need to override toString() method from Object

\* class. So for this reason we are getting proper value while printing object reference. Here while we are printing the object reference then we are not getting any class package name

\* with class name @ hexadecimal number, we are getting exact value available inside this object. Here below is the output of this program.

\* 20

\* abc

\* hello

\* [First Element, 2nd Element]

\*/

Hint:

String classes has three different kind of classes. String class, StringBuilder class and StringBuffer class. Here among all these three classes String class called as immutable class and StringBuilder and StringBuffer class is called as mutable class. Another difference between these two StringBuffer and StringBuilder class is that StringBuffer class synchronized where StringBuilder is not synchronized. Even String class is also not synchronized. But among all these difference one thing is common between them is toString() method is override inside all these classes.

Hint:

Inside all collection class, String related classes and all wrapper classes toString() method got override inside all these classes.

Examples:

**package**com.lara.ObjectNotes;

**class** Q

{

**int**i;

**public** String toString()

{

String s1 = "This is the reference variable"+"to Q & it is pointing to Q "+"object. \nQ object contains "+"i value as: "+i;

**return** s1;

}

}

**public class ONotes15**

{

**public static void** main(String[] args)

{

Q q1 = **new**Q();

System.*out*.println(q1);

System.*out*.println("------------------------");

q1.i = 100;

System.*out*.println(q1);

}

}

/\*

\* Here in this program we have created one class name Q inside this class we have created one variable i.ei which is int type one method i.etoString() method override from object

\* class. Inside this method user created one String object s1 which contains some concatenated String values with variable i also. Trying to check before initializing this i value and

\* after initializing i value. Here is the output of this program after and before initialize this i value.

\* This is the reference variable to Q & it is pointing to Q object.

Q object contains i value as: 0

------------------------

This is the reference variable to Q & it is pointing to Q object.

Q object contains i value as: 100

Here we are understanding that how toString() method is fetching value inside Java Program. Before not assigning any value to variable to i so JVM will assigned default value i.e 0. So

When we are checking that value that time its coming 0. But after user assigning value as 10 to variable i using the object reference then its value now change 0 to 10 and when we are

checking again variable i value the we are getting 10 as user already assigned this value to i. So we are getting 10.

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** R

{

**int**i;

R(**int**i)

{

**this**.i = i;

}

**public** String toString()

{

**return**"i: "+i;

}

}

**class** S **extends** R

{

**int**j;

S(**int**i, **int** j)

{

**super**(i);

**this**.j = j;

}

**public** String toString()

{

**returnsuper**.toString()+", j: "+j;

}

}

**public class ONotes16**

{

**public static void** main(String[] args)

{

R r1 = **new**R(10);

S s1 = **new**S(100, 200);

System.*out*.println(r1);

System.*out*.println(s1);

}

}

/\*

\* Here in this program we have created two class R and S. Here class S extends class R. Inside class R we have declared one variable i which is int type and one constructor which is

\* argumented and taking parameter as int type. Which is using to initialize this class level variable i. Inside class R also one method got override that is toString() method from object class,

\* which is returning String type of value (This method actually returning value of i inside this class R object.). Here user created another more class S which is extends class R. This class S

\* contains one class level variable inside this class S i.e j and one argumented constructor which is taking two paremeter both are int types. One paremeter of this class S constructor is

\* initializing value of i of class R inside using super() calling method and another parameter is using to initialize value of j inside class S. Inside class S also we have created one method which

\* is override form Object class, that method is toString() method. Inside this toString() method user implemented for fetching the value of both i and j. But here we are not directly calling i

\* value but using super keyword with toString() method we are fetching the value of i and getting the value of i inside class S toString() method. Super keyword is use to calling super class

\* data inside child class. So here we have use like this way.

\*

\* Inside class ONotes16 we have created objects of class R and S also. and pass parameter inside both constructor. Inside class R constructor we are passing one parameter which is int type

\* and inside class S constructor we have passed two parameter which is also both are int type. while we are trying to print object reference of class r then it will call check whether toString()

\* method inside class R is available or not. JVM will find that toString() method got override inside this class R. So for this reason its calling that method and return the value of i. Again while

\* in second line we are trying to print the reference variable of class S object then its checking that whether toString() method got override inside class S or not. JVM is found that toString()

\* method already got override inside class S. So its showing both values i.ei and j. Here below is the output of this program.

\*

\* i: 10

\* i: 100, j: 200

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** T

{

**int**i;

T(**int**i)

{

**this**.i = i;

}

**public** String toString()

{

**return**"i: "+i; //Here we are returning the value of inside this method.

}

}

**class** U

{

T t1;

**int**j;

U(T t1, **int** j)

{

**this**.t1 = t1; //Here we are initializing value of i of class T inside class U by creating object reference of class T inside class U.

**this**.j = j;

}

**public** String toString()

{

**return**"t1 : "+t1+", j: "+j; /\*Here we are returning value of i and j. But here we are printing t1 which is object reference of class T. While calling this object reference the it will call

toString() method of class T and print value of i. Here we are trying to print value of using the help of t1 object reference.

\*/

}

}

**public class ONotes17**

{

**public static void** main(String[] args)

{

T t1 = **new**T(100);

U u1 = **new**U(t1, 200);

System.*out*.println("Value of t1: "+t1);

System.*out*.println("Value of u1: "+u1);

}

}

/\*

\* Here in this program we are trying to call first class T object reference i.e t1. When it will call then JVM first check that whether toString() method is available inside class T or not. If this

\* toString() method is not got override form Object class then it will call Object class default toString() method and print the package name with class name @ hexadecimal number. If this

\* toString() method is available then its showing the value of i . So here while JVM is checking then its found that toString() method is already override inside class T so JVM will call this

\* method.

\*

\* While we are creating object of class U then inside constructor we have passed class T object reference and one int type parameter. Object class T reference we have used to initialize value of

\* i using this reference. So here is the output of this program.

\*

\* Value of t1: i: 100

\* Value of u1: t1 : i: 100, j: 200

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** V

{

String[] lovers;

**int**i;

V(String[] lovers, **int**i)

{

**this**.lovers = lovers;

**this**.i = i;

}

**public** String toString()

{

String s1 = "Lovers: ";

**for** (**int**i = 0; i<lovers.length; i++)

{

s1 = s1+lovers[i]+", "; //Here we are looping all elements inside String array

}

s1 = s1+"\n"; //This line used for after completion of loop its will come to next line and then print some other work.

s1 = s1+"i= "+i; //Here after loop it will print this line.

**return** s1;

}

}

**public class** ONotes18

{

**public static void** main(String[] args)

{

String[] lovers = {"Bhabbyalaxmi Swain", "Sita Jena", "SurendraNath Jena", "Ankit", "SwatyaSwarup", "Soumya", "Akash"};

V v1 = **new**V(lovers, 10);

System.*out*.println(v1);

}

}

/\*

\* Here inside this class ONotes18 we have created one String array inside this we have assigned some values. We have created one object of class V inside this class after creation of this String

\* array and we are passing this array value inside this object constructor. This class V constructor is taking two array one is String array and another is int value which is assigned this int

\* value which is assigned this value to i created inside class V. So inside class V we have created two variable i,e one is String array type and another is int type. While we are printing

\* class V reference then jvm first checking that whether toString() method got override inside this class or not, its found that already override inside this class and then jvm will call this

\* toString() method of class V.

\*

\* While calling toString() method of this class V then it will check implementation inside this method. It found that user trying to this String array and trying to print that value in a Single

\* line.

\*

\* Lovers: Bhabbyalaxmi Swain, SitaJena, SurendraNathJena, Ankit, SwatyaSwarup, Soumya, Akash,

\* i= 10

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** A3

{

**int**i;

A3(**int**i)

{

**this**.i = i;

}

}

**public class** ONotes19

{

**public static void** main(String[] args)

{

A3 a1 = **new**A3(10);

A3 a2 = **new**A3(20);

System.*out*.println(10 == 10);

System.*out*.println(a1 == a2);

System.*out*.println(a1.i == a2.i);

}

}

/\*

\* Here in this above program we have created two different object references because both object references values are different.

\* While we are trying to check whether is 10 is equals or not with 10 then as we know that 10 always be 10.

\* Second statement we are trying to compare two object reference. Both objects are belongs to same class but pointing to different location one is pointing to one object whose i value is 10,

\* another one is pointing to that location whose i value is 20. So in this case both objects memory locations are different.

\* Third statements we are trying to print the value of i inside both objects using their references. Even memory location for both the objects are different then value we have assigned also

\* different. So while we are comparing then we will get false.

\*

\* Here below is the output of this program:

\*

\* true

\* false

\* false

\*

\* Even though both object references are derived from same class but still failing because its checking only memory location by calling toString() method of Object class not this class. Due to

\* this reason its output is coming false.

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** B1

{

**int**i;

B1(**int**i)

{

**this**.i = i;

}

}

**public class** ONotes20

{

**public static void** main(String[] args)

{

B1 b1 = **new**B1(10);

B1 b2 = b1;

System.*out*.println(b1 == b2);

System.*out*.println(b1.i == b2.i);

}

}

/\*

\* Here both objects b1 and b2 are belongs to same class B1 so both memory location also same. So while we are checking the equality then in this case it showing true. Because still its

\* checking same memory location not contains inside these both objects. So we are getting true but actually its still failing. We want to compare on the basis of contains inside object.

\*

\* Here below is the output of this program:

\* true

\* true

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** C1

{

}

**class** D1 **extends** C1

{

}

**public class** ONotes21

{

**public static void** main(String[] args)

{

C1 c1 = **new**C1();

D1 d1 = **new**D1();

System.*out*.println(c1 == d1);

}

}

/\*

\* Here both objects are different so we can't compare each other. It will shows compilation error. If we want to compare this two class object references then we have to make any class as

\* parent and another class as child in that case we can compare.

\*

\* Here below is the output of this program:

\* false

\* Here both data types are different types. One is class C type and another one is class D type. \* So we are getting compilation time error as syntactical error. If both classes are in relation

\* then there will be no Compilation Error. But here both classes are not in any relation. So for

\* this reason this error is coming.

\*/

Examples:

**equals() method:**

equals() method is used for comparing purpose. This method is using for comparing between two objects for checking equality for value inside these objects and memory location. If both are matches then it will return true. Return type of this method is Boolean and another and most important points about this equals() method is it will return true by default if we are comparing between primitive types and if we are comparing between two objects then in that case it will not return true if and only if this method is not override inside that particular class whose objects comparing. So for getting true if we are comparing between two objects of a particular class then we have to first override equals() and hashCode() method from Object class then it will return exact output for comparison.

Examples:

**package**com.lara.ObjectNotes;

**class** E1

{

**int**i;

}

**public class** ONotes22

{

**public static void** main(String[] args)

{

E1 e1 = **new**E1();

E1 e2 = **new**E1();

e1.i = 10;

e2.i = 10;

System.*out*.println(e1.equals(e2)); //Here in this line we are calling Object class equals() method.

}

}

/\*

In this program we have created one class name E1 which has one class level variable "i" which is int type. Inside class ONotes22 we have created two objects of this class E1 and we have

assigned same values to this variable i. But this value is assigned by two different objects of this class E1. Here main task is to check if both objects are equals in content or not. But here we

have not override this equals() method inside this class. So while we are calling equals() method for comparing two objects then its calling Object class equals() method. First JVM will check

this method is available inside this class E or not. While checking JVM not found this method.So it will directly call the equals() method of Object class which is not checking by default

contents inside objects. Its checking the memory location of both objects and return false. Because both objects are different but belongs to same class. So for this reason its showing false.

Here below is the output of this programs.

false

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** F1

{

**int**i;

}

**public class** ONotes23

{

**public static void** main(String[] args)

{

F1 f1 = **new**F1();

f1.i = 10;

System.*out*.println(f1.equals(f1));

}

}

/\*

Here we have created only one object of class F1 i.e. f1 and we are comparing this object reference with itself. So that means both are belongs to same class and pointing to same location

but while we are checking the content of this object then it will not showing the content of the object of class F its showing memory address of these two objects of class F1.

Here is the output of this program:

true

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** G1

{

**int**i;

G1(**int**i)

{

**this**.i = i;

}

**public** String toString()

{

**return**"i: "+i;

}

}

**public class** ONotes24

{

**public static void** main(String[] args)

{

G1 g1 = **new**G1(100);

G1 g2 = **new**G1(100);

System.*out*.println("g1: "+g1);

System.*out*.println("g2"+g2);

System.*out*.println("g1.equals(g2): "+g1.equals(g2));

}

}

/\*

Here in this program we have created a class name G1 in which one class level variable i which is int type. and one argumented constructor which is taking one parameter as an argument used

for initialize class level variable i. One method also override from Object class i.etoString() method which is returning value of i.

In class ONotes24 we have created two objects of class G1 in which constructor we are passing same value i.e 100 and we are trying to check equality for both the objects using equals()

method of Object class and also we are checking the value of i in both objects. Here both objects are belongs to same class G1. So output will be like this

100

100

false

Still here in class G1 we have not override equals() method from Object class. We are just override toString() method which will return the value of i inside both the objects but not able to

check the equality. So while we are checking equality for these objects then its return false as we know the if we are checking equality of objects and calling Object class equals() method then

it will check memory address of both objects not the content. So output is false..

\*/

**Hint:**

Object class equals() method is not bother about content of the objects. Its checking the memory address of the objects. If both objects memory location is same then it return true and if not then false. If we are override this equals() method in our class and implements then its checking contents inside objects, so for this reason checking equality is not perfect 100%. So whenever we are override equals() method from object class then its bound that we have to override hashCode() method for checking equality. Then only equality will be successful. hashCode() method is returning an int value which is called as memory address into hexadecimal number. Using this hashCode we are comparing two objects if this hashCode is equal then we are checking content of the objects using equals() method. Then whatever result will come that is the actual output. So always prefer when use equals() method inside your class then better to override hashCode() method.

Examples:

**package**com.lara.ObjectNotes;

**class** H1

{

**int**i;

**publicboolean** equals(Object obj)

{

H1 h1 = (H1)obj;

**returnthis**.i == h1.i;

}

}

**public class** ONotes25

{

**public static void** main(String[] args)

{

H1 h1 = **new**H1();

h1.i = 100;

System.*out*.println(h1); //Here we have not override toString() method from Object class. So it will call toString() method of Object class which will return memory address.

H1 h2 = **new**H1();

h1.i = 100;

System.*out*.println(h2); //Here we have not override toString() method from Object class. So it will call toString() method of Object class which will return memory address.

System.*out*.println(h1.equals(h2));

}

}

/\*

Here in this program we have override equals() method inside class H1 and checking the contents of the class H1 objects. Here its not checking the memory address of the objects. So output

of this program will be true.

Here below is the output of this program:

com.lara.ObjectNotes.H1@89de832

com.lara.ObjectNotes.H1@36f72f09

false

Please refer class ONotes26 & ONotes27 where we have used same code with additionally we have override toString() method.

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** H2

{

**int**i;

**public** String toString()

{

**return**"i: "+i;

}

**publicboolean** equals(Object obj)

{

H2 h1 = (H2)obj;

**returnthis**.i == h1.i;

}

}

**public class** ONotes26

{

**public static void** main(String[] args)

{

H2 h1 = **new**H2();

h1.i = 100;

System.*out*.println("h1: "+h1);

H2 h2 = **new**H2();

h2.i = 100;

System.*out*.println("h2: "+h2);

System.*out*.println("h1.equals(h2): "+h1.equals(h2));

}

}

/\*

Here in this class H2 we have override both equals() and toString() method. So we are getting exact output of this program. Here we can see the value of i inside both objects as well as we can

check equality between objects.

Here is the output of this programs:

h1: i: 100

h2: i: 100

h1.equals(h2): true

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** H3

{

**int**i;

H3(**int**i)

{

**this**.i = i;

}

**public** String toString()

{

**return**"i: "+i;

}

**publicboolean** equals(Object obj)

{

H3 h1 = (H3)obj;

**returnthis**.i == h1.i;

}

}

**public class** ONotes27

{

**public static void** main(String[] args)

{

H3 h1 = **new**H3(100);

System.*out*.println("h1: "+h1);

H3 h2 = **new**H3(100);

System.*out*.println("h2: "+h2);

System.*out*.println("h1.equals(h2): "+h1.equals(h2));

}

}

/\*

Here in this program we have override both toString() and equals() method from Object class and also we are using help of constructor initialize value of i and checking equality. Here we will

get exact output of this program.

Here is the output of this program:

h1: i: 100

h2: i: 100

h1.equals(h2): true

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** Person

{

**int**age;

**public** String toString()

{

**return**"Age: "+age;

}

**publicboolean** equals(Object obj)

{

Person p = (Person)obj;

**boolean** flag = **this**.age == p.age;

**return** flag;

}

}

**public class** ONotes28

{

**public static void** main(String[] args)

{

Person p1 = **new**Person();

p1.age = 32;

System.*out*.println("Person Age: "+p1);

Person p2 = **new**Person();

p2.age = 32;

System.*out*.println("Person Age: "+p2);

System.*out*.println("p1.equals(p2): "+p1.equals(p2));

}

}

/\*

Here in this program we are checking equality of Person class objects as well as we are also fetching the values inside both objects.

Here is the output of this program:

Person Age: Age: 32

Person Age: Age: 32

p1.equals(p2): true

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** I1

{

}

**class** J

{

**int**i;

**publicboolean** equals(Object obj)

{

J j1 = (J)obj;

**returnthis**.i == j1.i;

}

}

**public class** ONotes29

{

**public static void** main(String[] args)

{

I1 i1 = **new**I1();

J j1 = **new**J();

System.*out*.println(j1.equals(i1));

}

}

/\*

here both classes I1 and J are different and there is no relation between them inside this program. I1 is different class which has no data inside it and J is class which has two date members are

there. One is class level variable i which is int type and another one is equals() method from Object class. While we are checking equality for these two classes objects then it will return false.

class J object having some data and I class object having default data. So it will return exception due to that here two objects are not belongs to same class these two objects are from different

class for this reason it will throw ClassCasteException that i1 is not able to caste into j1.

Here is the output of this program:

Exception in thread "main" java.lang.ClassCastException: com.lara.ObjectNotes.I1 cannot be cast to com.lara.ObjectNotes.J

atcom.lara.ObjectNotes.J.equals(ONotes29.java:14)

at com.lara.ObjectNotes.ONotes29.main(ONotes29.java:27)

\*/

**instanceOf()Operator:**

This operator is used for checking that the current object is belongs to this class or not. If this object is not belongs to this class then return false. We are using this operator inside a for loop and checking that current object is belongs to current class or not.

Examples:

**package**com.lara.ObjectNotes;

**class** I2

{

}

**class** J1

{

**int**k;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** J))

{

**return false**;

}

J1 j1 = (J1)obj; //Here we are Down Cast obj into class J1 type.

**returnthis**.k == j1.k;

}

}

**public class** ONotes30

{

**public static void** main(String[] args)

{

J1 j1 = **new** J1();

I2 i1 = **new**I2();

System.*out*.println("i1.equals(j1): "+j1.equals(i1));

}

}

/\*

Here in this program both objects (i1, j1) are different and also belongs to different classes (I2, J1). So while we are checking equality between these two objects then we are getting false.

Inside class J1 we have used instanceOf operator inside equals() method which is use to check whether coming data belongs to this class or not. If not then it will return false otherwise it will return

true. While we checking equality then JVM will check in which class equals() method got override based on that it will call that class method. If based on object in that class equals() method

not got override then JVM will call Object class equals() method and return default output. Here in this case JVM will call equals() method of class J1 while enter to this method, user first

check this object is belongs to this class or not. JVM found that this object is not belongs to class J1 its belongs to class I2. So it will return false.

Here is the output of this programs:

i1.equals(j1): false

\*/

Examples:

**package**com.lara.ObjectNotes;

**class** K1

{

**int**i;

}

**class** L1

{

**int**i;

**publicboolean** equals(Object obj)

{

L1 l1 = (L1)obj;

**returnthis**.i == l1.i;

}

}

**publicclass** ONotes31

{

**publicstaticvoid** main(String[] args)

{

K1 k1 = **new**K1();

k1.i = 10;

System.*out*.println("k1: "+k1);

L1 l1 = **new**L1();

l1.i = 10;

System.*out*.println("l1: "+l1);

System.*out*.println("k1.equals(l1): "+k1.equals(l1));

System.*out*.println("l1.equals(k1): "+l1.equals(k1));

}

}

/\*

We have created two classes i.e K1 and L1. Inside K1 class we have declared one variable which is int type i.ei. Inside class L1 we have created one variable same data types i.ei and also

implemented one override method i.e equals() method from Object class into this class L1. Here inside this equals() method we are not checking whatever object is coming using this equals()

method that time its checking that this object reference is belongs to this class or not. Here we have simply change the type of reference int o class L1 type and then checking whether content

inside this reference is same or not if same then it will return true or false. But here output will be as ClassCasteException due to reason that both object references are belongs to different

class. So for this reason while we are checking equality then JVM will check that both object references are belongs to same class or not. So for proper output please refer ONotes32 class.

There we have handled this exception by using instanceOf operator for checking these references are belongs to same class or not inside equals() method of refer object through which we are

calling equals() method. Here below is the output of this programs where we are getting exception.

Exception in thread "main" k1: com.lara.ObjectNotes.K1@36f72f09

l1: com.lara.ObjectNotes.L1@281c35ec

k1.equals(l1): false

java.lang.ClassCastException: com.lara.ObjectNotes.K1 cannot be cast to com.lara.ObjectNotes.L1

at com.lara.ObjectNotes.L1.equals(ONotes31.java:13)

at com.lara.ObjectNotes.ONotes31.main(ONotes31.java:31)

While we are checking equality using this statement "System.out.println("k1.equals(l1): "+k1.equals(l1));" then in this case its calling object class equals() method so we are always getting

false. Inside class K1 we have not override equals() method so its calling Object class equals() method. So its checking with object class data which will always return false. Because inside

Object class there is no such data available. So output is coming false. In this case JVM will not check

While we are checking equality using this statement "System.out.println("l1.equals(k1): "+l1.equals(k1));" then JVM will first check whether equals() method is available inside class L1 or

not if not then it will call default equals() method of Object class. But here in class L1 User already defined equals() method override from Object class. Using this method we have to

check equality properly. We are checking equality between l1 and k1 object reference then in this case we are checking first k1 object is an instance of class L1 or not, but while JVM

will found that this object is not actually not belongs to class L1 then it will not give any return as false. It will return as an exception as ClassCastException due to reason that

k1 reference is not belongs to class L1 so for this reason this exception is occur.

\*/

Example:

**package**com.lara.ObjectNotes;

**class** Book

{

**double**weight;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** Book))

{

**returnfalse**;

}

Book b = (Book)obj;

**returnthis**.weight == b.weight;

}

}

**class** Laptop

{

**double**weight;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** Laptop))

{

**returnfalse**;

}

Laptop l = (Laptop)obj;

**returnthis**.weight == l.weight;

}

}

**publicclass** ONotes32

{

**publicstaticvoid** main(String[] args)

{

Book b1 = **new**Book();

b1.weight = 10.0;

Book b2 = **new**Book();

b2.weight = 10.0;

Laptop l1 = **new**Laptop();

l1.weight = 10.0;

Laptop l2 = **new**Laptop();

l2.weight = 10.0;

System.*out*.println("b1.equals(b2): "+b1.equals(b2)); //True due to same class object reference as well as weight value inside both objects are same.

System.*out*.println("l1.equals(l2): "+l1.equals(l2)); //True. Due to same object references as well as weight value inside both objects are same.

System.*out*.println("b1.equals(l1): "+b1.equals(l1)); //False. Both objects are belongs to different object still even same weight value also.

System.*out*.println("l2.equals(b2): "+l2.equals(b2)); //False. Both objects are belongs to different class still even same weight value also.

}

}

/\*

Here in this program both classes i.e class Book and class Laptop already override equals() method inside their respective classes. So while we are checking equality using same class object

reference then JVM first checking their value inside both object and found as same value so that method will return true. So output will come as true. Same thing also happens with class

Laptop. If we are trying to check equality in case of different class reference then it will shows false. Because neither both classes are related to each other nor any class among

them parent of another. So for this reason its showing false due to checking instance of that particular class or not using instanceof operator. Here in third sop statement we required

reference for Laptop class type but we are passing Book type reference into equals() method, so its return false. Similarly in case of forth sop statement also same thing happens.

Here below is the output of this class.

b1.equals(b2): true

l1.equals(l2): true

b1.equals(l1): false

l2.equals(b2): false

\*/

Example:

**package**com.lara.ObjectNotes;

**class** M

{

**int**i;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** M))

{

**returnfalse**;

}

**return**i == ((M)obj).i;

}

}

**class** N **extends** M

{

**int**j;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** N))

{

**returnfalse**;

}

**returnsuper**.equals(obj) &&j == ((N)obj).j;

}

}

**publicclass** ONotes33

{

**publicstaticvoid** main(String[] args)

{

M m1 = **new**M();

m1.i = 100;

M m2 = **new**M();

m2.i = 100;

System.*out*.println("m1.equals(m2): "+m1.equals(m2));

N n1 = **new**N();

n1.i = 1000;

n1.j = 2000;

N n2 = **new**N();

n2.i = 1000;

n2.j = 2000;

System.*out*.println("n1.equals(n2): "+n1.equals(n2));

System.*out*.println("n1.equals(m1): "+n1.equals(m1));

System.*out*.println("m1.equals(n1): "+m1.equals(n1));

n1.i = 100;

System.*out*.println("m1.equals(n1): "+m1.equals(n1));

}

}

/\*

Here in this program we have created two classes M and N. Here class N is child of class M. Inside both classes we have declared one variable which is class level variable. i variable is belongs

to class M and j variable belongs to class M but both data type is int type. In both classes equals() method got override and implementation is also different in both cases. Here in case of

equals() method of class M user checking whatever reference is coming inside this class is belongs to same class or not. According to that basis its checking equality if reference is not belongs

to this class then JVM will not check equality. and return as false. In case of equality method we checking both classes reference and as we know that class N is the child class of class M. So

while calling equals() method then in that time it will also call equals() method of class M using super calling statement. Before that its checking that whatever reference is coming to class N

equals() method its belongs to same class or not. If coming reference is not belongs to class N then it will go super class M and check there whether this reference is belongs to this class or not.

After that it will return true.

Here below is the output of this program.

m1.equals(m2): true

n1.equals(n2): true

n1.equals(m1): false

m1.equals(n1): false

m1.equals(n1): true

Here in last statement while we are changing value of n1 and then checking equality based on value then its coming true. Because inside equals() method of class N its mentioned that

super.equals(obj) && j == ((N)obj).j; Using this statement JVM will check that what value available inside class M object after that getting that value of class M object then JVM will

compare with class N object value is equal or not.

\*/

Example:

**package**com.lara.ObjectNotes;

**class** P1

{

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** P1))

{

System.*out*.println(1);

**returnfalse**;

}

System.*out*.println(2);

**returntrue**;

}

}

**class** Q1 **extends** P1

{

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** Q1))

{

System.*out*.println(3);

**returnfalse**;

}

System.*out*.println(4);

**returnsuper**.equals(obj) &&**true**;

}

}

**publicclass** ONotes34

{

**publicstaticvoid** main(String[] args)

{

P1 p1 = **new**P1();

P1 p2 = **new**P1();

Q1 q1 = **new**Q1();

Q1 q2 = **new**Q1();

System.*out*.println("p1.equals(p2): "+p1.equals(p2)); //2,true

System.*out*.println("q1.equals(q2): "+q1.equals(q2)); //4, 2, true

System.*out*.println("p1.equals(q2): "+p1.equals(q2)); //1, false

System.*out*.println("q1.equals(p2): "+q1.equals(p2)); //3, false

}

}

/\*

In this program we have created two classes one is class P1 and another one is Q1. Here class Q1 is the child class of class P1. Inside both classes one common method got override i.e

equals() method of Object class. This method we are using in Java to compare between two similar objects not different objects. If we are using different objects then it will throws

ClassCastException. In both classes equals() method user first check whether object reference is belongs to same class object or not then it will check for the values inside that object.

But here in child class Q1 equals() method user first check whether object references is belongs to same class or not, if not then JVM will return output as false, but while it will check

that object reference is instance of that same class then JVM will not check inside if block it will come out of if block. After that it will check inside another return statement where user first check this

object reference inside parent class P1 and then it JVM will find that its compare with true.

So here below is the answer of this program.

2

p1.equals(p2): true

4

2

q1.equals(q2): true

2

p1.equals(q2): true

3

q1.equals(p2): false

\*/

Example:

**package**com.lara.ObjectNotes;

**class** R1

{

**int**i;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** R1))

{

**returnfalse**;

}

/\*

Here in this return statement we have first convert object into class R1 object and then check value of i inside this object and compare this value into class level variable i which is

int data type.

\*/

**return**i == ((R1)obj).i;

}

}

**class** S1

{

**int**j;

R1 r1;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** S1))

{

**returnfalse**;

}

/\*

Here in this return statement we are first convert Object class object to child class S1 and then check value value of r1 inside this object and compare this value with class level

variable of class R1 reference r1. While this r1 is calling equals() method then its calling equals() method of class R1 where its checking the value of i inside this r1 object. After

getting this value of i then HJVM again check the value of j inside class S1 and then compare these two variables value and JVM found as different.

\*/

**return**r1.equals(((S1)obj).r1) &&j == ((S1)obj).j;

}

}

**publicclass** ONotes35

{

**publicstaticvoid** main(String[] args)

{

//Here we have created two objects for class R1. Where we have assigned value of r1 is 10 and r2 as also 10.

R1 r1 = **new**R1();

r1.i = 10;

R1 r2 = **new**R1();

r2.i = 10;

//Here we have also created two object of class S1 where s1 value assigned as 200 and s2 value assigned as also 200.

S1 s1 = **new**S1();

s1.r1 = r1;

s1.j = 200;

S1 s2 = **new**S1();

s2.r1 = r2;

s2.j = 200;

//here we ae comparing two variable of class S1 i.e s1 and s2 by calling equals() method of class S1.

System.*out*.println("s1.equals(s2): "+s1.equals(s2));

}

}

/\*

Here below is the output of this program:

s1.equals(s2): true

\*/

Example:

**package**com.lara.ObjectNotes;

**class** T1

{

**int**i;

String s1;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** T1))

{

**returnfalse**;

}

/\* Here using this return statement user wants to check first value of i inside class T1 object. As we know equals() method already override inside String class. So while String

variable s1 is calling equals() method for checking s1 value in both classes that time String variable is calling String class equals() method, before calling String class equals()

method first obj is convert into class T1 reference then its calling class T1's s1 variable. After checking of s1 value of both classes then JVM compare i value with s1 variable and

then its return value as below.

\*/

**return**s1.equals(((T1)obj).s1) &&**this**.i == ((T1)obj).i;

}

}

**publicclass** ONotes36

{

**publicstaticvoid** main(String[] args)

{

//We have created two String class Object i.e s1 and s2 whose values are same in both cases.

String s1 = **new**String("abc");

String s2 = **new**String("abc");

//We have created two objects of class T1 whose value assigned below.

T1 t1 = **new**T1();

T1 t2 = **new**T1();

//Here we are calling s1 reference by class T1 reference i.e t1 and assigned value of s1 to it.

t1.s1 = s1;

//Here same thing is happening but little bit different is that value of s2 to it.

t2.s1 = s2;

//Here we are assigning value of i in class T1 as 200 using the reference t1.

t1.i = 200;

//Here we are assigning value of i in class T1 as 200 using the reference t2.

t2.i = 200;

//Here both references are belongs to same class T1. While we are checking the value of i inside these class T1 two objects then JVM found that its same. So its return as true.

System.*out*.println("t1.equals(t2): "+t1.equals(t2));

}

}

/\*

Here is the below output of this program:

t1.equals(t2): true

\*/

Example:

**package**com.lara.ObjectNotes;

**publicclass** ONotes37

{

**publicstaticvoid** main(String[] args)

{

Integer i1 = **new**Integer(100);

Integer i2 = **new**Integer(200);

System.*out*.println("i1.equals(i2): "+i1.equals(i2));

}

}

/\*

Inside all Wrapper classes equals() method got override. So while we are checking equality in case of Wrapper classes then there is no need to override equals() method separately.

Here both object references is belongs to same Wrapper class i.e Integer. But both references value is different. So while we are checking equality using this references then it will

return as false. Because both object contents are different. So we are getting output as false in this case.

Inside this Wrapper class equals() method implementation is like this as below:

publicboolean equals(Object obj) {

if (objinstanceof Integer) {

return value == ((Integer)obj).intValue();

}

return false;

}

Here inside this equals() method JVM first checking that whatever object reference is passing that reference is belongs to same Wrapper class or not. JVM found that its same

Wrapper class instance otherwise it will get output as false. But here its same Wrapper class instance so for that reason JVM is not enter into if block its coming out from if block and

check what next implementation is there. JVM found that first we have to type cast object class reference into Integer class reference and then it will call intValue() method which

will call value of object using same object class reference where this intValue() method return the value of this Wrapper class object value.

\*/

Example:

**package**com.lara.ObjectNotes;

**class** U1

{

Integer i1;

Double d1;

**publicboolean** equals(Object obj)

{

**if** (!(obj**instanceof** U1))

{

**returnfalse**;

}

U1 u1 = (U1)obj;

//Here in this return statement we have first check equality for i1 and d1 value inside class U1 object after that we are checking that whether both values are same in this case or not.

**return**i1.equals(u1.i1) &&d1.equals(u1.d1);

}

}

**publicclass** ONotes38

{

**publicstaticvoid** main(String[] args)

{

//Here we have created two different Integer Wrapper class objects whose value are different i.e i1 & i2.

Integer i1 = **new**Integer(1000);

Integer i2 = **new**Integer(1000);

//Here we have created two different objects of Double Wrapper classes whose values are different i.e d1 and d2.

Double d1 = **new**Double(1000.00);

Double d2 = **new**Double(1000.00);

/\*Here we have created one objects of class U1 i.e u1 where we have assigned i1 value from Integer Wrapper class i1 by calling i1 of class U1 object reference i.e u1. Same thing also

happening with d1 variable of class U1 from Double Wrapper class reference i.e d1.

\*/

U1 u1 = **new**U1();

u1.i1 = i1;

u1.d1 = d1;

/\*Here we have created one objects of class U1 i.e u2 where we have assigned i1 value from Integer Wrapper class i2 by calling i1 of class U1 object reference i.e u2. Same thing also

happening with d1 variable of class U1 from Double Wrapper class reference i.e d2.

\*/

U1 u2 = **new**U1();

u2.i1 = i2;

u2.d1 = d2;

/\*Here we are checking equality between two object reference of class U1 i.e u1 and u2. While checking equality JVM first check whether class U1 already override equals() method of

Object class. While checking equals() method then JVM found that class U1 already override this method. Then JVM will call this method using u1 reference and passing reference u2

inside this equals() method.

Inside equals() method of class U1 user want to check whatever reference object we are passing into this method that reference is belongs to this class U1 or not. Here user already pass

same class U1 reference i.e u2. So JVM while checking inside if block that found this reference is belongs to same class reference so JVM will not enter to this if block and come out of this

if block and start read next line.

Next line JVM found that user want to convert whatever reference is coming inside equals() method parameter into class U1 type object reference using type cast. So based on that JVM

also check that if both reference is not same kind of then it will return as ClassCastException. But here while checking JVM found that this reference is belongs to same class U1. So here

using another object reference of class U1 i.e u1 assigned value of u2. Then JVM read next statement.

While JVM will read this statement "d1.equals(u1.d1)" in the next line that time JVM found that using reference Double Wrapper class reference d1 user want to call equals() method and

inside this equals() method user passing value of d1 of class U1 using the help of reference u1. As we know that all Wrapper classes are override equals() method from Object class. So

While calling equals() method using the help of Wrapper class reference then its

\*/

System.*out*.println("u1.equals(u2): "+u1.equals(u2));

}

}

/\*

Here is the output of this program:

\*/

Example:

**package**com.lara.ObjectNotes;

**publicclass** ONotes39

{

**publicstaticvoid** main(String[] args)

{

String s1 = "abc";

String s2 = "abc";

System.*out*.println("s1.equals(s2): "+s1.equals(s2));

}

}

/\*

As we know that equals() method already override in all String related classes. So while we are checking equality between two String object then it will check the content inside both

objects. It will not check the memory address. While it will call equals() method then String object reference will call its own equals() method. So while checking equality in this case

it will return as true. Because both object references are belongs to same class i.e String class and also these objects having same values.

\*/

Hint:

While we are trying to evaluate or compare between two similar kind of objects then we have to use both method one is equals() method and another is hashCode() method for checking equality. If we are trying to evaluate or compare between two similar kind of primitive types then in that case we have to use this operator (==) i.e equal operator.

In all Wrapper classes equals() method got override, no need to override equals() method from object class. While we are trying to check equality of similar kind of wrapper classes references that time it will not call Object class equals() method. It will call its own equals() method.

String class also already override equals() &toString() method.

In StringBuilder&StringBufferequals() method not override.

In all Collection API equals() method got override.

**hashCode() Method:**

This method is used for generate int primitive type or Integer Wrapper class type or hexadecimal type number format of memory address. In short term its showing memory address in the format of this below kinds.

1. int primitive types.
2. Integer Wrapper class types.
3. Hexadecimal number formats.

hashCode() method also override inside Collection API from Object class and also override inside. In String class also both equals() method and hashCode() method.

hashCode() method represents memory address of an object which is refer by the reference variable of same class. This reference can be of two types.

1. Hexadecimal representation
2. Integer representation.

Here below is an example of this method.

Example:

**package**com.lara.ObjectNotes;

**class** A4

{

**int**i;

}

**publicclass** ONotes40

{

**publicstaticvoid** main(String args[])

{

A4 a1 = **new**A4();

a1.i = 10;

System.*out*.println(a1.hashCode());

}

}

/\*

Here inside class A4 we have not override hashCode() method from object classes. While we are trying to check memory address in terms of hashcode then we are getting hashcode value.

This hashcode value is normally coming from when JVM check that in user defined class whether equals() method is there or not if not found then it will call parent class equals() method. So

here in this case it will call hashCode() method from Object class. We can't check what implementation written inside hashCode() method of Object class because its a native method.

If you want to know little bit about native method then check hint below of this program.

Here is the output of this programs:

24052850

This is not a simple number. Its indicate that this is the memory address of that object which is calling this hashCode() method in the form of Integer class types or long primitive types value.

\*/

Hint:

Native method is the process through which we can integrate more than one language with another. Using this native method process we can call other language functions or methods or classes etc… If we want to check implementation of this native method using the help of source code then we can’t check because those classes written like that way so that no one can check that method.

From this program we got to know that if we are calling Object class hashCode() method then we will get hashcode generated value from that native method. If we are overriding hashCode() method from the Object class and then implements after that we will get the value of content inside object.

**How to override hashCode() method from Object class?**

**package**com.lara.ObjectNotes;

**class** B2

{

**int**i;

**publicint** hashCode()

{

**return**i;

}

}

**publicclass**ONotes41

{

**publicstaticvoid** main(String[] args)

{

B2 b1 = **new**B2();

b1.i = 100;

//Here b1 object reference of class B2 is calling its own hashCode() method not hashCode() method of Object class.

System.*out*.println("b1.hashCode(): "+b1.hashCode());

}

}

/\*

Here in this program we have created one class name as B2 where we have declared one variable int type i.ei and one method hashCode() which is override from Object class and

user implement inside this method which will return value of i from this method. Inside main class while we are calling hashCode() method using help of same class B2 reference

then it will not call Object class hashCode() method its calling only its own hashCode() method where its return the value of i.

Here below is the output of this program:

b1.hashCode(): 100

While we are initialize the value of i using reference of class B2 object i.e b1 with value as 100 then JVM first check what was the value i before this, then JVM will get that i value already

assigned as default value i.e 0. While user again assigned 100 to this i value then its changes that default value and place new value as 100. So while its printing this value of i then its

returning as 100. So output is coming as 100.

\*/

Hint:

From this Program we got to know that if we are override and check the content of the object then we will get proper output. If we are calling Object class hashCode() method then it will check the memory address

of then object and return same memory address in Integer format or hexadecimal format.

**State Management:**

State management of Object of a class means we have to keep maintain of content inside object and store pass this content into different states like store in session, request, cookies or some other way. In short form we have to transfer value of object from one form to another but content remain same as it is.

Here is the example below for this management.

Example:

**package**com.lara.ObjectNotes;

**class** C2

{

**int**i, j;

/\*Here hashCode() method got override inside class C2. So while we are checking hashCode value then it will return this hashCode() value. Here user adding to variable inside this method

\* and return same value.

\*/

**publicint** hashCode()

{

**return** (i+j);

}

}

**publicclass** ONotes42

{

**publicstaticvoid** main(String[] args)

{

C2 c1 = **new**C2();

c1.i = 100;

c1.j = 200;

System.*out*.println("(i+j): "+c1.hashCode());

}

}

/\*

\*Here we have created one object of class C2 i.e c1 and through which we are assigning value of i and j as 100 and 200. So while we are calling hashCode() method of class C2 then we are

\* getting 300 by adding the value of i and j. Here c1 is calling hashCode() method of class C2. So we are getting answer like this.

\*

\*Here below is the output of this program:

\*(i+j): 300

\*/

**Difference Between print(String x) and print(Object obj):**

**print(String x):**

This method is taking one parameter as argument as i.e String type variable. This method return String type value if the String variable is not null. Otherwise it will return null if the value is null.

Example:

**public** **void** println(String x) {

**synchronized** (**this**) {

print(x);

newLine();

}

}

**public** **void** print(String s) {

**if** (s == **null**) {

s = "null";

}

write(s);

}

**private** **void** write(String s) {

**try** {

**synchronized** (**this**) {

ensureOpen();

textOut.write(s);

textOut.flushBuffer();

charOut.flushBuffer();

**if** (autoFlush && (s.indexOf('\n') >= 0))

out.flush();

}

}

**catch** (InterruptedIOException x) {

Thread.*currentThread*().interrupt();

}

**catch** (IOException x) {

trouble = **true**;

}

}

**print(Object obj):**

Here in this method we have to pass Object class reference as parameter. While we are calling this method then we are passing user defined object references into this method argument. But while it will call then inside println() method one print() method is there which will convert user defined object reference into String type and then convert that value to bit types. While return this method then value will return as String type.

Example:

**public** **void** println(Object x) {

String s = String.*valueOf*(x);

**synchronized** (**this**) {

print(s);

newLine();

}

}

**public** **static** String valueOf(Object obj) {

**return** (obj == **null**) ? "null" : obj.**toString()**;

}

**public** **void** print(String s) {

**if** (s == **null**) {

s = "null";

}

write(s);

}

Here while we are alone calling only object reference in sop then JVM will call this println(Object obj) method where its calling valueOf() method using String class name and assigned value to a String variables like this below:

String s = String.*valueOf*(x);

While String class call this method then inside this method one login written y Sun Microsystem developer that first it will check whether reference value is null or not, if not then it will call again toString() method. While calling toString() method then also JVM first check whether user already override toString() method inside this reference class or not and found that its already override so it will not call toString() method of String class. After convert this value to String variable and assigned to a String variable i.e s then it will call print method where it will write the value inside console.

Example:

**package** com.lara.ObjectNotes;

**class** E2

{

**int** i, j;

/\*Here we have override this toString() method from Object class which is returning the value of i and j. This method is used for for checking of content inside the object.

\*/

**public** String toString()

{

**return** "i: "+i+", "+"j: "+j;

}

/\*Here we have override hashCode() method from Object class. But implementation given different by user. So here user return addition of two variable declared inside class E2 i. e

\* i and j.

\*/

**public** **int** hashCode()

{

**return** (i+j);

}

}

**public** **class** ONotes43

{

**public** **static** **void** main(String[] args)

{

/\*Here we have assigned different values to two variables declared inside class E2 i.e i and j. Before assigned values to these variables i.e i and j JVM first check what was the values

\* while checking this JVM found that these variables was assigned as default values. When user again assigned values to these variables then JVM will change default values to this

\* new values.

\*/

E2 e1 = **new** E2();

e1.i = 100;

e1.j = 200;

/\*Here while we are trying to print object of class E2 i.e e1 then JVM first check whether toString() method override inside class E2 or not. Here User already override inside class

\* E2. But as we have already know that while we are trying to print any class object reference then it will call first println() method of PrintStream class and there it will call String

\* class valueOf() method.

\*/

System.*out*.println(e1);

System.*out*.println("e1.hashCode(): "+e1.hashCode());

}

}

/\*

Here while we are printing this reference of class E2 i.e e1 then JVM will check first toString() method override inside this class or not and found its already override to this class E2.

So it will call toString() method of class E2 where user return value of i and j inside this method. So it will return then value of i and j after calling this method.

In second line while user want to call hashCode() method using the reference of class E2. So while calling hashCode() method then also JVM first check whether hashCode() method

already override inside this class E2 or not. Here JVM found that its already override. So it will call this hashCode() method from this class E2. While calling this method user return

concatenation of these two variable of class E2 i.e i and j. So it will return addition of these two values. So we will get 300. Here below is the output of this program.

i: 100, j: 200

e1.hashCode(): 300

\*/

Examples: