Object class in Java

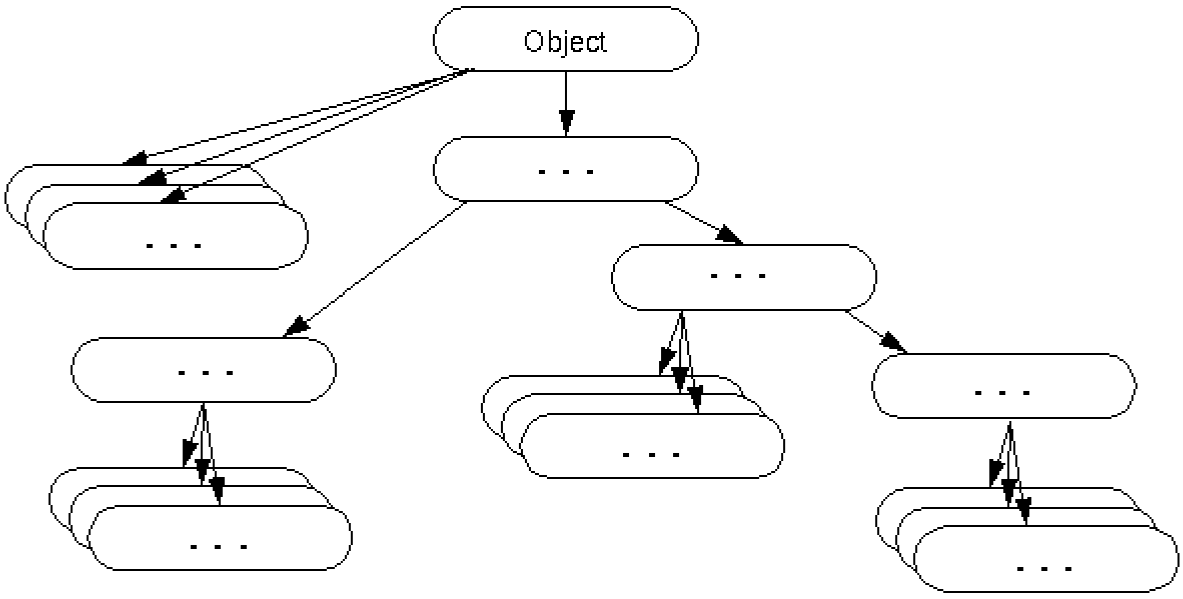
The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object. For example:

```
1. Object obj=getObject();//we don't know what object will be returned from this method
```

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.



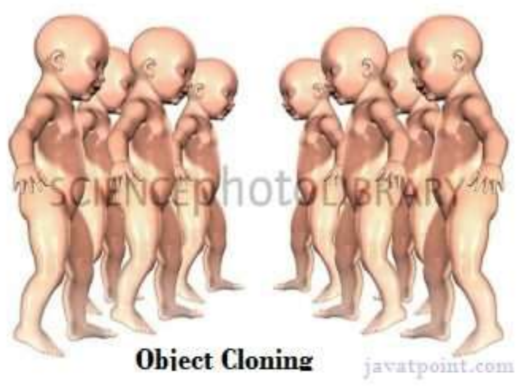
Methods of Object class

The Object class provides many methods. They are as follows:

Method	**Description**
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the this class.
public int hashCode()	returns the hashcode number for this object.

public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos) throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Object Cloning in Java



The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

The **clone() method** is defined in the Object class. Syntax of the clone() method is as follows:

```
1. protected Object clone() throws CloneNotSupportedException
```

Why use clone() method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.

Advantage of Object cloning

Although Object.clone() has some design issues but it is still a popular and easy way of copying objects. Following is a list of advantages of using clone() method:

- You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
- It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project. Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done.
- Clone() is the fastest way to copy array.

Disadvantage of Object cloning

Following is a list of some disadvantages of clone() method:

- To use the Object.clone() method, we have to change a lot of syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone() etc.
- We have to implement cloneable interface while it doesn't have any methods in it. We just have to use it to tell the JVM that we can perform clone() on our object.
- Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.
- Object.clone() doesn't invoke any constructor so we don't have any control over object construction.
- If you want to write a clone method in a child class then all of its superclasses should define the clone() method in them or inherit it from another parent class. Otherwise, the super.clone() chain will fail.
- Object.clone() supports only shallow copying but we will need to override it if we need deep cloning.

Example of clone() method (Object cloning)

Let's see the simple example of object cloning

```
class Student18 implements Cloneable{
int rollno;
String name;

Student18(int rollno,String name){
this.rollno=rollno;
this.name=name;
}

public Object clone()throws CloneNotSupportedException{
return super.clone();
}

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

```
Student18 s1=new Student18(101,"amit");


Student18 s2=(Student18)s1.clone();


System.out.println(s1.rollno+" "+s1.name);

System.out.println(s2.rollno+" "+s2.name);


}catch(CloneNotSupportedException c){}


}

}
```

```
101 amit

101 amit
```

As you can see in the above example, both reference variables have the same value. Thus, the clone() copies the values of an object to another. So we don't need to write explicit code to copy the value of an object to another.

If we create another object by new keyword and assign the values of another object to this one, it will require a lot of processing on this object. So to save the extra processing task we use clone() method.

```
package com.tutorialspoint;

import java.util.GregorianCalendar;

public class ObjectDemo {

   public static void main(String[] args) {

      // create a new ObjectDemo object
      GregorianCalendar cal = new GregorianCalendar();

      // print current time
      System.out.println("" + cal.getTime());

      // print the class of cal
      System.out.println("" + cal.getClass());

      // create a new Integer
      Integer i = new Integer(5);

      // print i
```

```
        System.out.println("" + i);

        // print the class of i
        System.out.println("" + i.getClass());
    }
}
```

```
public class HashCode {
    public static void main(String[] args)
    {
        //Create integer object
        Integer i = new Integer("155");
        //Returned hash code value for this object
        int hashValue = i.hashCode();
        System.out.println("Hash code Value for object is: " + hashValue);
    }
}
```

objequals

```
public class JavaObjectequalsExample1 {
   static int a = 10, b=20;
    int c;
    // Constructor
    JavaObjectequalsExample1()
    {
        System.out.println("Addition of 10 and 20 : ");
        c=a+b;
        System.out.println("Answer : "+c);
    }

    // Driver code
    public static void main(String args[])
    {
        System.out.println("1st object created...");
        JavaObjectequalsExample1 obj1 = new JavaObjectequalsExample1();
        System.out.println("2nd object created...");
        JavaObjectequalsExample1 obj2 = new JavaObjectequalsExample1();
                                System.out.println("Objects are equal:" + obj1.equals(obj2));
    }
}
```

Without tostring

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }

 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");

   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

With tostring

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
```

```
  this.city=city;
 }

public String toString(){//overriding the toString() method
 return rollno+" "+name+" "+city;
}
 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");

   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()
 }
}
```