Blog Site: [Java Guides] **Author: [Ramesh Fadatare]**

java.lang.Object Class API Guide

The Object class, in the java.lang package sits at the top of the class hierarchy tree. Every class is a descendant, direct or indirect, of the `Object` class. Every class you use or write inherits the instance methods of `Object`. You need not use any of these methods, but, if you choose to do so, you may need to override them with code that is specific to your class.
In this article we will discuss below methods inherited from Object class.

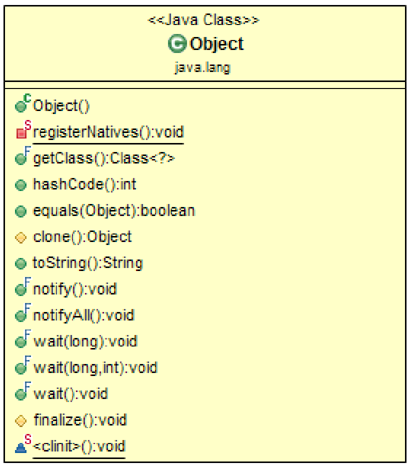
Object Class Methods

1.protected Object clone()
2.boolean equals(Object obj)
3.protected void finalize()
4.Class<?> getClass()
5.int hashCode()
6.void notify()
7.void notifyAll()
8.void wait()
9.String toString()

The `notify, notifyAll`, and `wait` methods of `Object` all play a part in synchronizing the activities of independently running threads in a program. There are five of these methods:

•public final void notify()
•public final void notifyAll()
•public final void wait()
•public final void wait(long timeout)
•public final void wait(long timeout, int nanos)

The below diagram is a Object class diagram shows a list of methods it provides.



Let's discuss above each method with examples.

1. protected Object clone() Method

`clone()` method creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object. The general intent is that for any object x, the expression:

```
x.clone() != x
```

will be true, and that the expression:

```
x.clone().getClass() == x.getClass()
```

will be true, but these are not absolute requirements. While it is typically the case that:

```
x.clone().equals(x)
```

will be true, this is not an absolute requirement.
By convention, the returned object should be obtained by calling super.clone. If a class and all of its superclasses (except Object) obey this convention, it will be the case that x.clone().getClass() == x.getClass().
`clone()` method throws a CloneNotSupportedException if the object's class does not support the Cloneable interface. Subclasses that override the clone method can also throw this exception to indicate that an instance cannot be cloned.

protected Object clone() Method Example

This example shows the usage of `clone()` method:

```
public class ObjectClass {

    public static void main(String[] args) {
        Date date = new Date();
        System.out.println(date.toString());
        Date date2 = (Date) date.clone();
        System.out.println(date2.toString());
    }
}
```

Output:

```
Tue Sep 04 14:15:00 IST 2018
Tue Sep 04 14:15:00 IST 2018
```

Let's discuss one more example using the `Closable` interface and `clone()` method together. First, create a `Person` class which implements a `Closable` interface.

```
public class Person implements Cloneable {
     private String firstName;
     private String lastName;

     public String getFirstName() {
         return firstName;
     }

     public void setFirstName(String firstName) {
         this.firstName = firstName;
     }

     public String getLastName() {
         return lastName;
     }

     public void setLastName(String lastName) {
         this.lastName = lastName;
     }

    @Override
    public Object clone() throws CloneNotSupportedException {
        Person person = (Person) super.clone();
        return person;
    }

    @Override
    public String toString() {
        return "Person [firstName=" + firstName + ", lastName=" + lastName + "]";
    }

    public static void main(String[] args) throws CloneNotSupportedException {
        Person person = new Person();
        person.setFirstName("Ramesh");
        person.setLastName("Fadatare");

        System.out.println(person.toString());

        Person person2 = (Person) person.clone();

        System.out.println(person2.toString());
    }
}
```

Output:

```
Person [firstName=Ramesh, lastName=Fadatare]
Person [firstName=Ramesh, lastName=Fadatare]
```

Note that we have to override the *clone()* method from *Object* class and made a call to Object *clone()*method using *super.clone().*

```
@Override
public Object clone() throws CloneNotSupportedException {
        Person person = (Person) super.clone();
        return person;
}
```

2. boolean equals(Object obj)

The java.lang.Object.equals(Object obj) indicates whether some other object is "equal to" this one. The equals method implements an equivalence relation on non-null object references:

- **•It is reflexive: for any non-null reference value x, *`x.equals(x)`* should return true.**
- **•It is symmetric: for any non-null reference values x and y, *`x.equals(y)`* should return true if and only if y.equals(x) returns true.**
- **•It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.**
- **•It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.**
- **•For any non-null reference value x, x.equals(null) should return false.**

boolean equals(Object obj) Example

Let's create a simple example to demonstrate the usage of equals method:

```
// get an integer, which is an object
  Integer x = new Integer(50);

  // get a float, which is an object as well
  Float y = new Float(50f);

  // check if these are equal,which is
  // false since they are different class
  System.out.println("" + x.equals(y));

  // check if x is equal with another int 50
  System.out.println("" + x.equals(50));
```

Output:

```
false
true
```

The equals() method for Person class is:

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Person other = (Person) obj;
    if (firstName == null) {
        if (other.firstName != null)
            return false;
    } else if (!firstName.equals(other.firstName))
            return false;
    if (lastName == null) {
        if (other.lastName != null)
            return false;
    } else if (!lastName.equals(other.lastName))
            return false;
    return true;
}
```

Let's test above equals method:

```
public class Person implements Cloneable {

    private String firstName;
    private String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }
```

```
      public void setLastName(String lastName) {
          this.lastName = lastName;
      }

     @Override
     public boolean equals(Object obj) {
        if (this == obj)
           return true;
        if (obj == null)
           return false;
        if (getClass() != obj.getClass())
           return false;
        Person other = (Person) obj;
        if (firstName == null) {
           if (other.firstName != null)
              return false;
        } else if (!firstName.equals(other.firstName))
              return false;
        if (lastName == null) {
           if (other.lastName != null)
              return false;
        } else if (!lastName.equals(other.lastName))
              return false;
        return true;
     }

 }
```

Output:

```
Both objects equal :: true
```

Note: If you override *`equals()`*, you must override *`hashCode()`* as well.

3. protected void finalize() Method

The *java.lang.Object.finalize()* is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. A subclass overrides the finalize method to dispose of system resources or to perform other cleanups.
The finalize() method may be called automatically by the system, but when it is called, or even if it is called, is uncertain. Therefore, you should not rely on this method to do your cleanup for you. For example, if you don't close file descriptors in your code after performing I/O and you expect finalize() to close them for you, you may run out of file descriptors.

protected void finalize() Method Example

Let's override finalize() method from Object class into Person class and test it using main() method.

```
public class Person {
    private String firstName;
    private String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    // This method is called just before an object is garbage collected
    @Override
    protected void finalize() throws Throwable {
        // TODO Auto-generated method stub
        super.finalize();
    }
    @Override
    public String toString() {
        return "Person [firstName=" + firstName + ", lastName=" + lastName + "]";
    }

    public static void main(String[] args) {
        Person person = new Person();
        person.setFirstName("Ramesh");
        person.setLastName("Fadatare");

        System.out.println("Before Finalize");
        try {
            person.finalize();
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
     System.out.println("After Finalize");
    }
}
```

Let us compile and run the above program, this will produce the following result −

```
Before Finalize
After Finalize
```

4. Class<?> getClass() Method

The *java.lang.Object.getClass()* method returns the runtime class of an object. That Class object is the object that is locked by static *synchronized* methods of the represented class.

Class<?> getClass() Method Example

The following example shows the usage of lang.Object.getClass() method.

```
public class Person {
    private String firstName;
    private String lastName;
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public static void main(String[] args) {
        Person person = new Person();
        System.out.println(person.getClass());
    }
}
```

Let us compile and run the above program, this will produce the following result −

```
class com.javaguides.corejava.lang.Person
```

4. int hashCode() Method

The java.lang.Object.hashCode() method returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by java.util.HashMap.

The general contract of hashCode is:

•Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

•If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
•It is not required that if two objects are unequal according to the java.lang.Object.equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

int hashCode() Method Example

The following example shows the usage of lang.Object.hashCode() method:

```
public class Person {
 private String firstName;
 private String lastName;

 public String getFirstName() {
  return firstName;
 }

 public void setFirstName(String firstName) {
  this.firstName = firstName;
 }

 public String getLastName() {
  return lastName;
 }

 public void setLastName(String lastName) {
  this.lastName = lastName;
 }

 @Override
 public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + ((firstName == null) ? 0 : firstName.hashCode());
  result = prime * result + ((lastName == null) ? 0 : lastName.hashCode());
  return result;
```

```
   }

   public static void main(String[] args) {
    Person person = new Person();
    person.setFirstName("Ramesh");
    person.setLastName("Fadatare");

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

    Person person1 = new Person();
    person1.setFirstName("Ramesh");
    person1.setLastName("Fadatare");
    System.out.println(person1.hashCode());
   }
 }
```

Let us compile and run the above program, this will produce the following result −

```
-1066062211
-1066062211
```

By definition, if two objects are equal, their hashcode must also be equal. If you override the equals() method, you change the way two objects are equated and Object's implementation of hashCode() is no longer valid. Therefore, if you override the equals() method, you must also override the hashCode() method as well.

6. notify() , 7. notifyAll() and 8. wait() Methods

- •void notify() - This method wakes up a single thread that is waiting on this object's monitor.
- •void notifyAll() - This method wakes up all threads that are waiting on this object's monitor.
- •void wait() - This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

notify() , 7. notifyAll() and 8. wait() Methods Example

The following example shows the usage of notify() , notifyAll() and wait() Methods:

```
import java.util.Collections;
import java.util.LinkedList;

import java.util.List;



public class ObjectClassNotifyNotifyAllAndWaitExample {

 private List<String> synchedList;



 public ObjectClassNotifyNotifyAllAndWaitExample() {

   // create a new synchronized list to be used

   synchedList = Collections.synchronizedList(new LinkedList<>());

 }

 // method used to remove an element from the list

 public String removeElement() throws InterruptedException {

   synchronized (synchedList) {

    // while the list is empty, wait

    while (synchedList.isEmpty()) {

     System.out.println("List is empty...");

         synchedList.wait();

         System.out.println("Waiting...");

    }

    String element = synchedList.remove(0);

    return element;

   }
 }

 // method to add an element in the list

 public void addElement(String element) {

   System.out.println("Opening...");

   synchronized (synchedList) {

     // add an element and notify all that an element exists

     synchedList.add(element);

     System.out.println("New Element added:'" + element + "'");

     synchedList.notifyAll();

     System.out.println("notifyAll called!");

   }

   System.out.println("Closing in AddElement method...");

 }

 public static void main(String[] args) {

   final ObjectClassNotifyNotifyAllAndWaitExample demo = new
ObjectClassNotifyNotifyAllAndWaitExample();

   Runnable runA = () -> {

     try {

       String item = demo.removeElement();

       System.out.println("" + item);

     } catch (InterruptedException ix) {

       System.out.println("Interrupted Exception!");

     } catch (Exception x) {
       System.out.println("Exception thrown.");
     }
   };

   Runnable runB = () -> {
     // run adds an element in the list and starts the loop
     demo.addElement("Hello!");
   };
   try {
     Thread threadA1 = new Thread(runA, "A");
     threadA1.start();
     Thread.sleep(500);
     Thread threadA2 = new Thread(runA, "B");
     threadA2.start();
     Thread.sleep(500);
     Thread threadB = new Thread(runB, "C");
     threadB.start();
     Thread.sleep(1000);
     threadA1.interrupt();
     threadA2.interrupt();
   } catch (InterruptedException x) {
   }
```

```
 }

}
```

Output:

```
List is empty...
List is empty...
Opening...
New Element added:'Hello!'
notifyAll called!
Waiting...
Closing in AddElement method...
Hello!
Waiting...
List is empty...
Interrupted Exception!
```

9. String toString() Method

The `java.lang.Object.toString()` method returns a string representation of the object. In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

String toString() Method Example

The following example shows the usage of `lang.Object.toString()` method.

```
public class Person {
    private String firstName;
    private String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return "Person [firstName=" + firstName + ", lastName=" + lastName + "]";
    }

    public static void main(String[] args) {
        Person person = new Person();
        person.setFirstName("Ramesh");
        person.setLastName("Fadatare");
        System.out.println(person.toString());
    }
}
```

Let us compile and run the above program, this will produce the following result −

```
Person [firstName=Ramesh, lastName=Fadatare]
```