```
Object class methods
++++++++++++++++++++
public class Object {

  //Consturctor of the Class
  public Object();

  //To know the name of the class which implements an interface at the runtime.
  public final native Class<?> getClass();

  //At the time of creating of Object,JVM will mantain unique address.
  public native int hashCode();

  //To compare the objects
  public boolean equals(java.lang.Object);


  //It is used during cloning of Object
  protected native Object clone() throws CloneNotSupportedException;


  //While printing the reference the method gets called automatically
  public String toString();


  //methods used while working with MultiThreading Environment[Interthread
communication]
  public final native void notify();
  public final native void notifyAll();
  public final void wait() throws InterruptedException;
  public final native void wait(long) throws InterruptedException;
  public final void wait(long, int) throws   InterruptedException;


  //Called by Garbage Collector beforing cleaning the object from the Heap memory.
  protected void finalize() throws Throwable;

}

toString()
++++++++++
=> We can use this method to get the String representation of the Object.
=> When we try to print the reference,this method gets called automatically(Magic
method).
=> if our class doesn't contain this method, then Object class toString() will be
called automatically.

Note::
class Object
{
      public String toString() {
                  //method body
            return getClass().getName() + "@" + Integer.toHexString(hashCode());
      }
}

eg#1.
Before Overriding toString() of Object class
++++++++++++++++++++++++++++++++++++++++++++
```

```
class Student
{
	String name;
	int rollNo;

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

public class Test
{
	public static void main(String[] args)
	{
			Student s1 = new Student("sachin",10);
			Student s2 = new Student("dhoni",7);
			System.out.println(s1);
			System.out.println(s2);

			System.out.println();

			System.out.println(s1.toString());
			System.out.println(s2.toString());
	}
}

Output
Student@15db9742
Student@6d06d69c

Student@15db9742
Student@6d06d69c

After Overriding toString() in Student class
++++++++++++++++++++++++++++++++++++++++++
class Student
{
	String name;
	int rollNo;

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

	@Override
	public String toString()
	{
		return name +"-----" + rollNo;
	}
}

Output
sachin-----10
dhoni-----7
```

sachin-----10
dhoni-----7

Can u name few inbuilt classes where toString() is overriden to print the Object data than the Object reference?
Ans. String,StringBuilder,StringBuffer,Wrapper classes,Thread class ,All Collection objects......

2.
Class<?> getClass();
  This method is used to get the runtime object details while dealing with "Implementation classes".

eg#1.

```
interface INotification
{
	public void notifyUsers();
}

class EmailNotification implements INotification
{
	@Override
	public void notifyUsers(){
		//logic to notify the user through email...
		System.out.println("Notification through EMAIL....");
	}
}
class SMSNotification implements INotification
{
	@Override
	public void notifyUsers(){
		//logic to notify the user through sms...
		System.out.println("Notification through SMS....");
	}
}

class Factory{

	public INotification getInstance(String mode){

		INotification notification =null;

		if (mode.equalsIgnoreCase("SMS"))
			notification = new SMSNotification();
		else if(mode.equalsIgnoreCase("EMAIL"))
			notification = new EmailNotification();
		else if(mode.equalsIgnoreCase("PUSH"))
			notification =new PushNotification();
		else
			throw new RuntimeException("Mode is invalid...");

		return notification;
	}
}

class PushNotification implements INotification
{
```

```
	//loose coupling,abstraction
	@Override
	public void notifyUsers(){
		//logic to notify the user through push service...
		System.out.println("Notification through PUSH ....");
	}

}

//Client Code
public class Test
{
	public static void main(String[] args)
	{
		Factory factory= new Factory();

		//loose coupling,abstraction
		INotification obj1 = factory.getInstance("SMS");
		obj1.notifyUsers();
		System.out.println("Notification object is :: "+obj1.getClass());

		System.out.println();

		//loose coupling,abstraction
		INotification obj2 = factory.getInstance("EMAIL");
		obj2.notifyUsers();
		System.out.println("Notification object is :: "+obj2.getClass());

		System.out.println();

		INotification obj3 = factory.getInstance("PUSH");
		obj3.notifyUsers();
		System.out.println("Notification object is :: "+obj3.getClass());

		System.out.println();

		INotification obj4 = factory.getInstance("Call");
		obj4.notifyUsers();
		System.out.println("Notification object is :: "+obj4.getClass());


	}
}

Output
Notification through SMS....
Notification object is :: class SMSNotification

Notification through EMAIL....
Notification object is :: class EmailNotification

Notification through PUSH ....
Notification object is :: class PushNotification

Exception in thread "main" java.lang.RuntimeException: Mode is invalid...
	at Factory.getInstance(Test.java:36)
	at Test.main(Test.java:79)
```

hashcode()
++++++++++
public int hashCode()

=> For every object jvm will generate a unique number which is nothing but a hashCode.
=> For all hashing related datastructure like HashSet,HashMap,...jvm will use hashcode to save the objects.
=> If objects are stored according to the hashcode searching for a data becomes more efficient.

Note:
 if we don't override hashCode(),then object class hashCode() will be executed which generates the hashCode based on the address
 of the object.

=> Overriding the hashCode() is said to be proper, iff for every object we have to generate a unique number as a hashcode for
   every object.

```
class Object
{
      public String toString(){
            return getClass().getName() + "@"+ Integer.toHexString(hashCode());
        }

      //body will come at the runtime:: by linking with other language libraries
      public native int hashCode(); // object address -> hashcode
}

eg#1.
class Student
{
      @Override
      public int hashCode()
      {
            return 10;
      }
}

//Client Code
public class Test
{
      public static void main(String[] args)
      {
                  Student s1 = new Student();
                  System.out.println(s1.hashCode());

                  Student s2 = new Student();
                  System.out.println(s2.hashCode());
      }
}
```

Hashcode for 2 objects should not be same it is a improper approach.
Output
 10
 10

eg#2.

```
class Student
{
	int rollNo;

	Student(int rollNo){
		this.rollNo = rollNo;
	}

	@Override
	public int hashCode()
	{
		return rollNo;
	}
}

//Client Code
public class Test
{
	public static void main(String[] args)
	{
			Student s1 = new Student(10);
			System.out.println(s1.hashCode());

			Student s2 = new Student(7);
			System.out.println(s2.hashCode());
	}
}
```

Output

```
10
7
```

toString() vs hashCode() method

++++++++++++++++++++++++++++++

=> Which class toString() and hashCode() will be called?

Ans.toString() :: object class

hashCode() :: Object class

eg#1.

```
class Student
{
	int rollNo;

	Student(int rollNo){
		this.rollNo = rollNo;
	}
}

//Client Code
public class Test
{
	public static void main(String[] args)
	{
			Student s1 = new Student(10);
			System.out.println(s1);

			Student s2 = new Student(7);
```

```
                System.out.println(s2);
        }
}

=> Which class toString() and hashCode() will be called?
Ans.toString() :: Object class
    hashCode() :: Student class

eg#1.
class Student
{
        int rollNo;

        Student(int rollNo){
                this.rollNo = rollNo;
        }

        @Override
        public int hashCode(){
                return rollNo;
        }
}

//Client Code
public class Test
{
        public static void main(String[] args)
        {
                Student s1 = new Student(10);
                System.out.println(s1);

                Student s2 = new Student(7);
                System.out.println(s2);
        }
}
Output
Student@a
Student@7

eg#3.
class Student
{
        int rollNo;
        String name;

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

        @Override
        public int hashCode(){
                return rollNo;
        }

        @Override
        public String toString(){
                return name + " "+ rollNo;
        }
```

```
}

//Client Code
public class Test
{
      public static void main(String[] args)
      {
                  Student s1 = new Student(10,"sachin");
                  System.out.println(s1);

                  Student s2 = new Student(7,"dhoni");
                  System.out.println(s2);
      }
}

Output
sachin  10
dhoni    7

Note:
1. If we are overriding toString(), then making a call to hashCode() is in the
hands of programmer, hashCode() won't be called
   automatically.
2. if we don't override toString(), then toString() internally will make a call to
hashCode(), so overriding
   toString() and hashCode() is totally decided by the programmer.




equals()
+++++++
=> To check whether two objects are equal or not we use this method.
=> If our class doesn't contain equals() then object class equals() will be
executed.

public boolean equals(Object obj){
      return (this == obj);//reference check
}


eg#1.
class Student
{
      int rollNo;
      String name;

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

//Client Code
public class Test
{
      public static void main(String[] args)
      {
                  Student s1 = new Student(10,"sachin");
```

```
                Student s2 = new Student(7,"dhoni");
                Student s3 = new Student(10,"sachin");
                Student s4 = s1;

                System.out.println(s1.equals(s2));
                System.out.println(s1.equals(s3));
                System.out.println(s1.equals(s4));
        }
}
```

while overridng equals() we need to make sure, we handle the following cases
1. Compare the entire object data, based on the result return true or false.
2. if we are passing different type of data, it would result in "ClassCastException"
    a. handle the ClassCastException and return false.
3. if we are passing null type of data, it would result in "NullPointerException"
    a. handle the NullPointerException and return false.

eg#1.

```
class Student
{
      int rollNo;
      String name;

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


      //overriding equals() as per our requirement
      @Override
      public boolean equals(Object obj)
      {
            try
            {
                  //compare the contents and return boolean result
                  int     id1= this.rollNo;
                  String name1 = this.name;

                  Student std = (Student)obj;
                  int id2 = std.rollNo;
                  String name2 = std.name;

                  if (id1==id2  && name1.equals(name2))
                  {
                        return true;
                  }
                  else
                  {
                        return false;
                  }
            }
            catch (ClassCastException ce)
            {
                  //write the logic
                  return false;
            }
```

```
            catch(NullPointerException ne)
            {
                  //write the logic
                  return false;
            }
            catch(Exception e)
            {
                  //write the logic
                  return false;
            }

      }
}

//Client Code
public class Test
{
      public static void main(String[] args)
      {
                  Student s1 = new Student(10,"sachin");
                  Student s2 = new Student(7,"dhoni");
                  Student s3 = new Student(10,"sachin");
                  Student s4 = s1;

                  System.out.println(s1.equals(s2));//false
                  System.out.println(s1.equals(s3));//true
                  System.out.println(s1.equals(s4));//true
                  System.out.println(s1.equals("divya"));
                  System.out.println(s1.equals(null));
      }
}
Output
false
true
true
false
false

More simplified version of equals()
++++++++++++++++++++++++++++++++++
public boolean equals(Object obj) {
            try
            {
                  Student std = (Student)obj;
                  //compare the contents and return boolean result
                  if (rollNo==std.rollNo  && name.equals(std.name))
                        return true;
            }
            catch (ClassCastException ce)
            {
                  //write the logic
                  return false;
            }
            catch(NullPointerException ne)
            {
                  //write the logic
                  return false;
            }
            catch(Exception e)
```

```
            {
                  //write the logic
                  return false;
            }
            return false;
}

More simplified version of equals()
++++++++++++++++++++++++++++++++++
public boolean equals(Object obj) {
            if (this == obj)
            {
                  return true;
            }

            if (obj instanceof Student)
            {
                  Student std = (Student)obj;
                  //compare the contents and return boolean result
                  if (rollNo==std.rollNo  && name.equals(std.name))
                        return true;
            }
            return false;
}
```