Object class-

It is the parent class of all the classes in java. It is called as topmost class of java which is present in java.lang package.

If our class doesn't extends any other class then it is the direct child class of object. If our class extends any other class then it is the indirect child class of Object.

There are different methods of object class are as follows.

1. **Public final Class getClass()-**
   This class is used to get the metadata of class.i.e. returns runtime class definition of an object

```
package com.wipro.jpmorgan;

public class Example {

	public static void main(String[] args) {

		Example example = new Example();
		System.out.println(example.getClass().getName());
		System.out.println(example.getClass().getSimpleName());

	}
}
```

2. **Public int hashCode()-**

   For every object unique number is generated by JVM called as hashcode. It is based on address of the object but it doesn't mean hashCode represents address of the object. Jvm will be using hashCode while saving objects into hashing related data structures like HashSet, HashMap, and Hashtable etc.

   The most of java native method are written in c or c++ that why not possible to show the body.

Note-

1) If two objects are equal, their hashcode will be same.

2) If two object hashcode are same, you cannot guaranty that objects are equal.

3) Overriding hashCode() method is said to be proper if and only if for every object we have to generate a unique number as hashcode for every object

Example

```
package com.wipro.jpmorgan;

public class Example {

	public static void main(String[] args) {

		Example example1 = new Example();
		Example example2 = new Example();

		System.out.println(example1.hashCode());
		System.out.println(example2.hashCode());
	}
}
```

3) Public Boolean equals (Object obj)-

It compare the given object to this object.

If our class doesn't contain .equals() method then object class .equals() method will be executed which is always meant for reference comparison[address comparison]. i.e., if two references pointing to the same object then only .equals() method returns true .

Example-

```
package com.velocity;

public class Employee {

	int empId;
	String empName;
```

```
	public static void main(String[] args) {
		Employee emp1 = new Employee();
		emp1.empId = 1;
		emp1.empName = "Ashok";

		Employee emp2 = new Employee();
		emp2.empId = 2;
		emp2.empName = "Sachin";

		System.out.println(emp1.equals(emp2));
	}
}
```

Output- false

4) protected Object clone() throws CloneNotSupportedException-

It creates and returns the exact copy (clone) of this object.

The main objective of cloning is to maintain backup purposes.(i.e., if something goes wrong we can recover the situation by using backup copy.)

Example-

```
package com.wipro.jpmorgan;

public class Example implements Cloneable {

	int x;

	public static void main(String[] args) throws
CloneNotSupportedException {

		Example example1 = new Example();
		example1.x = 50;

		System.out.println("First Object data is>>" + example1.x);

		Object example2 = example1.clone();

		System.out.println("Second Object data is>>" + example2);
	}
}
```

5) **public String toString() –**

It returns the string representation of this object.

Whenever we are try to print any object reference internally toString() method will be executed.

```
package com.wipro.jpmorgan;

public class Example {

	int x;

	@Override
	public String toString() {
		return "Example [x=" + x + "]";
	}

	public static void main(String[] args) throws
CloneNotSupportedException {

		Example example1 = new Example();
		example1.x = 50;

		System.out.println("First Object data is>>" + example1);

	}
}
```

6) public final void notify()-

It wakes up single thread, waiting on this object's monitor.

7) **public final void notifyAll()-**

It wakes up all the threads, waiting on this object's monitor.

8) public final void wait(long timeout)throws InterruptedException()-

It causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).

9) public final void wait(long timeout,int nanos)throws InterruptedException-

It causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).

10) public final void wait()throws InterruptedException

It causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).

11) protected void finalize()throws Throwable

It is invoked by the garbage collector before object will be destroyed to perform clean up activity.

Int ques : Contract between equals and hashcode method

1) If 2 objects are equal by .equals() method compulsory their hashcodes must be equal (or) same. That is If r1.equals(r2) is true then r1.hascode()==r2.hashcode( ) must be true.
2) If 2 objects are not equal by .equals() method then there are no restrictions on hashCode() methods. They may be same (or) may be different. That is If r1.equals(r2) is false then r1.hashCode()==r2.hashCode() may be same (or) may be different.
3) If hashcodes of 2 objects are equal we can't conclude anything about .equals() method it may returns true (or) false. That is If r1.hashCode()==r2.hashCode() is true then r1.equals(r2) method may returns true (or) false.
4) If hashcodes of 2 objects are not equal then these objects are always not equal by .equals() method also. That is If r1.hashCode()==r2.hashCode() is false then r1.equals(r2) is always false.

To maintain the above contract between .equals() and hashCode() methods whenever we are overriding .equals() method compulsory we should override hashCode() method.

Final keyword-

We can apply final to variables, method and class.

Final variable-

A variable which is declared with final keyword called as final variables.

Once you assigned any value to that variables then it won't be changed. It works like constants in java.

How to declare the final variables-

final int a=5;

Example-1

```
class FinalDemo {
	public static void main(String args[]) {
		final int a = 5;
		System.out.println(a);
	}
}
```

Output-

5

Example-2

```
class FinalDemo {
	public static void main(String args[]) {
		final int a = 5;
		for (int a = 5; a <= 10; a++) {
			System.out.println(a);
		}
	}
}
```

In this example, we will get compile time error, final variable values does not changed.

Final method-

Method which is defined with final keyword called as final method.

How to declare the final method-

```
public final void test(){
        //business logic here.
}
```

Note- Final method cannot be overridden.

Example-3

```
class X {
final void test(){
System.out.println("This is x class-test method");
}

        class Y extends X {
final void test(){
System.out.println("This is y class-test method");
}

                public static void main(String args[]) {
                        X x = new Y();
                        x.test();
                }
}
```

Output-

Nothing

In this example, we will get compile time error final method cannot be override final method from X

Final class-

The class which is defined with final keyword called as final class.

How to declare the final class

```
final class Test {
        // business logic
}
```

How you stop others from inheriting your class-

By making class as final.

```
final class X {
	public final void test() {
		System.out.println("x class-test method");
	}
}

class Y extends X {

}

public class FinalKeyword {
	public static void main(String[] args) {
		Y y = new Y();
		y.test();
	}
}
```

Wrapper class-

It provides the mechanism to convert primitive's data type into object and object into primitives data type called as wrapper class.

Process of converting primitive's data type into object called as "Autoboxing." And process of converting object into primitive's data type called as "Unboxing."

There are 8 classes of java.lang.package are known as wrapper classes in java.

Integer

Short

Byte

Long

Double

Character

Boolean

Float

Example-

```
package com.object;

public class WrapperDemo {

    public static void main(String[] args) {

        int a = 20; // primitive data type

        Integer i = new Integer(a); // autoboxing
        System.out.println("i>>" + i);

        int b = i.intValue();   //unboxing
        System.out.println("b>>"+b);
    }
}
```