**Object class**

The [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) class, in the java.lang package, sits at the top of the class hierarchy tree. The Object class is the parent class of all the classes in java by default. The Object class is beneficial if you want to refer any object whose type you don't know. Every class you use or write inherits the instance methods of Object.

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: 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.

The Object class provides many methods. They are

* public boolean equals(Object obj)🡺Indicates whether some other object is "equal to" this one.

The equals() method provided in the Object class uses the identity operator (==) to determine whether two objects are equal. For primitive data types, this gives the correct result. For objects, however, it does not. The equals() method provided by Object tests whether the object *references* are equal—that is, if the objects compared are the exact same object.

* public int hashCode()🡺 Returns a hash code value for the object. The value returned by hashCode() is the object's hash code, which is the object's memory address in hexadecimal.
* public String toString()🡺Returns a string representation of the object. The Object's toString() method returns a String representation of the object, which is very useful for debugging. The String representation for an object depends entirely on the object, which is why you need to override toString() in your classes.
* public final Class getClass()🡺 Returns the runtime class of an object. The getClass() method returns a Class object, which has methods you can use to get information about the class, such as its name (getSimpleName()), its superclass (getSuperclass()), and the interfaces it implements (getInterfaces()).
* protected Object clone() throws CloneNotSupportedException🡺 If a class, or one of its superclasses, implements the Cloneable interface, you can use the clone() method to create a copy from an existing object. To create a clone, you write:

*aCloneableObject*.clone();

Object's implementation of this method checks to see whether the object on which clone() was invoked implements the Cloneable interface. If the object does not, the method throws a CloneNotSupportedException exception. clone() must be declared as:

protected Object clone() throws CloneNotSupportedException

* protected void finalize() throws Throwable🡺Called by the garbage collector on an object when garbage collection determines that there are no more references to the object. The Object class provides a callback method, finalize(), that *may be* invoked on an object when it becomes garbage. Object's implementation of finalize() does nothing—you can override finalize() to do cleanup, such as freeing resources.

**Custom Exception:** If the developer is creating his/her own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.With the help of custom exception, we can have our own exception and message.