Java OOPs Misc

Object class in Java

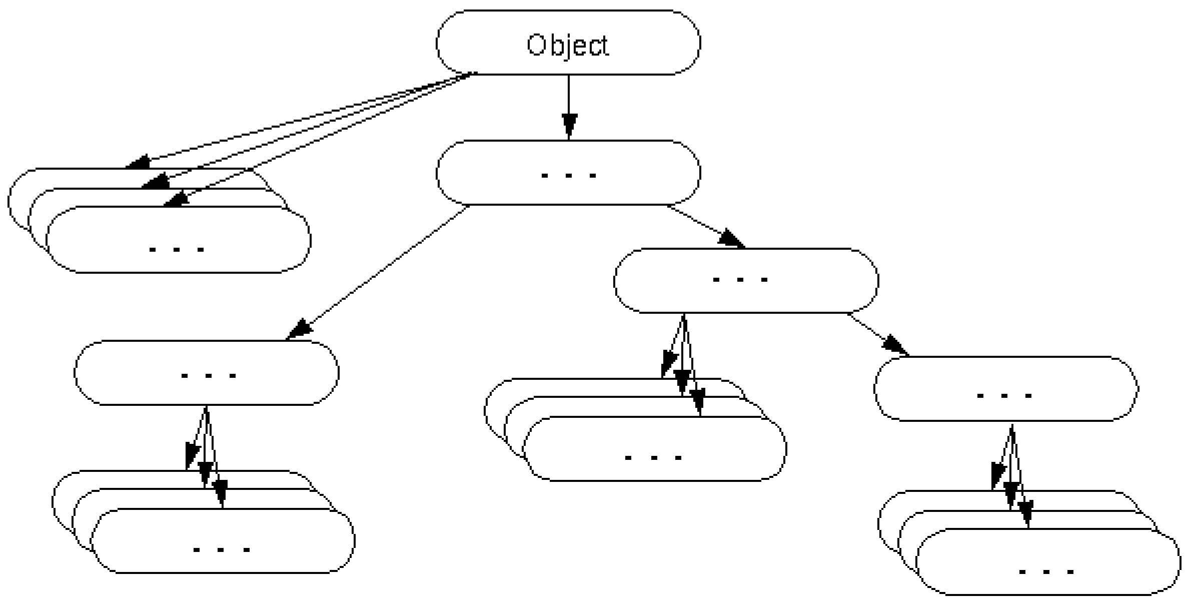
The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

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

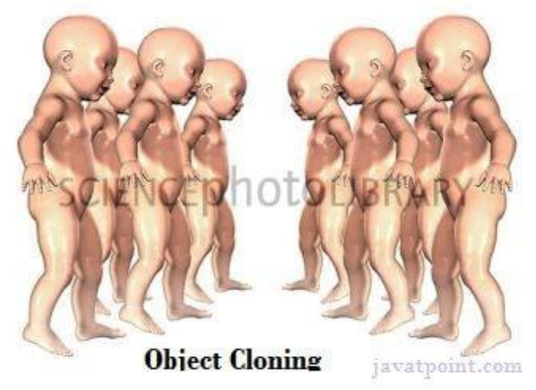
The Object class provides many methods. They are as follows:

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of 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.

We will have the detailed learning of these methods in next chapters.

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.

Keep Watching

Competitive questions on Structures in Hindi

00:00/03:34

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){}

}
}
```

Test it Now

```
Output:101 amit
       101 amit
```

download the example of object cloning

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.

Java Math class

Java Math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

Unlike some of the StrictMath class numeric methods, all implementations of the equivalent function of Math class can't define to return the bit-for-bit same results. This relaxation permits implementation with better-performance where strict reproducibility is not required.

If the size is int or long and the results overflow the range of value, the methods addExact(), subtractExact(), multiplyExact(), and toIntExact() throw an ArithmeticException.

For other arithmetic operations like increment, decrement, divide, absolute value, and negation overflow occur only with a specific minimum or maximum value. It should be checked against the maximum and minimum value as appropriate.

Example 1

```java
public class JavaMathExample1
{
    public static void main(String[] args)
    {
        double x = 28;
        double y = 4;

        // return the maximum of two numbers
        System.out.println("Maximum number of x and y is: " +Math.max(x, y));

        // return the square root of y
        System.out.println("Square root of y is: " + Math.sqrt(y));

        //returns 28 power of 4 i.e. 28*28*28*28
        System.out.println("Power of x and y is: " + Math.pow(x, y));

        // return the logarithm of given value
        System.out.println("Logarithm of x is: " + Math.log(x));
        System.out.println("Logarithm of y is: " + Math.log(y));

        // return the logarithm of given value when base is 10
        System.out.println("log10 of x is: " + Math.log10(x));
        System.out.println("log10 of y is: " + Math.log10(y));

        // return the log of x + 1
        System.out.println("log1p of x is: " +Math.log1p(x));

        // return a power of 2
        System.out.println("exp of a is: " +Math.exp(x));

        // return (a power of 2)-1
        System.out.println("expm1 of a is: " +Math.expm1(x));
    }
}
```

Test it Now

Output:

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
log1p of x is: 3.367295829986474
exp of a is: 1.446257064291475E12
expm1 of a is: 1.446257064290475E12
```

Example 2

```java
public class JavaMathExample2
{
    public static void main(String[] args)
    {
        double a = 30;

        // converting values to radian
        double b = Math.toRadians(a);

        // return the trigonometric sine of a
        System.out.println("Sine value of a is: " +Math.sin(a));

        // return the trigonometric cosine value of a
        System.out.println("Cosine value of a is: " +Math.cos(a));

        // return the trigonometric tangent value of a
        System.out.println("Tangent value of a is: " +Math.tan(a));

        // return the trigonometric arc sine of a
        System.out.println("Sine value of a is: " +Math.asin(a));

        // return the trigonometric arc cosine value of a
        System.out.println("Cosine value of a is: " +Math.acos(a));

        // return the trigonometric arc tangent value of a
        System.out.println("Tangent value of a is: " +Math.atan(a));

        // return the hyperbolic sine of a
        System.out.println("Sine value of a is: " +Math.sinh(a));

        // return the hyperbolic cosine value of a
        System.out.println("Cosine value of a is: " +Math.cosh(a));

        // return the hyperbolic tangent value of a
        System.out.println("Tangent value of a is: " +Math.tanh(a));
    }
}
```
Test it Now

Output:

```
Sine value of a is: -0.9880316240928618
Cosine value of a is: 0.15425144988758405
Tangent value of a is: -6.405331196646276
Sine value of a is: NaN
Cosine value of a is: NaN
Tangent value of a is: 1.5374753309166493
Sine value of a is: 5.343237290762231E12
Cosine value of a is: 5.343237290762231E12
Tangent value of a is: 1.0
```

Java Math Methods

The **java.lang.Math** class contains various methods for performing basic numeric operations such as the logarithm, cube root, and trigonometric functions etc. The various java math methods are as follows:

Basic Math methods

Method	Description
Math.abs()	It will return the Absolute value of the given value.
Math.max()	It returns the Largest of two values.
Math.min()	It is used to return the Smallest of two values.
Math.round()	It is used to round of the decimal numbers to the nearest value.
Math.sqrt()	It is used to return the square root of a number.
Math.cbrt()	It is used to return the cube root of a number.

Math.pow()	It returns the value of first argument raised to the power to second argument.
Math.signum()	It is used to find the sign of a given value.
Math.ceil()	It is used to find the smallest integer value that is greater than or equal to the argument or mathematical integer.
Math.copySign()	It is used to find the Absolute value of first argument along with sign specified in second argument.
Math.nextAfter()	It is used to return the floating-point number adjacent to the first argument in the direction of the second argument.
Math.nextUp()	It returns the floating-point value adjacent to d in the direction of positive infinity.
Math.nextDown()	It returns the floating-point value adjacent to d in the direction of negative infinity.
Math.floor()	It is used to find the largest integer value which is less than or equal to the argument and is equal to the mathematical integer of a double value.
Math.floorDiv()	It is used to find the largest integer value that is less than or equal to the algebraic quotient.
Math.random()	It returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
Math.rint()	It returns the double value that is closest to the given argument and equal to mathematical integer.
Math.hypot()	It returns sqrt(x^2 +y^2) without intermediate overflow or underflow.
Math.ulp()	It returns the size of an ulp of the argument.
Math.getExponent()	It is used to return the unbiased exponent used in the representation of a value.
Math.IEEEremainder()	It is used to calculate the remainder operation on two arguments as prescribed by the IEEE 754 standard and returns value.
Math.addExact()	It is used to return the sum of its arguments, throwing an exception if the result overflows an int or long.
Math.subtractExact()	It returns the difference of the arguments, throwing an exception if the result overflows an int.
Math.multiplyExact()	It is used to return the product of the arguments, throwing an exception if the result overflows an int or long.
Math.incrementExact()	It returns the argument incremented by one, throwing an exception if the result overflows an int.
Math.decrementExact()	It is used to return the argument decremented by one, throwing an exception if the result overflows an int or long.
Math.negateExact()	It is used to return the negation of the argument, throwing an exception if the result overflows an int or long.
Math.toIntExact()	It returns the value of the long argument, throwing an exception if the value overflows an int.

Logarithmic Math Methods

Method	Description
Math.log()	It returns the natural logarithm of a double value.
Math.log10()	It is used to return the base 10 logarithm of a double value.
Math.log1p()	It returns the natural logarithm of the sum of the argument and 1.
Math.exp()	It returns E raised to the power of a double value, where E is Euler's number and it is approximately equal to 2.71828.
Math.expm1()	It is used to calculate the power of E and subtract one from it.

Trigonometric Math Methods

Method	Description
Math.sin()	It is used to return the trigonometric Sine value of a Given double value.
Math.cos()	It is used to return the trigonometric Cosine value of a Given double value.
Math.tan()	It is used to return the trigonometric Tangent value of a Given double value.
Math.asin()	It is used to return the trigonometric Arc Sine value of a Given double value
Math.acos()	It is used to return the trigonometric Arc Cosine value of a Given double value.
Math.atan()	It is used to return the trigonometric Arc Tangent value of a Given double value.

Hyperbolic Math Methods

Method	Description
Math.sinh()	It is used to return the trigonometric Hyperbolic Cosine value of a Given double value.
Math.cosh()	It is used to return the trigonometric Hyperbolic Sine value of a Given double value.
Math.tanh()	It is used to return the trigonometric Hyperbolic Tangent value of a Given double value.

Angular Math Methods

Method	Description
Math.toDegrees	It is used to convert the specified Radians angle to equivalent angle measured in Degrees.
Math.toRadians	It is used to convert the specified Degrees angle to equivalent angle measured in Radians.

Wrapper classes in Java

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the *java.lang* package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

Wrapper class Example: Primitive to Wrapper

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

Output:

```
20 20 20
```

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

Wrapper class Example: Wrapper to Primitive

```java
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```

Output:

```
3 3 3
```

Java Wrapper classes Example

```java
//Java Program to convert all primitives into its corresponding
//wrapper objects and vice-versa
public class WrapperExample3{
public static void main(String args[]){
byte b=10;
short s=20;
int i=30;
long l=40;
float f=50.0F;
double d=60.0D;
char c='a';
boolean b2=true;

//Autoboxing: Converting primitives into objects
Byte byteobj=b;
Short shortobj=s;
Integer intobj=i;
Long longobj=l;
Float floatobj=f;
Double doubleobj=d;
Character charobj=c;
Boolean boolobj=b2;

//Printing objects
System.out.println("---Printing object values---");
System.out.println("Byte object: "+byteobj);
System.out.println("Short object: "+shortobj);
System.out.println("Integer object: "+intobj);
System.out.println("Long object: "+longobj);
System.out.println("Float object: "+floatobj);
System.out.println("Double object: "+doubleobj);
System.out.println("Character object: "+charobj);
System.out.println("Boolean object: "+boolobj);

//Unboxing: Converting Objects to Primitives
byte bytevalue=byteobj;
short shortvalue=shortobj;
int intvalue=intobj;
long longvalue=longobj;
float floatvalue=floatobj;
double doublevalue=doubleobj;
char charvalue=charobj;
boolean boolvalue=boolobj;

//Printing primitives
System.out.println("---Printing primitive values---");
System.out.println("byte value: "+bytevalue);
System.out.println("short value: "+shortvalue);
System.out.println("int value: "+intvalue);
System.out.println("long value: "+longvalue);
System.out.println("float value: "+floatvalue);
System.out.println("double value: "+doublevalue);
System.out.println("char value: "+charvalue);
System.out.println("boolean value: "+boolvalue);
}}
```

Output:

```
---Printing object values---
Byte object: 10
Short object: 20
Integer object: 30
Long object: 40
Float object: 50.0
Double object: 60.0
Character object: a
Boolean object: true
---Printing primitive values---
byte value: 10
short value: 20
int value: 30
long value: 40
float value: 50.0
double value: 60.0
char value: a
boolean value: true
```

Custom Wrapper class in Java

Java Wrapper classes wrap the primitive data types, that is why it is known as wrapper classes. We can also create a class which wraps a primitive data type. So, we can create a custom wrapper class in Java.

```
//Creating the custom wrapper class
class Javatpoint{
private int i;
Javatpoint(){}
Javatpoint(int i){
this.i=i;
}
public int getValue(){
return i;
}
public void setValue(int i){
this.i=i;
}
@Override
public String toString() {
  return Integer.toString(i);
}
}
//Testing the custom wrapper class
public class TestJavatpoint{
```

```
public static void main(String[] args){
Javatpoint j=new Javatpoint(10);
System.out.println(j);
}}
```

Output:

```
10
```

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```
class Operation{
 int data=50;

 void change(int data){
 data=data+100;//changes will be in the local variable only
 }

 public static void main(String args[]){
   Operation op=new Operation();

   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);

 }
}
```

download this example

```
Output:before change 50
       after change 50
```

Another Example of call by value in java

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:

```
class Operation2{
 int data=50;

 void change(Operation2 op){
 op.data=op.data+100;//changes will be in the instance variable
 }


 public static void main(String args[]){
   Operation2 op=new Operation2();

   System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);

 }
}
```

download this example

```
Output:before change 50
       after change 150
```

Java Strictfp Keyword

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform. So, now you have better control over the floating-point arithmetic.

Legal code for strictfp keyword

The strictfp keyword can be applied on methods, classes and interfaces.

```
strictfp class A{}//strictfp applied on class
```

```
strictfp interface M{}//strictfp applied on interface
```

```
class A{
strictfp void m(){}//strictfp applied on method
}
```

Illegal code for strictfp keyword

The strictfp keyword **cannot** be applied on abstract methods, variables or constructors.

```
class B{
strictfp abstract void m();//Illegal combination of modifiers
}
```

```
class B{
strictfp int data=10;//modifier strictfp not allowed here
}
```

```
class B{
strictfp B(){}//modifier strictfp not allowed here
}
```

Creating API Document | javadoc tool

We can create document api in java by the help of **javadoc** tool. In the java file, we must use the documentation comment /**... */ to post information for the class, method, constructor, fields etc.

Let's see the simple class that contains documentation comment.

```
package com.abc;
/** This class is a user-defined class that contains one methods cube.*/
public class M{

/** The cube method prints cube of the given number */
public static void  cube(int n){System.out.println(n*n*n);}
}
```

To create the document API, you need to use the javadoc tool followed by java file name. There is no need to compile the javafile.

On the command prompt, you need to write:

```
javadoc M.java
```

to generate the document api. Now, there will be created a lot of html files. Open the index.html file to get the information about the classes.

Java Command Line Arguments

1. Command Line Argument

2. Simple example of command-line argument
3. Example of command-line argument that prints all the values

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```java
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

```
compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo
```

```
Output: Your first argument is: sonoo
```

Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

```java
class A{
public static void main(String args[]){

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}
```

```
compile by > javac A.java
run by > java A sonoo jaiswal 1 3 abc
```

```
Output: sonoo
        jaiswal
        1
        3
        abc
```

Difference between object and class

There are many differences between object and class. A list of differences between object and class are given below:

No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}

5)	Object is created **many times** as per requirement.	Class is declared **once**.
6)	Object **allocates memory when it is created**.	Class **doesn't allocated memory when it is created**.
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class keyword.

Let's see some real life example of class and object in java to understand the difference well:

Class: Human **Object:** Man, Woman

Class: Fruit **Object:** Apple, Banana, Mango, Guava wtc.

Class: Mobile phone **Object:** iPhone, Samsung, Moto

Class: Food **Object:** Pizza, Burger, Samosa

Difference between method overloading and method overriding in java

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Java Method Overloading example

```
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
```

Java Method Overriding example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
```

```
void eat(){System.out.println("eating bread...");}
}
```