**Java:**

**OOPS:**

Object

Class

Inheritance

Polymorphism

Abstraction

Encapsulation

Class:

Class is logical entity

class is a blueprint or template

class has

methods

fields

blocks

constructor

nested class and interfaces.

Object:

Object is a physical and logical entity.

Object is run and real time entity.

Object is an instance of a class. Since class is a blueprint or template. Object is created from class.

Object has state, behavior and identity.

Object can be created in 4 different ways.

1. By reference variable
2. By method
3. By constructor
4. By new keyword

|  |  |
| --- | --- |
| Using new keyword | } → constructor gets called |
| Using [newInstance()](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html" \l "newInstance--" \t "_blank) method of Class class | } → constructor gets called |
| Using [newInstance()](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Constructor.html" \l "newInstance-java.lang.Object...-" \t "_blank) method of Constructor class | } → constructor gets called |
| Using clone() method | } → no constructor call |
| Using deserialization | } → no constructor call |

**New keyword:** - applies to both no arg and arg constructor

Employee obj1 = new Employee();

**New instance of Class class**

Calls the default constructor

Employee obj1 = (Employee) Class.forName(“xxxx”).newInstance();

Or

Employee obj2 = Employee.class.newInstance();

**New instance of Constructor class**

Similar to the newInstance() method of Class class, There is one newInstance() method in the java.lang.reflect.Constructor class which we can use to create objects. We can also call parameterized constructor, and private constructor by using this newInstance() method.

Both newInstance() methods are known as reflective ways to create objects. In fact newInstance() method of Class class internally uses newInstance() method of Constructor class. That's why the later one is preferred and also used by different frameworks like Spring, Hibernate, Struts etc.

Constructor<Employee> constructor = Employee.class.getConstructor();

Employee emp3 = constructor.newInstance();

**Clone()**

Whenever we call clone() on any object, the JVM actually creates a new object for us and copies all content of the previous object into it. Creating an object using the clone method does not invoke any constructor.

To use clone() method on an object we need to implement Cloneable and define the clone() method in it.

Employee emp4 = (Employee) emp3.clone();

**Deserialization:**

Whenever we serialize and deserialize an object, the JVM creates a separate object for us. In deserialization, the JVM doesn’t use any constructor to create the object.

To deserialize an object we need to implement a Serializable interface in our class.

ObjectInputStream in = new ObjectInputStream(new FileInputStream(“data.obj”));

Employee emp5 = (Employee) in.readObject();

As we can see in the above bytecode snippets, all 4 methods are called and get converted to invokevirtual (object creation is directly handled by these methods) except the first one, which got converted to two calls: one is new and other is invokespecial (call to constructor).

Conversion:

**String to int**

int i=Integer.parseInt("200"); (to - primitive)

Integer i=Integer.valueOf(s); (to - non primitive)

# int to String

int i=10;

String s=String.valueOf(i); (P)

int i=10;

String s=Integer.toString(i); (NP)

int i=200;

String s=String.format("%d",i);

# String to long

long l=Long.parseLong("200");

# long to String

long i=9993939399L;//L is the suffix for long

String s=String.valueOf(i);

long i=9993939399L;

String s=Long.toString(i);

# String to float

float f=Float.parseFloat("23.6");

# float to String

 float f=12.3F;//F is the suffix for float

 String s=String.valueOf(f);

 float f=89.7F;

 String s=Float.toString(f);

# String to double

1. double d=Double.parseDouble("23.6");

# double to String

1. double d=12.3;//floating literal is double by default
2. String s=String.valueOf(d);
3. double d=89.7;
4. String s=Double.toString(d);

# String to Date

String sDate1="31/12/1998";

Date date1=new SimpleDateFormat("dd/MM/yyyy").parse(sDate1);

# Date to String

 Date date = Calendar.getInstance().getTime();

 DateFormat dateFormat = new SimpleDateFormat("yyyy-mm-dd hh:mm:ss");

 String strDate = dateFormat.format(date);

# String to char

 String s="hello";

 char c=s.charAt(0);

 String s1="hello";

 char[] ch=s1.toCharArray();

 for(int i=0;i<ch.length;i++){

 System.out.println("char at "+i+" index is: "+ch[i]);

 }

# char to String

char c='S';

String s=String.valueOf(c);

char c='M';

String s=Character.toString(c);

# String to Object

String s="hello";

Object obj=s;

## String to Class object

 Class c=Class.forName("java.lang.String");

 System.out.println("class name: "+c.getName());

 System.out.println("super class name: "+c.getSuperclass().getName());

# Object to String

 Emp e=new Emp();

 String s=e.toString();

 String s2=String.valueOf(e);

## StringBuilder

 StringBuilder sb=new StringBuilder(s);

 sb.reverse();

 String rev=sb.toString();

# int to long

 int i=200;

 long l=i;  equivalent to Long l2=Long.valueOf(i);

 Long l= new Long(i);//first way

 Long l2=Long.valueOf(i);//second way

# long to int

 long l=500;

 int i=(int)l;

 Long l= new Long(10);

 int i=l.intValue();

# int to double

 int i=200;

 double d=i;

 Double d= new Double(i);//first way

 Double d2=Double.valueOf(i);//second way

# double to int

 double d=10.5;

 int i=(int)d;

 Double d=new Double(10.5);

 int i=d.intValue();

# char to int

## Get ASCII value

 char c='a';

 char c2='1';

 int a=c;

 int b=c2;

## Character.getNumericValue()

 char c='1';

 int a=Character.getNumericValue(c);

## String.valueOf()

 char c='1';

 int a=Integer.parseInt(String.valueOf(c));

# int to char

 int a=65;

 char c=(char)a;

If you add '0' with int variable, it will return actual value in the char variable. The ASCII value of '0' is 48. So, if you add 1 with 48, it becomes 49 which is equal to 1. The ASCII character of 49 is 1.

 int a=1;

 char c=(char)(a+'0');

If you store integer value in a single quote, it will store actual character in char variable.

 int a='1';

 char c=(char)a;

## Character.forDigit()

 int REDIX=10;//redix 10 is for decimal number, for hexa use redix 16

 int a=1;

 char c=Character.forDigit(a,REDIX);

To get the hexa value, use redix 16 in Character.forDigit() method.

# String to boolean

String s1="true";

boolean b1=Boolean.parseBoolean(s1);

## Boolean.valueOf()

String s1="true";

Boolean b1=Boolean.valueOf(s1);

# boolean to String

boolean b1=true;

String s1=String.valueOf(b1);

 boolean b2=false;

 String s1=Boolean.toString(b1);

# Date to Timestamp

 Date date = new Date();

 Timestamp ts=new Timestamp(date.getTime());

 Date date = new Date();

                 Timestamp ts=new Timestamp(date.getTime());

                 SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

# Timestamp to Date

 Timestamp ts=new Timestamp(System.currentTimeMillis());

                 Date date=new Date(ts.getTime());

# Binary to Decimal

 String binaryString="1010";

 int decimal=Integer.parseInt(binaryString,2);

# Decimal to Binary

Integer.toBinaryString(10)

# Hexadecimal to Decimal

 String hex="a";

 int decimal=Integer.parseInt(hex,16);

# Decimal to Hexadecimal

Integer.toHexString(10)

# Octal to Decimal

Integer.parseInt(octalString,8)

# Decimal to Octal

Integer.toOctalString(8)