Today topics
==========
Small topic
      a. serialVersionUID
  1. Cloneable
       shallow copy, deep copy
  2. Different ways of Creating an object
  3. Difference b/w ClassNotFoundException vs NoClassDefFoundError
  4. Command line arguments
  5. Singleton class/Design pattern using factory methods
Navin sir => SQL

serialVersionUID
===============
=> To perform Serialization & Deserialization internally JVM will use a unique identifier,which is nothing but serialVersionUID .
=> At the time of serialization JVM will save serialVersionUID with object.
=> At the time of Deserialization JVM will compare serialVersionUID and if it is matched then only object will be
     Deserialized otherwise we will get RuntimeException saying "InvalidClassException".

The process in depending on default serialVersionUID are :
1. After Serializing object if we change the .class file then we can't perform deserialization   because of mismatch in serialVersionUID of
    local class and serialized object in this case at the time of Deserialization we will get RuntimeException saying in "InvalidClassException".
2. Both sender and receiver should use the same version of JVM if there any incompatability in JVM versions then receive anable to
     deserializable because of different serialVersionUID , in this   case receiver will get RuntimeException saying "InvalidClassException".
3. To generate serialVersionUID internally JVM will use complexAlgorithm which may create performence problems.

Serialization
==========
```
class Dog implements Serializable{
      public static final long serialVersionUID = 1L;
      int i=10;
      int j=20;
}

FileOutputStream fos= new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
System.out.println("Serialization ended");
```

DeSerialization
=============
```
class Dog implements Serializable{
      public static final long serialVersionUID = 1L;
      int i=10;
      int j=20;
}
System.out.println("Deserialization started");
FileInputStream fis=new FileInputStream("abc.ser");
```

```
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog) ois.readObject();
System.out.println("Deserialization ended");
```

We can solve above problems by configuring our own serialVersionUID .

eg#1.

```
import java.io.Serializable;
public class Dog implements Serializable {
      private static final long serialVersionUID=1L;
      int i=10;
      int j=20;
}
```

```
import java.io.*;
public class Sender {
      public static void main(String[] args)throws IOException {
            Dog d=new Dog();
            FileOutputStream fos=new FileOutputStream("abc.ser");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            oos.writeObject(d);
      }
}
```

```
import java.io.*;
public class ReceiverApp {
      public static void main(String[] args) throws
IOException,ClassNotFoundException{
            FileInputStream fis=new FileInputStream("abc.ser");
            ObjectInputStream ois=new ObjectInputStream(fis);
            Dog d2=(Dog) ois.readObject();
            System.out.println(d2.i+"=====>"+d2.j);
      }
}
```

```
D:\TestApp>javac Dog.java

D:\TestApp>java Sender

D:\TestApp>javac Dog.java

D:\TestApp>java ReceiverApp
10=====>20
```

=> In the above program after serialization even though if we perform any change to Dog.class file we can deserialize object.
=> We can configure our own serialVersionUID both sender and receiver not required to maintain the same JVM versions.
      Note : some IDE's generate explicit serialVersionUID

Clone () method:
1. The process of creating exactly duplicate object is called cloning.
2. 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.)
3. We can perform cloning by using clone() method of Object class.

Signature

```
    protected native object clone() throws CloneNotSupportedException;
```

eg#1.

```
public class Test implements Cloneable{
      int i=10;
      int j=20;
      public static void main(String[] args)throws CloneNotSupportedException{
            Test t1=new Test();
            Test t2=(Test)t1.clone();
            t2.i=100;
            t2.j=200;
            System.out.println("Acutal object => "+t1.i+" "+t1.j);
            System.out.println("Cloned object => "+t2.i+" "+t2.j);
   }
}
```

Output

```
Acutal object => 10 20
Cloned object => 100 200
```

KeyPoints about Cloneable interface

==========

=> We can perform cloning only for Cloneable objects.

=> An object is said to be Cloneable if and only if the corresponding class implements Cloneable interface.

=> Cloneable interface present in java.lang package and does not contain any methods.

It is a marker interface where the required ability will be provided automatically by the JVM.

=> If we are trying to perform cloning on non-clonable objects then we will get RuntimeException saying "CloneNotSupportedException".

eg#1.

```
class Cat
{
      int i;
      Cat(int i){
            this.i=i;
      }
}
class Dog implements Cloneable
{
      Cat cat;
      int j;

      Dog(Cat cat,int j){
            this.cat=cat;
            this.j=j;
      }
      public Object clone()throws CloneNotSupportedException{
            return super.clone();
      }
}

public class Test{
      public static void main(String[] args)throws CloneNotSupportedException{
            Cat cat=new Cat(10);
            Dog d1=new Dog(cat,20);
            System.out.println("Acutal object => "+d1.cat.i+" "+d1.j);
```

```
            System.out.println("Perfoming cloning");
            Dog d2=(Dog)d1.clone();
            d2.cat.i=100;
            d2.j=200;

            System.out.println("Acutal object after cloning => "+d1.cat.i+"
"+d1.j);
            System.out.println("Cloned object data          => "+d2.cat.i+"
"+d2.j);
   }
}
Output
Acutal object => 10 20
Perfoming cloning
Acutal object after cloning => 100 20
Cloned object data          => 100 200

Note:
=> Shallow cloning is the best choice , if the Object contains only primitive
values.
=> In Shallow cloning by using main object reference , if we perform any change to
the contained object then those changes will be reflected
      automatically in cloned copy.
=> To overcome this problem we should go for Deep cloning

Deep Cloning :
1. The process of creating exactly independent duplicate object(including contained
objects also)   is   called deep cloning.
2. In Deep cloning , if main object contain any reference variable then the
corresponding Object   copy will also be created in cloned object.
3. Object class clone( ) method meant for Shallow Cloning , if we want Deep cloning
then the  programmer is responsible to implement by
     overriding clone( ) method.

eg#1.
class Cat
{
      int i;
      Cat(int i){
            this.i=i;
      }
}
class Dog implements Cloneable
{
      Cat cat;
      int j;

      Dog(Cat cat,int j){
            this.cat=cat;
            this.j=j;
      }
      public Object clone()throws CloneNotSupportedException{

            Cat c1= new Cat(cat.i);
            Dog d1=new Dog(c1,j);
            return d1;

      }
}
```

```
public class Test{
      public static void main(String[] args)throws CloneNotSupportedException{
            Cat cat=new Cat(10);
            Dog d1=new Dog(cat,20);
            System.out.println("Acutal object => "+d1.cat.i+" "+d1.j);

            System.out.println("Perfoming cloning");
            Dog d2=(Dog)d1.clone();
            d2.cat.i=100;
            d2.j=200;
            System.out.println("Acutal object after cloning => "+d1.cat.i+"
"+d1.j);
            System.out.println("Cloned object data              => "+d2.cat.i+"
"+d2.j);
   }
}
Output
Acutal object => 10 20
Perfoming cloning
Acutal object after cloning => 10 20
Cloned object data               => 100 200
```

Note:
In Deep cloning by using main Object reference if we perform any change to the contained Object those changes won't be reflected to the cloned object.

Example:

```
Test t1=new Test();
Test t2=(Test)t1.clone();
System.out.println(t1==t2); //false
System.out.println(t1.hashCode()==t2.hashCode()); //false
```

Singleton classes :
For any java class if we are allow to create only one object, such type of class is said to be singleton class.
Example:
1) Runtime class
2) ActionServlet
3) ServiceLocator
4) BusinessDelegate

eg#1

```
Runtime r1=Runtime.getRuntime();//getRuntime() method is a factory method
Runtime r2=Runtime.getRuntime();
Runtime r3=Runtime.getRuntime();
.............................................
.............................................
System.out.println(r1==r2);//true
System.out.println(r1==r3);//true
```

Advantage of Singleton class :
If the requirement is same then instead of creating a separate object for every person we will create only one object and
we can share that object for every required person we can achieve this by using singleton classes.

That is the main advantages of singleton classes are Performance will be improved and memory utilization will be improved.

Creation of our own singleton classes:
We can create our own singleton classes for this we have to use private constructor, static variable and factory method.

```
class Test {
            private static Test t=null;
            private Test(){}//to avoid object creation by the user using new
keyword

            public static Test getTest() //getTest() method is a factory method
            {
                  if(t==null){
                        t=new Test();
                  }
                  return t;
            }
}
class Client{
            public static void main(String[] args){
                  System.out.println(Test.getTest().hashCode());//1671711
                  System.out.println(Test.getTest().hashCode());//1671711
                  System.out.println(Test.getTest().hashCode());//1671711
                  System.out.println(Test.getTest().hashCode());//1671711
            }
}
```

We can create any xxxton classes like(double ton,triple ton...etc)

Example:

```
class Test {
            private static Test t1=null;
            private static Test t2=null
            private Test(){}

            public static Test getTest()//getTest() method is a factory method
            {
                  if(t1==null){
                        t1=new Test();
                        return t1;
                  }
                  else if(t2==null){
                        t2=new Test();
                        return t2;
                  }
                  else{
                        if(Math.random()<0.5) //Math.random() limit : 0<=x<1
                              return t1;
                        else
                              return t2;
                  }
      }
}
public class Client{
      public static void main(String[] args){
                  System.out.println(Test.getTest().hashCode());//1671711
```

```
                System.out.println(Test.getTest().hashCode());//11394033
                System.out.println(Test.getTest().hashCode());//11394033
                System.out.println(Test.getTest().hashCode());//1671711
      }
}
```

Factory method:
By using class name if we are calling a method and that method returns the same class object such type of method is called factory method.

Example:

```
Runtime r=Runtime.getRuntime();//getRuntime is a factory method.
DateFormat df=DateFormat.getInstance();
```

If object creation required under some constraints then we can implement by using factory method.

```
Calendar calendar = Calendar.getInstance();//static factory methods
String result = "name".toUpperCase();//instance factory methods
```

Different ways of Creating an Object

1. using new Operator

```
Test t=new Test();
```

2. using newInstance()

```
Class.forName("com.abc.main.Test").newInstance()
```

3. using clone()

```
Test t2=(Test)t1.clone();
```

4. using factorymethods

```
Runtime r=Runtime.getRuntime();
DateFormat df=DataFormat.getInstance();
```

5. using Serialization and DeSerialization

```
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Test t=(Test)ois.readObject();
```

new Vs newInstance( ) :
1. new is an operator to create an objects , if we know class name at the beginning then we can create an object by using new operator .
2. newInstance( ) is a method presenting class " Class " , which can be used to create object.
3. If we don't know the class name at the beginning and its available dynamically Runtime then we should go for newInstance() method

```
public class Test {
      public static void main(String[] args) throws Exception {
             Object o=Class.forName(arg[0]).newInstance( ) ;
            System.out.println(o.getClass().getName( ) );
      }
}
```

If dynamically provide class name is not available then we will get the RuntimeException saying ClassNotFoundException
To use newInstance( ) method compulsory corresponding class should contains no argument constructor ,
otherwise we will get the RuntimeException saying "InstantiationException".

if the constuctor is private then it would result in "IllegalAccessException"

Difference between new and newInstance( ) :
new
===
new is an operator , which can be used to create an object.
We can use new operator if we know the class name at the beginning.
      Test t= new Test( );
If the corresponding .class file not available at Runtime then we will get RuntimeException saying NoClassDefFoundError , It is unchecked.
To used new operator the corresponding class not required to contain no argument constructor

newInstance( )
============
newInstance( ) is a method , present in class Class , which can be used to create an object .
We can use the newInstance( ) method , If we don't class name at the beginning and available dynamically Runtime.
      Object o=Class.forName(arg[0]).newInstance( );
If the corresponding .class file not available at Runtime then we will get RuntimeException saying ClassNotFoundException , It is checked.
To used newInstance( ) method the corresponding class should compulsory contain no argument constructor , Other wise we will get
RuntimeException saying InstantiationException.

Difference between ClassNotFoundException & NoClassDefFoundError :
1. For hard coded class names at Runtime in the corresponding .class files not available we will get NoClassDefFoundError ,
    which is unchecked
            Test t = new Test( );
            In Runtime Test.class file is not available then we will get "NoClassDefFoundError"

2. For Dynamically provided class names at Runtime , If the corresponding .class files is not available then we will get the
    RuntimeException saying "ClassNotFoundException".
            Ex : Object o=Class.forname("Test").newInstance( );
      At Runtime if Test.class file not available then we will get the "ClassNotFoundException" , which is checked exception.

Difference between instanceof and isInstance( ) :
instanceof
instanceof an operator which can be used to check whether the given object is particular type or not We know at the type at
beginning it is available.

eg: String s = new String("sachin");
System.out.println(s instanceof Object);//true
      //If we know the type at the beginning only.

isInstance( )
isInstance( ) is a method , present in class Class , we can use isInstance( ) method to checked whether the given object is particular type or
not   We don't know at the type at beginning it is available Dynamically at Runtime.

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

```
                Test t = new Test( ) ;

      System.out.println(Class.forName(args[0]).isInstance(t));////arg[0] --- We
don't know the type at beginning
      }
}
java Test Test //true
java Test String //false
java Test Object //true
```