Serialization w.r.t Inheritance

===============================

Case 1:

  If parent class implements Serializable then automatically every child class by default implements Serializable.
  That is Serializable nature is inheriting from parent to child.
  Hence even though child class doesn't implements Serializable , we can serialize child class object if parent class implements
  serializable interface.

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;


class Animal implements Serializable{
      int i=10;
}
class Dog extends Animal{
      int j=20;
}

public class Test {
      public static void main(String[] args)throws
IOException,ClassNotFoundException{

      Dog d=new Dog();

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


      System.out.println("******************************");

      System.out.println("DeSerialization started");
      FileInputStream fis=new FileInputStream("abc.ser");
      ObjectInputStream ois=new ObjectInputStream(fis);
      Dog d1=(Dog)ois.readObject();
      System.out.println(d1.i+"====> "+d1.j);
      System.out.println("DeSerialization ended");


   }
}
```

Output

```
Serialization started
Serialization ended
******************************
DeSerialization started
10====> 20
DeSerialization ended
```

Even though Dog class does not implements Serializable interface explicitly but we can Serialize Dog object because its parent class
Animal already implements Serializable interface.
Note :Object class doesn't implement Serializable interface.

Case 2:
1. Even though parent class does not implementsSerializable we can serialize child object if child class implements Serializable
interface.

2. At the time of serialization JVM ignores the values of instance variables which are coming
from non Serializable parent then instead of original value JVM saves default values for those variables to the file.

3. At the time of Deserialization JVM checks whether any parent class is non Serializable or not.
If any parent class is nonSerializable JVM creates a separate object for every non Serializabl parent and
shares its instance variables to the current object.

4. To create an object for non-serializable parent JVM always calls no arg constructor
(default constructor) of that non Serializable parent hence every non Serializable parent should compulsory contain
no arg constructor otherwise we will get runtime exception "InvalidClassException".

eg#1.
```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;


class Animal {
      int i=10;
      Animal(){
            System.out.println("No arg Animal constructor");
      }
}
class Dog extends Animal implements Serializable{
      int j=20;
      Dog(){
            System.out.println("No arg Dog constructor");
      }
}
public class Test {
      public static void main(String[] args)throws
IOException,ClassNotFoundException{

      Dog d=new Dog();
      d.i=888;
      d.j=999;

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


      System.out.println("******************************");

      System.out.println("DeSerialization started");
      FileInputStream fis=new FileInputStream("abc.ser");
      ObjectInputStream ois=new ObjectInputStream(fis);
      Dog d1=(Dog)ois.readObject();
      System.out.println(d1.i+"====> "+d1.j);
      System.out.println("DeSerialization ended");
   }
}
```

Output

```
No arg Animal constructor
No arg Dog constructor
Serialization started
Serialization ended
******************************
DeSerialization started
No arg Animal constructor
10====> 999
DeSerialization ended
```

Agenda :
1. Externalization
2. Difference between Serialization & Externalization
3. SerialVersionUID

Externalization : (1.1 v)
1. In default serialization every thing takes care by JVM and programmer doesn't have any control.
2. In serialization total object will be saved always and it is not possible to save part of the object , which creates performance problems at certain point.
3. To overcome these problems we should go for externalization where every thing takes care by programmer and JVM doesn't have any control.
4. The main advantage of externalization over serialization is we can save either total object or part of the object based on our requirement.
5. To provide Externalizable ability for any object compulsory the corresponding class should   implements externalizable interface.
6. Externalizable interface is child interface of serializable interface.

Externalizable interface defines 2 methods :
1. writeExternal(ObjectOutput out) throws IOException
2. readExternal(ObjectInput in) throws IOException,ClassNotFoundException

public void writeExternal(ObjectOutput out) throws IOException
    This method will be executed automatically at the time of Serialization with in this method , we have to write code to save required variables to the file .

public void readExternal(ObjectInput in) throws IOException,ClassNotFoundException

   This method will be executed automatically at the time of deserialization with in this
   method , we have to write code to save read required variable from file and assign to the
   current object.

At the time of deserialization JVM will create a seperate new object by executing public no-arg constructor
on that object JVM will call readExternal() method.
Every Externalizable class should compusory contains public no-arg constructor otherwise we will get
RuntimeExcepion saying "InvaidClassException" .

eg#1.

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;
import java.io.Externalizable;
import java.io.ObjectOutput;
import java.io.ObjectInput;

class ExternalizableDemo implements Externalizable{
      String i;
      int j;
      int k;

      ExternalizableDemo(String i,int j,int k){
            this.i=i;
            this.j=j;
            this.k=k;
      }

      public ExternalizableDemo(){
            System.out.println("Zero arg constructor");
      }

      //Performing Serialization as per our requirement
      public void writeExternal(ObjectOutput out) throws IOException{
            System.out.println("call back method used while Serialization");
            out.writeObject(i);
            out.writeInt(j);
      }

      //Performing DeSerialization as per our requirement
      public void readExternal(ObjectInput in) throws
IOException,ClassNotFoundException{
            System.out.println("call back method used while DeSerialization");
            i=(String)in.readObject();
            j=in.readInt();
      }

}
public class Test {
      public static void main(String[] args)throws
IOException,ClassNotFoundException{
      ExternalizableDemo d=new ExternalizableDemo("nitin",100,200);

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


      System.out.println("******************************");

      System.out.println("DeSerialization started");
      FileInputStream   fis=new FileInputStream("abc.ser");
      ObjectInputStream ois=new ObjectInputStream(fis);
      d=(ExternalizableDemo)ois.readObject();
      System.out.println(d.i+"======>"+d.j+"======>"+d.k);
      System.out.println("DeSerialization ended");
   }
}
```

```
Output
Serialization started
call back method used while Serialization
Serialization ended
******************************
DeSerialization started
Zero arg constructor
call back method used while DeSerialization
nitin======>100======>0
DeSerialization ended
```

1. If the class implements Externalizable interface then only part of the object will be saved in the case output is
2. public no-arg constructor
3. nitin---- 10 ----- 0
4. If the class implements Serializable interface then the output is nitin --- 10 --- 20
5. In externalization transient keyword won't play any role , hence transient keyword not   required.

Difference b/w Serialization and Externalization
==

Serialization
==========
1. It is meant for default Serialization
2. Here every thing takes care by JVM and programmer doesn't have any control doesn't have any control.
3. Here total object will be saved always and it is not possible to save part of the object.
4. Serialization is the best choice if we want to save total object to the file.
5. relatively performence is low.
6. Serializable interface doesn't contain any method
7. It is a marker interface.
8. Serializable class not required to contains public no-arg constructor.
9. transient keyword play role in serialization

Externalization

============
1. It is meant for Customized Serialization
2. Here every thing takes care by programmer and JVM does not have any control.
3. Here based on our requirement we can save either total object or part of the object.
4. Externalization is the best choice if we want to save part of the object.
5. relatively performence is high
6. Externalizable interface contains 2 methods :
    1. writeExternal()
    2. readExternal()
7. It is not a marker interface.
8. Externalizable class should compulsory contains public no-arg constructor otherwise we will get
   RuntimeException saying "InvalidClassException"
9. transient keyword don't play any role in Externalization.

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 unable 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.

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.

Usage of StringTokenzier
====================
=> It is a part of java.util package
=> It is used to split the entire string into mulitple tokens based on the delimiter we supply

```
            eg: String  data= "sachin ramesh tendulkar";
                  StringTokenzier stk = new StringTokenizer(data);
                  StringTokenzier stk = new StringTokenizer(data," ");
```

=> public boolean hasMoreTokens()
=> public String nextToken()

eg#1.
```
import java.util.*;
class TestApp {
      public static void main(String[] args) {
            StringTokenizer stk = new
StringTokenizer("sachin$ramesh$tendulkar","$");
            System.out.println(stk);
            int tokenCount = stk.countTokens();
            System.out.println(tokenCount);
            while (stk.hasMoreTokens())
            {
                  String data = stk.nextToken();
                  System.out.println(data);
            }
      }
}
```