DESERIALIZATION SOLUTION OF SINGLETON

Override **readResolve()** method either returning already available object or throwing exception in out singleton class. This method is internally called by **readObject()** while performing deserialization…

Note:- "readResolve() is not declared in any interface or class of java api..like main() method but it will be internally called by OIS.readObject() on the invoking object of Deserialization its signature is public Object readResolve()".

Ex:

```
1.    package com.nit.printer;
2.    import com.nit.cloneble.UtilsClass;
3.    public class Printer extends UtilsClass {
4.
5.          private static Printer instence;
6.          private static long serialVersionUID=561;
7.          //private constructor
8.          private Printer() {
9.                System.out.println("0 param constructor ");
10.         }
11.         //static factory method
12.         public static Printer getInstence() {
13.               if(instence==null) {
14.                     instence=new Printer();
15.               }
16.               return instence;
17.         }
18.         public void printData(String data) {
19.               System.out.println(data);
20.         }
21.         //solving cloning problem
22.         //.by throwing exception (this is the best practice )
23.
24.         /*@Override
25.         public Object clone() throws CloneNotSupportedException {
26.               throw new CloneNotSupportedException("cloning is not allowed");
27.         }*/
28.         //2.by returning existing object
29.         @Override
30.         public Object clone() throws CloneNotSupportedException {
31.               return instence;
32.         }
33.         public Object readResolve() {
34.               return instence;
35.         }
36.   }
```

DESERIALIZATION SOLUTION OF SINGLETON

"the best practice is **throw an exception not return existing object**

Ex:-

```
1.  package com.nit.printer;
2.  import com.nit.cloneble.UtilsClass;
3.  public class Printer extends UtilsClass {
4.
5.     private static Printer instence;
6.     private static long serialVersionUID=56l;
7.     //private constructor
8.     private Printer() {
9.           System.out.println("0 param constructor ");
10.    }
11.    //static factory method
12.    public static Printer getInstence() {
13.          if(instence==null) {
14.                instence=new Printer();
15.          }
16.          return instence;
17.    }
18.    public void printData(String data) {
19.          System.out.println(data);
20.    }
21.
22.    //solving cloning problem
23.    //.by throwing exception (this is the best practice )
24.
25.    /*@Override
26.    public Object clone() throws CloneNotSupportedException {
27.          throw new CloneNotSupportedException("cloning is not allowed");
28.    }*/
29.    //2.by returning existing object
30.    @Override
31.    public Object clone() throws CloneNotSupportedException {
32.          return instence;
33.    }
34.    /*public Object readResolve() {
35.          return instence;
36.    }*/
37.
38.    public Object readResolve() {
39.          System.out.println("Printer.readResolve()");
40.          throw new IllegalArgumentException("printer do not want to support
     deserialization");
41.    }
42. }
```

DESERIALIZATION SOLUTION OF SINGLETON

```
1.  package com.nit.deserilizable;
2.  import java.io.FileInputStream;
3.  import java.io.FileNotFoundException;
4.  import java.io.FileOutputStream;
5.  import java.io.ObjectInputStream;
6.  import java.io.ObjectOutputStream;
7.  import java.io.Serializable;
8.
9.  public class DeSerializationTest implements Serializable {
10.         public static void serilize(Object obj) {
11.                 try {
12.         ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream("abc.txt"));
13.                         oos.writeObject(obj);
14.                         oos.flush();
15.                         oos.close();
16.                         System.out.println("object serialized ");
17.                 } catch (FileNotFoundException e) {
18.                         e.printStackTrace();
19.                 }
20.                 catch (Exception e) {
21.                         e.printStackTrace();
22.                 }
23.         }
24.         public static Object desrialize() {
25.                 Object obj1 = null;
26.                 try {
27.         ObjectInputStream ios=new ObjectInputStream(new FileInputStream("abc.txt"));
28.                         obj1=  ios.readObject();
29.                         System.out.println("Deserialised object");
30.                         ios.close();
31.                 } catch (Exception e) {
32.                         // TODO: handle exception
33.                 }
34.                 return obj1;
35.         }
36.}
```

DESERIALIZATION SOLUTION OF SINGLETON

```
1.  package com.nit.deserilizable;
2.
3.  import com.nit.printer.Printer;
4.
5.  public class DeserializeTest {
6.
7.     public static void main(String[] args) {
8.          Printer p1=null, p2=null;
9.          p1=Printer.getInstence();
10.         DeSerializationTest.serilize(p1);
11.         p2=(Printer) DeSerializationTest.desrialize();
12.         System.out.println(p1.hashCode()+" "+p2.hashCode());
13.    }
14. }
```

Output:-

```
0 param constructor
object serialized
Deserialised object
1028566121 1028566121
```

Or

```
0 param constructor
object serialized
Printer.readResolve()
Exception in thread "main" java.lang.NullPointerException
    at
com.nit.deserilizable.DeserializeTest.main(DeserializeTest.ja
va:12)
```

Note:- “as part network/file related Deserlization process the readResolve() method availability will be verified in the serilizable class, if not available new object will be created otherwise same/old object will be returned (or) exception will be thrown”

Note:- “as of now, there is not provision to stop Serialization of serializable objects but using readResolve() method we stop Deserilization.”