Telegram link :: https://web.telegram.org/k/#-4132683294

Serialization
=> Converting the java object to file/network supported format
=> To implement Serialziation, we use "Serializable".
=> If the object doesnot support Serialization it would result in "NotSerializableException".
=> Streams used for Serialization

```
          new ObjectOutputStream(new
FileOutputStream(".ser")).writeObject(object)
```

DeSerialization
=> Converting the file/network supported object format to java supported Object format
=> To implement DeSerialziation also the class should implements "Serializable".
=> If the object doesnot support De-Serialization it would result in "NotSerializableException".
=> Streams used for De-Serialization

```
          new ObjectInpuStream(new FileInputStream(".ser")).readObject()
```

Usage of tranisent keyword in Serialziation and DeSerialization
a. transient => variable value won't participate in serialization rather default value will be stored in serialized file.

Note:
Combination of final, static vs transient keyword
a. final transient => final means variable won't participate value will participate so transient on final variables
has no impact.
b. static transient => static means class variables, so during serialization only instance variable will participate so
transient on static variables has no impact.

We can serialize any no of objects,but during DeSerialization order of Serialized objects is important otherwise it would result
in "ClassCastException".

=> Object graph in serialization
a. Whenever we serialize any object, the set of all the objects which are reachable from that object will be serialized
automatically. this group is nothing but object graph in serialization.
b. In object graph every objet must be serialized otherwise it would result in "NotSerializableException".

=> Customized Serialization
By default when we do serialization all the instance variable will be serialized, if the instance variable is marked with
trasient keyword then variable value won't be serialized, As a result of which the there would be loss of data at the time
of DeSerialization, to resolve this problem we need to go for "Customized Serialization".

-> method used for Customized Serialization and DeSerialization

```
      private void writeObject(ObjectOutputStream oos) throw Exception{

            //Default serialization
            oos.defaultWriteObject();

            //customization
            oos.writeObject(),oosWriteInt(),oosWriteFloat(),........
      }

      private void readObject(ObjectInputStream ois) throw Exception{

            //Default serialization
            ois.defaultReadObject();

            //customization
            ois.readObject(),ois.readInt(),ois.readFloat(),........
      }
```

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.

eg#1.

```
import java.io.*;

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

public class Test
{
      public static void main(String... args) throws Exception{

            Dog d1 = new Dog();

            new ObjectOutputStream(
                  new FileOutputStream("Dog.ser")).writeObject(d1);

            Dog d2 = (Dog)new ObjectInputStream(
                  new FileInputStream("Dog.ser")).readObject();
            System.out.println("Dog object :: "+d2.i+"......."+d2.j);
      }
}
```

Output

```
Dog object :: Dog object :: 10........20
```

Does Object class implements Serializable?
Ans. No, Because every Object we will not be sent to the network or to the file for storage purpose,so Object class doesn't implement

     Serializable interface.

Case 2:
1. Even though parent class does not implements Serializable 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 Serializable 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".

5. If non-serializable parent is abstract class then just instance control flow will be performed and share it's instance variable
   to the current object.

eg#1.

```
import java.io.*;

abstract class Animal
{
      int i = 10;

      Animal(){
            System.out.println("Animal constructor called");
      }
}
class Dog extends Animal implements Serializable
{
      int j = 20;

      Dog(){
            System.out.println("Dog constructor called");
      }
}

public class Test
{
      public static void main(String... args) throws Exception{

            Dog d1 = new Dog();
            d1.i = 888;
            d1.j = 999;
            System.out.println("Serialization started....");
            new ObjectOutputStream(
                  new FileOutputStream("Dog.ser")).writeObject(d1);
            System.out.println("Serialization ended....");

            System.in.read();


            Dog d2 = (Dog)new ObjectInputStream(
                  new FileInputStream("Dog.ser")).readObject();
            System.out.println("Dog object :: "+d2.i + "........"+d2.j);
      }
}
```

```
Output
Dog constructor called
Animal constructor called
Serialization started....
Serialization ended....

Animal constructor called
Dog object :: 10........999
```

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 automaticcay 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.

eg#1.

```
import java.io.*;

class Dog implements Externalizable
{
      //instance variable
      String s;
      int i;
      int j;

      //parameterized constructor
      Dog(String s,int i, int j){
            this.s = s;
            this.i = i;
            this.j = j;
      }

      public Dog()
      {
            //To avoid InvalidClassException during "DeSerialization"
            System.out.println("Dog constructor called...");
      }

      //Serialization
      public void writeExternal(ObjectOutput oo) throws IOException
      {
            System.out.println("Serializing the required fields of the Object");
            oo.writeObject(s);
            oo.writeInt(i);
      }

      //DeSerialization
      public void readExternal(ObjectInput in)
                        throws IOException,ClassNotFoundException
      {
            System.out.println("DeSerializing the required fields of the Object");
            s = (String)in.readObject();
            i = in.readInt();
      }

}

public class Test
{
      public static void main(String... args) throws Exception{

            Dog d1 = new Dog("bruno",10,15);
            System.out.println("Dog Object :: "+d1.s+"...."+d1.i+"...."+d1.j);
            System.out.println("Serialization started....");

            new ObjectOutputStream(
                  new FileOutputStream("Dog.ser")).writeObject(d1);
            System.out.println("Serialization ended....");


            System.in.read();


            System.out.println("DeSerialziation Started...");

            Dog d2 = (Dog)new ObjectInputStream(
                  new FileInputStream("Dog.ser")).readObject();

            System.out.println("DeSerialziation ended...");
            System.out.println("Dog Object :: "+d2.s+"...."+d2.i+"...."+d2.j);

      }
}
```

```
Output
Dog Object :: bruno....10....15
Serialization started....
Serializing the required fields of the Object
Serialization ended....

DeSerialziation Started...
Dog constructor called...
DeSerializing the required fields of the Object
DeSerialziation ended...
Dog Object :: bruno....10....0
```

CaseStudy
+++++++++
1. If the class implements Externalizable interface then only part of the object will be saved in the case output is
       public no-arg constructor
       bruno---- 10 ----- 0
2. If the class implements Serializable interface then the output is nitin --- 10 --- 20
3. 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.
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 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.

```
import java.io.*;

class Dog implements Serializable
{
      static final long serialVersionUID = 1L;
      int i =10;
      int j =20;
      int k =30;
}


import java.io.*;

class SenderApp
{
      public static void main(String[] args) throws Exception
      {
            System.out.println("*****Serialization Started*********");
            new ObjectOutputStream(
                  new FileOutputStream("Dog.ser"))
                  .writeObject(new Dog());
            System.out.println("*****Serialization Ended*********");
      }
}
```
Output
```
*****De-Serialization Started*********
Dog Object :: 10....20
*****De-Serialization Ended*********
```

++

```
import java.io.*;

class Dog implements Serializable
{
      static final long serialVersionUID = 1L;
      int i =10;
      int j =20;
      int k =30;
}

import java.io.*;
class ReceiverApp
{
      public static void main(String[] args) throws Exception
      {
            System.out.println("*****De-Serialization Started*********");
            Dog d = (Dog)new ObjectInputStream(
                  new FileInputStream("Dog.ser"))
                  .readObject();
            System.out.println("Dog Object :: "+d.i+"...."+d.j);
            System.out.println("*****De-Serialization Ended*********");
      }
}
```
++
Output
```
*****De-Serialization Started*********
Dog Object :: 10....20
*****De-Serialization Ended*********
```

KeyPoints
++++++++++
=> 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

What are the different ways to create an Object in java?
 a. using new keyword
```
      Test t =new Test();
```

 b. using clonning
```
      class Test implements Cloneable{}
      t.clone();
```

 c. using Serialization and DeSerialization

```
      Dog d1 = (Dog)new ObjectInputStream(new
FileInputStream("Dog.ser")).readObject();

 d. using FactoryDesign pattern
      Runtime r = Runtime.getInstance();

 e. using class.forName() approach
      Test t1 = (Test)Class.forName(Test).newInstance();
```

Question
++++++++
1. Difference b/w ClassNotFoundException vs NoClassDefFoundError?
2. Difference b/w instanceof vs isInstance()?

CoreJava
 a. Generics and Composition vs Aggreation
 b. Enum,Annotation
 c. Packages,Accessmodifier
 d. Creation of Userdefined jars
 e. JVM Architecture[JDK9 features]