Serialization: (1.1 v)

=> The process of saving (or) writing the state of an object to a file is called serialization but strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.

=> By using FileOutputStream and ObjectOutputStream classes we can achieve a serialization process.

|=> writeObject(Object obj)

Ex: big balloon.

De-Serialization:

=> The process of reading the state of an object from a file is called DeSerialization but strictly speaking it is the process of converting an object from file supported form (or) network supported form to java supported form.

=> By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

|=> readObject()

Transient keyword:

1. transient is the modifier applicable only for variables,but not for classes and methods.
2. While performing serialization, if we don't want to save the value of a particular variable to meet the security constant of such a type of variable , then we should declare that variable with the "transient" keyword.
3. At the time of serialization JVM ignores the original value of the transient variable and saves the default value to the file .
4. That is transient means "not to serialization".

static Vs transient :

1. static variable is not part of the object state hence they won't participate in serialization because of this declaring a static variable as transient there is no use.

Transient Vs Final:

1. final variables will be participated into serialization directly by their values.
   Hence declaring a final variable as transient there is no use.
   //the compiler assign the value to final variable

```
eg: final int x= 10;
    int y = 20;
    System.out.println(x);// compiler will replace this as System.out.println(20) becoz x is final.
    System.out.println(y);
```

Externalization : (1.1 v)

1. In default serialization everything is taken care of by JVM and the programmer doesn't have any control.
2. In serialization the total object will always be saved and it is not possible to save part of the object , which creates performance problems at certain points.
3. To overcome these problems we should go for externalization where everything is taken care of by programmers and JVM doesn't have any control.
4. The main advantage of externalization over serialization is that we can save either the total object or part of the object based on our requirement.
5. To provide Externalizable ability for any object, the corresponding class should implement an externalizable interface.
6. Externalizable interface is child interface of serializable interface.

Serialization: (1.1 v)
=> The process of saving (or) writing the state of an object to a file is called serialization but strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.
=> By using FileOutputStream and ObjectOutputStream classes we can achieve a serialization process.
|=> writeObject(Object obj)
Ex: big balloon.

De-Serialization:
=> The process of reading the state of an object from a file is called DeSerialization but strictly speaking it is the process of converting an object from file supported form (or) network supported form to java supported form.
=> By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.
|=> readObject()

Transient keyword:
1. transient is the modifier applicable only for variables,but not for classes and methods.
2. While performing serialization, if we don't want to save the value of a particular variable to meet the security constant of such a type of variable , then we should declare that variable with the "transient" keyword.
3. At the time of serialization JVM ignores the original value of the transient variable and saves the default value to the file .
4. That is transient means "not to serialization".

static Vs transient :
1. static variable is not part of the object state hence they won't participate in serialization because of this declaring a static variable as transient there is no use.

Transient Vs Final:
1. final variables will be participated into serialization directly by their values.
Hence declaring a final variable as transient there is no use.
//the compiler assign the value to final variable

```
eg: final int x= 10;
    int y = 20;
    System.out.println(x);// compiler will replace this as System.out.println(20) becoz x is final.
    System.out.println(y);
```

Externalization : (1.1 v)
1. In default serialization everything is taken care of by JVM and the programmer doesn't have any control.
2. In serialization the total object will always be saved and it is not possible to save part of the object , which creates performance problems at certain points.
3. To overcome these problems we should go for externalization where everything is taken care of by programmers and JVM doesn't have any control.
4. The main advantage of externalization over serialization is that we can save either the total object or part of the object based on our requirement.
5. To provide Externalizable ability for any object, the corresponding class should implement an 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. Within this method , we have to write code to save the required variable from the file and assign it 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 compulsorily contain a public no-arg constructor otherwise we will get RuntimeExcepion saying "InvaidClassException" .

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 the total object will always be saved and it is not possible to save part of the object.
4. Serialization is the best choice if we want to save a total object to the file.
5. relatively performance is low.
6. Serializable interface doesn't contain any method
7. It is a marker interface.
8. Serializable class not required to contain public no-arg constructor.
9. transient keyword play role in serialization

Externalization
1. It is meant for Customized Serialization
2. Here everything is taken care of by the 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 performance 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 the 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 a 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 is any incompatibility 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 performance problems.

We can solve the above problems by configuring our own serialVersionUID .
eg#.

```
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 we perform any change to Dog.class file we can deserialize the 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