Agenda
======
1. Serialization
2. Deserialization
3. transient keyword
4. static Vs transient
5. transient Vs final
6. Object graph in serialization.
7. customized serialization.
8. Serialization with respect inheritance.
9. Externalization
10. Difference between Serialization & Externalization
11. SerialVersionUID

Serialization: (1.1 v)
=> The process of saving (or) writing 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 serialization process.
|=> writeObject(Object obj)

Ex: using flipkart booking an iPhone and iPhone reaching to the user.

De-Serialization:
=> The process of reading 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()

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

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

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

            Dog d1=new Dog();

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

```
            System.out.println("Serialization ended");


            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");

            System.out.println("Dog object data");
            System.out.println(d2.i+"\t" +d2.j);

      }
}
eg#2.
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.Serializable;

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

class Cat implements Serializable{
      int i=100;
      int j=200;
}

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

            Dog d1=new Dog();
            Cat c1=new Cat();

            System.out.println("serialization started");
            FileOutputStream fos= new FileOutputStream("abc.ser");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            oos.writeObject(d1);
            oos.writeObject(c1);
            System.out.println("Serialization ended");


            System.out.println("Deserialization started");
            FileInputStream fis=new FileInputStream("abc.ser");
            ObjectInputStream ois=new ObjectInputStream(fis);
            Dog d2=(Dog) ois.readObject();
            Cat c2=(Cat) ois.readObject();
            System.out.println("Deserialization ended");

            System.out.println("Dog object data");
            System.out.println(d2.i+"\t" +d2.j);

            System.out.println("Cat object data");
            System.out.println(c2.i+"\t" +c2.j);
```

```
      }
}
```

```
Output
serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      20
Cat object data
100     200
```

Note:
1. We can perform Serialization only for Serilizable objects.
2. An object is said to be Serilizable if and only if the corresponding class implements Serializable interface.
3. Serializable interface present in java.io package and does not contain any methods. It is marker interface.
The required ability will be provided automatically by JVM.
4. We can add any no. Of objects to the file and we can read all those objects from the file but in which order we wrote objects in the
same order only the objects will come back. That is order is important.if there is a mismatch in order it would result in
"ClassCastException".
5. If we are trying to serialize a non-serializable object then we will get RuntimeException saying "NotSerializableException"

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 security constant such type of variable ,
then we should declare that variable with "transient" keyword.
3. At the time of serialization JVM ignores the original value of transient variable and save default value to the file .
4. That is transient means "not to serialize".

eg#1.

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


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

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

            Dog d1=new Dog();
```

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


            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");

            System.out.println("Dog object data");
            System.out.println(d2.i+"\t" +d2.j);

      }
}
Output
serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      0

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

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

class Dog implements Serializable{
       static transient int i=10;
       int j=20;
}

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

            Dog d1=new Dog();

            System.out.println("serialization started");
            FileOutputStream fos= new FileOutputStream("abc.ser");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            oos.writeObject(d1);

            System.out.println("Serialization ended");


            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");

            System.out.println("Dog object data");
            System.out.println(d2.i+"\t" +d2.j);
      }
}
Output
serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      20

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);

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

class Dog implements Serializable{
     int i=10;
     transient final int j=20;
}


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

            Dog d1=new Dog();

            System.out.println("serialization started");
            FileOutputStream fos= new FileOutputStream("abc.ser");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            oos.writeObject(d1);

            System.out.println("Serialization ended");


            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");

            System.out.println("Dog object data");
            System.out.println(d2.i+"\t" +d2.j);

      }
}
Output
Serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      20


Declaration output
================
1.
int i=10;
int j=20;
output:: 10   20

2.
transient int i=10;
int j=20;
output:: 0 20

3.
transient int i=10;
transient static int j=20;
output:: 0  20

4.
transient final int i=10;
transient int j=20;
output::  10  0

5.
transient final int i=10;
transient static int j=20;
output: 10 20

Note:
We can serialize any no of objects to the file but in which order we serialized in
the same order only we have to deserialize,
if we change the order then it would result in "ClassCastException".

Example :
Dog d1=new Dog( );
Cat c1=new Cat( );
Rat r1=new Rat( );

FileOutputStreamfos=new FileOutputStream("abc.ser");
ObjectOutputStreamoos=new ObjectOutputStream(fos);
oos.writeObject(d1);
```

```
oos.writeObject(c1);
oos.writeObject(r1);

FileInputStreamfis=new FileInputStream("abc.ser");
ObjectInputStreamois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
Cat c2=(Cat)ois.readObject();
Rat r2=(Rat)ois.readObject();
```

=> If we don't know the order of Serialization then we need to use the following code

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

      Object obj=ois.readObject();/runtime object can be Dog,Cat,Rat
      if(obj instanceof Dog){
            Dog d=(Dog)obj;// parent type type casting
            //perform operation related to Dog
      }
      if(obj instanceof Cat){
            Cat C=(Cat)obj;
          //perform operation related to Cat
      }
      if(obj instanceof Rat){
             Rat r=(Rat)obj;
           //perform operation related to Rat
      }
```

Object graph in serialization:

1. Whenever we are serializing an object the set of all objects which are reachable from that object will be serialized automatically.
   This group of objects is nothing but object graph in serialization.
2. In object graph every object should be Serializable otherwise we will get runtime exception saying "NotSerializableException".

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 Dog implements Serializable{
      Cat c=new Cat();
}

class Cat implements Serializable{
      Rat r=new Rat();
}

class Rat implements Serializable{
       int i=10;
}
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.c.r.i);
            System.out.println("DeSerialization ended");


   }
}
```

```
Output
======
Serialization Started
Serialization ended
**********************************
DeSerialization Started
10
DeSerialization ended
```

=> In the above example whenever we are serializing Dog object automatically Cat and Rat objects will be serialized because these are part of object graph of Dog object.
=> Among Dog, Cat, Rat if at least one object is not serializable then we will get runtime exception saying "NotSerializableException".

CustomizedSerialization
=======================
During default Serialization there may be a chance of lose of information due to transient keyword.
example: remember mango and money inside it.

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 Account implements Serializable{
      String name="sachin";
      transient String password="tendulkar";
}
public class Test {
      public static void main(String[] args)throws
IOException,ClassNotFoundException{
```

```
            Account acc=new Account();
            System.out.println(acc.name +"=====> "+ acc.password);

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

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

            System.out.println("DeSerialization Started");
            FileInputStream fis= new FileInputStream("abc.ser");
            ObjectInputStream ois=new ObjectInputStream(fis);
            acc=(Account)ois.readObject();
            System.out.println(acc.name +"=====> "+ acc.password);
            System.out.println("DeSerialization ended");
   }
}
```

=> In the above example before serialization Account object can provide proper username and password.

But after Deserialization Account object can provide only username but not password. This is due to declaring password as transient.

Hence doing default serialization there may be a chance of loss of information due to transient keyword.

=> We can recover this loss of information by using customized serialization.

We can implements customized serialization by using the following two methods.

1. private void writeObject(ObjectOutputStream os) throws Exception.

=> This method will be executed automatically by jvm at the time of serialization.

=> It is a callback method. Hence at the time of serialization if we want to perform any extra work we have to define that in this method only. (prepare encrypted password and write encrypted password seperate to the file)

2. private void readObject(ObjectInputStream is) throws Exception.

=> This method will be executed automatically by JVM at the time of Deserialization.

Hence at the time of Deserialization if we want to perform any extra activity we have to define that in this method only.

(read encrypted password , perform decryption and assign decrypted password to the current object password variable)

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 Account implements Serializable{
      String name="sachin";
      transient String password="tendulkar";

      private void writeObject(ObjectOutputStream oos)throws Exception{
            oos.defaultWriteObject();//performing default Serialization
```

```
            String epwd="123"+password;//performing encryption

            oos.writeObject(epwd);//write the encrypted data to file(abc.ser)


      }
      private void readObject(ObjectInputStream ois)throws Exception{
            ois.defaultReadObject();//performing default Serialization

            String epwd=(String)ois.readObject();//performing decryption

            password=epwd.substring(3);//writing the extra data to Object
      }
}
public class Test {
      public static void main(String[] args)throws
IOException,ClassNotFoundException{

            Account acc=new Account();
            System.out.println(acc.name +"=====> "+ acc.password);

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

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

            System.out.println("DeSerialization Started");
            FileInputStream fis= new FileInputStream("abc.ser");
            ObjectInputStream ois=new ObjectInputStream(fis);
            acc=(Account)ois.readObject();
            System.out.println(acc.name +"=====> "+ acc.password);
            System.out.println("DeSerialization ended");
   }
}
```

=> At the time of Account object serialization JVM will check is there any writeObject() method in Account class or not.
=> If it is not available then JVM is responsible to perform serialization(default serialization).
 => If Account class contains writeObject() method then JVM feels very happy and executes that   Account class writeObject() method.
      The same rule is applicable for readObject() method also.

eg#2.

```
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 Account implements Serializable{
      String name="sachin";
      transient String password="tendulkar";
```

```
	transient int pin=4444;

	private void writeObject(ObjectOutputStream oos)throws Exception{
		oos.defaultWriteObject();//performing default Serialization

		String epwd="123"+password;//performing encryption
		int epin=1234+pin;//performing encryption


		oos.writeObject(epwd);//write the encrypted data to file(abc.ser)
		oos.writeInt(epin);//write the encrypted data to file(abc.ser)
	}
	private void readObject(ObjectInputStream ois)throws Exception{
		ois.defaultReadObject();//performing default Serialization

		String epwd=(String)ois.readObject();//performing decryption
		int epin=ois.readInt();//performing decryption

		password=epwd.substring(3);//writing the extra data to Object
		pin=epin-1234;//writing the extra data to Object
	}
}
public class Test {
	public static void main(String[] args)throws
IOException,ClassNotFoundException{

		Account acc=new Account();
		System.out.println(acc.name +"=====> "+
acc.password+"======>"+acc.pin);

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

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

		System.out.println("DeSerialization Started");
		FileInputStream fis= new FileInputStream("abc.ser");
		ObjectInputStream ois=new ObjectInputStream(fis);
		acc=(Account)ois.readObject();
		System.out.println(acc.name +"=====> "+
acc.password+"========>"+acc.pin);
		System.out.println("DeSerialization ended");
   }
}
Output
sachin=====> tendulkar======>4444
Serialization Started
Serialization ended
**********************************
DeSerialization Started
sachin=====> tendulkar========>4444
DeSerialization ended

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

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 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.
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

```
      public no-arg constructor
       nitin---- 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 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.

Topics pending
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