Serialization

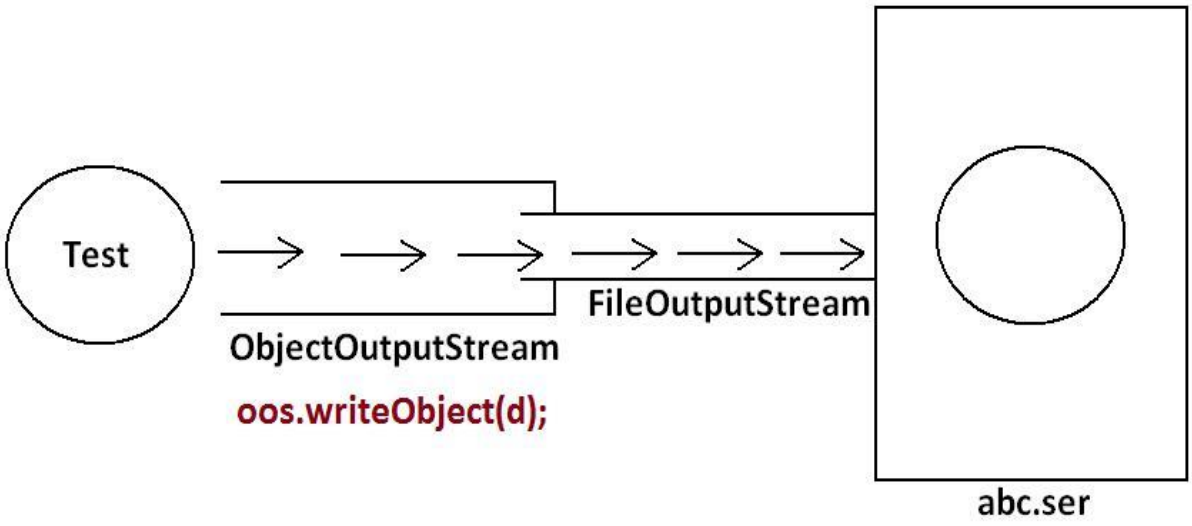
Agenda :

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)

1. **The process of saving (or) writing state of an object to a file is called serialization**
2. **but strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.**
3. **By using FileOutputStream and ObjectOutputStream classes we can achieve serialization process.**
4. **Ex: big ballon**

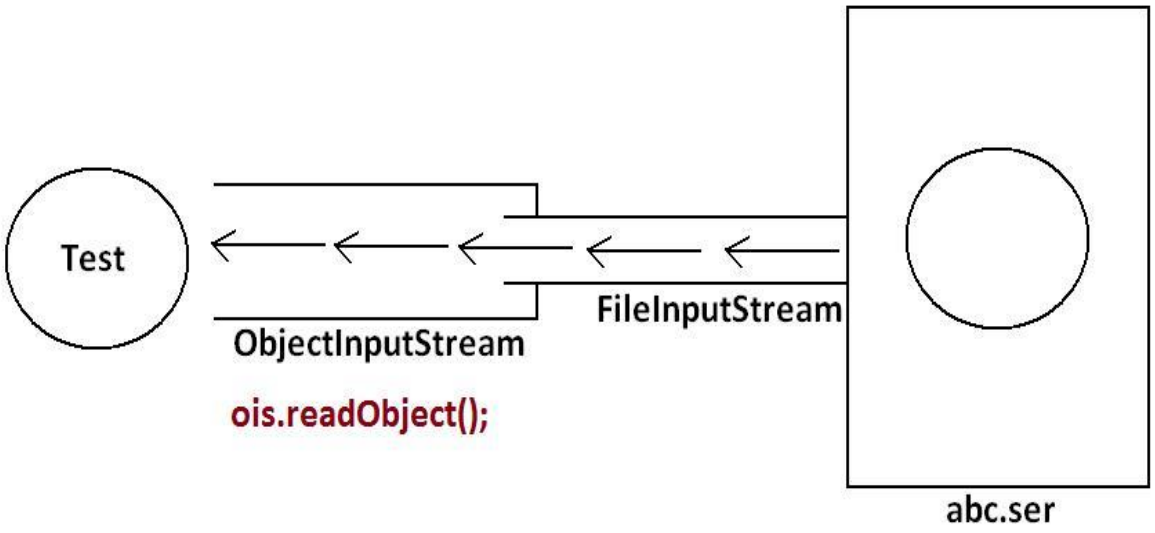
Diagram:



De-Serialization:

1. **The process of reading state of an object from a file is called DeSerialization**
2. **but strictly speaking it is the process of converting an object from file supported form (or) network supported form to java supported form.**
3. **By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.**

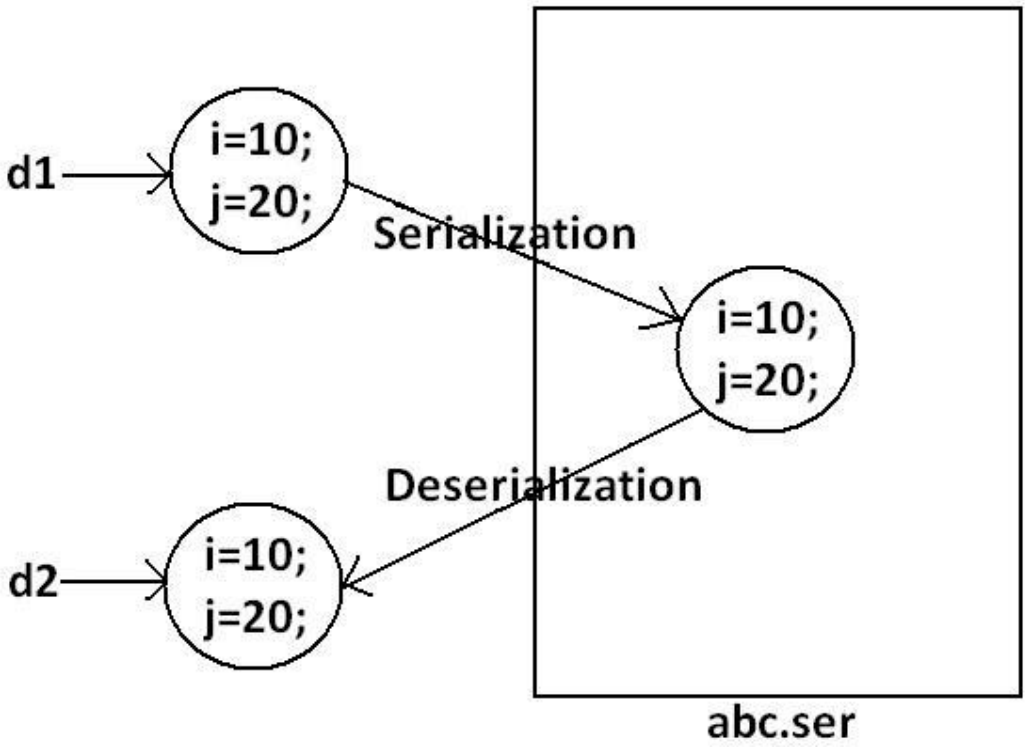
Diagram:




```
Example 1: import
java.io.*;
class Dog implements Serializable
{
int i=10;
int j=20;
}
class SerializableDemo
{
public static void main(String args[])throws
Exception{ 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(d2.i+"..............."+d2.j);

}
}
```

```
 Output: Serialization
started Serialization
ended Deserialization
started
Deserialization ended
10.............20
```

Diagram:



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.**
5. **If we are trying to serialize a non-serializable object then we will get RuntimeException saying "*NotSerializableException*".**

Example2:

```
import java.io.*;
class Dog implements Serializable
{
int i=10;
int j=20;
}
```

```
class Cat implements Serializable
{
int i=30;
int j=40;
}
class SerializableDemo
{
public static void main(String args[])throws Exception{
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(d2.i+".............."+d2.j);
System.out.println(c2.i+".............."+c2.j);
}
}
Output:
Serialization started
Serialization ended
Deserialization started
Deserialization ended
10..............20

30..............40
```

Transient keyword:

1. **transient is the modifier applicable only for variables.**
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".**

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.**

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

```
Example 3: import
java.io.*;
class Dog implements Serializable
{
static transient int i=10;
final transient int j=20;
}
class SerializableDemo
{
public static void main(String args[])throws
Exception{ Dog d1=new Dog();
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis); Dog
d2=(Dog)ois.readObject();
System.out.println(d2.i+".............."+d2.j);
}
}
Output:
10..............20
```

Diagram:

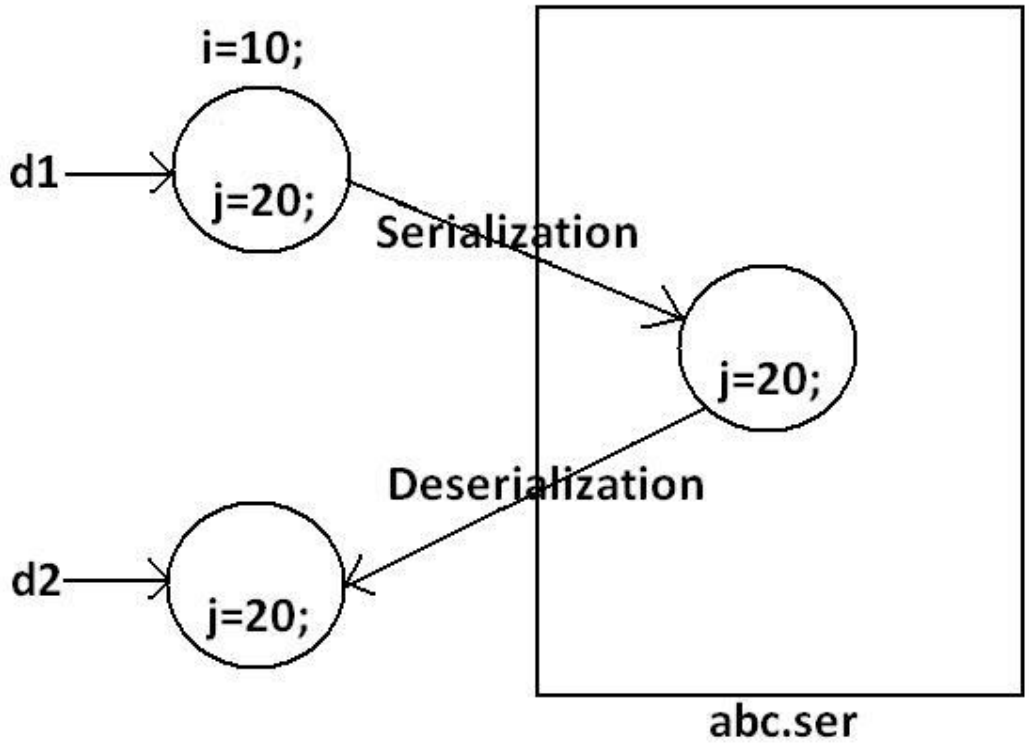


Table:

declaration	output
int i=10; int j=20;	10..... 20
transient int i=10; int j=20;	0..... 20
transient int i=10; transient static int j=20;	0..... 20
transient final int i=10; transient int j=20;	10..... 0
transient final int i=10; transient static int j=20;	10..... 20

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.

```
Example :

Dog d1=new Dog( );
Cat c1=new Cat( );
Rat r1=new Rat( );
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
oos.writeObject(c1);
oos.writeObject(r1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog  d2=(Dog)ois.readObject();
Cat c2=(Cat)ois.readObject();
Rat r2=(Rat)ois.readObject();
```

If we don't know order of objects :

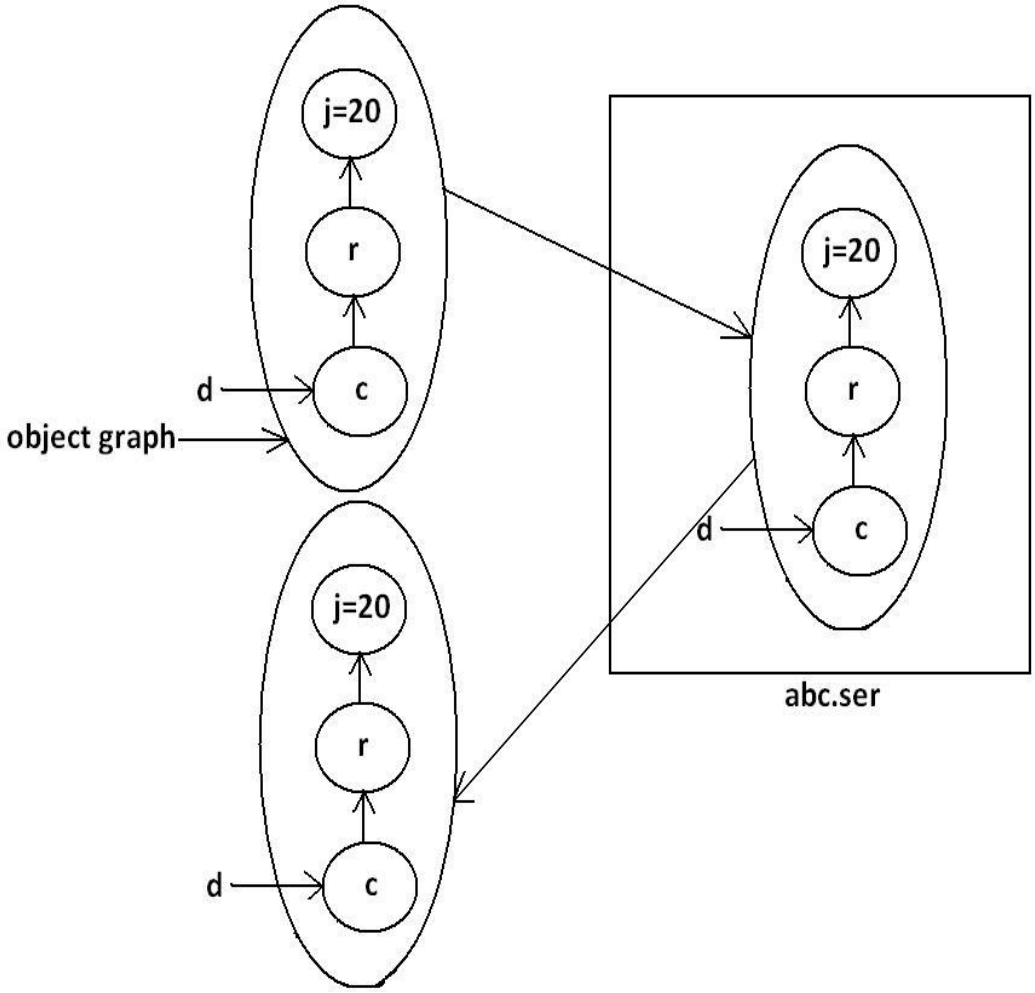
```
Example :

FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Object o=ois.readObject( );
        if(o instanceof Dog) {
          Dog d2=(Dog)o;
            //perform  Dog  specific  functionality
        }
        else if(o instanceof Cat) {
          Cat c2=(Cat)o;
            //perform Cat  specific functionality
        }
.
.
.}
```

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*".**

```
Example 4: import
java.io.*;
class Dog implements Serializable
{
Cat c=new Cat();
}
class Cat implements Serializable
{
Rat r=new Rat();
}
class Rat implements Serializable
{
int j=20;
}
class SerializableDemo
{
public static void main(String args[])throws
Exception{ Dog d1=new Dog();
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis); Dog
d2=(Dog)ois.readObject();
System.out.println(d2.c.r.j);
}
}
Output:
20
```

Diagram:



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*".

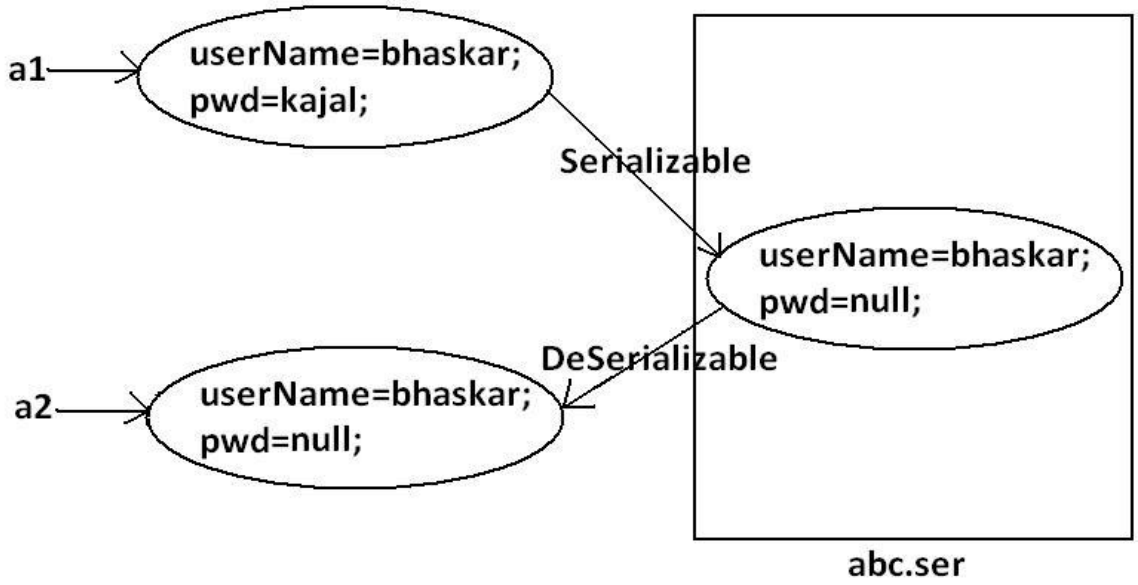
Customized serialization:

During default Serialization there may be a chance of lose of information due to transient keyword.(Ex : mango ,money , box)

```
Example 5: import
java.io.*;
```

```
class Account implements Serializable
{
String userName="Bhaskar";
transient String pwd="kajal";
}
class CustomizedSerializeDemo
{
public static void main(String[] args)throws Exception{
Account a1=new Account();

System.out.println(a1.userName+".........."+a1.pwd);

FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(a1);

FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Account a2=(Account)ois.readObject();
System.out.println(a2.userName+".........."+a2.pwd);
}

}

Output:
Bhaskar.........kajal

Bhaskar.........null
```

Diagram:



In the above example before serialization Account object can provide proper username and password. But after Deserialization Account object can provide only username bur 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.

<u>We can implements customized serialization by using the following two methods.</u>

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)

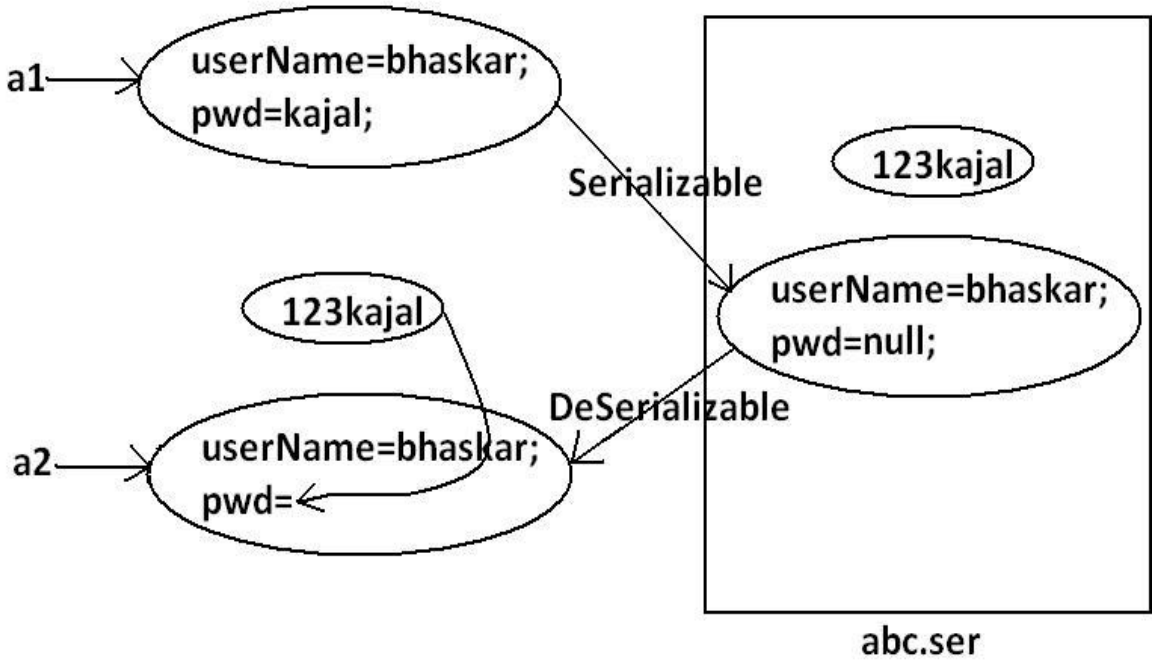
<u>Example 6:</u>

Demo program for customized serialization to recover loss of information which is happen due to transient keyword.

```
import java.io.*;
class Account implements Serializable
{
String userName="Bhaskar";
transient String pwd="kajal";
private void writeObject(ObjectOutputStream os)throws Exception
{
os.defaultWriteObject();
String epwd="123"+pwd;
os.writeObject(epwd);
}
private void readObject(ObjectInputStream is)throws
Exception{ is.defaultReadObject();
String epwd=(String)is.readObject();
pwd=epwd.substring(3);
}
}
class CustomizedSerializeDemo
{
public static void main(String[] args)throws
Exception{ Account a1=new Account();
System.out.println(a1.userName+"........."+a1.pwd);
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(a1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Account a2=(Account)ois.readObject();
System.out.println(a2.userName+"........."+a2.pwd);
}
}
```

```
Output:
Bhaskar.........kajal

Bhaskar.........kajal
```

Diagram:



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.

Serialization with respect to 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.

```
Example 7: import
java.io.*;
class Animal implements Serializable
{
int i=10;
}
class Dog extends Animal
```

```
{
int j=20;
}
class SerializableWRTInheritance
{
public static void main(String[] args)throws Exception{
Dog d1=new Dog();
System.out.println(d1.i+"........"+d1.j);
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
System.out.println(d2.i+"........"+d2.j);
}
}
Output:
10........20

10........20
```

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.***

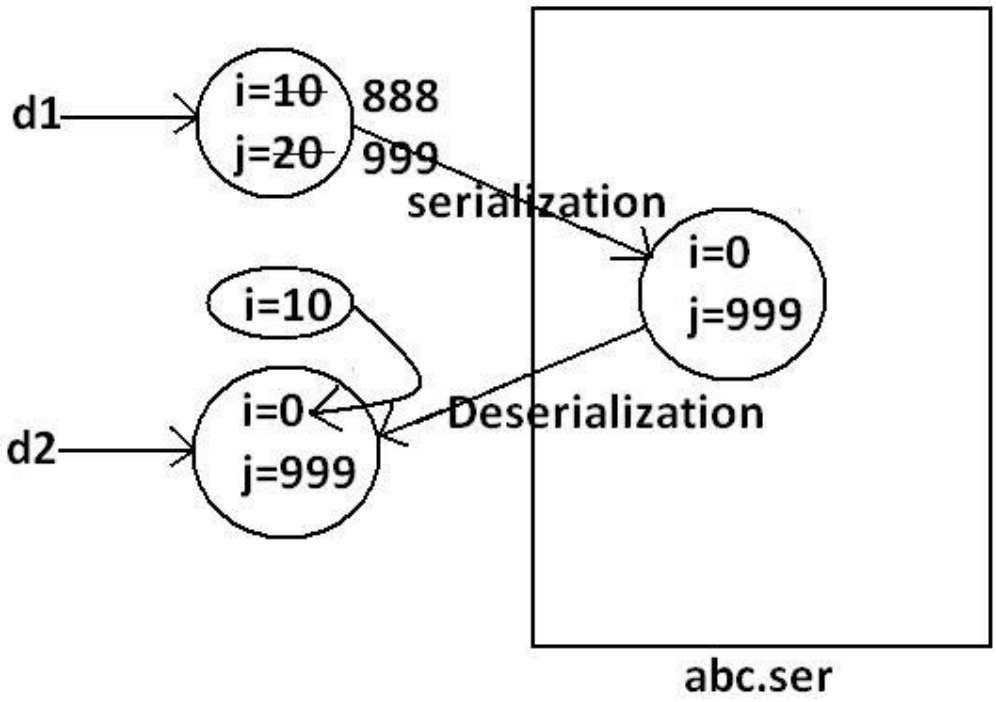
Case 2:

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 non Serializable 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.**

```
Example 8: import
java.io.*; 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");
}
}
class SerializableWRTInheritance
{
public static void main(String[] args)throws Exception{
Dog d1=new Dog();
d1.i=888;
d1.j=999;
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
System.out.println("Deserialization started");
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
System.out.println(d2.i+"........."+d2.j);
}
}
Output:
Animal constructor called
Dog constructor called
Deserialization started
Animal constructor called

10.........999
```

Diagram:



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 performence 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( )**
2. **readExternal( )**

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.

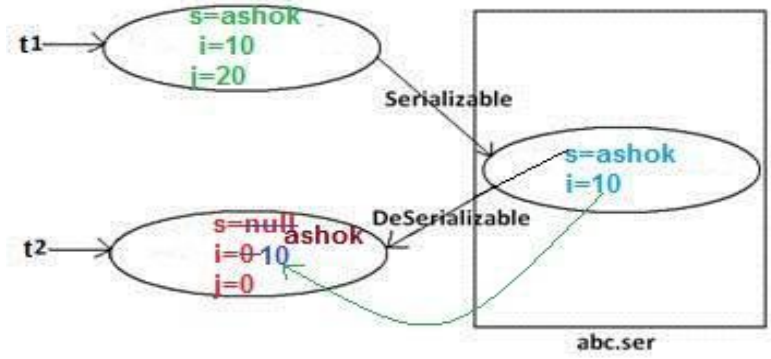
Every Externalizable class should compusory contains public no-arg constructor otherwise we will get RuntimeExcepion saying "InvaidClassException" .

Example :

```
import java.io.*;

class ExternalDemo implements Externalizable {
 String s ;
 int i ;
 int j ;
 public ExternalDemo() { System.out.println("public
  no-arg constructor");
```

```
 }
public ExternalDemo(String s , int i, int j)
        { this.s=s ;
        this.i=i ;
        this.j=j ;
}
public void writeExternal(ObjectOutput out) throws IOException
        { out.writeObject(s);
        out.writeInt(i);
}
public void readExternal(ObjectInput in) throws IOException ,
ClassNotFoundException {
        s=(String)in.readObject();
        i= in.readInt();
}
}
public class Externalizable1 {
public static void main(String[] args)throws Exception
{ ExternalDemo t1=new ExternalDemo("ashok", 10, 20);
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(t1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
ExternalDemo t2=(ExternalDemo)ois.readObject();
System.out.println(t2.s+"               -------"+t2.i+"--------"+t2.j);
}

}

output :
public --------no-argconstructor------
 ashok            10         0
```

Diagram :



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. `ashok ---- 10 ----- 0`
4. **If the class implements Serializable interface then the output is ashok --- 10 --- 20**

5. **In externalization transient keyword won't play any role , hence transient keyword not required.**

Difference between Serialization & Externalization :

Serialization	Externalization
It is meant for default Serialization	It is meant for Customized Serialization
Here every thing takes care by JVM and programmer doesn't have any control	Here every thing takes care by programmer and JVM doesn't have any control.
Here total object will be saved always and it is not possible to save part of the object.	Here based on our requirement we can save either total object or part of the object.
Serialization is the best choice if we want to save total object to the file.	Externalization is the best choice if we want to save part of the object.
relatively performence is low	relatively performence is high
Serializable interface doesn't contain any method , and it is marker interface.	Externalizable interface contains 2 methods : 1.writeExternal() 2. readExternal() It is not a marker interface.
Serializable class not required to contains public no-arg constructor.	Externalizable class should compulsory contains public no-arg constructor otherwise we will get RuntimeException saying "InvalidClassException"
transient keyword play role in serialization	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".

<u>The process in depending on default serialVersionUID are :</u>

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.**

We can solve above problems by configuring our own serialVersionUID .

<u>we can configure serialVersionUID as follows :</u>

private static final long serialVersionUID = 1L;

Example :

```
class Dog implements Serializable {
        private static final long serialVersionUID=1L;
        int i=10;
        int j=20;
}
class Sender {
        public static void main(String[] args) throws Exception
                { Dog d1=new Dog();
                FileOutputStream                                fos=new
                FileOutputStream("abc.ser"); ObjectOutputStream oos=
                new ObjectOutputStream(fos); oos.writeObject(d1);
        }
}
class Receiver {
        public static void main(String[] args)throws Exception {
                FileInputStream fis=new FileInputStream("abc.ser");
                ObjectInputStream ois=new ObjectInputStream(fis); Dog
                d2=(Dog) ois.readObject(); System.out.println(d2.i+"-
                ----"+d2.j);
        }
}
```

In the above program after serialization even though if we perform any change to Dog.class file , we can deserialize object.

We if configure our own serialVersionUID both sender and receiver not required to maintain the same JVM versions.
<u>Note :</u> some IDE's generate explicit serialVersionUID.