1) Introduction
- Serialization
- De - Serialization
- transient Key Word
- static Vs transient
- final Vs transient

2) Object Graphs in Serialization

3) Customized Serialization

4) Serialization with Respect to Inheritance

5) Externalization

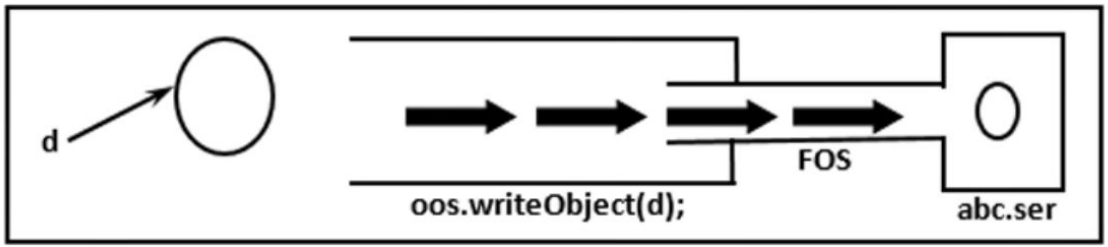
6) serialVersionUID

INTRODUCTION

SERIALIZATION:

- The Process of 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 File Supported Form OR Network Supported Form.

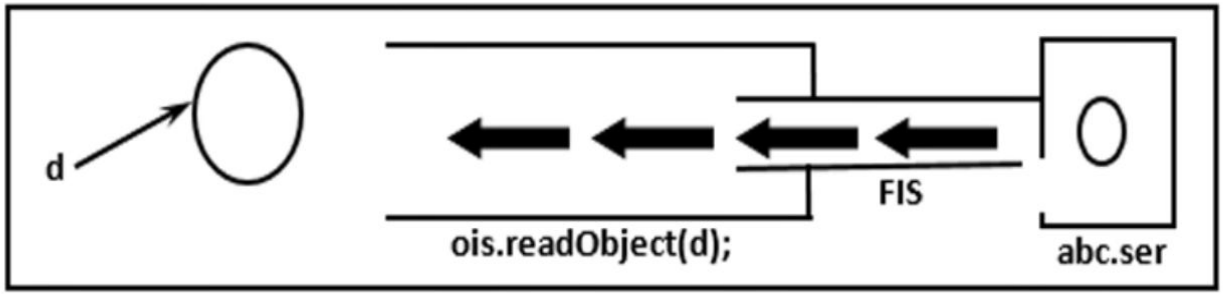
- By using FileOutputStream and ObjectOutputStream Classes we can Achieve Serialization.



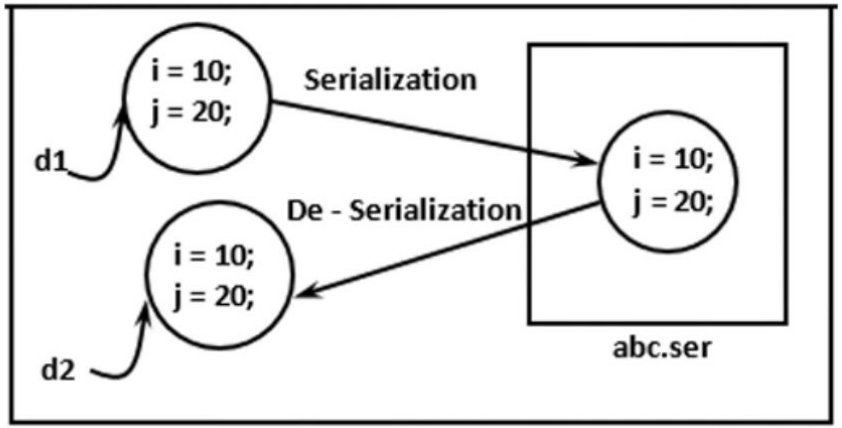
De - Serialization:

- 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 Either File OR Network Supported Form into Java Supported Form.

- By using FileInputStream and ObjectInputStream Classes we can Achieve



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

            FileInputStream fis = new FileInputStream("abc.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            Dog d2 = (Dog) ois.readObject();                              } De - Serialization

            System.out.println(d2.i+"....."+d2.j); //10.....20
        }
}
```



- We can Serialize Only Serializable Objects.
- An Object is Said to be Serializable if and Only if Corresponding Class implements Serializable Interface.
- Serializable Interface Present in java.io Package and it doesn't contain any Methods. It is a Marker Interface.
- If we are trying to Serialize Non Serializable Object then we will get Runtime Exception Saying NotSerializableException.

transient Key Word:

- transient is a Modifier Applicable Only for Variables.
- At the Time of Serialization if we don't want to Save the Value of a Particular Variable to Meet Security Constraints then we should go for transient Key Word.
- At the Time of Serialization JVM Ignores Original Value of transient Variables and Save Default Value to the File.
- Hence transient Means Not to Serialize.

static Vs transient:

- Static Variables are Not Part of Object State and Hence they won't Participate in Serialization.
- Due to this Declaring a Static Variable as transient there is No Use or impact.

final Vs transient:

- final Variables will be participated in Serialization Directly by their Values.
- Due to this declaring a final Variable as transient there is No Use or impact.

Summary:

Declaration	Output
int i = 10; int j = 20;	10....20
transient int i = 10; int j = 20;	0....20
transient static int i = 10; transient int j = 20;	10....0
transient int i = 10; transient final int j = 20;	0....20
transient static int i = 10; transient final int j = 20;	10....20

Note:

- **We can Serialize any Number of Objects to the File,But in which Order we Serialized in the Same Order Only we have to De - Serialize.That is in Serialization the order of objects is important.**

```
Dog d1 = new Dog();
Cat  c1 = new Cat();
Rat r1 = new Rat();
FOS fos = new FOS("abc.ser");
OOS oos = new OOS(fos);
oos.writeObject(d1);
oos.writeObject(c1);
oos.writeObject(r1);

FIS fis = new FIS("abc.txt");
OIS ois = new OIS(fis);
Dog d2 = (Dog)ois.readObject();
Cat c2 = (Cat)ois.readObject();
Rat r2 = (Rat)ois.readObject();
```

If we don't Know Order of Objects in Serialization:

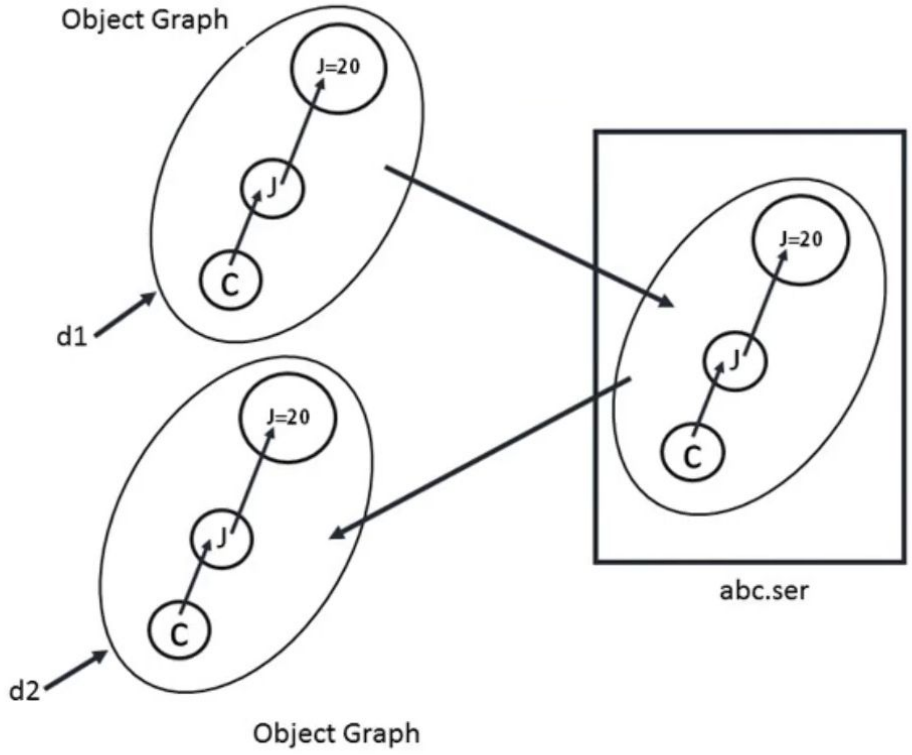
```
FIS fis = new FIS("abc.ser");
OIS ois = new OIS(fis);
Object o = ois.readObject();
if(o instanceof Dog) {
        Dog d2 = (Dog)o;
        //Perform Dog Specific Functionality
}
else if(o instanceof Cat) {
        Cat c2 = (Cat)o;
        //Perform Cat Specific Functionality
}
```

Object Graph in Serialization:

- 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 Object Graph Every Object should be Serializable. If at least One Object is Non Serializable then we will get Runtime Exception Saying NotSerializableException.

```
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 SerializeDemo {
        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); //20
        }
}
```


Object Graph
J=20
J
C
d1
J=20
J
C
d2
Object Graph
J=20
J
C
abc.ser

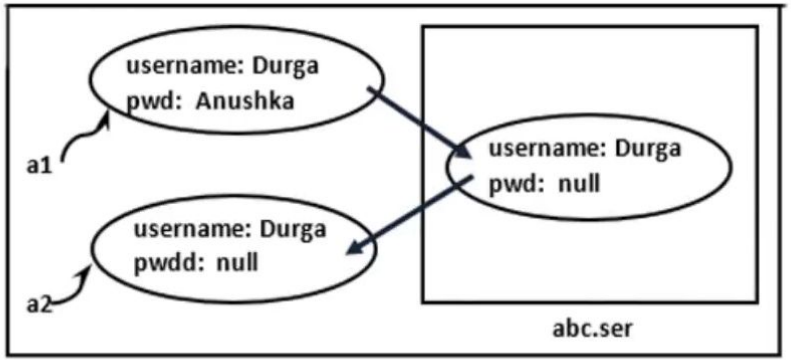
- In the Above Example whenever we are serializing Dog Object Automatically Cat and Rat Objects will be Serialized, because these are Part of Object Graph of Dog Object.

- Among Dog, Cat and Rat if at least One Object is Non Serializable then we will get Runtime Exception Saying NotSerializableException

Customized Serialization:

Need of Customized Serialization: In Default Serialization there May be a Loss of Information because of transient Key Word.

```
import java.io.*;
class  Account implements Serializable{
        String username = "Durga";
        transient String pwd = "Anushka";
}
class CustSerializeDemo{
        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);
        }
}
```



- In the Above Example before Serialization Account Object can Provide Proper User Name and Password. But after De - Serialization Account Object can Provide Only User Name but Not Password. This is Due to declaring Password Variable as transient.

- Hence during Default Serialization there May be a Chance of Loss of Information Due to transient Key Word.

- To Recover this Loss of Information we should go for Customized Serialization.

We can Implement Customized Serialization by using the following 2 Methods :

1) private void writeObject(ObjectOutputStream os) throws Exception

- This Method will be executed Automatically at the Time of Serialization.
- Hence while performing Serialization if we want do any Extra Work we have to write corresponding Code in this Method Only.

2) private void readObject(ObjectInputStream is) throws Exception

- This Method will be executed Automatically at the Time of De - Serialization.
- Hence while performing De - Serialization if we want do any Extra Work we have to Define the corresponding code in this Method Only.

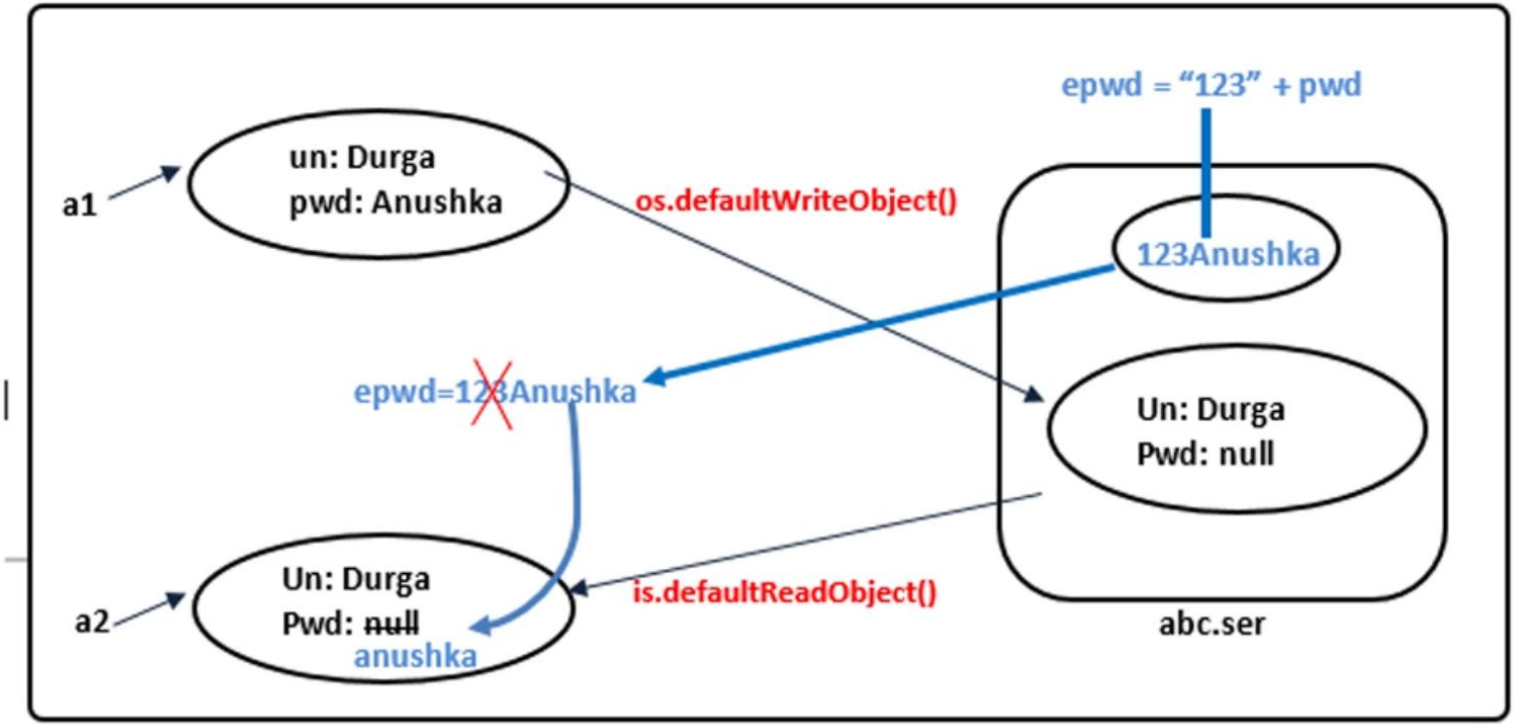
Note:

1. The above methods are callback methods because these methods will be executed automatically by JVM.
2. While performing which Object Serialization we have to do Extra Work, in the Corresponding Class we have to Define Above Methods.

Eg: For Example while performing Account Object Serialization if we required to do Extra Work, then in Account Class we have to Define Above Methods.

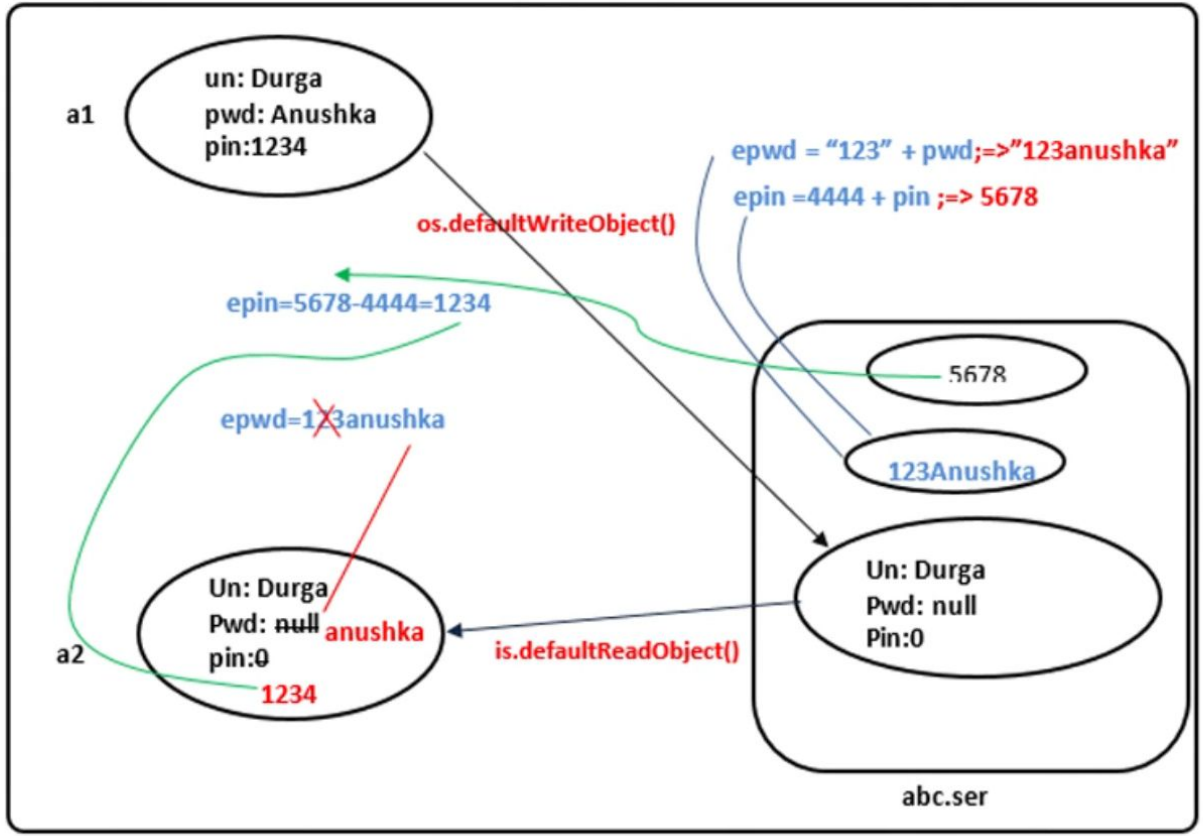
```
import java.io.*;
class  Account implements Serializable {
    String un= "Durga";
    transient String pwd = "Anushka";

    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 CustSerializeDemo {
    public static void main(String[] args) throws Exception {
        Account a1 = new Account();
        System.out.println(a1.un+"...."+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.un+"..."+a2.pwd);
    }
}
```


epwd = "123" + pwd
un: Durga
pwd: Anushka
a1
os.defaultWriteObject()
123Anushka
epwd=123Anushka
Un: Durga
Pwd: null
Un: Durga
Pwd: null
anushka
a2
is.defaultReadObject()
abc.ser

```
import java.io.*;
class Account implements Serializable
{
    String un="durga";
    transient String pwd = "anushka";
    transient int pin=1234;
    private void writeObject(ObjectOutputStream os) throws Exception
    {
        os.defaultWriteObject();
        String epwd="123"+pwd;
        int epin=4444+pin;
        os.writeObject(epwd);
        os.writeInt(epin);
    }
    private void readObject(ObjectInputStream is) throws Exception
    {
        is.defaultReadObject();
        String epwd = (String)is.readObject();
        pwd=epwd.substring(3);
        int epin=is.readInt();
        pin=epin-4444;
    }
}
```

```
class CustSerializeDemo2
{
    public static void main(String[] args) throws Exception
    {
        Account a1 = new Account();
        System.out.println(a1.un+"..."+a1.pwd+"..."+a1.pin);
        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.un+"..."+a2.pwd+"...."+a2.pin);

    }
}
```


a1
un: Durga
pwd: Anushka
pin:1234
os.defaultWriteObject()
epwd = "123" + pwd;=>"123anushka"
epin =4444 + pin ;=> 5678
epin=5678-4444=1234
5678
epwd=123anushka
123Anushka
Un: Durga
Pwd: null
Pin:0
Un: Durga
Pwd: null anushka
pin:0
a2
1234
is.defaultReadObject()
abc.ser

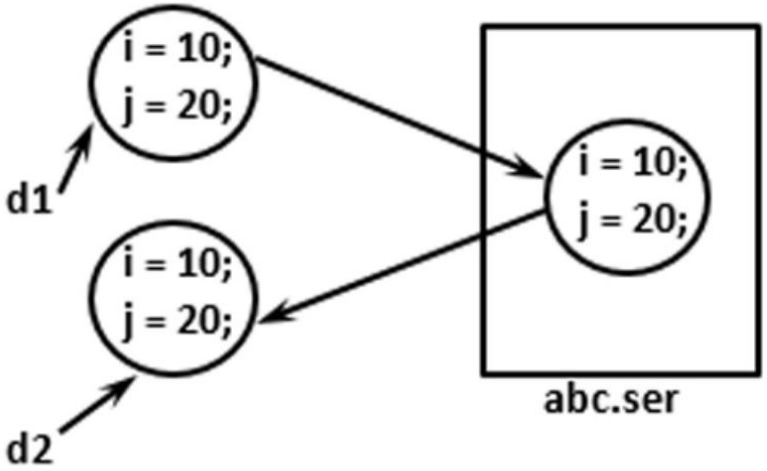
Serialization with Respect to Inheritance:

Case 1:

- If Parent is Seralizable then by Default Every Child is Serializable. That is Seralizable Nature is inheriting from Parent to Child. Hence Even though Child doesn't Implement Serializable if Parent Class Implements Serializable then we can Serialize Child Class Object.

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

```
class SerializeDemo5
{
    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 );

    }
}
```



Note: Object Class doesn't Implement Serializable Interface.

Case 2:

- Even though Parent Class does't Implement Serializable Interface we can Serialize Child Class Object if Child Class Implements Serializable Interface i.e. to Serialize Child Class Object, Parent Class Need Not be Serializable.

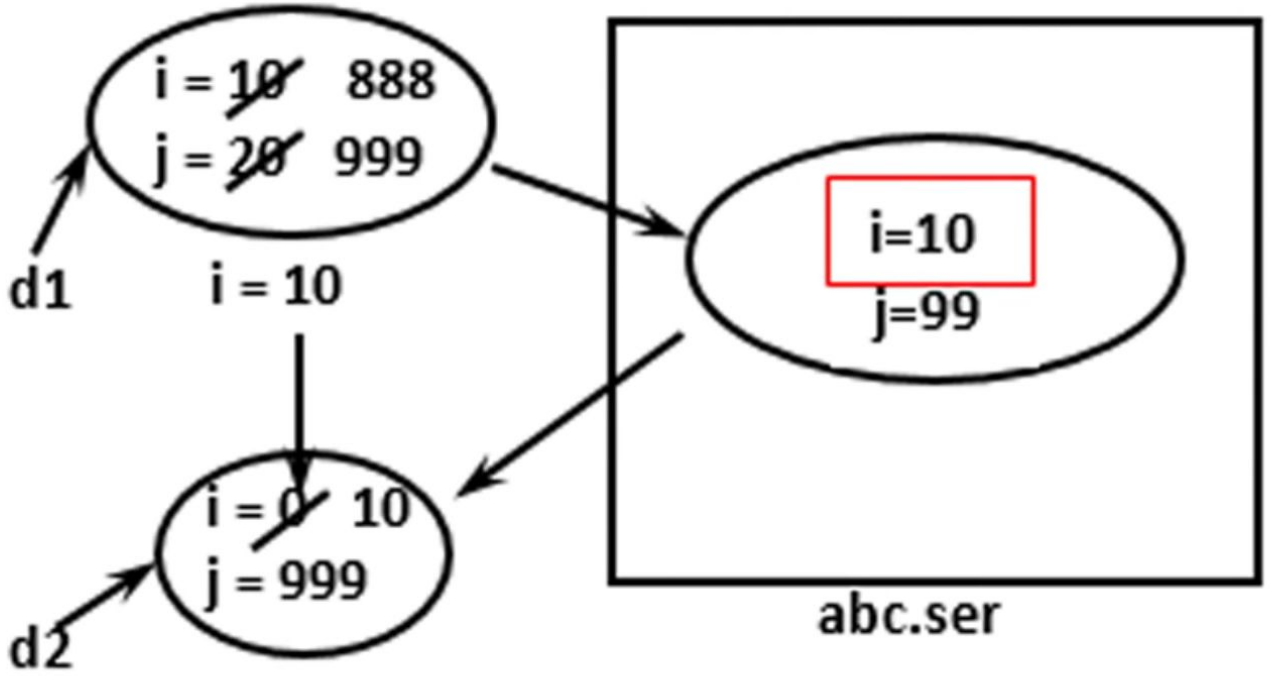
- At the Time of Serialization JVM will Check is any Instance Variable is inheriting form Non - Serializable Parent OR Not. If any Variable inheriting from Non - Serializable Parent then JVM Ignores Original Value and Save Default Value to the File.

- At the Time of DeSerializatin JVM will Check is any Parent Class is Non - Serializable OR Not. If any Parent Class is Non - Serializable then JVM will Execute Instance Control Flow in that Non - Serializable Parent and Share it Instance Variables to the Current

- To Execute Instance Control Flow execution of Non - Serializable Parent JVM will Always Invoke No Argument Constructor. Hence Every Non - Serializable Class should Compulsory contain No - Argument Constructor. Otherwise we will get Runtime Exception Saying InvalidClassExceptin

```
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 SerializeDemo  {
    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);
    }
}
```

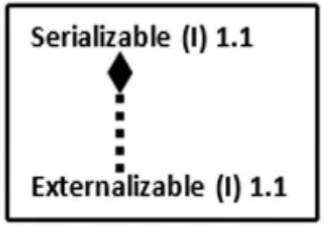
```
Animal Constructor Called
Dog Constructor Called
Deserialization  Started
Animal Constructor Called
10...999
```


i = 10 888
j = 20 999
d1
i = 10
i=10
j=99
i = 0 10
j = 999
d2
abc.ser

Externalization

- In Serialization Everything Takes Care by JVM and Programmer doesn't have any Control.In Serialization Total Object will be Saved Always to the File and it is Not Possible to Save Part of the Object which Creates Performance Problems.To Overcom these Problems we should go for Externalization where Everything Takes Care by Programmer and JVM doesn't have any Control.
- The Main Advantage of Externalization Over Serialization is based on Our Requirement we can Save Either Total Object OR Part of the Object so that Relatively Performance will be Improved.
- To Provide Externalizable Ability for any Java Object Compulsory the Corresponding Class should Implements Externalizable Interface.

- **Extenalizable Interface Present in java.io Package and it is the Child Interface of Serializable.**



Externalizable Interface contains 2 Methods.

- **writeExternal()**
- **readExternal()**

1) public void writeExternal(ObjectOutput out) throws IOException

- This Method will be executed Automatically at the Time of Serialization.
- Within this Method we have to Write Code to Save required Variables to the File.

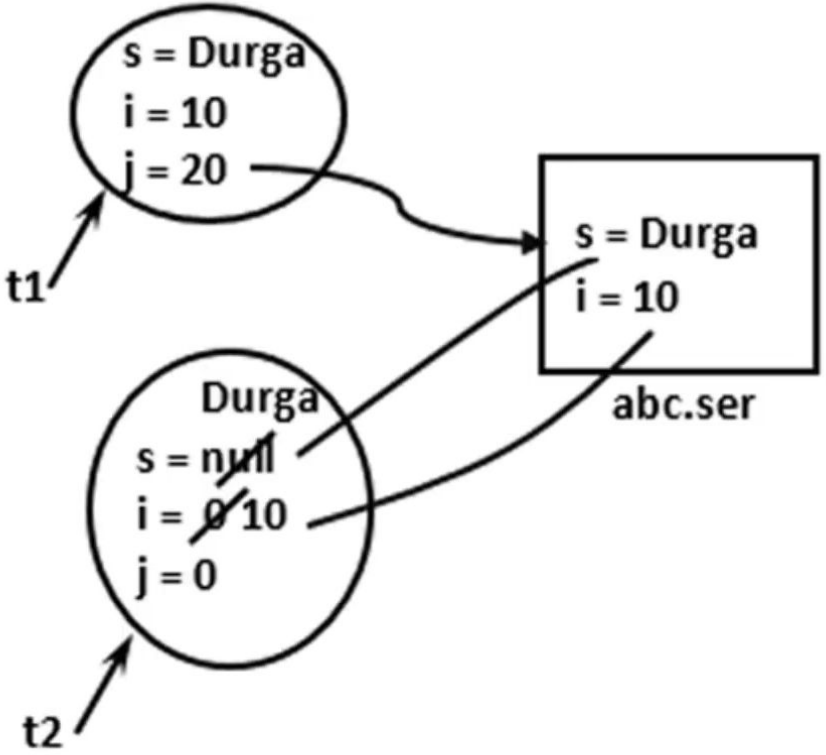
2) public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException

- This Method will be exectued Automatically at the Time of De – Serialization.
- Within this Method we have to Write Code to Read required Variables from the File and Assign to the Current Object.
- Strictly Speaking at the Time of De – Serialization JVM will Create a Separate New Object by executing public No – Argument Constructor. On that Object readExternal() method will be executed.
- Hence Every Externalizable interface implemented Class should Compulsory contain public No – Argument Constructor. Otherwise we will get Runtime Exception Saying InValidClassException.

```
import java.io.*;
class ExternalizableDemo implements Externalizable {
    String s;
    int i;
    int j;
    public ExternalizableDemo() {
        System.out.println("Public No - Arg Constructor");
    }
    ExternalizableDemo(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 static void main(String[] args)throws Exception {
        ExternalizableDemo t1 = new ExternalizableDemo("Durga", 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);
        ExternalizableDemo t2 = (ExternalizableDemo)ois.readObject();
        System.out.println(t2.s+"..."+t2.i+"...."+t2.j);
    }
}
```

```
Public No - Arg Constructor
Durga...10....0
```


s = Durga
i = 10
j = 20
t1
s = Durga
i = 10
abc.ser
Durga
s = null
i = 0 10
j = 0
t2

- If the Class Implements Serializable Interface then total object will be saved to the file and the Output is
- Durga...10...20

- If the Class Implements Externalizable Interface then part of the object will be saved to the file and Output is
- Public No – Arg Constructor
- Durga...10...20

Note:

- transient Key Word Play Role only in Serialization and it won't Play any Role in Externalization. Of Course transient Key Word is Not required in Externalization.
- If All Variables declared as transient and
- If Class Implements Serializable then the output is
- null .... 0.....0
- If Class Implements Externalizable then the output is
- Public No – Arg Constructor
- Durga .... 0.....0

Differences between Serialization and Externalization?

Serialization	Externalization
It is Meant for Default Serialization.	It is Meant for Customized Serialization.
Here Everything Takes Care by JVM and Programmer doesn't have any Control.	Here Everything Takes Care by Programmer and JVM doesn't have any Control.
In Serialization Total Object will be Saved to the File Always whether it is required OR Not.	In Externalization Based on Our Requirement we can Save Either Total Object OR Part of the Object.
Relatively Performance is Low.	Relatively Performance is High.
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 to the file.
Serializable Interface doesn't contain any Method. It is a *Marker* Interface.	Externalizable Interface contains 2 Methods, *wrteExternal*() and *readExternal()*. So it is Not Marker Interface.
Serializable implemented Class Not required to contain public No - Argument Constructor.	Externalizable implemented Class should Compulsory contain public No - Argument Constructor. Otherwise we will get Runtime Exception Saying *InvalidClassException*.
transient Key Word will Play Role in Serialization.	transient Key Word won't Play any Role in Externalization. Of Course it is Not Required.

serialVersionUID:

In Serialization both sender and receiver need not be same and need not be from the same location and need not to use same machine. Persons may be different, locations may be different and machines may be different.

At the time of Serialization JVM will save a unique id with every object. This unique id will be generated by JVM based on .class file. At the time of deserialization Receiver side JVM will compare object unique id with local .class unique id. If both are matched then only deserialization will be performed otherwise receiver unable to deserialize and we will get Runtime Exception saying InvalidClassException.

This unique identifier is nothing but serialVersionUID.

The Problems of Depending on Default serialVerisionUID generated by JVM are:

- Both Sender and Receiver should Use Same JVM wrt to Vendor and Version. If there is any Incompatability in JVM Versions, then Receiver is Unable to DeSerialize because of different serialVersionUID's. In this Case Receiver will get Runtime Exception Saying InvalidClassException.

- After Serialization if we Change .class File at Receiver Side then we can't Perform DeSerialization because of Miss Match in serialVersionUID's of Local Class of Receiver and Serialized Object. In this Case at the Time of DeSerialization we will get Runtime Exception Saying InvalidClassException.

- To Generate serialVersionUID Internally JVM will Use Complex Alogirithm which May Create Performance Problems.

- We can Solve Above Problems by configuring Our Own serialVersionUID.
- We can Configure serialVersionUID as follows

```
private static final long serialVersionUID = 1L;
```

```
import java.io.Serializable;
class Dog implements Serializable {
        private static final long serialVersionUID = 1L;
        int i = 10;
        int j = 20;
}
```

```
import java.io.*;
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);
        }
}
```

```
import java.io.*;
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 are performing any Change to the .class File we can DeSerialize Object.
- If we Configure Our Own serialVersionUID Both Sender and Receiver are Not required to Maintain Same JVM Versions.

Note:

- Some IDE's Explicitly Prompt the Programmer to Enter serialVersionUID.
- Some IDE's May Generate Explicit serialVersionUID's Automatically Instead of depending on JVM generated Default serialVersionUID.