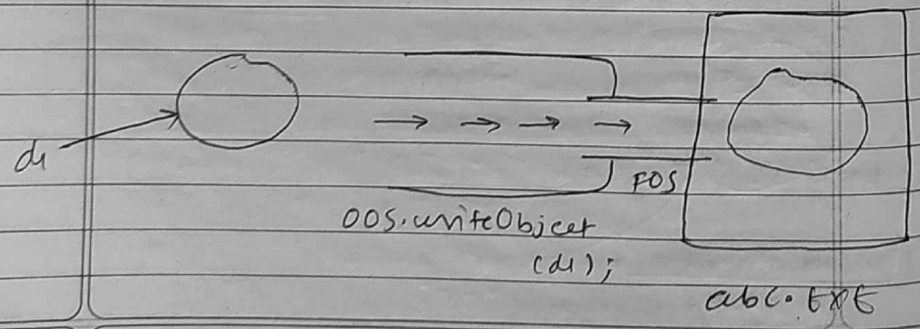
# Serialization

1. Introduction
2. Object Graph in serialization
3. Customized Serialization
4. Serialization w.r.t Inheritance
5. Externalization
6. SerialVersionUID

## Introduction

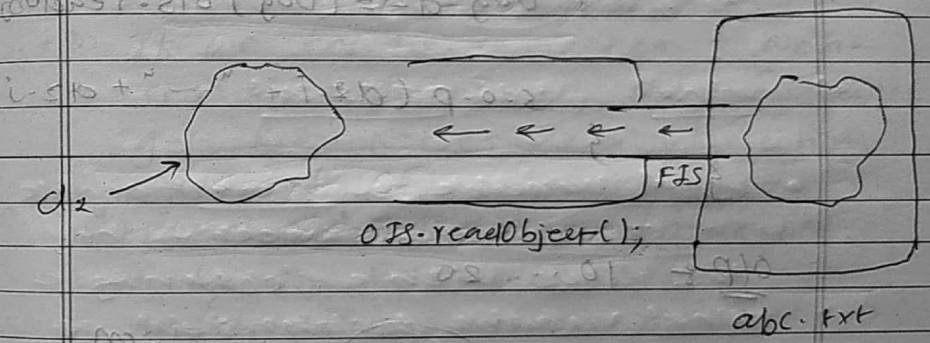
⇝ Serialization is a process of writing state of an object to a file, strictly speaking it is process of converting an object from Java supported form into either File supported form or Network supported form.

⇝ By using FileOutputStream & ObjectOutputStream classes we can acheive serialization.




⇝ Deserialization is a process of reading state of an object from the file ~~is~~, but strictly speaking it is a process of converting an object from either File supported form or Network supported form into Java supported form.

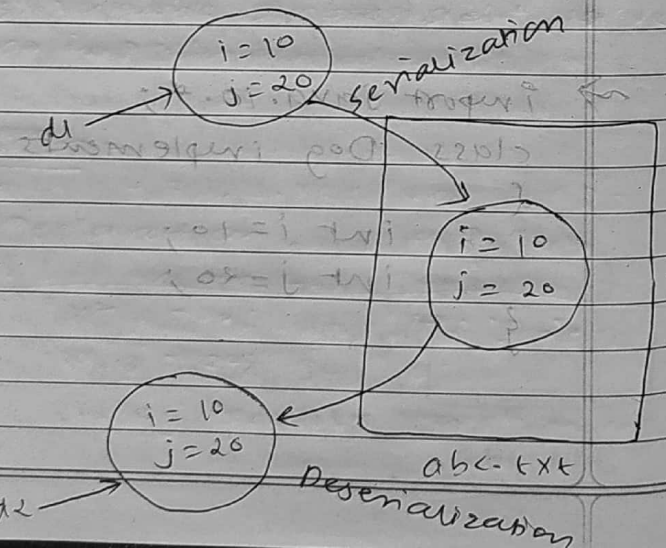
⇝ By using FileInputStream & ObjectInputStream classes we can acheive deserialization.




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

```
class SerializeDemo
{
    p s v main(Str..) throws Exception
    {
        Dog d1 = new Dog();

Serialization {
        FOS fos = new FOS("abc.txt");
        OOS oos = new OOS(fos);
        oos.writeObject(d1);

Deserialization {
        FIS fis = new FIS("abc.txt");
        OIS ois = new OIS(fis);
        Dog d2 = (Dog) ois.readObject();

        S.O.P(d2.i + "..." + d2.j);
    }
}
```

O/P : 10 ... 20




⇒ we can serialize only serializable object.

⇒ An object is said to be serializable if and only if the corresponding class implements serializable interface.

⇒ Serializable interface present in java.io package & it doesn't contain any method (It is Marker Interface)

⇒ If we try to serialize a Non-serializable object then we will get R.E. NotSerializableException.

⇒ "transient" modifier applicable only for variables but not for methods & classes.

⇒ At the time of serialization, if we don't want to save the value of particular variable to meet security constraints then we should declare that variable as "transient"

⇒ while performing serialization, JVM ignores the ~~value~~ original value of transient variable & save default value to the file.

→ Hence

transient means Not to serialize

→ transient vs static

- static variable is not part of object-state ; hence it won't participate in ~~object~~ serialization due to this declaring static variable as transient there is no use

→ final vs transient

- final variable will participate in serialization directly by the value, Hence declaring a final variable as transient there is no impact.

→ Conclusion :

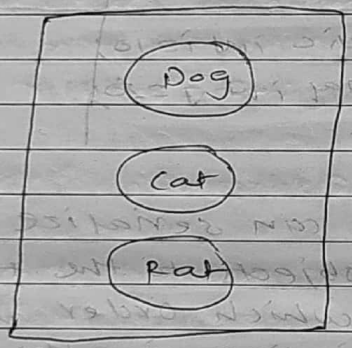
| Declaration | o/p |
|---|---|
| int i = 10;<br>int j = 20; | 10 . . . 20 |
| transient int i = 10;<br>int j = 20; | 0 . . . 20 |
| transient static<br>int i = 10;<br>transient int j = 10; | 10 . . . 0 |
| transient int i = 10;<br>transient final<br>int j = 20; | 0 . . . 20 |
| transient static int i = 10;<br>transient final int j = 20; | 10 . . . 20 |

→ Note : we can serialize any no. of objects to the file, but in which order we serialize, in the same order we have to deserialize so, order of object is important in serialization

Ex:

```
Dog d = new Dog();
Cat c = new Cat();
Rat r = new Rat();
FOS fos = new FOS("abc.ser");
OOS oos = new OOS(fos);
```

```
oos.writeObject(d);
oos.writeObject(c);
oos.writeObject(r);

FIS fis = new FIS("abc.ser");
OIS ois = new OIS(fis);

Dog d1 = (Dog) ois.readObject();
Cat c1 = (Cat) ois.readObject();
Rat r1 = (Rat) ois.readObject();
```




abc.ser

⇒ If we don't know order of object in serialization :-

```
Dog
Object o = ois.readObject();
if (o instanceof Dog)
{
    Dog d2 = (Dog) o;
    // Dog specific functionality
}
else if (o instanceof Cat)
{
    Cat c2 = (Cat) o;
    // Cat specific functionality
}
else if (o instanceof Rat)
{
    Rat r2 = (Rat) o;
    // Rat specific functionality
}
```

## Object Graphs in Serialization

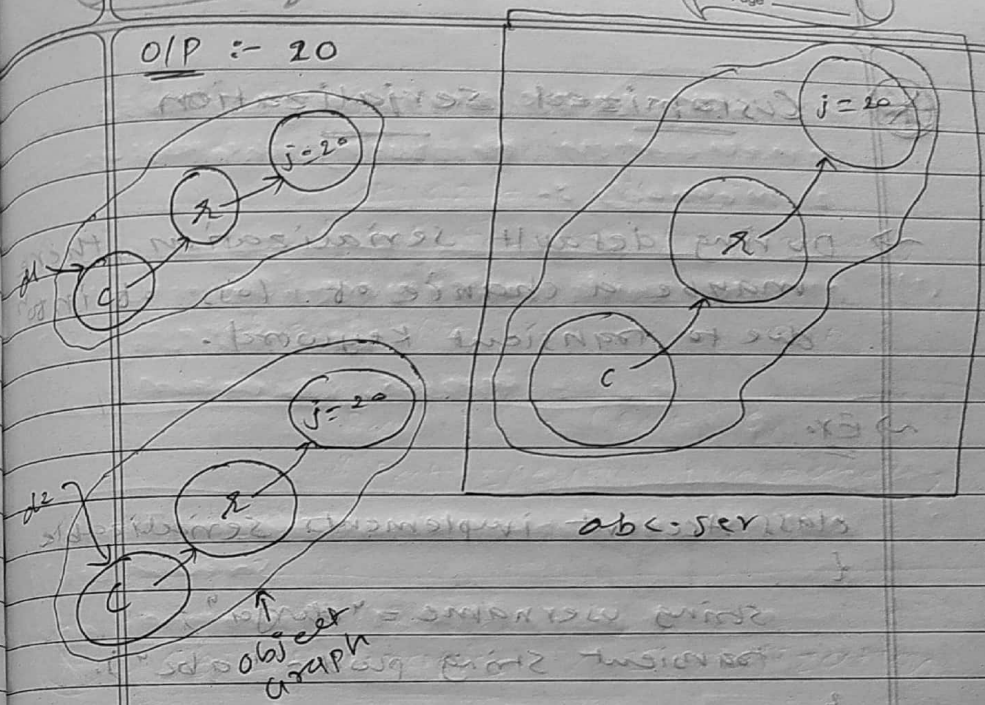
⇒ Whenever we are serializing an object, the set of all objects which are reachable from that object will be serialized automatically. This group of objects is nothing but "Object Graph"

⇒ In Object Graph, every object should be serializable, If atleast one object is not serializable then we will get R.E: NotSerializableException.

Ex-

```
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
{
    p s v main(String[] args) throws Exception
    {
        Dog d1 = new Dog();
        FOS fos = new FOS("abc.ser");
        OOS oos = new OOS(fos);
        oos.writeObject(d1);

        FIS fis = new FIS("abc.ser");
        OIS ois = new OIS(fis);
        Dog d2 = (Dog) ois.readObject();

        S.O.P(d2.c.r.j);
    }
}
```

O/P :- 20




→ In above program whenever we are serializing Dog object automatically Cat & Rat objects will be serialized because they are part of object graph of Dog.

→ Among Dog, Cat & Rat object, if at least one object is not serializable then we will get

R.E. NotSerializableException.

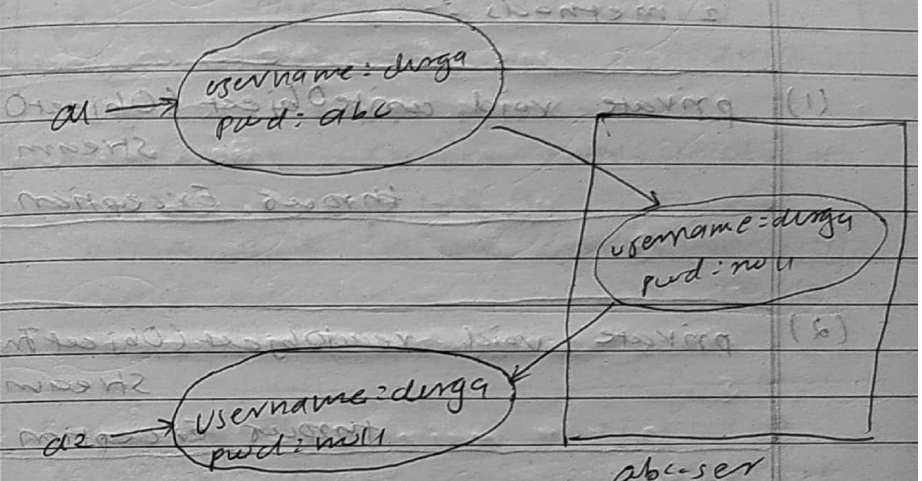
# Customized serialization

→ During default serialization, there maybe a chance of loss of info due to transient keyword.

→ Ex.

```
class Account implements Serializable
{
    String username = "durga";
    transient String pwd = "abc";
}
class CustomizedSerializationDemo
{
    p s v main(Str--) throws Exception
    {
        Account a1 = new Account();
        S.O.P(a1.username + "..." +
                a1.pwd);
        FOS fos = new FOS("abc.ser");
        OOS oos = new OOS(fos);
        oos.writeObject(a1);

        FIS fis = new FIS("abc.ser");
        OIS ois = new OIS(fis);
        Account a2 = (Account)
                ois.readObject();
        S.O.P(a2.username + "..." +
                a2.pwd);
    }
}
```

O/P : durga ... abc
durga ... null




→ In the above example, before serialization, Account object can provide proper username & password But after deserialization, Account object can provide only username but not password, this is due to declaring pwd variable as transient.

→ Hence, during default serialization there may be chance of loss of information bcz of transient keyword, To recover the loss of info we should go for customized serialization.

→ we can implement customized serialization by using the following 2 methods :-

(1) `private void writeObject(ObjectOutputStream oos) throws Exception`

(2) `private void readObject(ObjectInputStream ois) throws Exception`

→ method-(1) will be executed automatically at the time of serialization, hence, at the time of serialization if we want to perform any activity, we have to define that in this method only.

→ method-(2) will be executed automatically at the time of De-serialization hence, at the time of deserialization if we want to perform any activity we have to define that in this method only.

→ **Note :** Above methods are "call back methods" bcz these are executed automatically by JVM.

→ While performing which object serialization we have to do extra work in the corresponding class we have to define above methods.

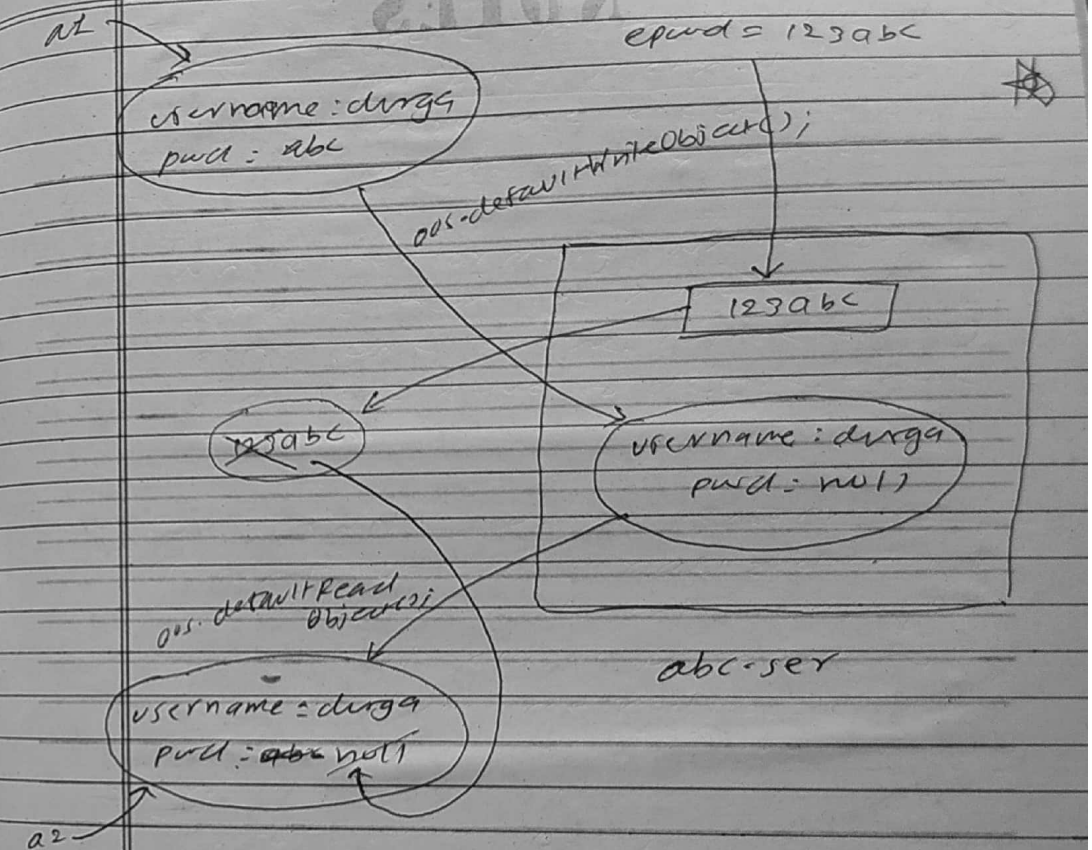
→ For example, while performing Account object serialization, if we require to do extra work, In the Account class we have to define above methods.

→ Ex:

```
import java.io.*;
class Account implements Serializable
{
    String username = "durga";
    transient String pwd = "abc";

    private void writeObject(OOS oos)
                    throws Exception
    {
        oos.defaultWriteObject();
        String epwd = "123" + pwd;
        oos.writeObject(epwd);
```

```
private void readObject(OIS ois)
        throws Exception
{
    ois.defaultReadObject();
    String epwd = (String)
                  ois.readObject();
    pwd = epwd.substring(3);
}
}
class customizedSerializationDemo
{
    p s v main(str) throws Exception
    {
        Account a1 = new Account();
        s.o.p(a1.username + "---" +
              a1.pwd);
        FOS fos = new FOS("abc.ser");
        OOS oos = new OOS(fos);
        oos.writeObject(a1);

        FIS fis = new FIS("abc.ser");
        OIS ois = new OIS(fis);
        Account a2 = (Account)
                     ois.readObject();
        s.o.p(a2.username + "---" +
              a2.pwd);
    }
}
```

O/P : durga---abc
durga---abc




→ In the above program, before serialization & after serialization Account object can provide proper username & pwd.

→ **Note:** Programmer can't call private methods from outside of class. But JVM can call private methods directly from outside of class.

→ If multiple variables are there which should be kept secret then for each variable, we will do some extra work (encryption / decryption) in the same method only.

Ex.

```
private void writeObject(OOS oos)
        throws Exception
{
    oos.defaultWriteObject();
    String epwd = "123" + pwd;
    int epin = 4444 + pin;
    oos.writeObject(epwd);
    oos.writeInt(epin);
}
```

Two secret variables

serializing/writing them individually in file.

→ we will do same process as before to deserialize them.

## ✳ Serialization w.r.t Inheritance

→ Here, 2 cases are available as follows :-

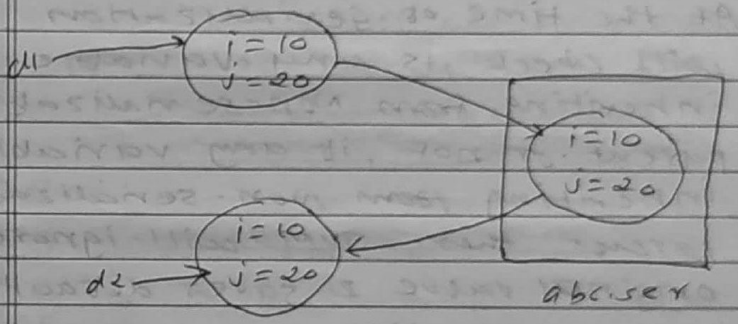
## Case-I

⇝ Even though child class doesn't implement serializable, we can serialize child class object if parent class implements serializable interface i.e serializable nature is inheriting from parent to child hence if parent is serializable then every child is serializable.

⇝ Ex-

```
class Animal implements Serializable
{
    int i=10;
}
class Dog extends Animal
{
    int j=20;
}
class SerializeDemo
{
    p s v main(Strn ) throws Exception
    {
        Dog d1 = new Dog();
        S.O.p(d1.i + "---" + d2.j);
        FOS fos = new FOS("abc.ser");
        OOS oos = new OOS(fos);
        oos.writeObject(d1);
        FIS fis = new FIS("abc.ser");
        OIS ois = new OIS(fis);
        Dog d2 = (Dog) ois.readObject();
        S.O.p(d2.i + "---" + d2.j);
    }
}
```

O/P = 10 ---20
10 ---20




⇝ In the above example even though Dog class doesn't implement serializable, we can serialize Dog object bcz it's parent class Animal implements Serializable.

⇝ Note : Object class doesn't implement serializable interface

## Case - II

→ Even though parent class doesn't implement serializable, we can serialize child class object, if child class implements serializable interface. i.e To serialize child class object, parent class need not to be serializable always.

→ At the time of serialization, JVM will check, is any variable inheriting from Non-serializable parent or not, if any variable inheriting from Non-serializable parent then JVM will ignore original value & saves default value to the file.

→ At the time of deserialization, JVM will check, is any parent class Non-serializable or not, if any parent class is Non-serializable then JVM will execute instance control flow in every Non-serializable parent & shares its instance variable to the current object.

→ While executing instance control flow of non-serializable parent, JVM will always call no-arg constructor, hence every Non-serializable class should compulsory contain no-arg constructor, it may be default constructor generated by compiler or customized constructor explicitly provided by programmer otherwise we will get R.E.
InvalidClassException.

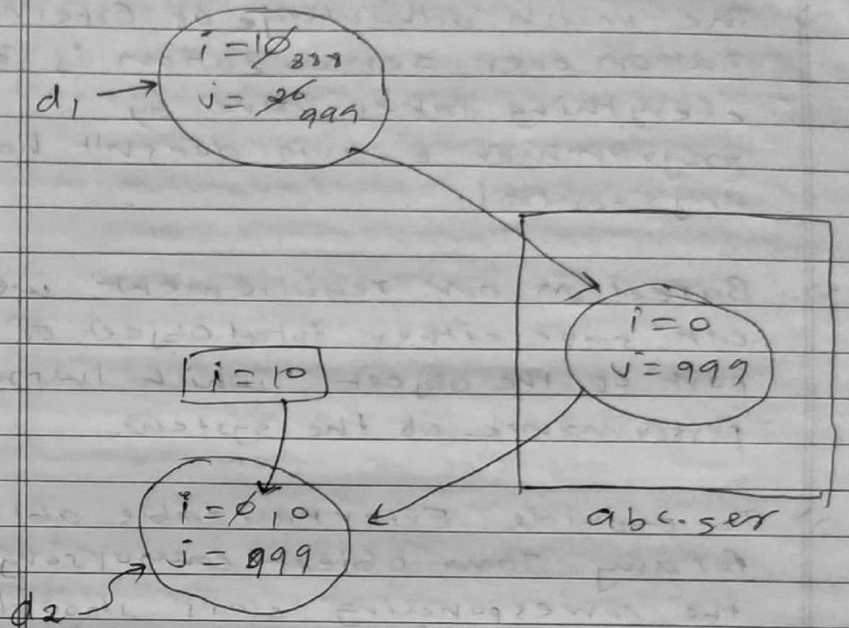
→ Ex.

```
class Animal
{
    int i=10;
    Animal()
    {
        S.O.P("Animal constructor called");
    }
}
class Dog extends Animal
        implements serializable
{
    int j=20;
    Dog()
    {
        S.O.P("Dog constructor called");
    }
}
```

```
class SerializeDemo
{
    p s v main (str[]) throws Exception
    {
        Dog d1 = new Dog();
        d1.i = 888;
        d1.j = 999;

        FOS fos = new FOS("abc.ser");
        OOS oos = new OOS(fos);
        oos.writeObject(d1);

        S.O.P("Deserialization
                        started");
        FIS fis = new FIS("abc.
                        ser");

        OIS ois = new OIS(fis);
        Dog d2 = (Dog) ois.readed
                        Object();

        S.O.P("d2.i + "---" + d2.j);
    }
}
```

O/P :- Animal constructor called
Dog constructor called
Deserialization started
Animal constructor called
10---999






## Externalization

→ In serialization everything takes care by JVM & programmer doesn't have any control.

→ In serialization it is always possible to save total object to the file but it is not possible to save part of the object, which may creates performance problem.

→ To overcome this problem, we should go for Externalization.

→ The main advantage of Externalization over serialization is everything takes care by programmer & JVM doesn't have any control.

→ Based on our requirement we can save either total object or part of the object, which improves performance of the system.

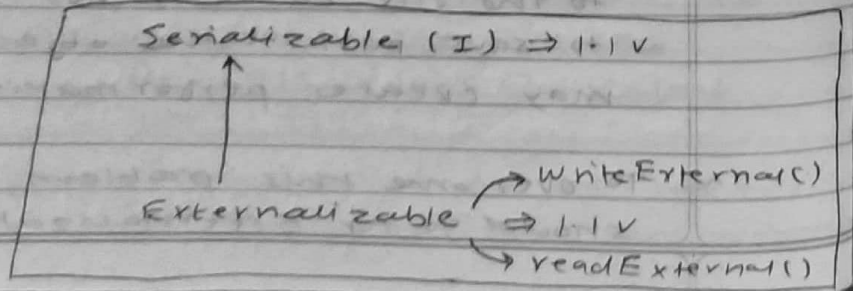
→ To provide Externalizable ability for any Java object compulsory the corresponding class should implement "Externalizable" interface.

→ Externalizable interface defines two methods:

① writeExternal()
② readExternal()

so, it is not Marker Interface.

→ Externalizable is the child interface of serializable




→ 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.

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

- This method will be executed automatically at the time of De-serialization.
- Within this method we have to write code to read required variables from file & assign to current object.
- But strictly speaking, at the of De-serialization, JVM will create a separate new object by executing public no-arg constructor, on that object JVM

will call readExternal() method.

- Hence every Externalizable implemented class should compulsory contain public no-arg constructor, otherwise we will get R.E. InvalidClassException: no valid constructor

→ Ex.

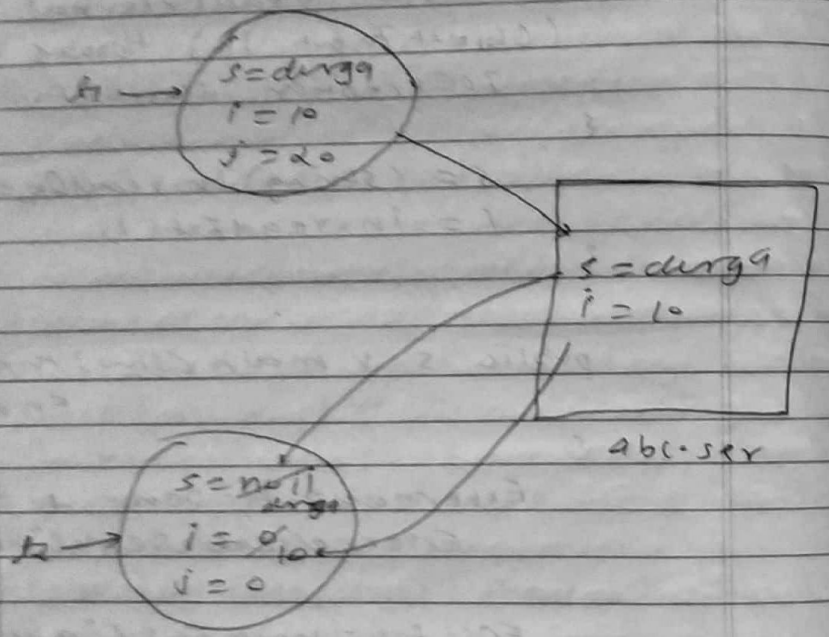
```
import java.io.*;
public class ExternalizableDemo
        implements Externalizable
{
    String s;
    int i;
    int j;

    public ExternalizableDemo()
    {
        S.O.P ("public no-arg
                constructor");
    }

    public 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 s v main (Str) throws
                        Exception
    {
        ExternalizableDemo t1 = new
            ExternalizableDemo("abc", 10, 20);

        FOS fos = new FOS ("abc.ser");
        OOS oos = new OOS (fos);
        oos.writeObject (t1);

        FIS fis = new FIS ("abc.ser");
        OIS ois = new OIS (fis);
        ExternalizableDemo te =
            (ExternalizableDemo)
                ois.readObject();
```

```
        S.O.P("t2.s+"--"+t2.i+"--"+t2.j);
    }
}
```

O/P : public no-arg constructor
durga --10--0




→ If the class implements Serializable then total object will be saved to the file, so JVM will not create new object via public-no-arg constructor at the time of deserialization & O/P will be
durga --10--20

→ Note : In serialization, "transient" keyword will play role But in Externalization "transient" keyword won't play any role ofcourse "transient" keyword not required in Externalization bcz it doesn't affect the O/P

→ Differences between serialization & Externalization :-

| | Serialization | Externalization |
|---|---|---|
| (1) | It is meant for default serialization | It is meant for customized serialization |
| (2) | Here, everything takes care by JVM & programmer doesn't have any control | Here everything takes care by programmer & JVM doesn't have control |
| (3) | It is always possible to save total object to the file but not possible to save part of the object | Always possible to save both total & part of the object to the file. |

| | | |
|---|---|---|
| 4 | Relatively performance is low | Relatively performance is high |
| 5 | Best choice if we want to save total object to the file. | Best choice if we want to save part of the object to file. |
| 6 | Serializable (I) doesn't contains any method. It is marker (I) | Externalizable (I) contains 2 methods writeExternal() & readExternal(). It is Not marker (I) |
| 7 | Serializable implemented class not require to contain public no-arg constructor. | Externalizable implemented class should compulsory contain public no-arg constructor. or we will get R.E InvalidClassEx. |
| 8 | "transient" keyword will play role in serialization | "transient" keyword won't play any role in ~~serialization~~ Externalization ofcourse it won't be required. |

## Serial Version UID

→ In serialization, Both sender & Receiver need not be same person, need not to use same machine & need not be from location, The person may be different, The machine may be different or The location may be different.

→ In Serialization, Both sender & Receiver should have .class file at the beginning only, Just state of object is travelling from sender to receiver.

→ At the time of serialization, with every object sender side JVM will save a Unique Identifier JVM is responsible to generate this Unique Identifier based on .class file

→ At the time of Deserialization, receiver side JVM will compare Unique Identifier associated with Object with local class Unique Identifier, If both are matched then only Deserialization will

be performed, otherwise we will get R.E.

InvalidClassException.

⇝ This Unique Identifier is nothing but "serialVersionUID".

⇝ Problems of depending on default serialVersionUID generated by JVM:-

(1) Both sender & receiver should use same JVM w.r.t vendor & plateform & version otherwise Receiver unable to deserialize bcz of different serial version UID.

(2) Both sender & receiver should use same .class version, After serialization, If there is any change in .class at receiver side then receiver won't be able to Deserialize.

(3) To generate serialVersionUID, internally JVM may use complex algorithms which may create performance problems.



⇝ we can solve above problems by configuring our own serialversionUID

⇝ we can configure our own serial versionUID as follows:

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

⇝ Ex- Dog1.java

```
class Dog1 implements Serializable
{
    private static final long
        serialVersionUID = 1L;
    int i = 10;
    int j = 20;
}
```

Sender.java

```
class Sender
{
    p s v main (sr-) throws Exception
    {
        Dog1 d1 = new Dog1();
        FOS fos = new FOS("abc.ser");
        OOS oos = new OOS(fos);
        oos.writeObject(d1);
    }
}
```

Receiver.java

```
class Receiver
{
    p s v main(Str[]) throws Exception
    {
        FIS fis = new FIS("abc.ser");
        OIS ois = new OIS(fis);
        Dog1 d2 = (Dog1) ois.readObject();
        S.O.p(d2.i + "---" + d2.j);
    }
}
```

⇝ In the above program after serialization if we perform any change to the .class file at receiver side, we won't get any problems at the time of Deserialization.

⇝ In this case sender & Receiver not require to maintain same JVM versions.

⇝ Note: Some IDEs prompt programmer to enter serialVersionUID explicitly

- Some IDEs may generate serialVersionUID automatically.

End of Serialization

Start of Generics