

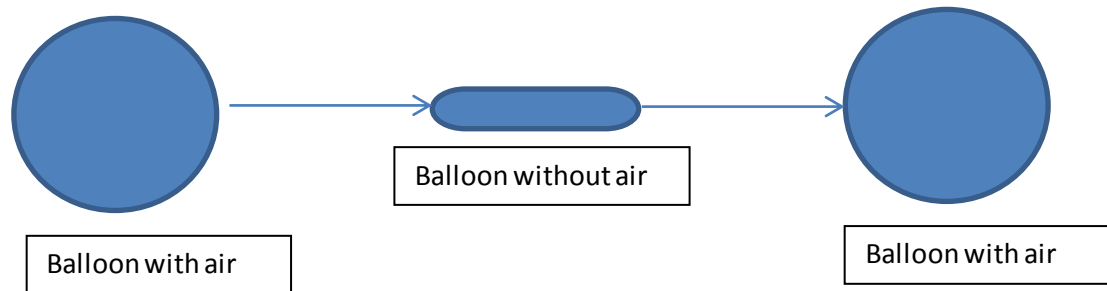
Table of Contents

1	Introduction	2
1.1	Serialization	3
1.2	Deserialization	3
1.3	Transient Keyword	5
1.4	Static vs Transient Keyword	6
1.5	Final vs Transient.....	7
2	Object graphs in serialization.....	9
3	Customized serialization.....	10
4	Serialization with respect to inheritance	13
5	Externalization	16
6	Serialization vs Externalization.....	18
7	Serial Version UID.....	19

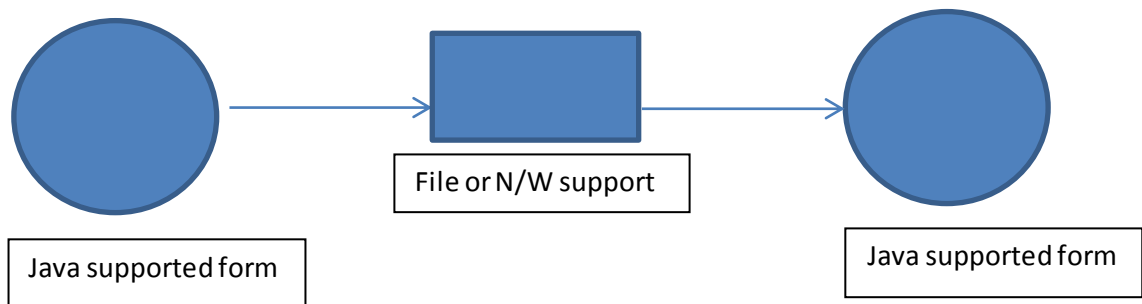
1 Introduction

What is serialization?

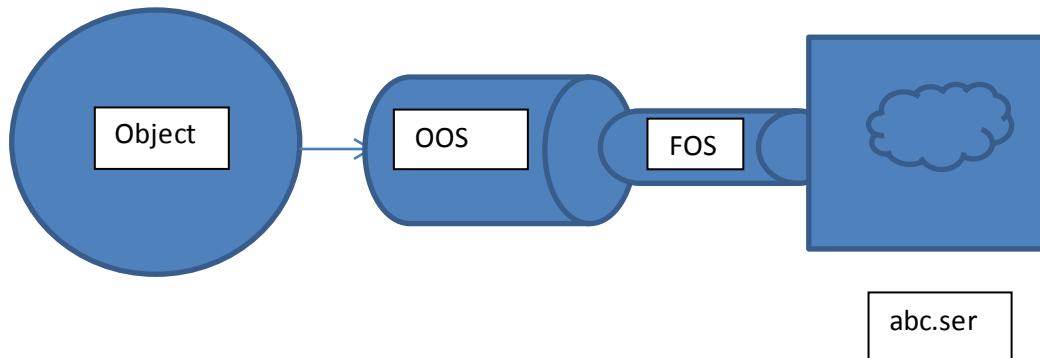
Analogy: Consider sending balloon to Bangalore from Hyderabad, To do that, In Hyderabad we need to take out the air from balloon and parcel the balloon without air. Once balloon reached to Bangalore we can refill the balloon with air.



- a) Original form into transport supported form (Serialization)
- b) Transported form into usable form (De-Serialization)



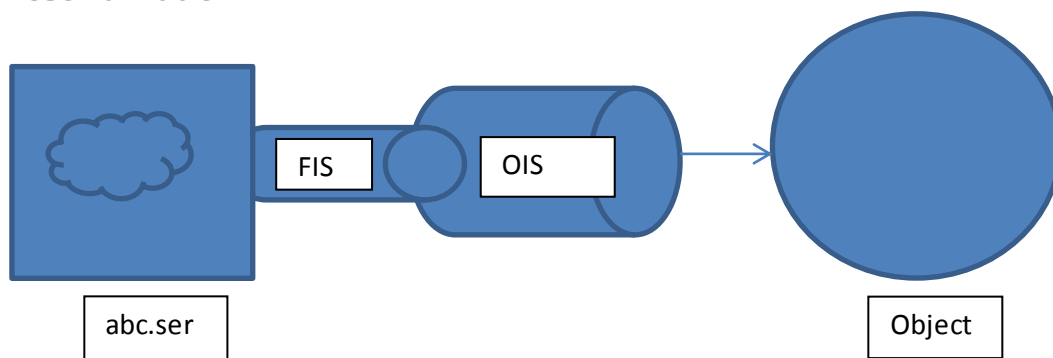
1.1 Serialization



Process of convert (writing) state of an object to a file is called serialization but strictly speaking it is process of converting an object from java supported form to either Fiile supported form or network supported form.

By using `fileOutputStream` and `ObjectOutputStream` classes we can achive serialization.

1.2 Deserialization



Process of reading the state of an object from a file is called Deserialization, But strictly speaking it is the process of converting an object from wither file or network supported form into java supported form.

By using `FileInputStream` and `Object inputstream` classes we can achieve deserialization.

Example:

```
Class Dog {
```

```
    int i=10;
```

```
    int j=20;
```

```
}
```

```
Class SerializationDemo throws Exception {
```

```
    Public static void main (String[] args) {
```

```
        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 d1 = (Dog) ois.readObject();
```

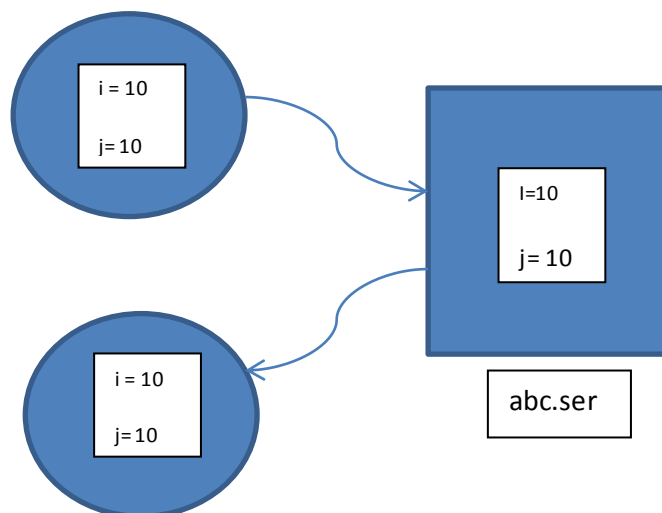
```
        System.out.print(d1.i + "----" + d2.j);
```

```
    }
```

```
}
```

Serialization

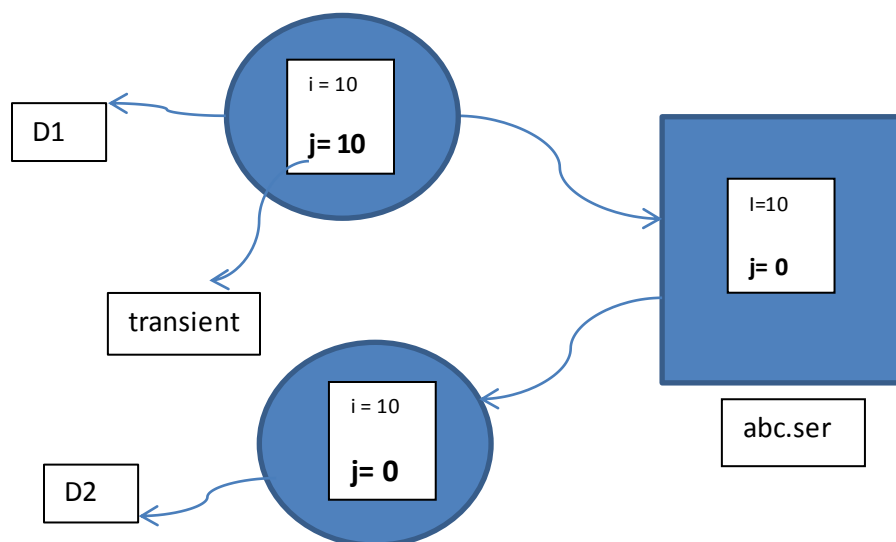
De-Serialization



- In the above case Dog is not Serializable
- If we implement Serializable interface then Dog will be Serialized.
- Without implementing, code compiles, But in runtime “NotSerializableException” will be thrown.
- Abc.ser will be created in current working directory, If it is available then it will not create that file.
- **Serializable interface available in “java.io” package**, which is a marker interface.

1.3 Transient Keyword

- Transient modifier only applicable for variables.
- In serialization, we are saving the data into file or transfer across the network.
- In some case we need to deal with sensitive data which needs protection and not recommended to transfer across the network or save it in a file.
- To protect such kind of data while doing the serialization, we can declare sensitive variable as “transient”. With transient data will not be serialized into a file or across a network.



```

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

```

1.4 Static vs Transient Keyword

Class Dog implements Serializable {

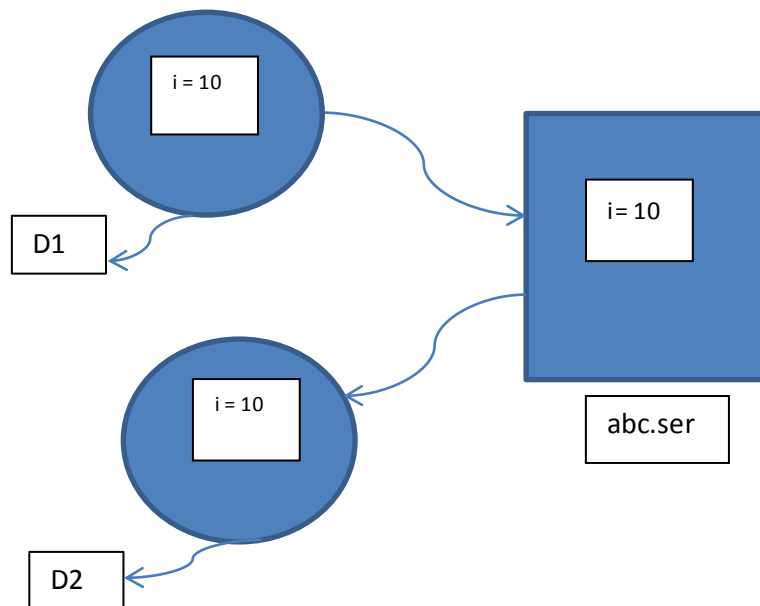
int i=10;

static int j=20;

}

- Static variable is not part of “**object state**”, hence it will not be part of in serialization.
- Static variable + Serialization is having no relation.
- Static + Serialization is having no impact.
- J=20 will be shared by the All the objects of a class and not part of a call hence it will be not participated in serialization.

J= 20 - J is Static variable = is class level variable



1.5 Final vs Transient

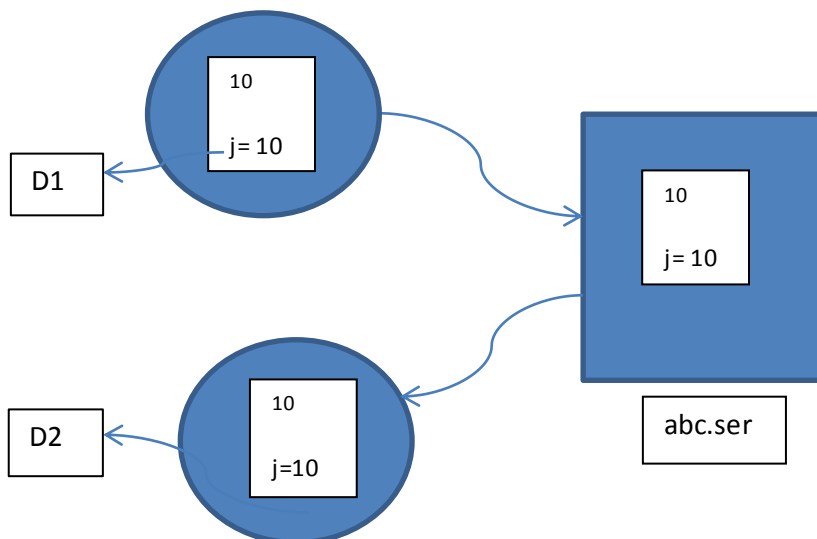
Class Dog implements Serializable {

 Final int a=10;

 int b=20;

}

- Every final variable values will be replaced with its value at compile time itself. Meaning values will be calculated at compile time and assigned to a variable
- So final key will not be having an impact along with “transient”.



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

Conclusion: Final and static variable doesn't participate/no impact in serialization.

- How many objects you can serializable
- Any number of objects.
- While de-serializing objects, the object which is serialized first will be de-serialized.
 - o Same order it will be followed
 - o So order of objects is important while serializing.

Class SerializationDemo throws Exception {

```
    Public static void main (String[] args) {
```

```
        Dog d1 = new Dog ();
```

```
        Cat c1 = new Cat ();
```

```
        Rat r1 = new Rat ();
```

```
        FileOutputStream fos = new FileOutputStream (abc.ser);
```

```
        ObjectOutputStream oos = new ObjectOutputStream (fos);
```

```
        os.writeObject(d1);
```

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

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

```
        FileInputStream fis = new FileInputStream (abc.ser);
```

```
        ObjectInputStream ois = new ObjectInputStream (fos);
```

```
        Dog d2 = (Dog) ois.readObject();
```

```
        Cat c2 = (Cat) ois.readObject();
```

```
        Rat r2 = (Rat) ois.readObject();
```

```
        Object o1 = ois.readObject();
```

```
        If (o1 instanceof Dog) {
```

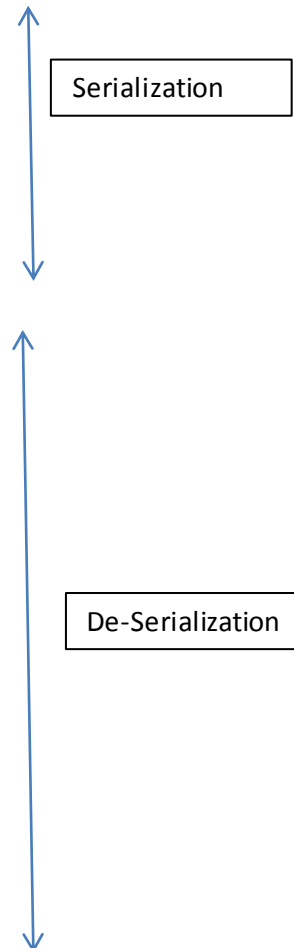
```
            Dog d2 = (Dog) ois.readObject();
```

```
        } else if (o1 instanceof Cat) {
```

```
            Cat c2 = (Cat) ois.readObject();
```

```
        }
```

```
    }}
```



2 Object graphs in serialization


- All the objects that are reachable from serialized objects will be serialized automatically. This is nothing but object graph in serialization.
- In this context all the Objects in the object graph should implement serializable interface.
- In the below Example all the Dog, Cat, Rat object should implement serializable interface.
- If **any of the object doesn't implement** "Serializable" interface then at Runtime "NotSerializableException" exception will be thrown.

```
Class Dog {  
    Cat c = new Cat();  
}
```

```
Class Cat {  
    Rat r = new Rat ();  
}
```

```
Class Rat {  
    int j = 20;  
}
```

```
Class SerializationDemo throws Exception {  
    Publicstaticvoid main (String[] args) {  
        Dog d1 = new Dog ();  
        FileOutputStream fos = new FileOutputStream (abc.ser);  
        ObjectOutputStream oos = new ObjectOutputStream (fos);  
        oos.writeObject(d1);  
    }  
}
```



Serialization

3 Customized serialization

What is the need of Customized serialization ?

In case of default serialization there is may a loss of information because of transient variables, to avoid loss of information we need to go for customized serialization

```
Class Account implements Serializable {  
    String username = "naren";  
    Transient String password = "test";  
}  
  
Class CustSerializationDemo throws Exception {  
    Public static void main (String[] args) {  
        Account a1 = new Account ();  
        FileOutputStream fos = new FileOutputStream (abc.ser);  
        ObjectOutputStream oos = new ObjectOutputStream (fos);  
        os.writeObject(a1);  
        FileInputStream fis = new FileInputStream (abc.ser);  
        ObjectInputStream ois = new ObjectInputStream (fos);  
        Account a2 = (Account) ois.readObject();  
    }  
}
```

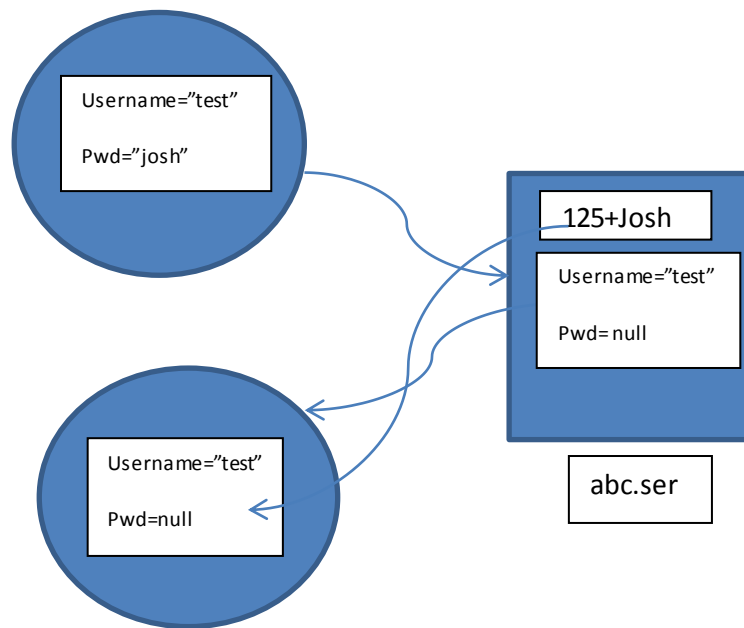
Analogy:

- Sending money to home 3 lakhs with friend from Hyderabad. Friend doesn't carry the money because of security reasons.
- Sending money 7 lakhs along with mangos
- At first instance friend didn't take the money because of security reasons.
- Second time friend doesn't know about security sensitive information (money), so he carried the money along with mangos.
- At senders and receivers programs has to do so extra work to make sure sensitive information is protected in serialization.

For Customized serialization programmer has to do some extra work to protect sensitive information which will be participated in serialization, it's kind of encryption.

Extra work:

Encrypt the password and write to abc.ser file.
Decrypt while doing the deserilization.



Callback methods called by JVM:

Below callback methods executed by jvm on serialization objects, So these methods can be used to perform customized serialization.

```
Private void writeObject(OOS oos) throws Exception;
```

```
Private void readObject(OIS ois) throws Exception;
```

```

Class Account implements Serializable {
    String username = "naren";
    Transient String password = "test";
    Private void writeObject(OOS oos) throws Exception {
        Oos.defaultWriteObject();
        String encrypt = "125"+pwd;
        Oos.writeObject();
    }
    Private void readObject(OIS ois) throws Exception {
        Ois.defaultReadObject();
        String decrypt = (String) ois.readObject();
        Pwd = decrypt.substring(3);
    }
}

```

Below things will happen when SerializableDemo class executed-

When `oos.writeObject()` methods executed, JVM calls `writeObject(OOS oos)` method in Account class automatically.

When `oos.readObject()` methods executed, JVM calls `readObject(OIS ois)` method in Account class automatically.

In this case JVM hands over serialization task to Programmer.

So, **its programmers responsibility to call default serialization mechanism performed by JVM by calling “`defaultWriteObject()`” method inside `writeObject()` method** of serialized custom Account class.

You can add as many properties as you want in Serializable class “Account”

4 Serialization with respect to inheritance

Case 1:

Parent implements serializable,

Child extends parent

Case 2

Parent doesn't implements serializable

Child extends parent implements serializable

Case 1 Explanation :

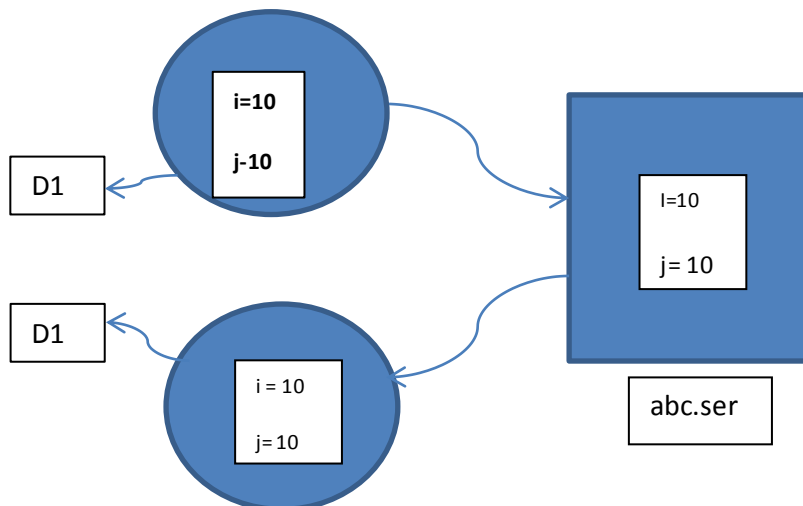
Serializing nature is inherited from parent to child, if parent implements serializable then all its child are serializable.

Example – Object is not serializable, But GenericServlet is serializable so all the servlets are serializable.

```
Class Animal implements serializable {  
    Int i = 10;  
}  
Class Dog extends Animal {  
    Int j = 20;  
}
```

In this case :

```
Dog d = new Dog ();  
Oos.writeObject(d1);  
Dog d2 = (Dog)ois.readObject();  
System.out.print(d1.i + "----" + d2.j);
```



Case 2 Explanation :

```
Class Animal {  
    Int i = 10;  
}  
  
Class Dog extends Animal implements Serilizable {  
    Int j = 20;  
}
```

In this case :

```
Dog d = new Dog ();  
d.j = 999;  
Oos.writeObject(d1);  
Dog d2 = (Dog)ois.readObject();  
System.out.print( d1.i + "----" + d2.j);
```

Conclusion: Parent class need not be Serilizable to achieve serialization on parent class variables.

While doing the deserialization: -

JVM will perform "Instance control flow"

- a) Identification of instance members
- b) Execution of instance variable assignment and instantiation.
- c) Execution of constructors.

No Argument constructor in parent class should be available otherwise "Invalid Class Exception"

```
Class Animal {  
    Int i = 10;  
    Animal () {  
        System.out.print("Animal type");  
    }  
}  
  
Class Dog extends Animal implements Serilizable {  
    Int j = 20;  
    Dog () {  
        System.out.print("Dog created");  
    }  
}
```



If not provided an Exception will be thrown

Class SerializationDemo throws Exception {

Public static void main (String[] args) {

Dog d1 = new Dog ();

d.i = 888;

d2.j = 999;

FileOutputStream fos = new FileOutputStream (abc.ser);

ObjectOutputStream oos = new ObjectOutputStream (fos);

os.writeObject(d1);

System.out.print("Deserialization started");

FileInputStream fis = new FileInputStream (abc.ser);

ObjectInputStream ois = new ObjectInputStream (fos);

Dog d2 = (Dog) ois.readObject();

System.out.print(d2.i+ "-----" + d2.j);

}

}



Serialization

De-Serialization

Output :

Animal created

Dog constructor

Desrialization started

Animal Constructor

10 ----- 999

Explanation:

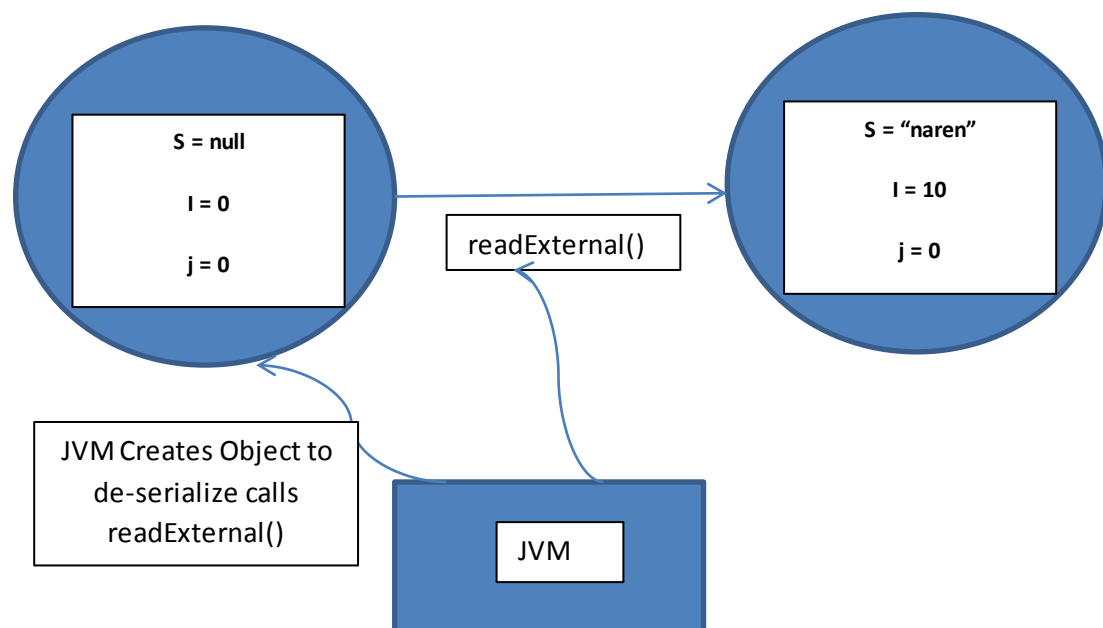
- At the time of serialization JVM will check whether any instance variables are inheriting from non-serializable parent or not. **If any variable is inheriting from non-serializable parent then JVM ignores original value and save default value to file.**
- At the time of deserialization JVM checks if any parent class is non-serializable or not if any parent class is **non-serializable then JVM executes "Instance control flow" in that non-serializable parent and share its instance variables to the current object.**
- To execute instance control flow execution of non-serializable parent JVM will always invoke No argument constructor. Hence every non-serializable class should compulsorily contain a No argument constructor, otherwise we will get run time exception saying "InvalidClassException".

5 Externalization

- To overcome problems of serialization we need to go for externalization.
- In serialization everything is taken care by JVM, and programmers have no control.
- In serialization all the properties then there might be a performance issues as cannot serialize part of properties.
- In Externalization everything will be taken care by programmer.
- To provide externalizable ability class has to implement “Externalizable” Interface.
- Externalizable contains following methods
 - writeExternal()
 - readExternal()
- Because of programmers laziness, serialization became so popular concept as he is lazy to implement logic to do serialization.

1. writeExternal() method –
 - a. Used for serialization
 - b. To save required properties to a file
2. readExternal() method –
 - a. Used for deserialization
 - b. To read required properties to a file and assigned to object.

- At the time of deserialization JVM creates a new Object and call “readExternal” on that object.
- Class should have a no-argument constructor.
- In serialization need not to have a no-argument constructor.
- No use/required of transient key word in case of Externalization.
- A new Object will be created like below in case of Deserialization.



Example:

```
Class ExternalizableEx implements Externalizable {  
    String s;  
    Int i;  
    Int j;  
    ExternalizableEx(){  
        System.out.print("no argument constructor");  
    }  
    ExternalizableEx(String s, int i, int j){  
        this.s=s;  
        this.i=i;  
        this.j=j;  
    }  
  
    public void writExternal(OOS oos) throws IOException {  
        oos.writObject(s);  
        oos.writInt(i);  
    }  
  
    public void readExternal(OIS ois) throws IOException, ClassNotFoundException {  
        S = (String) ois.readObject();  
        i = (String) ois.readInt();  
    }  
  
    Public static void main (String[] args) {  
        ExternalizableEx e1 = new ExternalizableEx ("nare", 10, 20);  
        FileOutputStream fos = new FileOutputStream (abc.ser);  
        ObjectOutputStream oos = new ObjectOutputStream (fos);  
        os.writeObject(e1);  
        FileInputStream fis = new FileInputStream (abc.ser);  
        ObjectInputStream ois = new ObjectInputStream (fos);  
        ExternalizableEx e2 = (ExternalizableEx) ois.readObject();  
        System.out.print(e2.i + e2.j + e.s);  
    }  
}
```

6 Serialization vs Externalization

Serialization	Externalization
It is meant for default serialization	It is meant for customized serialization
Here everything take care by JVM and programmer doesn't have control	Here everything takes care by programmer, JVM doesn't have a 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 or part of the object
Relatively performance is low	Relatively performance is high
serializations is best choice if you want to save total object to a file	This best choice if we want to save part of a file
Serializable interface doesn't contain any methods. It's a marker interface.	Externalizable contains 2 methods – writeExternal(), readExternal(). So it is not a marker interface
Serializable implemented class doesn't need to have a no-argument constructor	Externalizable implemented class should have a public no argument constructor else it will throw "InvalidClassException"
Transient keyword will play a role in serialization	Transient keyword won't play any role in externalization, of course it is not required.

7 Serial Version UID

In serializations 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

Machines may be different.

At the time of serialization JVM will see 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 de-serialization will be performed otherwise receiver cannot de-serialize and will get “Runtime exception” saying “InvalidClassException”.

This unique ID is nothing but serial version ID.

Problem of depending on default Serial version UID generated by JVM:

1. Both sender and receiver should use same JVM with respect to vendor and version. Here if there is any incompatibility in JVM versions, then receiver is unable to de-serialize because of different “SerialVersion UID’s in this case receiver will get runtime exception saying “InvalidClassException”
2. After Serialization, if we change “.class” file at receiver side then we can’t perform deserialization because of miss match in serialversionUID’s or because of miss match in serialization object. In this case at the time of deserialization we will get runtime exception saying “InvalidClassException”
3. To generate “SerialVersionUID” internally JVM will use complex algorithm which creates performance problems.

We can solve above problems by configuring our own serialversionUID like below : –

```
Class Dog {  
  
    Private static final long serialVersionUID = 1L;  
  
    Int i=0, j=0, k=0;  
  
}
```

If we provide serialVersionUID value in a serializable class then there is no problem while doing the serialization.