Serialization

① Introduction

② Object Graphs in Serialization

③ Customized Serialization

④ Serialization w.r.t Inheritance.

Serilization:-

→ The process of writing state of an object to a file is called Serialization.

→ But strictly it is a process of Converting an object from java Supported form to either file Supported form on network Supported form.

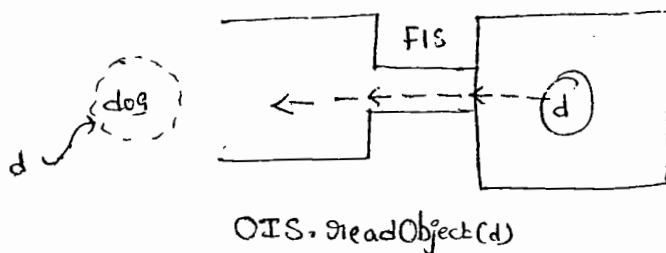→ By using FileOutputStream and ObjectOutputStream classes we can achieve Serialization.

OOS.writeObject (d)   abc.txt
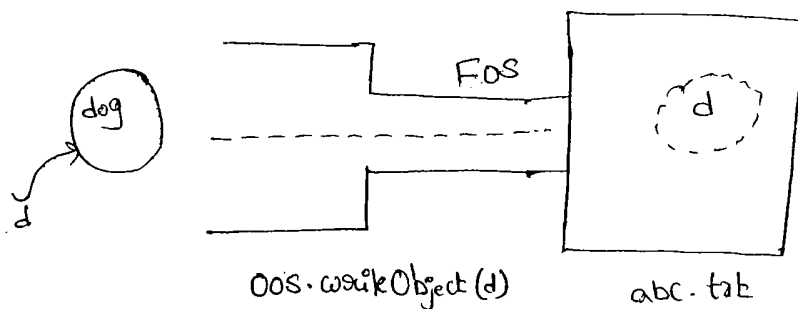
Deserialization:-

→ The process of reading state of an object from a file is called Deserialization.

→ But Strictly Speaking, It is not the process of Converting an Object from either network Supported form or File Supported form to java Supported form.

→ By using │FileInputStream│ and │ObjectInputStream│ Classes we Can achieve De-Serialization.

OIS. ReadObject(d)

Ex:-
```
import java.io.*;
Class Dog implements Serializable
{
    int i = 10;
    int j = 20;
}
Class SerializableDemo
{
    p.s.v.m(————)throws Exception
    {
        Dog d₁ = new Dog();
        FileOutputStream fos = new FileOutputStream("abc.txt");
        ObjectOutputStream oos = new Object OutputStream(fos);
        oos.write Object(d₁);
```

Serialization

```
FileInputStream  fis = new FIS ("abc.tab");

   OIS   ois = new OIS(fis);

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

   S.o.pln (d2.i + " - - - - " + d2.j);
      }
   }
```

→ We Can Perform Serialization only for Serializable Objects.

→ An Object is said to be Serializable iff the Corresponding class

   implements Serializable interface.

→ Serializable interface present in <u>java.io</u> package

   and doesn't Contain any methods, it is a marker interface.

→ If we are trying to serialize a non-Serializable Object we will

   get Run-time exception saying <u>Not Serializable Exception</u>.

transient Keyword :-

→ At the time of Serialization if we dont want to Serialize the

   Value of a particular variable to meet the Security Constraint

   we have to declare those variables with "<u>transient</u>" keyword.

→ At the time of Serialization Jvm ignores original value of

   transisent variable and saves default value.

transient Vs Static :-

→ Static variables are not part of object hence they won't participate in Serialization process. Due to this declaring a Static variable as transistent There is no impact.

transistent Vs final :-

→ final variables will be participated into Serialization directiy by their values Hence declaring a final variable with transistent There is no impact.

Summary :-

declaration	o/p
① int i =10 int j =20	10 - - - - - 20
② transient int i =10; int j = 20	0 - - - - - 20
③ transient final int i=10; transient int j =20;	10 - - - - - 0
④ transient int i = 10; transient static int j=20;	0 - - - - 20

Object Graph in Serialization :-

→ when ever we are trying to Serialize an object the Set of all objects which are reachable from that object will be Serialized automatically this group of objects is Called "Object Graph".

→ In Object Graph every object should be Serializable other wise we will get "Not - Serializable - Exception".

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 SerializeDemo2
{
    P.S.v.m(String[] args) throws Exception
    {
        Dog d = new Dog();
        FileOutputStream fos = new FileOutputStream("abc.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(d);
```
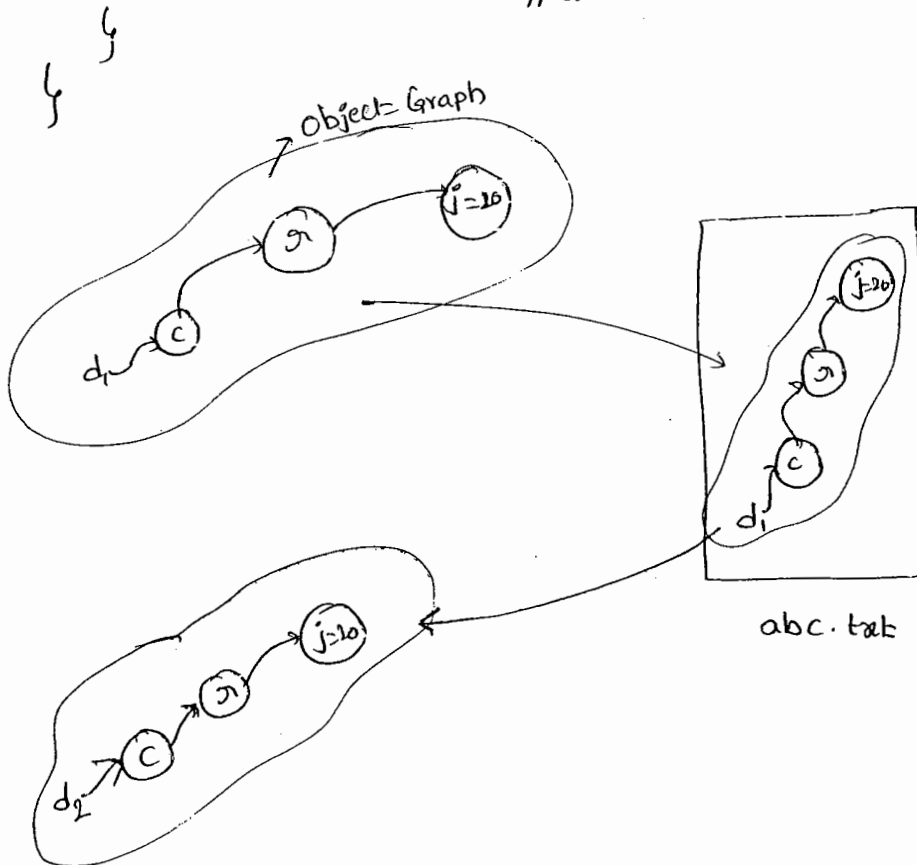
```
FileInputStream  fis = new  FileInputStream ("abc-ser")
Object InputStream  ois = new  ObjectInputStream (fis);

  Dog  d1  = (Dog) ois . read Object ();

  S.o.pln (d1.c.ar.j);   //20
 }
}
}
```


abc. txt

→ In the above prog. whenever we are Serializing a Dog object
automatically Cat & Rat objects will be Serialized. because these are
the part of object graph of dog.

→ Among dog, Cat & Rat if atleast one class is not Serializable
Then we will get Not Serializable Exception.

Customized Serialization :-

→ In the default Serialization there may be a chance of loss of information because of <u>transient</u> Keyword.

Ex:-
```
class Account implements Serializable
{
    String username = "durga";
    transient String password = "anushka";
}
Class SerializeDemo2
{
    P.S.v.m (String[] args) throws Exception
    {
        Account a1 = new Account();
        S.o.pln(a1.username + "-----" a.password);    durga ... anushka
        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)read Object();
        S.o.pln(a2.username + "-----" + a2.password);
    }
}
```
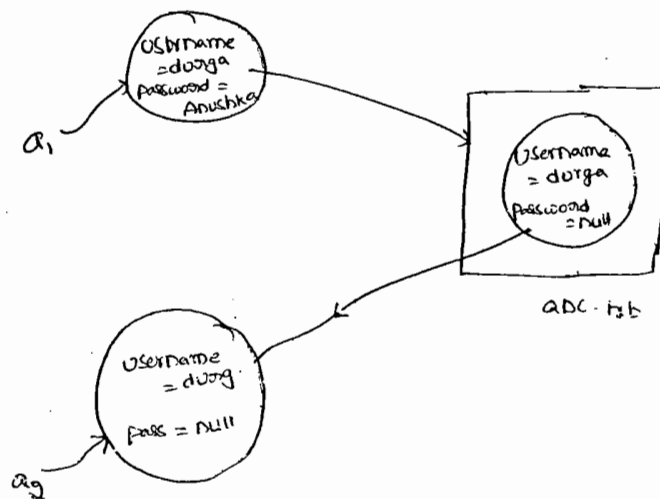
Username
=durga
password =
Anushka

a₁

Username
= durga
Password
= null

adc.txt

Username
= durg
pass = null

a₂

→ In the above Example before Serialization Account object can provide proper username & password But after deserialization Account object Can't provide the Original password. Hence during default Serialization there may be a chance of loss of information due to <u>transient Key word</u>. We Can recover this loss of information by using <u>Customized Serialization</u>

→ We Can implement Customized Serialization by using the following 2 methods

(1) <u>private void writeObject(OOS os) throws Exception</u>

→ This method will be Executed automatically at the time of Serialization it is a Call back method.

(2) private void readObject(OIS is) throws Exception

→ This method will be Executed automatically at the time of deserialization it is a Callback method.

→ The above 2 methods We have to define in the Corresponding Class of Serialized Object.

ex:-

```java
import java.io.*;
class Account implements Serializable
{
    String username = "durga";
    transient String pwd = "anushka";
    private void writeObject(ObjectOutputStream os) throws Exception
    {
        os.defaultWriteObject();
        String epwd = (String)is.readObject();
        pwd = epwd.substring(3);
    }
}
class CustSerializeDemo1
{
    P.S.V.m(String[] args) throws Exception
    {
        Account a1 = new Account();
        S.o.pln(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 = new(Account)ois.readObject();
```
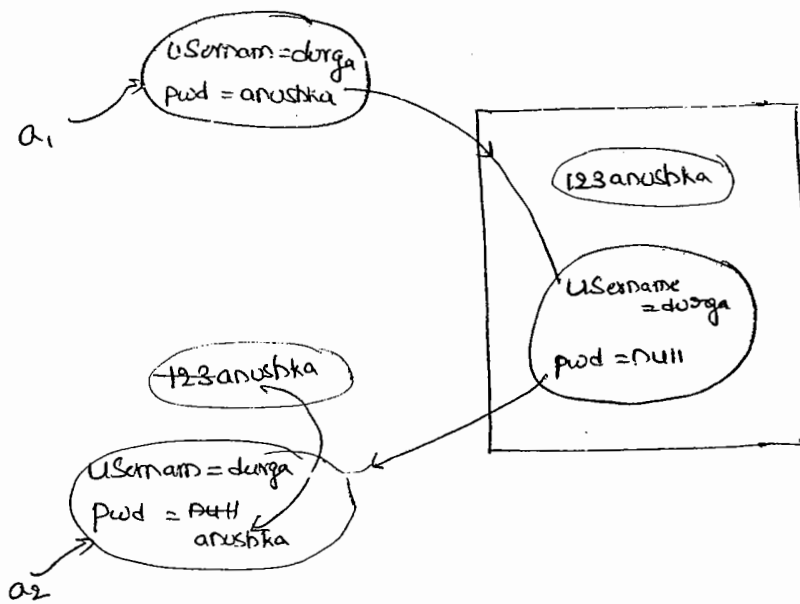
S.O.pln (a2.username + '----' + a2.pwd);
}
}

Serialization w.r.t Inheritance:-

Serialization w.r.t Inheritance:-

Case1:-

→ If the parent class implements Serializable then every child class is by default Serializable. i.e, Serializable nature is inheriting from parent to child (P→C).

Ex:-   class Animal implements Serializable
          {
              int x=10;
          }
          class Dog extends Animal
          {
              int y=20;
          {

→ we Can Serialize dog object eventhough dog class doesn't implement Serializable interface Explicitly. because it's parent class Animal is Serializable.

Case 2:-

→ Eventhough parent class doesn't implement Serializable & if the child is Serializable then we Can Serialize child class object. At the time of Serialization JVM ignores the Original values of instance variables which are Comeing from NON-Serializable parent & Store default Values.

→ At the time of deserialization JVM checkes is any parent class is non-Serializable or not, JVM Creates a Seperate Object for every non-Serializable parent & share its instance variables to the Current Object.

→ for this JVM always Calls no arguement Constructor of the non-Serializable parent. If the non-Serializable parent doesn't have no arguement Constructor then we will get RuntimeException.

Ex:-

```
import java.io.*;
class Animal
{
   int i=10;
    Animal()
    {
       S.o.pln("Animal Constructor Called");
    }  }
```

```java
Class Dog extends Animal implements Serializable
{
    int j = 20;

    Dog()
    {
        S.o.pln("Dog Constructor Called");
    }
}

Class SerializeDemo6
{
    P.S.V.m(String[] args) throws Exception
    {
        Dog d = new Dog();
        d.i = 888;
        d.j = 999;

        FileOutputStream fos = new FileOutputStream("abc.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(d);
        S.o.pln("Deserialization Started");

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

        Dog di = (Dog)ois.readObject();
        S.o.pln(di.i + "---" + di.j);
    }
}
```

O/P :- Animal Constructor Called

Dog   "   "

Deserialization started

* Animal Constructor Called

10 - - - - 999

d

i =888
j =999

i = 0
j =999

i =10

i=A10
j =999

d1