Serialization

① Introduction

② Object Graphs in Serialization

③ Customized Serialization

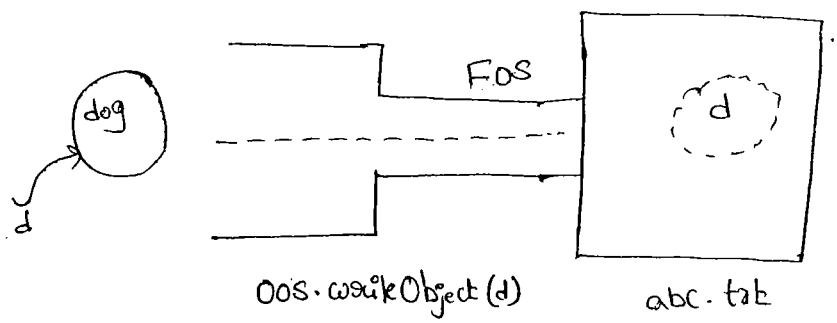
④ Serialization w.r.t Inheritance.

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 or network Supported form.

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

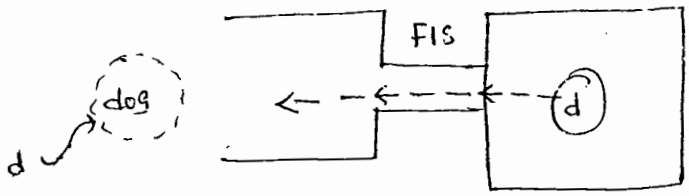


DeSerialization :-

→ 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 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();
```

Serialization

```
FileOutputStream fos = new FileOutputStream("abc.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(d1);
```

```
FileInputStream fis = new FIS("abc.txt");
OIS oos = new OIS(fis);
Dog d2 = (Dog) ois.readObject();
S.o.println(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 java.io 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 NotSerializableException.

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 "transient" keyword.

→ At the time of Serialization JVM ignores original value of transient 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.

transisent Vs final :-

→ final variables will be participated into Serialization directly 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".

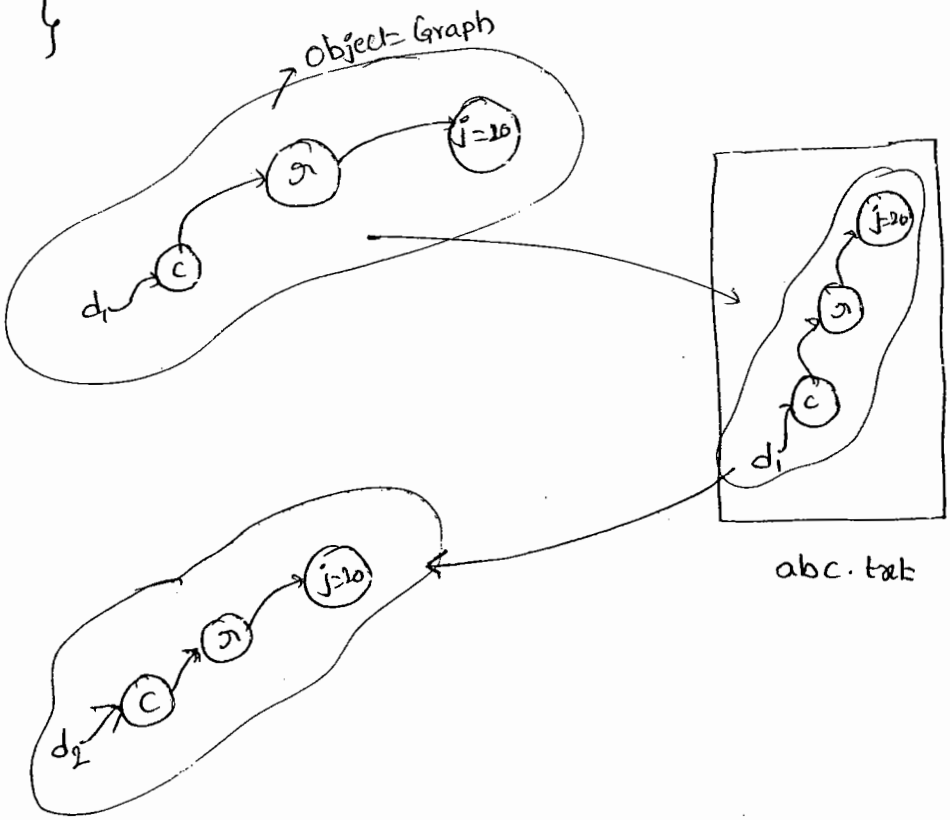
→ In Object Graph every object should be Serializable otherwise we will get "Not-Serializable-Exception".

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");
ObjectInputStream ois = new ObjectInputStream(fis);
Dog d1 = (Dog)ois.readObject();
S.o.pln(d1.c.r.j); //20
}
}
```



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

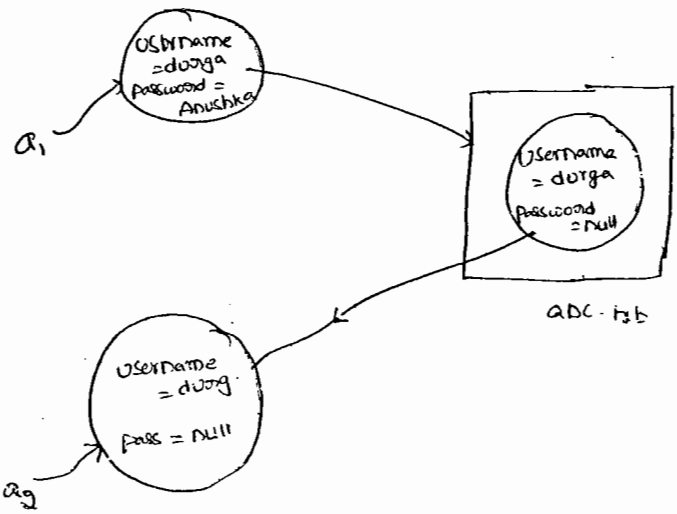
→ Among dog, Cat & Rat if atleast one class is not Serializable Then we will get NotSerializableException.

Customized Serialization :-

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

Ex:-

```
class Account implements Serializable
{
  String username = "durga";
transient String password = "anushka";
}
Class SerializeDemo3
{
 P.S.V.m(String[] args) throws Exception
 {
   Account a1 = new Account();
   S.o.println(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)readObject();
   S.o.println(a2.username + "-----" + a2.password);
 }
}
```



→ 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 transient Key word. We Can recover this loss of information by using Customized Serialization

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

(1) private void writeObject(OOS os) throws Exception

→ 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.

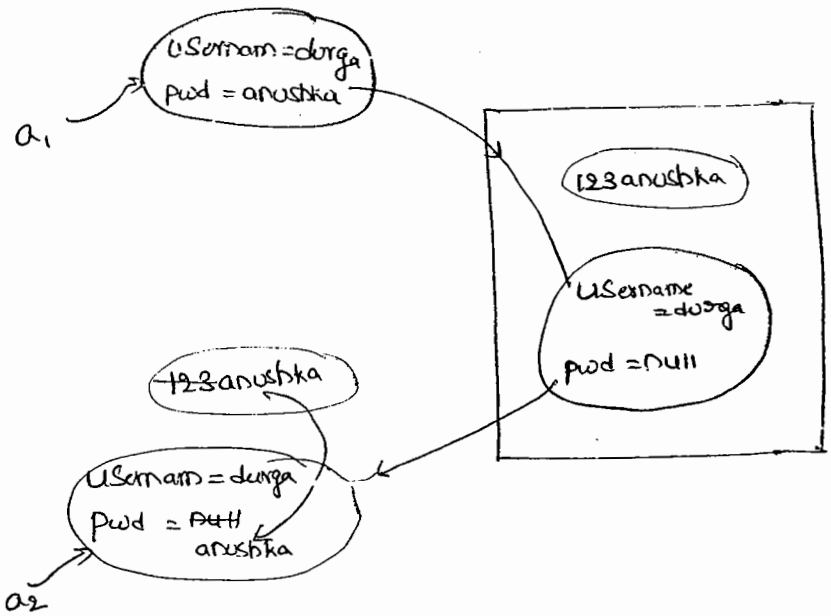
→ The above 2 methods we have to define in the Corresponding Class of Serialized Object.

ex:-

```
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 = (Account)ois.readObject();
```

```
        S.O.println(a2.username + "----" + a2.pwd);
    }
}
```

~~Serialization w.r.t Inheritance~~ :-



Serialization w.r.t Inheritance :-

Case 1:-

→ 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 coming 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 arrguement Constructor then we will get RuntimeException.

Ex:-

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

```
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 d1 = (Dog)ois.readObject();
        S.o.pln(d1.i+"---"+d1.j);
    }
}
```

O/P :- Animal Constructor Called

Dog " "

Deserialization started

* Animal Constructor Called

10 ---- 999