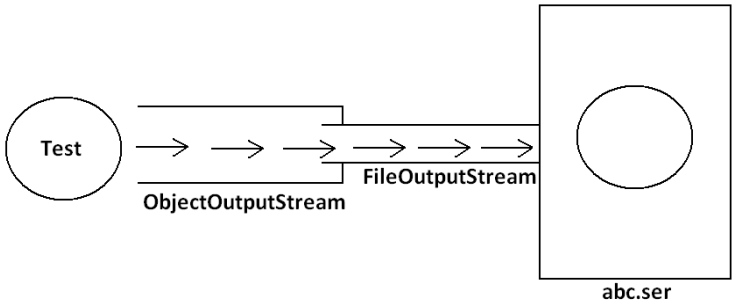
SERIALIZATION

1. Introduction.
2. Object graph in serialization.
3. customized serialization.
4. Serialization with respect inheritance.

Serialization: The process of saving (or) 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 network supported form (or) file supported form.

- By using FileOutputStream and ObjectOutputStream classes we can achieve serialization process.

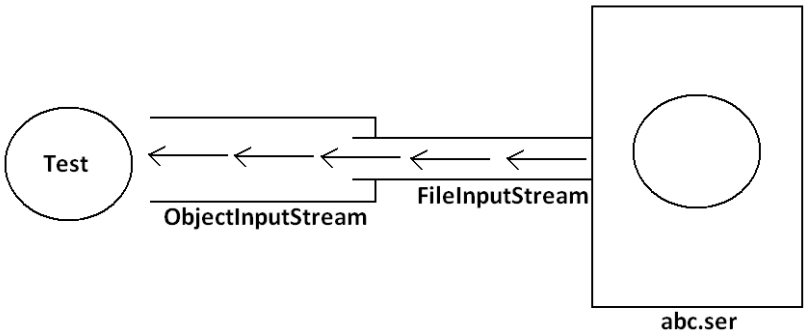
Diagram:



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 file supported form (or) network supported form to java supported form.

- By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

Diagram:



Example 1:

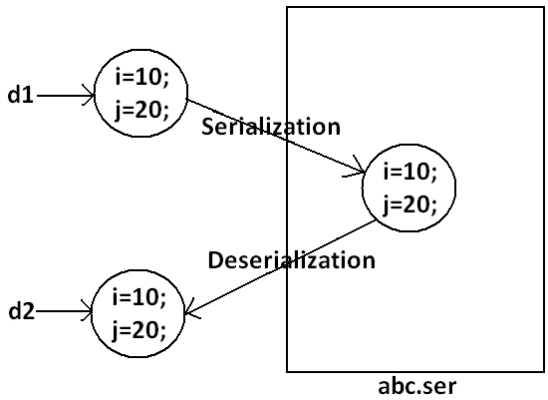
```
import java.io.*;
class Dog implements Serializable
{
int i=10;
int j=20;
```

```
}
class SerializableDemo
{
public static void main(String args[])throws Exception{
Dog d1=new Dog();
System.out.println("Serialization started");
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
System.out.println("Serialization ended");
System.out.println("Deserialization started");
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
System.out.println("Deserialization ended");
System.out.println(d2.i+"....."+d2.j);
}
}
```

Output:

```
Serialization started
Serialization ended
Deserialization started
Deserialization ended
10.....20
```

Diagram:



Note: We can perform Serialization only for Serilizable objects.

- An object is said to be Serilizable if and only if the corresponding class implements Serializable interface.

- Serializable interface present in **java.io package** and does not contain any methods. It is marker interface. The required ability will be provided automatically by JVM.
- We can add any no. Of objects to the file and we can read all those objects from the file but in which order we wrote objects in the same order only the objects will come back. That is order is important.

Example2:

```
import java.io.*;
class Dog implements Serializable
{
int i=10;
int j=20;
}
class Cat implements Serializable
{
int i=30;
int j=40;
}
class SerializableDemo
{
public static void main(String args[])throws Exception{
Dog d1=new Dog();
Cat c1=new Cat();
System.out.println("Serialization started");
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
oos.writeObject(c1);
System.out.println("Serialization ended");
System.out.println("Deserialization started");
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
Cat c2=(Cat)ois.readObject();
System.out.println("Deserialization ended");
System.out.println(d2.i+"....."+d2.j);
System.out.println(c2.i+"....."+c2.j);
}
}
```

Output:

Serialization started
Serialization ended
Deserialization started
Deserialization ended
10.....20
30.....40

Transient keyword:

- While performing serialization if we don't want to serialize the value of a particular variable then we should declare that variable with "transient" keyword.
- At the time of serialization JVM ignores the original value of transient variable and save default value.
- That is transient means "not to serialize".

Static Vs Transient:

- static variable is not part of object state hence they won't participate in serialization because of this declaring a static variable as transient these is no use.

Transient Vs Final:

- final variables will be participated into serialization directly by their values. Hence declaring a final variable as transient there is no use.

Example 3:

```
import java.io.*;
class Dog implements Serializable
{
static transient int i=10;
final transient int j=20;
}
class SerializableDemo
{
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.i+"....."+d2.j);
}
```

}

Output:

10.....20

Diagram:

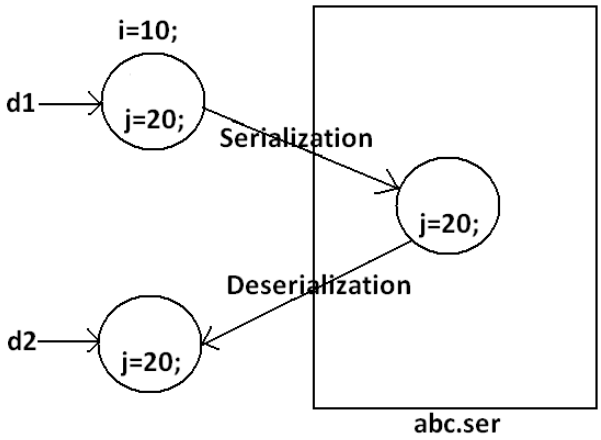


Table:

declaration	output
int i=10; int j=20;	10.....20
transient int i=10; int j=20;	0.....20
transient int i=10; transient static int j=20;	0.....20
transient final int i=10; transient int j=20;	10.....0
transient final int i=10; transient static int j=20;	10.....20

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 serialization.
- In object graph every object should be Serializable otherwise we will get runtime exception saying "NotSerializableException".

Example 4:

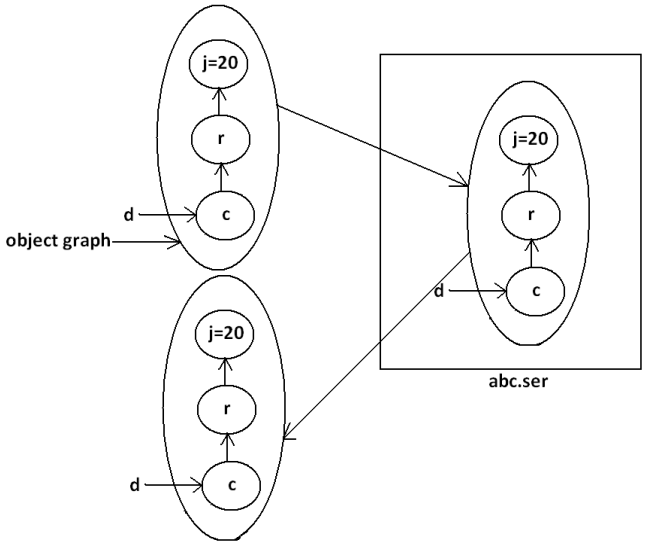
```
import java.io.*;
class Dog implements Serializable
{
Cat c=new Cat();
}
```

```
class Cat implements Serializable
{
Rat r=new Rat();
}
class Rat implements Serializable
{
int j=20;
}
class SerializableDemo
{
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);
}
}
```

Output:

20

Diagram:



- 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, Rat if at least one object is not serializable then we will get runtime exception saying “NotSerializableException”.

Customized serialization:

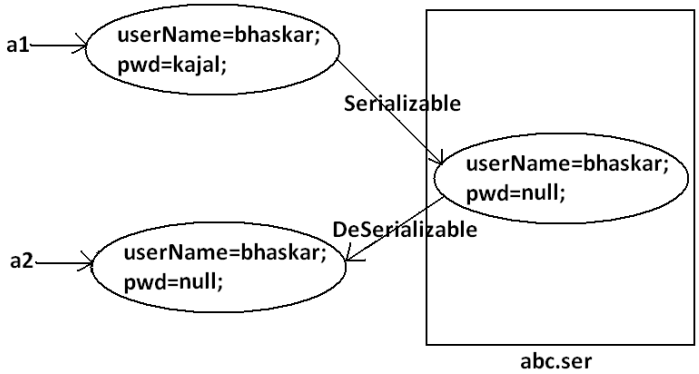
Example 5:

```
import java.io.*;
class Account implements Serializable
{
String userName="Bhaskar";
transient String pwd="kajal";
}
class CustomizedSerializeDemo
{
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);
}
}
```

Output:

```
Bhaskar.....kajal
Bhaskar.....null
```

Diagram:



- In the above example before serialization Account object can provide proper username and password. But after Deserialization Account object can provide only username bur not password. This is due to declaring password as transient. Hence doing default serialization there may be a chance of loss of information due to transient keyword.
- We can recover this loss of information by using customized serialization.
- We can implements customized serialization by using the following two methods.
    1) private void writeObject(OutputStream os) throws Exception.
- This method will be executed automatically by jvm at the time of serialization.
- It is a callback method. Hence at the time of serialization if we want to perform any extra work we have to define that in this method only.
    2) private void readObject(InputStream is)throws Exception.
- This method will be executed automatically by JVM at the time of Deserialization. Hence at the time of Deserialization if we want to perform any extra activity we have to define that in this method only.

<u>Example 6:</u> Demo program for customized serialization to recover loss of information which is happen due to transient keyword.

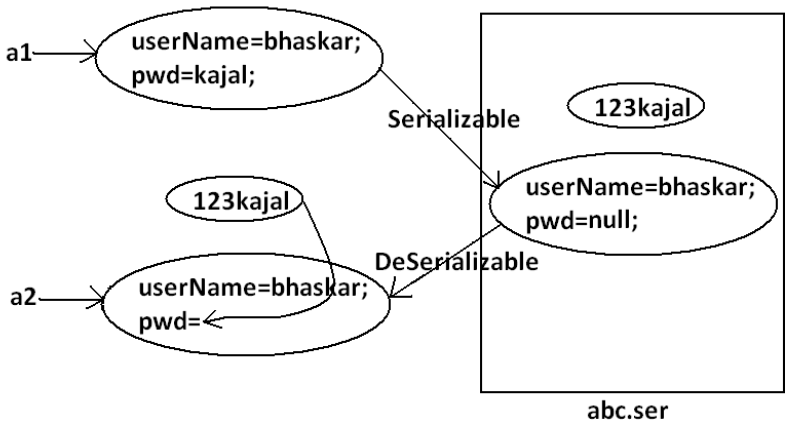
```
import java.io.*;
class Account implements Serializable
{
String userName="Bhaskar";
transient String pwd="kajal";
private void writeObject(ObjectOutputStream os)throws Exception
{
os.defaultWriteObject();
String epwd="123"+pwd;
os.writeObject(epwd);
}
private void readObject(ObjectInputStream is)throws Exception{
is.defaultReadObject();
String epwd=(String)is.readObject();
pwd=epwd.substring(3);
}
}
class CustomizedSerializeDemo
{
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);
}
}
```

Output:

Bhaskar.....kajal
Bhaskar.....kajal

Diagram:



- At the time of Account object serialization JVM will check is there any writeObject() method in Account class or not. If it is not available then JVM is responsible to perform serialization(default serialization). If Account class contains writeObject() method then JVM feels very happy and executes that Account class writeObject() method. The same rule is applicable for readObject() method also.

Serialization with respect to inheritance:

Case 1:

- If parent class implements Serializable then automatically every child class by default implements Serializable. That is Serializable nature is inheriting from parent to child.

Example 7:

```
import java.io.*;
class Animal implements Serializable
{
int i=10;
```

```
}
class Dog extends Animal
{
int j=20;
}
class SerializableWRTInheritance
{
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);
}
}
```

Output:

10.....20

10.....20

- Even though Dog class does not implements Serializable interface explicitly but we can Serialize Dog object because its parent class animal already implements Serializable interface.

Case 2:

- Even though parent class does not implements Serializable we can serialize child object if child class implements Serializable interface.
- At the time of serialization JVM ignores the values of instance variables which are coming from non Serializable parent JVM saves default values for those variables.
- At the time of Deserialization JVM checks whether any parent class is non Serializable or not. If any parent class is non Serializable JVM creates a separate object for every non Serializable parent and shares its instance variables to the current object.
- For this JVM always calls no arg constructor(default constructor) of that non Serializable parent hence every non Serializable parent should compulsory contain no arg constructor otherwise we will get runtime exception.

Example 8:

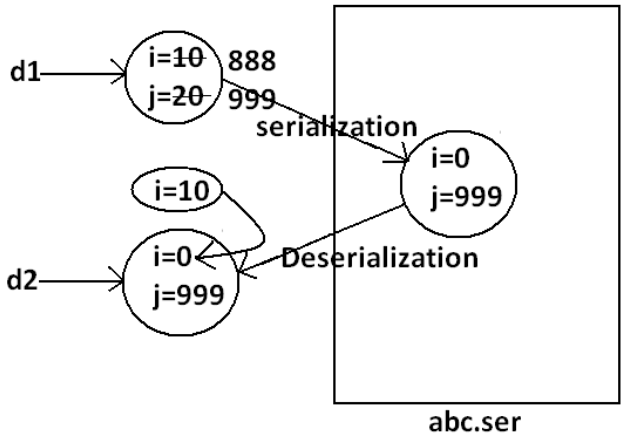
```
import java.io.*;
```

```
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 SerializableWRTInheritance
{
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);
}
}
```

Output:

```
Animal constructor called
Dog constructor called
Deserialization started
Animal constructor called
10.....999
```

Diagram:


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