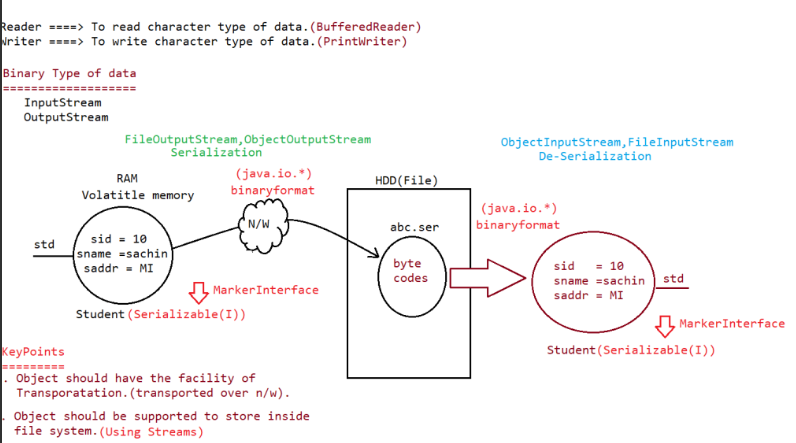
Serialization and Deserialization:



Serialization:

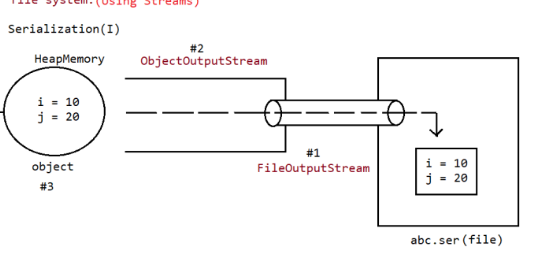
=> The process of saving (or) writing state of an object to a file is called serialization but strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.

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

|=> writeObject(Object obj)

Note:

We can serialize any no of objects to the file but in which order we serialized in the same order only we have to deserialize, if we change the order then it would result in "ClassCastException".

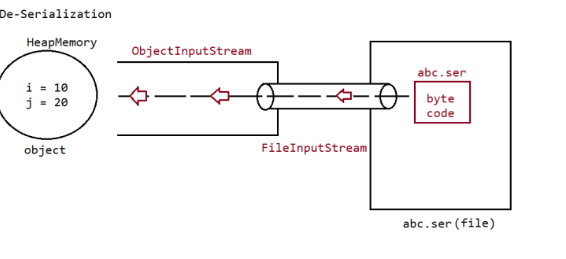


DeSerialization:

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

=> By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

|=> readObject()



Program:

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.io.FileInputStream;

import java.io.ObjectInputStream;

import java.io.Serializable;

class Dog implements Serializable{

int i=10;

int j=20;

}

class Cat implements Serializable{

int i=100;

int j=200;

}

public class TestApp {

public static void main(String[] args)throws

IOException,ClassNotFoundException {

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("Dog object data");

System.out.println(d2.i+"\t" +d2.j);

System.out.println("Cat object data");

System.out.println(c2.i+"\t" +c2.j);

}

}

Output

serialization started

Serialization ended

Deserialization started

Deserialization ended

Dog object data

10 20

Cat object data

100 200

Note:

1. We can perform Serialization only for Serilizable objects.

2. An object is said to be Serilizable if and only if the corresponding class implements Serializable interface(MI).

3. Serializable interface present in java.io package and does not contain any abstract methods. It is marker interface.The required ability will be provided automatically by JVM.

4. 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.

5. If we are trying to serialize a non-serializable object then we will get RuntimeException saying "NotSerializableException".

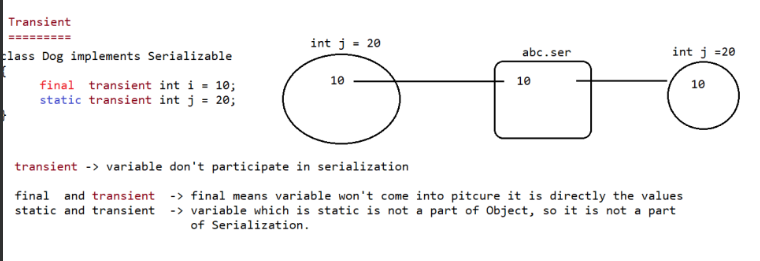
Transient keyword:

1. transient is the modifier applicable only for variables,but not for classes and methods.

2. While performing serialization if we don't want to save the value of a particular variable to meet security constant such type of variable ,then we should declare that variable with "transient" keyword.

3. At the time of serialization JVM ignores the original value of transient variable and save default value to the file .

4. That is transient means "not to serialize".



static Vs transient :

1. static variable is not part of object state hence they won't participate in

serialization because of this declaring a static variable as

transient there is no use.

Transient Vs Final:

1. final variables will be participated into serialization directly by their values.

Hence declaring a final variable as transient there is no use.

//the compiler assign the value to final variable

eg: final int x= 10;

int y = 20;

System.out.println(x);// compiler will replace this as System.out.println(20) becoz x is final.

System.out.println(y);

Note:

We can serialize any no of objects to the file but in which order we serialized in

the same order only we have to deserialize,

if we change the order then it would result in "ClassCastException".

Example :

Dog d1=new Dog( );

Cat c1=new Cat( );

Rat r1=new Rat( );

FileOutputStreamfos=new FileOutputStream("abc.ser");

ObjectOutputStreamoos=new ObjectOutputStream(fos);

oos.writeObject(d1);

oos.writeObject(c1);

oos.writeObject(r1);

FileInputStreamfis=new FileInputStream("abc.ser");

ObjectInputStreamois=new ObjectInputStream(fis);

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

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

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

=> If we don't know the order of Serialization then we need to use the following code

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

ObjectInputStream ois=new ObjectInputStream(fis);

Object obj=ois.readObject();

if(obj instanceof Dog){

Dog d=(Dog)obj;

//perform operation related to Dog

}

if(obj instanceof Cat){

Cat C=(Cat)obj;

//perform operation related to Cat

}

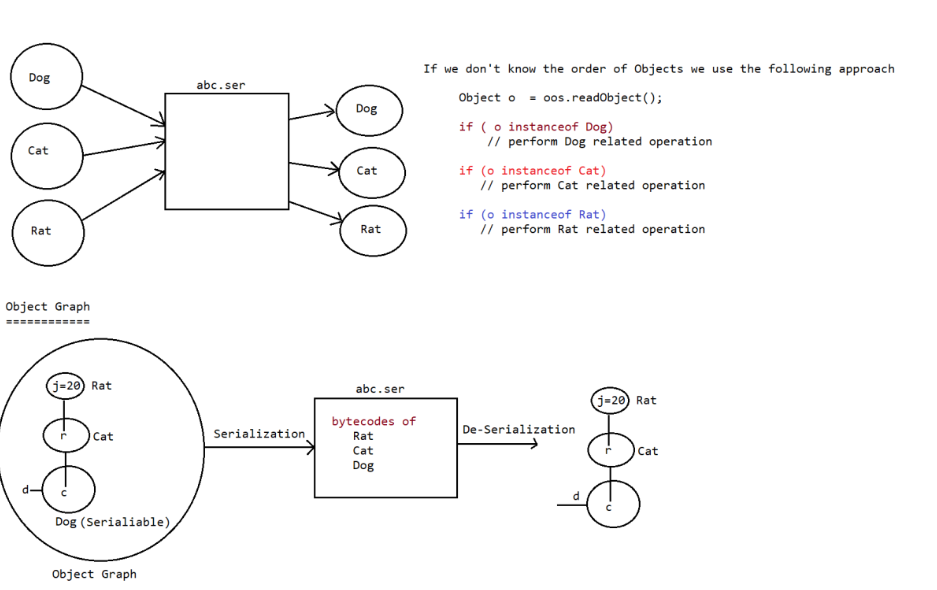
if(obj instanceof Rat){

Rat r=(Rat)obj;

//perform operation related to Rat

}

Object graph in serialization:



1. 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.

2. In object graph every object should be Serializable otherwise we will get runtime exception saying

"NotSerializableException".

eg#1.

import java.io.Serializable;

import java.io.FileOutputStream;

import java.io.ObjectOutputStream;

import java.io.FileInputStream;

import java.io.ObjectInputStream;

import java.io.IOException;

class Dog implements Serializable{

Cat c=new Cat();

}

class Cat implements Serializable{

Rat r=new Rat();

}

class Rat implements Serializable{

int i=10;

}

public class Test {

public static void main(String[] args)throws

IOException,ClassNotFoundException{

Dog d= new Dog();

System.out.println("Serialization Started");

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

ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(d);

System.out.println("Serialization ended");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("DeSerialization Started");

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

ObjectInputStream ois=new ObjectInputStream(fis);

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

System.out.println(d1.c.r.i);

System.out.println("DeSerialization ended");

}

}

Output

======

Serialization Started

Serialization ended

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DeSerialization Started

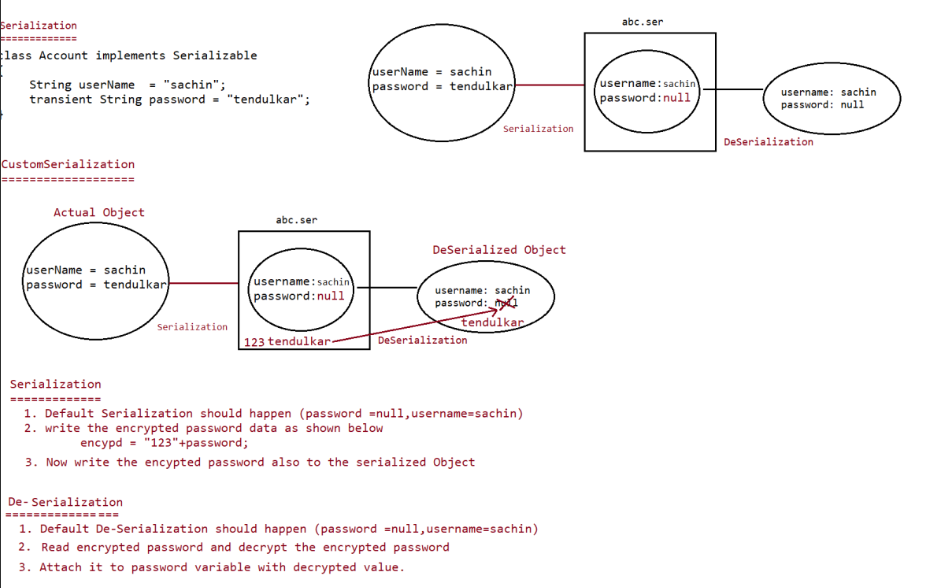
10

DeSerialization ended

CustomizedSerialization

Why:

During default Serialization there may be a chance of lose of information due to transient keyword.



eg#1.

import java.io.Serializable;

import java.io.FileOutputStream;

import java.io.ObjectOutputStream;

import java.io.FileInputStream;

import java.io.ObjectInputStream;

import java.io.IOException;

class Account implements Serializable{

String name="sachin";

transient String password="tendulkar";

}

public class Test {

public static void main(String[] args)throws

IOException,ClassNotFoundException{

Account acc=new Account();

System.out.println(acc.name +"=====> "+ acc.password);

System.out.println("Serialization Started");

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

ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(acc);

System.out.println("Serialization ended");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("DeSerialization Started");

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

ObjectInputStream ois=new ObjectInputStream(fis);

acc=(Account)ois.readObject();

System.out.println(acc.name +"=====> "+ acc.password);

System.out.println("DeSerialization ended");

}

}

=> In the above example before serialization Account object can provide proper username and password.

But after Deserialization Account object can provide only username but 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(ObjectOutputStream os) throws Exception.

=> This method will be executed automatically by jvm at the time of serialization.

=> Hence at the time of serialization if we want to perform any extra work we have to define that in this method only. (prepare encrypted password and write encrypted password seperate to the file )

2. private void readObject(ObjectInputStream 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.

(read encrypted password , perform decryption and assign decrypted password

to the current object password variable )

=> 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.

import java.io.\*;

/\*

public java.io.ObjectOutputStream(java.io.OutputStream) throws

java.io.IOException;

public java.io.FileOutputStream(java.lang.String) throws

java.io.FileNotFoundException;

public final void writeObject(java.lang.Object) throws java.io.IOException;

public java.io.ObjectInputStream(java.io.InputStream) throws java.io.IOException;

public java.io.FileInputStream(java.lang.String) throws

java.io.FileNotFoundException;

public final java.lang.Object readObject() throws java.io.IOException,

java.lang.ClassNotFoundException;

\*/

class Account implements Serializable

{

String userName = "sachin";

transient String password = "tendulkar";//loss of information

transient int pin=4444;//loss of information

//Write a logic of Serialization

private void writeObject(ObjectOutputStream oos) throws Exception{

System.out.println("writeObject method is called....");

// perform default serialization

oos.defaultWriteObject();

// perform encryption on password

String encypwd = "123" + password;// 123tendulkar

int encypin = 1111 + pin;// 5555

// write the encrytped data as object to serialized file

oos.writeObject(encypwd);

oos.writeInt(encypin);

}

//Write a logic of Deserialization

private void readObject(ObjectInputStream ois) throws Exception{

System.out.println("readObject method is called....");

//perform default deserialization

ois.defaultReadObject();

//read encrypted data from serialized file

String encypwd = (String)ois.readObject();

int encypin = ois.readInt();

// perform decryption and attach it to instance variable

password = encypwd.substring(3);// tendulkar

pin = encypin - 1111;// 4444

}

}

class Test

{

public static void main(String[] args)throws Exception

{

Account account =new Account();

System.out.println("Serialization started.....");

String fileName = "abc.ser";

FileOutputStream fos = new FileOutputStream(fileName);

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(account);

System.out.println("Serialization ended.....");

//To pause the execution till we press some key from keyboard

System.in.read();

System.out.println("De-Serialization started.....");

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

ObjectInputStream ois = new ObjectInputStream(fis);

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

System.out.println("Username is :: "+acc.userName);

System.out.println("Password is :: "+acc.password);

System.out.println("Pin is :: "+acc.pin);

System.out.println("De-Serialization ended.....");

}

//JVM shutdown now

}

Output

D:\IOOperations>javac Test.java

D:\IOOperations>java Test

Serialization started.....

writeObject method is called....

Serialization ended.....

De-Serialization started.....

readObject method is called....

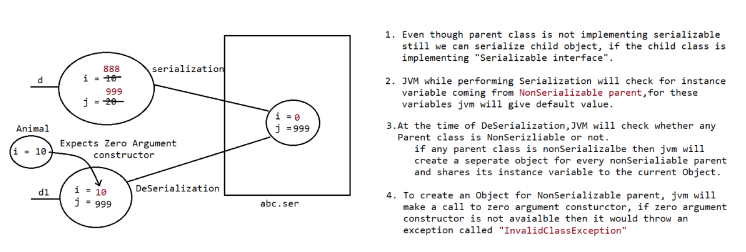
Username is :: sachin

Password is :: tendulkar

Pin is :: 4444

De-Serialization ended.....

Serialization w.r.t 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.

Hence even though child class doesn't implements Serializable , we can serialize

child class object if parent class implements

serializable interface.

import java.io.Serializable;

import java.io.FileOutputStream;

import java.io.ObjectOutputStream;

import java.io.FileInputStream;

import java.io.ObjectInputStream;

import java.io.IOException;

class Animal implements Serializable{

int i=10;

}

class Dog extends Animal{

int j=20;

}

public class Test {

public static void main(String[] args)throws

IOException,ClassNotFoundException{

Dog d=new Dog();

System.out.println("Serialization started");

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

ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(d);

System.out.println("Serialization ended");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("DeSerialization started");

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

ObjectInputStream ois=new ObjectInputStream(fis);

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

System.out.println(d1.i+"====> "+d1.j);

System.out.println("DeSerialization ended");

}

}

Output

Serialization started

Serialization ended

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DeSerialization started

10====> 20

DeSerialization ended

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.

Note :Object class doesn't implement Serializable interface.

Case 2:

1. Even though parent class does not implementsSerializable we can serialize child

object if child class implements Serializable

interface.

2. At the time of serialization JVM ignores the values of instance variables which

are coming

from non Serializable parent then instead of original value JVM saves default

values for those variables to the file.

3. At the time of Deserialization JVM checks whether any parent class is non

Serializable or not.

If any parent class is nonSerializable JVM creates a separate object for every

non Serializabl parent and

shares its instance variables to the current object.

4. To create an object for non-serializable parent 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

"InvalidClassException".

eg#1.

import java.io.Serializable;

import java.io.FileOutputStream;

import java.io.ObjectOutputStream;

import java.io.FileInputStream;

import java.io.ObjectInputStream;

import java.io.IOException;

class Animal {

int i=10;

Animal(){

System.out.println("No arg Animal constructor");

}

}

class Dog extends Animal implements Serializable{

int j=20;

Dog(){

System.out.println("No arg Dog constructor");

}

}

public class Test {

public static void main(String[] args)throws

IOException,ClassNotFoundException{

Dog d=new Dog();

d.i=888;

d.j=999;

System.out.println("Serialization started");

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

ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(d);

System.out.println("Serialization ended");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("DeSerialization started");

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

ObjectInputStream ois=new ObjectInputStream(fis);

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

System.out.println(d1.i+"====> "+d1.j);

System.out.println("DeSerialization ended");

}

}

Output

No arg Animal constructor

No arg Dog constructor

Serialization started

Serialization ended

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DeSerialization started

No arg Animal constructor

10====> 999

DeSerialization ended