Day 2: Serialization, transient keyword

Serialization:

Serialization is the conversion of the state of an object into a byte stream. which we can then save to a database or transfer over a network.

The reverse operation of serialization is called *deserialization* where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

We can’t serialize any java class object, There is an interface called **java.io.Serializable,** The class which implements this Serializable interface, that class object is only eligible for serialization.

Note: If we try to serialize a non-serializable object (If the object of a class has not implemented the Serializable interface)then we will get a runtime exception called NotSerializableExcception.

Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability.

Note: all the Wrapper classes, String classes and collection framework related classes internally implements the Serializable interface.

The **java.io.ObjectOutputStream** class helps us to serialize an object of a java class (converts it into a sequence (stream) of bytes)

Similarly, The **java.io.ObjectInputStream** class helps us read a stream of bytes and convert it back into a Java object.(Deserialization)

ObjectOutputStream class has a method by name **writeObject(Serializable s)** ,it takes an object of a class whose class implements Serializable interface. and

converts it into a sequence (stream) of bytes.

Note: we can serialize multiple object, but in which order we serialized those objects, in the same order only we have to deserialize those objects.

Example: Serialization

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

 class A implements Serializable
{

    int i=10;

    public void funA(){

        System.out.println("inside funA() of A");
        System.out.println(i);
    }

}

class Main {

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

        A a1=new A();

        a1.i=22; //change the state of a1 object

        FileOutputStream fos=new FileOutputStream("file1.txt");
        ObjectOutputStream oos=new ObjectOutputStream(fos);

        oos.writeObject(a1);

        oos.writeObject("Welcome");//String class object
        oos.writeObject(10); //autoboxing

        oos.close();

        System.out.println("a1 object is serailized");
    }
}
```

Example of Java Deserialization:

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization

To deserialize an object we need to use following method **ObjectInputStream** class.

```
  public Object readObject();
```

To get back our object we have to downcast the object of Object class into object of Original class.

```
class Main {

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

        FileInputStream fis=new FileInputStream("file1.txt");

        ObjectInputStream ois=new ObjectInputStream(fis);

        Object obj=ois.readObject();

        A a1=(A)obj;

        a1.funA();

        String ss=(String)ois.readObject();
        System.out.println(ss);

        int z=(Integer)ois.readObject();
        System.out.println(z);

        ois.close();


    }
}
```

transient keyword:

This keyword is applicable only for the variables, at the time of serialization if we don't want to save the original value of a particular variable for some security reason, such type of variable we should declare with transient keyword..

For example, if a program accepts a user's login details and password. But we don't want to store the original password in the file. Here, we can use transient keyword

and when JVM reads the transient keyword it ignores the original value of the object and instead stores the default value of the object.

We Problem:

Serializing an object with transient keyword:

```
import java.io.*;
public class Student implements Serializable{

 int id;
 String name;
 transient int age;//Now it will not be serialized

public Student(int id, String name,int age) {
  this.id = id;
  this.name = name;
  this.age=age;
 }
}
class Main{
 public static void main(String args[])throws Exception{
  Student s1 =new Student(211,"ravi",22);//creating object
  //writing object into file
  FileOutputStream f=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(f);

  out.writeObject(s1);

  out.flush();
  out.close();

  System.out.println("success");
 }
}
```

You Problem:

Write a application to de-serialize the above Student object and print their values.

transient vs static:

static variables are not part of object, hence they will also not participate in serialization.

Object Graph: (Serialization with respect to Has-A relationship)

Whenever we are serializing an object, the set of all the objects which are reachable from that object will be serialized automatically. this group of object is nothing but object-graph..

In object-graph every object should be serializable otherwise we will get a Runtime exception NotSerializableException.

Example:

```
import java.io.*;
class Dog implements Serializable{

  Cat c=new Cat();


}

class Cat impelements Serializable{

  Rat r=new Rat();

}

class Rat implements Serializable {

}

class Main{

  public static void main(String[] args){

    Dog d=new Dog();

    FileOutputStream fos=new FileOutputStream("file1.txt");
    ObjectOoutputStream oos=new ObjectOoutputStream(fos);
    oos.writeObect(d);
    oos.flush();
    oos.close();

    System.out.println("done");

}
}
```

In the above program whenever we are serializing Dog obj, automatically Cat and Rat object will be serialized. because these are the part of the object-graph of the dog object.

Among above classes if at-least one class is not Serializable then we will get a Runtime exception.

Serialization with the respect of Inheritance:

If the parent class is implementing Serializable interface then automatically every child class implements Serializable. weather we mention it or not in a child class.

It is a Serializable nature of inheriting parent to child.

Example:

```
import java.io.*;
class Animal {

    int i=10;

}

class Dog extends Animal implements Serializable{

    int j=20;
}


class Main {

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

        FileOutputStream fos=new FileOutputStream("file1.txt");

        ObjectOutputStream oos = new ObjectOutputStream(fos);

        Dog d = new Dog();

        oos.writeObject(d);;

        System.out.println("done..");


    }
}
```

Here we have serialized the dog object, even though Dog class has not implemented the Serializable interface.

If the parent class doesn't implements the Serializable interface still we can serialize the child objects. if the child class implements Serializable interface.

At the time of serialization JVM will ignores the updated values of instance variables which are inherited from the parent.

At the time of Deserialization JVM will check if the parent class is serializable or not. if the parent class is non-serializable then JVM will creates an object for non-serializable parent class by executing default constructor of the parent class and share its default instance variable value to the child objects.

Example

```
import java.io.*;
class A {

    int i=10;

}

class B extends A implements Serializable{

    int j=20;

}


class Main {

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


        B b1 = new B();

        b1.i = 200;
        b1.j = 500;


        FileOutputStream fos=new FileOutputStream("file1.txt");

        ObjectOutputStream oos = new ObjectOutputStream(fos);

        oos.writeObject(b1);;

        System.out.println("serialized the Child object ");
```

```
        System.out.println("After Deserialization");

        ObjectInputStream ois = new ObjectInputStream(new FileInputStream("file1.txt"));

        B b2 = (B)ois.readObject();

        System.out.println(b2.i);
        System.out.println(b2.j);

    }
}

Output:
serialized the Child object
After Deserialization
10
500
```