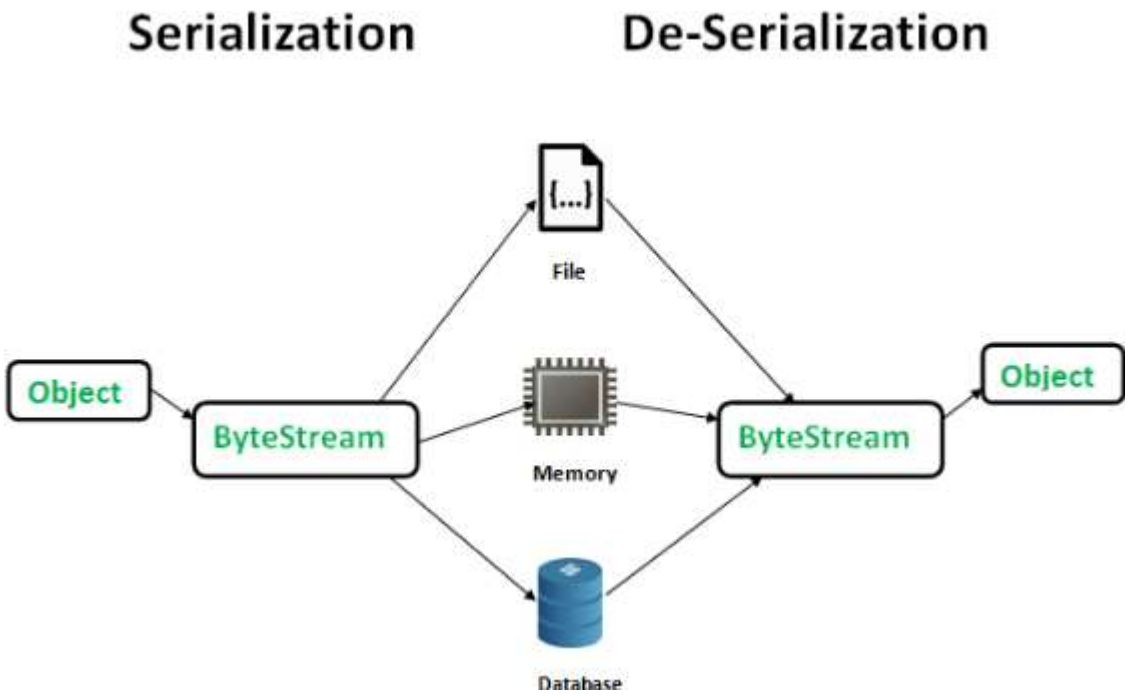
Java Serialization, Deserialization,examples and transient keyword with examples

-K.L.MADHAVI

Java Serialization and Deserialization

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.



Continue…

Serialization and deserialization are crucial for saving and restoring the state of objects in Java.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform. To make a Java object serializable we implement the **java.io.Serializable** interface. The ObjectOutputStream class contains **writeObject()** method for serializing an Object.

```
public final void writeObject(Object obj)
        throws IOException
```

The ObjectInputStream class contains **readObject()** method for deserializing an object.

```
public final Object readObject()
throws IOException,
ClassNotFoundException
```

Continue…

Advantages of Serialization

1. To save/persist state of an object.

2. To travel an object across a network.

Example for Serialization

```
import java.io.*;
class Persist{
 public static void main(String args[]){
 try{
 //Creating the object
 Student s1 =new Student(211,"ravi");
 //Creating stream and writing the object
 FileOutputStream fout=new FileOutputStream("f.txt");
 ObjectOutputStream out=new ObjectOutputStream(fout);
 out.writeObject(s1);
 out.flush();
 //closing the stream
 out.close();
 System.out.println("success");
 }catch(Exception e){System.out.println(e);}
 }
}
```

Example for Deserialization

```
import java.io.*;
class Depersist{
public static void main(String args[]){
try{
//Creating stream to read the object
 ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
Student s=(Student)in.readObject();
 //printing the data of the serialized object
System.out.println(s.id+" "+s.name);
 //closing the stream
 in.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

Transient keyword

In Java, Serialization is used to convert an object into a stream of the byte. The byte stream consists of the data of the instance as well as the type of data stored in that instance. Deserialization performs exactly opposite operation. It converts the byte sequence into original object data. During the serialization, when we do not want an object to be serialized we can use a **transient** keyword.

Why to use the transient keyword?

The transient keyword can be used with the data members of a class in order to avoid their serialization. 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.

Syntax:

private **transient** <member variable>;

Continue…

When to use the transient keyword?

1. The transient modifier can be used where there are data members derived from the other data members within the same instance of the class.
2. This transient keyword can be used with the data members which do not depict the state of the object.
3. The data members of a non-serialized object or class can use a transient modifier.

Example of Java 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 PersistExample{
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();
 f.close();
 System.out.println("success");
}
}
```