3.3 Serialization

Serializable Objects

To*serialize*an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object. A Java object is*serializable*if its class or any of its superclasses implements either thejava.io.Serializableinterface or its subinterface,java.io.Externalizable .*Deserialization*is the process of converting the serialized form of an object back into a copy of the object.

For example, thejava.awt.Buttonclass implements theSerializableinterface, so you can serialize ajava.awt.Buttonobject and store that serialized state in a file. Later, you can read back the serialized state and deserialize into ajava.awt.Buttonobject.

The Java platform specifies a default way by which serializable objects are serialized. A (Java) class can override this default serialization and define its own way of serializing objects of that class. TheObject Serialization Specificationdescribes object serialization in detail.

When an object is serialized, information that identifies its class is recorded in the serialized stream. However, the class's definition ("class file") itself is not recorded. It is the responsibility of the system that is deserializing the object to determine how to locate and load the necessary class files. For example, a Java application might include in its classpath a JAR file that contains the class files of the serialized object(s) or load the class definitions by using information stored in the directory, as explained later in this lesson.

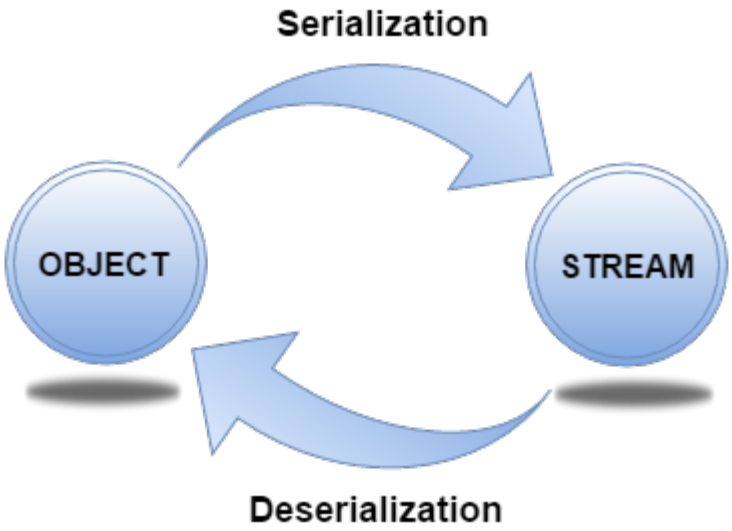
Serialization in java is a mechanism of *writing the state of an object into a byte stream*.

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called *deserialization*.

Advantage of Java Serialization

It is mainly used to travel object's state on the network (known as marshaling).



java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" java classes so that objects of these classes may get certain capability. The Cloneable and Remote are also marker interfaces.

It must be implemented by the class whose object you want to persist.

The String class and all the wrapper classes implements *java.io.Serializable* interface by default.

Let's see the example given below:

```
1. import java.io.Serializable;
2. public class Student implements Serializable{
3.  int id;
4.  String name;
5.  public Student(int id, String name) {
6.   this.id = id;
7.   this.name = name;
8.  }
9. }
```

In the above example, Student class implements Serializable interface. Now its objects can be converted into stream.

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types and Java objects to an OutputStream. Only objects that support the java.io. Serializable interface can be written to streams.

Constructor

1) public ObjectOutputStream(OutputStream out) throws IOException {}creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException {}	writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	flushes the current output stream.
3) public void close() throws IOException {}	closes the current output stream.

Example of Java Serialization

In this example, we are going to serialize the object of Student class. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

```
1. import java.io.*;
2. class Persist{
3.  public static void main(String args[])throws Exception{
4.   Student s1 =new Student(211,"ravi");
5.
6.   FileOutputStream fout=new FileOutputStream("f.txt");
7.   ObjectOutputStream out=new ObjectOutputStream(fout);
8.
9.   out.writeObject(s1);
10.  out.flush();
11.  System.out.println("success");
12.  }
13. }
success
```

download this example of serialization

Deserialization in java

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

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

1) public ObjectInputStream(InputStream in) throws IOException {}	creates an ObjectInputStream that reads from the specified InputStream.

Important Methods

Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	reads an object from the input stream.
2) public void close() throws IOException {}	closes ObjectInputStream.

Example of Java Deserialization

```
1. import java.io.*;
2. class Depersist{
3.  public static void main(String args[])throws Exception{
4.
5.   ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
6.   Student s=(Student)in.readObject();
7.   System.out.println(s.id+" "+s.name);
8.
9.   in.close();
10.  }
11. }
211 ravi
```

download this example of deserialization

Java Serialization with Inheritance (IS-A Relationship)

If a class implements serializable then all its sub classes will also be serializable. Let's see the example given below:

```
1. import java.io.Serializable;
2. class Person implements Serializable{
3.  int id;
4.  String name;
5.  Person(int id, String name) {
6.   this.id = id;
7.   this.name = name;
8.  }
9. }
1. class Student extends Person{
2.  String course;
3.  int fee;
4.  public Student(int id, String name, String course, int fee) {
5.   super(id,name);
6.   this.course=course;
7.   this.fee=fee;
8.  }
9. }
```

Now you can serialize the Student class object that extends the Person class which is Serializable.Parent class properties are inherited to subclasses so if parent class is Serializable, subclass would also be.

Java Serialization with Aggregation (HAS-A Relationship)

If a class has a reference of another class, all the references must be Serializable otherwise serialization process will not be performed. In such case, *NotSerializableException* is thrown at runtime.

```
class Address{
 String addressLine,city,state;
 public Address(String addressLine, String city, String state) {
  this.addressLine=addressLine;
  this.city=city;
  this.state=state;
 }
}
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 Address address;//HAS-A
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
```

Since Address is not Serializable, you can not serialize the instance of Student class.

Note: All the objects within an object must be Serializable.

Java Serialization with static data member

If there is any static data member in a class, it will not be serialized because static is the part of class not object.

```
class Employee implements Serializable{
 int id;
 String name;
 static String company="SSS IT Pvt Ltd";//it won't be serialized
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
```

Java Serialization with array or collection

Rule: In case of array or collection, all the objects of array or collection must be serializable. If any object is not serialiizable, serialization will be failed.

Externalizable in java

The Externalizable interface provides the facility of writing the state of an object into a byte stream in compress format. It is not a marker interface.

The Externalizable interface provides two methods:

- **public void writeExternal(ObjectOutput out) throws IOException**
- **public void readExternal(ObjectInput in) throws IOException**

Java Transient Keyword

If you don't want to serialize any data member of a class, you can mark it as transient.

Visit next page for more details.