By Peter V. Mikhalenko

February 14, 2007, 3:24pm PST

The serialization of objects in Java allows you to make a byte sequence from any object that has implemented the Serializable interface; it also allows you to turn that byte sequence back into an object. The mechanism does not depend on the operating system, which means you can transfer objects via your network and restore them at the other side of the wire.

With serialization, you can easily implement a so-called lightweight persistence, prolonging an object's life beyond the life of the application. The serialization mechanism has been added into the Java language for two reasons: (1) the JavaBeans mechanism uses serialization, and (2) remote method invocation (RMI) allows you to automatically use objects located at another host in the network just like any local objects.

**The Serializable interface**

You can easily serialize any object if it implements the Serializable interface. This interface does not contain any methods, so it's just a sign for the compiler and the Java Virtual Machine (JVM) that this class is serializable. When the serialization mechanism was added into the language, many standard Java classes (all primitive type wrappers like Integer and Double, all container classes, and Class class) were modified to support serialization.

Get developer tips in your inbox

Delivered each Thursday, our free Java newsletter provides insight and hands-on tips you need to unlock the full potential of this programming language.

**Automatically sign up today!**

In order to serialize an object, you need the output stream OutputStream, which must be put into the special serialization stream called ObjectOutputStream. After that, you only need to call the method writeObject() to serialize the object and send it to the output stream. In order to deserialize an object, you need to convert InputStream into ObjectInputStream and then call the readObject() method. As usual, you will get a reference to an Object type, so you'll also need to make a class cast to get an object of required type.

The serialization mechanism correctly handles all references contained in the object. All connected objects to which there are live references are also serialized, as well as all connected objects of all connected objects, etc. This is sometimes called objects web. Listing A shows how you can deserialize an object.

import java.io.\*;

public class Data implements Serializable { public static void main(String[] args) throws ClassNotFoundException, IOException { Data d = new Data(); ObjectOutputStream out = new ObjectOutputStream( new FileOutputStream("data.out")); out.writeObject("Data storage"); out.writeObject(d); out.close(); ObjectInputStream in = new ObjectInputStream( new FileInputStream("data.out")); String s = (String)in.readObject(); Data d2 = (Data)in.readObject(); }

}

After you create an ObjectOutputStream (based on another output stream), the writeObject() method writes an object into the stream. Note that I write the String object into the stream as well and then easily deserialize it. The JVM looks after object length, so don't care about object sizes and their structure.You can write objects into any stream; for instance, RMI writes an object to the network stream. Also note that no methods, including constructors, will be called at deserialization.

**Find the class**

Let's assume you transfer an object into another host. Can it really be used there based on the information received from the network? Yes, it can, but only if the JVM knows about the class of the object.

In Listing B, I read the serialized object from the file and then try to get information about its class with the getClass() method of an object. If a class is not contained in the classpath, you will get the ClassNotFoundException exception.

importjava.io.\*;

public class GetAlienClass { public static void main(String[] args) throws IOException, ClassNotFoundException { ObjectInputStream in = new ObjectInputStream( new FileInputStream("alien.obj")); Object obj = in.readObject();

// Here you will get ClassNotFoundException System.out.println(obj.getClass()); }

}

**The Externalizable interface**

There might be times when you have special requirements for the serialization of an object. For example, you may have some security-sensitive parts of the object, like passwords, which you do not want to keep and transfer somewhere. Or, it may be worthless to save a particular object referenced from the main object because its value will become worthless after restoring.

You can control the process of serialization by implementing the Externalizable interface instead of Serializable. This interface extends the original Serializable interface and adds writeExternal() and readExternal(). These two methods will automatically be called in your object's serialization and deserialization, allowing you to control the whole process.

There is one major difference between serialization and externalization: When you serialize an Externalizable object, a default constructor will be called automatically; only after that will the readExternal() method be called. Listing C shows how you can use externalization.

If you inherit some class from a class implementing the Externalizable interface, you must call writeExternal() and readExternal() methods when you serialize or deserialize this class in order to correctly save and restore the object.

**The "transient" keyword**

If you don't need to save and restore any member variable (e.g., the password kept in a String object), the private modifier will not help you. Serialized information can be read in a file or in a captured network packet. You may implement the Externalizable interface, which is demonstrated in the previous paragraph. In this case, nothing is written automatically, and you can control the entire process.

However, serializable objects are much more convenient because everything is serialized there automatically. You can forbid serialization of any member variable object with the transient modifier. It tells the JVM: "Do not save and restore this field, please; somebody else will take care of this field." Listing D shows how it looks.