SE 550: Lecture 5

Overview

Object serialization

XML Serialization

RMI and object serialization

What is RMI? What is a remote pointer? What is object marshalling and unmarshalling?

What is object serialization? What is an ObjectOutputStream and ObjectInputStream?

How does RMI use object serialization?

Object serialization

To serialize an object:

```
OutputStream out = ...;
MyObject anObject = ...;
ObjectOutputStream oOut = new ObjectOutputStream (out);
oOut.writeObject (anObject);
```

To unserialize an object:

```
InputStream in = ...;
ObjectInputStream oIn = new ObjectInputStream (in);
MyObject anObject = (MyObject)(oIn.readObject ());
```

Object serialization

Object output streams provide:

```
writeObject (Object);
writeInt (int);
writeBoolean (boolean);
...
```

Object input streams provide:

```
Object readObject ();
int readInt ();
boolean readBoolean ();
...
```

By some ‘magical process’, data is sent down the line.

Object serialization

Not all objects can be serialized. Why not?

Objects which can be serialized are called *serializable*.

Serializable objects are instances of classes:

```
class MyClass implements Serializable { ... }
```

Note that the Serializable interface is empty! This is a really bad hack!

Object serialization

What happens if we write:

```
class Unserializable {
  stuff you can’t send across the network
}
class OhDear implements Serializable {
  Unserializable stupid = new Unserializable ();
}
```

then try:

```
Serializable oh = new OhDear ();
oo.writeObject (oh);
```

Not one of the best bits of Java!

Object serialization

```
class Example implements Serializable {
  String sendMe = "hello";
  transient String dontSendMe = "world";
}
```

At the sender:

```
  Example test = new Example ();
  test.sendMe = "fred";
  test.dontSendMe = "wilma";
  oOut.writeObject (test);
```

What gets printed by the receiver:

```
  Example copy = (Example)(oIn.readObject ());
  System.out.println (copy.sendMe);
  System.out.println (copy.dontSendMe);
```

What are transient fields good for?

Object serialization

Sometimes you want more control over how an object is serialized.

For example, you might want to encrypt a password before saving it.

Java allows this by implementing the Externalizable interface:

```
public interface Externalizable extends Serializable {

   public void writeExternal(ObjectOutput out)
     throws IOException;

   public void readExternal(ObjectInput in)
     throws IOException, ClassNotFoundException;

}
```

How could we implement a serializable `User` class with a secret `password` field?

Object serialization

Under the hood, the objects are converted into a stream of bytes.

For most classes, this is easy:

- To serialize ints, booleans, etc. just send the binary data (doubles are a bit trickier but not much).
- To serialize an object, send the name of the class, then recursively serialize all the fields.

For example, what is sent when:

```
class Foo { String a = "AString"; int b = 37; }
class Bar { Foo c = new Foo (); }
oOut.writeObject (new Foo ());
oOut.writeObject (new Bar ());
```

Object serialization

What is sent by:

```
interface List { ... }
class Empty implements List { ... }
class Cons implements List { final Object hd; final List tl; ... }
List foo = new Cons ("hello", new Cons ("world", new Empty ()));
oOut.writeObject (foo);
```

What is sent by:

```
class Foo { Bar a; }
class Bar { Foo b; }
Foo foo = new Foo (); Bar bar = new Bar ();
foo.a = bar; bar.b = foo;
oOut.writeObject (foo);
```

Oops...

Object serialization

To deal with cyclic heap, we need a better algorithm:

- The first time an object is sent, we use the simple algorithm, but give the object a name called a *handle*.
- The second and later times the object is sent, we just send the handle.

With this improved algorithm, what is sent by:

```
class Foo { Bar a; }
class Bar { Foo b; }
Foo foo = new Foo (); Bar bar = new Bar ();
foo.a = bar; bar.b = foo;
oOut.writeObject (foo);
```

Object serialization

With this improved algorithm, what is sent by:

```
class MutableInteger { int contents = 0; }
MutableInteger i = new MutableInteger ();
oOut.writeObject (i);
i.contents = 37;
oOut.writeObject (i);
```

A good rule of thumb: *never serialize a mutable object*.

Exception 1: if the last thing you do with an object is serialize it, then you're OK.

Exception 2: if you know this is a new object stream (eg you just created it), then you're OK.

XML Serialization

What is document markup? What is SGML? HTML? XML?

What does an XML document look like?

What is SOAP? How does SOAP relate to HTTP? XML? RMI? Corba?

The SOAP home page: http://www.w3.org/TR/SOAP/. (Note which companies are working on it!)

XML Serialization

An example XML fragment:

```
<Object class="Cons" id="h">
  <int value="2"/>
  <String value="Hello"/>
  <Object class="Cons" id="w">
    <int value="1"/>
    <String value="World"/>
    <Object class="Empty">
    </Object>
  </Object>
</Object>
```

What is an XML element? An attribute? PCDATA?

What would a grammar for XML document fragments look like?

(For this lecture we are ignoring XML headers, DTDs, namespaces...)

XML Serialization

XML can be used as a replacement for Java object streams.

Example...

```
java ajeffrey.teaching.test.TestSoapServer
java ajeffrey.teaching.test.TestSoapClient fred wilma
```

These send and receive XML:

```
<Object class="ajeffrey.teaching.test.TestSoapCons" id="h">
  <int value="2"/>
  <String value="Hello"/>
  <Object class="ajeffrey.teaching.test.TestSoapCons" id="w">
    <int value="1"/>
    <String value="World"/>
    <Object class="ajeffrey.teaching.test.TestSoapEmpty">
    </Object>
  </Object>
</Object>
```

Uses a small fragment of SOAP (we'll call this mini-SOAP).

XML Serialization

How can we convert an object to an XML document and back?

```
public interface SoapWriter {
    public void serialize (Object obj) throws IOException;
    ...
}
public interface SoapReader {
    public Object unserialize () throws IOException;
    ...
}
```

What is *introspection* (also called *reflection*)? Why might it be useful?

XML Serialization

Code for converting a Java object to XML:

```
public void serialize (final Object obj) throws IOException {
    final Class objClass = obj.getClass ();
    final Field[] objFields = objClass.getFields ();
    try {
        out.println ("<Object class=\"" + objClass.getName () + "\">");
        ...serialize each field...
        out.println ("</Object>");
    } catch (final IllegalAccessException ex) {
        throw new SoapException ("Caught " + ex);
    }
}
```

What do getClass and getFields do?

XML Serialization

The loop to serialize the fields:

```
for (int i=0; i < objFields.length; i++) {
  final Class fieldType = objFields[i].getType ();
  final Object fieldContents = objFields[i].get (obj);
  if (fieldType.isPrimitive ()) {
    out.println ("<" + fieldType + " value=\"" + fieldContents + "\"/>");
  } else if (fieldContents instanceof String) {
    out.println ("<String value=\"" + fieldContents + "\"/>");
  } else {
    serialize (fieldContents);
  }
}
```

What do `getType`, `get` and `isPrimitive` do?

XML Serialization

A JavaCC grammar for a fragment of XML...

Tokens:

```
TOKEN : {
    <BEGINSTRING: "<String">
  | <BEGININT: "<int">
  | <BEGINOBJECT: "<Object">
  | <ENDOBJECT: "</Object>">
  | <SLASHGT: "/>">
  | <GT: ">">
  | <EQUALS: "=">
  | <CLASS: "class">
  | <VALUE: "value">
  | <LT: "<">
  | <SPACE: ([" ","\n","\r","\t"])+>
  | <STRING_LITERAL: "\"" (
        ~["\"","\\","\n","\r"] |
        "\\" ["n","t","b","r","f","\\","\'","\""]
      )* "\"">
}
```

XML Serialization

Parse an object:

```
Object object () throws ClassNotFoundException, IllegalAccessException, InstantiationExceptio
{
    String className; Object result;
}{
   <BEGINOBJECT> <SPACE> <CLASS> <EQUALS> className=stringLiteral() <GT>
    result=fields (className)
    <ENDOBJECT>
    { return result; }
}
```

XML Serialization

Parse the fields of an object:

```
Object fields (final String className) throws ClassNotFoundException, IllegalAccessException
{
    final Class objClass = Class.forName (className);
    final Object result = objClass.newInstance ();
    final Field[] fields = objClass.getFields ();
    int fieldNum = 0;
    Object tmpObject; String tmpString; int tmpInt;
}{
    (
        ( tmpObject = object () { fields[fieldNum++].set (result, tmpObject); } )
      | ( tmpString = string () { fields[fieldNum++].set (result, tmpString); } )
      | ( tmpInt = integer () { fields[fieldNum++].setInt (result, tmpInt); } )
      | <SPACE>
    )*
    {
        if (fieldNum == fields.length) { return result; }
        else { throw new SoapException ("Not enough fields for " + objClass); }
    }
}
```

XML Serialization

Some gotchas:

- The code only works with public classes and fields.
- The code requires a zero argument constructor.
- The code requires fields to be non-final.
- The code doesn’t work for cyclic heap (why not?)

Your homework is to fix the last problem!

XML Serialization

This is only a small fragment of SOAP!

The real thing:

- Contains HTTP header and footer information for RMI-like calls.
- Provides error handling functionality.
- Uses namespaces and DTDs ‘properly’.
- Is generally better.

But mini-SOAP contains many of the important features of SOAP serialization.

Summary

Object serialization transmits objects over a stream.

Object serialization uses object caching to cope with cyclic heap, so doesn’t work well with mutable objects.

One possible stream representation is XML, for example in SOAP.

Next week: midterm.

Week after that: RMI.