OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

CT326 Programming III

LECTURE 14-15

OBJECT SERIALIZATION

DR ADRIAN CLEAR
SCHOOL OF COMPUTER SCIENCE

Object Serialization

- Two streams in java.io - `ObjectInputStream` and `ObjectOutputStream` - are special in that they can read and write actual objects.
- The key to writing an object is to represent its state in a serialized form sufficient to reconstruct the object as it is read.
- Thus reading and writing objects is a process called object serialization.
- Object serialization can be useful in lots of application domains.

Uses of Object Serialization

- You can use object serialization in the following ways:
  - Remote Method Invocation (RMI) - communication between objects via sockets i.e. to pass various objects back and forth between the client and server.
  - Lightweight persistence - the archival of an object for use in a later invocation of the same program.
- As a Java programmer, you need to know about object serialization from two points of view:
  - How to serialize existing objects.
  - How to provide serialization for new classes.

Object Serialization in Practice

- Reconstructing an object from a stream requires that the object first be written to a stream.
- Writing objects to a stream is a straight-forward process.
- For example, the following code sample gets the current time in milliseconds by constructing a `Date` object and then serializes that object:
  - `ObjectOutputStream` is a processing stream, so it must be constructed on another stream.

Ollscoil na Gaillimhe
University of Galway

Object Serialization in Practice

```
FileOutputStream out = new FileOutputStream("theTime");
ObjectOutputStream s = new ObjectOutputStream(out);
s.writeObject("Today");
s.writeObject(new Date());
s.flush();
```

- This code constructs an `ObjectOutputStream` on a `FileOutputStream`, thereby serializing the object to a file named `theTime`.
- Next, the string `Today` and a `Date` object are written to the stream with the `writeObject` method of `ObjectOutputStream`.

Object Serialization with Related Objects

- If an object refers to other objects, then all of the objects that are reachable from the first must be written at the same time so as to maintain the relationships between them.
- Thus the `writeObject` method serializes the specified object, traverses its references to other objects recursively, and writes them all.
  - The `writeObject` method throws a `NotSerializableException` if it's given an object that is not serializable.
  - An object is serializable only if its class implements the `Serializable` interface.

Reconstructing serialized objects

- Reading from an `ObjectInputStream`
  - Once you've written objects and primitive data types to a stream, you'll likely want to read them out again and reconstruct the objects.
  - Here's code that reads in the `String` and the `Date` object that was written to the file named `theTime` in the last example:

```
FileInputStream in = new FileInputStream("theTime");
ObjectInputStream s = new ObjectInputStream(in);
String today = (String) s.readObject();
Date date = (Date) s.readObject();
```

  - Note that there’s no standard file extension for files that store serialized objects

Reconstructing serialized objects

- Like `ObjectOutputStream`, `ObjectInputStream` must be constructed on another stream.
- In this example, the objects were archived in a file, so the code constructs an `ObjectInputStream` on a `FileInputStream`.
- Next, the code uses `ObjectInputStream`'s `readObject` method to read the `String` and the `Date` objects from the file.
- The objects must be read from the stream in the same order in which they were written.

Reconstructing serialized objects

- Note that the return value from `readObject` is an object that is cast to and assigned to a specific type.
- The `readObject` method deserializes the next object in the stream and traverses its references to other objects recursively to deserialize all objects that are reachable from it.
- In this way, it maintains the relationships between the objects.
- The methods in `DataInput` parallel those defined in `DataOutput` for writing primitive data types.

Serializing classes

- Providing Serialization for Your Own Classes
  - An object is serializable only if its class implements the `Serializable` interface.
  - Thus, if you want to serialize the instances of one of your classes, the class must implement the `Serializable` interface.
  - The good news is that `Serializable` is an empty interface.
  - That is, it doesn't contain any method declarations; it's purpose is simply to identify classes whose objects are serializable.

The `Serializable` Interface

- Here's the complete definition of the `Serializable` interface:

```
package java.io;
    public interface Serializable {
        // there's nothing in here!
    };
```

- To make instances of your classes serializable, just add the `implements Serializable` clause to your class declaration.

A Serializable class

- Example of serializable class ...

```
public class MySerializableClass implements Serializable {
            ...
}
```

- You don't have to write any methods.
- The serialization of instances of this class are handled by the `defaultWriteObject` method of `ObjectOutputStream`.

Instance variables in Serializable classes

- All instance variables to be serialized must be serializable
- Referenced objects must be serializable, including those within referenced data structures
- In Java, all primitive type variables are serializable by default
- Can ignore instance variables in the process by declaring them as `transient`

The `defaultWriteObject` method

- This method automatically writes out everything required to reconstruct an instance of the class, including the following:
    - Class of the object
    - Class signature
    - Values of all non-transient and non-static members, including members that refer to other objects.
- For many classes, the default behaviour is fine.
- However, default serialization can be slow, and a class might want more explicit control over the serialization.

Customising Serialization

- You can customise serialization for your classes by providing two methods for it: `writeObject` and `readObject`.
- The `writeObject` method controls what information is saved and is typically used to append additional information to the stream.
- The `readObject` method either reads the information written by the corresponding `writeObject` method or can be used to update the state of the object after it has been restored.

Customising Serialization

- The `writeObject` and `readObject` methods must be declared exactly as shown in the following example.
- Also, it should call the stream's `defaultWriteObject` as the first thing it does to perform default serialization (any special arrangements can be handled afterwards):

```
private void writeObject(ObjectOutputStream s)
                                     throws IOException {
        s.defaultWriteObject();
        // customised serialization code
    }
```

Customising Serialization

- The `readObject` method must read in everything written by `writeObject` in the same order in which it was written.
- Here's the `readObject` method that corresponds to the `writeObject` method just shown:

```
    private void readObject(ObjectInputStream s)
                                    throws IOException  {
            s.defaultReadObject();
           // customised deserialization code
           // followed by code to update the object,
           //if necessary
    }
```

Customising Serialization

- Also, the `readObject` method can perform calculations or update the state of the object in some way.
- The `writeObject` and `readObject` methods are responsible for serializing only the immediate class.
- Any serialization required by the superclasses is handled automatically.
  - However, a class that needs to explicitly co-ordinate with its superclasses to serialize itself can do so by implementing the `Externalizable` interface.

Externalizable Interface

- For complete, explicit control of the serialization process, a class must implement the `Externalizable` interface.
- For `Externalizable` objects, only the identity of the object's class is automatically saved by the stream.
- The class is responsible for writing and reading its contents, and it must co-ordinate with its superclasses to do so.

Next time…

- Object Serialization demo