Java Programming 2 – Lecture #16 – Jeremy.Singer@glasgow.ac.uk

Writing to a text file

The `FileWriter`[1] is the standard library class used for generating text file output. To avoid constant byte-level filesystem access, these objects are generally wrapped in `BufferedWriter`[2] objects. The simple source code below writes a 'hello world' file in the current working directory. Note the use of a `try`-with-resources statement, which will automatically close the `BufferedWriter` object when it completes. Also note that the `newline()` method inserts the system-specific characters that encode a newline (different for DOS and Unix[3]).

```
public class HelloFile {
  public static void main(String [] args) {
    try(BufferedWriter bw =
              new BufferedWriter(new FileWriter("hello.txt"))) {
      bw.write("hello world");
      bw.newLine();
    }
    catch(IOException e) {
      e.printStackTrace();
    }
  }
}
```

Storing Objects in Files

Java *serialization*[4] is the process of writing Java objects as binary data, e.g. for transmission over a network socket or for saving to a file. Objects that can be serialized in this way implement the `Serializable`[5] *marker* interface. Most of the standard library classes are serializable. If you define a custom class, it can be serialized using the default serialization code, or you can define custom `writeObject()` and `readObject()` methods. Note that `transient` fields of objects are not serialized.

---

[1] See http://docs.oracle.com/javase/7/docs/api/java/io/FileWriter.html
[2] See http://docs.oracle.com/javase/7/docs/api/java/io/BufferedWriter.html
[3] See http://en.wikipedia.org/wiki/Newline#In_programming_languages
[4] See http://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html
[5] See http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html

Java Serialization for Object Output

We use an `ObjectOutputStream` instance to write out objects in a binary format.

```
String s = "save me!";
Calendar c = Calendar.getInstance();
Foo f = new Foo();
f.i = new Integer(42);
try(ObjectOutputStream oos =
 new ObjectOutputStream(new FileOutputStream("objects.tmp"))) {
    oos.writeObject(s);
    oos.writeObject(c);
    oos.writeObject(f);
}
catch (IOException e) {
    e.printStackTrace();
}
```

Reading Serialized Objects from a File

We use an `ObjectInputStream` instance to read objects back into memory from a serialized binary file. Objects must be read in the same order that they were written. The `readObject()` method returns an `Object` reference – this needs to be cast to the appropriate type to invoke class-specific behaviour. If the class is not defined on the current CLASSPATH, or the class versions are somehow inconsistent, then a `ClassNotFoundException`[6] is thrown.

```
try(ObjectInputStream ois =
       new ObjectInputStream(new FileInputStream("objects.tmp"))) {
  while(true) {
    Object o = ois.readObject();
    System.out.println("found object: " + o);
  }
}
catch (ClassNotFoundException e) {
  System.err.println("serialization error, wrong class version?");
  e.printStackTrace();
}
catch (IOException e) {
  e.printStackTrace();
}
```

Question

In what circumstances might you want to use serialization in your programs?

[6] See http://docs.oracle.com/javase/7/docs/api/java/lang/ClassNotFoundException.html