1.00 Lecture 31

Streams 2

Reading for next time: Big Java 18.6-18.8, 20.1-20.4

The 3 Flavors of Streams

In Java, you can read and write data to a file:

- **as text using `FileReader` and `FileWriter`**
- **as binary data using `DataInputStream` connected to a `FileInputStream` and as a `DataOutputStream` connected to a `FileOutputStream`**
- **as objects using an `ObjectInputStream` connected to a `FileInputStream` and as an `ObjectOutputStream` connected to a `FileOutputStream`**

Binary data files

	0	30	34	42
0	Jennifer Wang	198	5.	
42	Helen Smithson	198	5.	
84	Rashika Mathews	198	5.	
126	Ferd Johnson	198	5.	
168				

The file is just a stream of bytes on disk:

0	30	34	42	72	76	84	114	118	126	156	164
Jen..	1984	5.0	Hel..	1985	5.0	Ras.	1983	5.0	Ferd	1981	5.0

We can access this file at any place within it. It has a file pointer that indicates the position of the next byte to be read or written. It can be set by the `seek(int bytes)` method.

Unlike the text files we just read and wrote, we can't view binary files in a text editor and make any sense of them.

Students in binary files

```
import java.io.*;

public class RandomStudentFile {
   public static final int NAME_SIZE = 15;
   public static final int RECORD_SIZE = NAME_SIZE * 2 + 4 + 8;
   public static void main(String[] args) {
     Student[] team= new Student[4];
     // All Strings in Student MUST be of length NAME_SIZE   !!!!!!!!!!!!
     team[0]= new Student("Jennifer Wang  ", 1984, 5.0);
     team[1]= new Student("Helen Smithson ", 1985, 5.0);
     team[2]= new Student("Rashika Mathews", 1983, 5.0);
     team[3]= new Student("Ferd Johnson   ", 1981, 5.0);
     try {
       FileOutputStream f= new FileOutputStream("studentRandom.txt");
       DataOutputStream out= new DataOutputStream(f);
       writeData(team, out);
       out.close();
       RandomAccessFile in=new RandomAccessFile("studentRandom.txt","r");
       Student[] newTeam= readData(in);
       in.close();
       for (int i=0; i < newTeam.length; i++)
         System.out.println(newTeam[i]);
     } catch(IOException e) { System.out.println(e); }  }
```

Students in binary files, p.2

```
public static void writeData(Student[] s, DataOutputStream out)
            throws IOException {
    for (int i= 0; i < s.length; i++) {
            String name= s[i].getName();
            int year= s[i].getYear();
            double gpa= s[i].getGpa();
            out.writeChars(name);
            out.writeInt(year);
            out.writeDouble(gpa);
    }
}
public static Student[] readDataRecord(RandomAccessFile in)
            throws IOException {
    int n= (int) (in.length()/ RECORD_SIZE);
    Student[] s= new Student[n];
    for (int i= n-1; i >= 0; i--) {
            int j= (n-1) - i;      // Reverse order of students
            s[j]= new Student();
            in.seek(i*RECORD_SIZE);
            readDataFields(in, s[j]);
    }
    return s;
}
```

Students in binary files, p.3

```
public static void readDataFields(DataInput in,
             Student s) throws IOException {
     StringBuffer b = new StringBuffer(NAME_SIZE);
     for (int i = 0; i < NAME_SIZE; i++) {
            char ch = in.readChar();
            b.append(ch);
     }
     s.setName(b.toString());
     s.setYear(in.readInt());
     s.setGpa(in.readDouble());
 }
}
// Strings are immutable (implicitly final) in Java
// To mess with a string, we use a StringBuffer, which we can
// then change to a String when we're done. A bit weird.
```

Exercise 1

- **Download RandomStudentFile**
- **Questions:**
  - **In older languages we spent a lot of time counting and manipulating bytes. Were they the good old days? Or is this a big pain?**
  - **Why do you think we stack DataOutputStream on FileOutputStream?**

Exercise 2

- **Modify readDataRecord() to read the students in order 0, 2, 3, 1 (Wang, Mathews, Johnson, Smithson) into Team**
  - **Don't need to be fancy about implementing the order**
    - **Hint: Create int[ ] order= {0,2,3,1}; and a loop**
    - **Or just use 4 seek statements, one per student**
  - **Side note: closing the "last" stream (the one you read or write) closes all the other streams stacked with it**

Comment on Exercise 2

- **Comment:**
  - **Databases have replaced random access files in most current applications if sophisticated data manipulation is required.**
  - **If sophisticated data manipulation is not required, text files are used: much easier to debug, more portable**
  - **Binary files used to be common but shouldn't be used much these days**
  - **RandomAccessFile is the only Java stream that allows reading and writing; it's the exception!**

The 3 Flavors of Streams

In Java, you can read and write data to a file:

- **as text using `FileReader` and `FileWriter`**
- **as binary data using `DataInputStream` connected to a `FileInputStream` and as a `DataOutputStream` connected to a `FileOutputStream`**
- **as objects using an `ObjectInputStream` connected to a `FileInputStream` and as an `ObjectOutputStream` connected to a `FileOutputStream`**

Students in object files

```
import java.io.*;
public class ObjectStudentFile {
   public static void main(String[] args) {
     TA head= new TA("Elana M Wang", 1982, 5.0, 24.0);
     Student[] team= new Student[4];
     team[0]= head;
     team[1]= new Student("Helen Smithson", 1985, 5.0);
     team[2]= new Student("Rashika Mathews", 1983, 5.0);
     team[3]= new Student("Ferd Johnson", 1981, 5.0);
     try {
        FileOutputStream f= new FileOutputStream("studentObject.txt");
        ObjectOutputStream out= new ObjectOutputStream(f);
        out.writeObject(team);
        out.close();
        FileInputStream fin= new FileInputStream("studentObject.txt");
        ObjectInputStream in= new ObjectInputStream(fin);
        Student[] newTeam= (Student[]) in.readObject();
        in.close();
        for (int i=0; i < newTeam.length; i++)
                        System.out.println(newTeam[i]);
        } catch(IOException e) { System.out.println(e);
        } catch(ClassNotFoundException e) { System.out.println(e); }
} }
```

Class TA

```
public class TA extends Student {
   private double hours;
   public TA(String n, int y, double g, double h) {
        super(n, y, g);
        hours= h;
   }
   public String toString() {
        return super.toString() + "\t\t" + hours;
   }
}
```

Object Files

- **Class Student must implement Serializable**
  - **How many methods does this interface have?**
  - **Look in Javadoc, or select from the following options (more than one may be correct):**

    0 0 0 0 524
- **Class StudentObjectFile can read and write Student objects without using getXXX() and setXXX() methods, even though all Student data is private**
  - **Does this trouble you?**
  - **If you were a programmer on an MIT system with a Serializable Student object, could you set your gpa to 5.0 without using setGpa()? Could you set someone else's to 0.7?**
- **This is why most Java classes do NOT implement Serializable**
  - **There are security measures you must take if you do implement it**

Object Streams

- **Object streams preserve object structure**
  - **They are self-describing, which is a good thing**
  - **Your example reads and writes two kinds of objects, with different fields, without requiring you to know anything about their structure**
  - **Object streams are useful for communicating between Java programs or to restore/retrieve data into a Java program.**
    - **They are not a common archival or data storage format**
  - **It's easiest to store just one aggregate (array, array list, linked list, …) object in an object stream file**
    - **Otherwise it's messy to read the correct Object type from the file.**

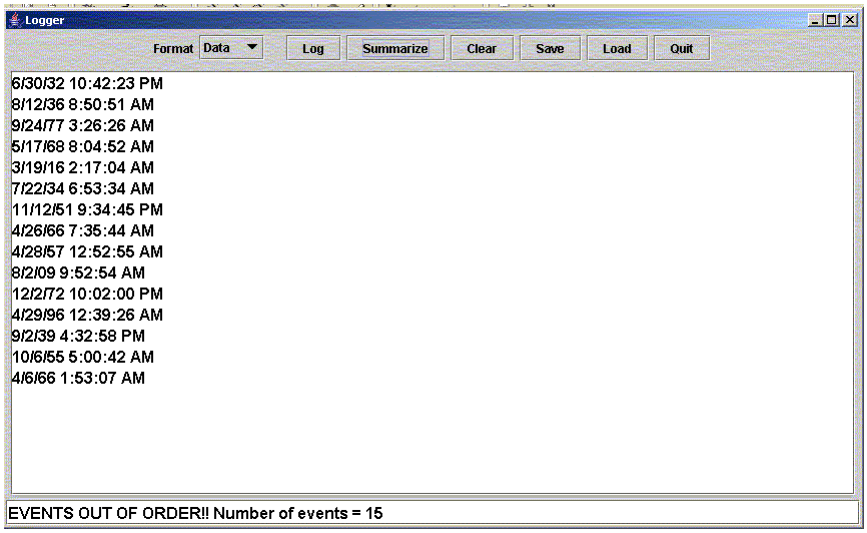
Exercise 3

- **Write a subclass of Student, GradStudent, with one more field: int yearsAtMIT**
  - **The current year field in Student is year of birth**
- **In main():**
  - **Create a GradStudent g, "Anthony Chung"**
  - **Dimension team to be size 5**
  - **Add g to the team as the fifth member**
  - **Write and read the team as before**
- **Could you handle multiple object types with a random access file?**
- **Could you handle multiple object types with a text file?**

Exercise 4

- **Download Logger and LoggerIO**
  - **Run Logger:**
    - **When you click on 'Log', Logger writes a date/time stamp into a Java List**
    - **Select a format (text, data, object) and save and load some log files.**
    - **Try to look at these files in Notepad, etc.**
  - **Set the type to data and open one of your text log files. What happens?**
    - **This is a type conversion error, very common in the C programming language, and somewhat in C++.**
    - **Java has sharply reduced the possibilities to do this, except when data is stored, which is when the worst of these mistakes occur… Oh well. It's still progress,**
    - **But… beware of data files! They don't have many safeguards for correctness. Use text files for simple things, databases for complex things, binary or object files rarely**

Exercise 4: Logger



Optional Exercise 5

- **If you're done with Exercise 4:**
  - **Download, read and run FindFolder**
    - **Example of managing files in folders**

Class FindFolder

```
import java.io.*;

public class FindFolder {
   public static void main(String[] args) {
        find("..");              // Find in parent of this directory
   }
   public static void find(String s) {
        try {
                File start = new File(s);
                String[] files = start.list();
                for (int i = 0; i < files.length; i++) {
                        File f = new File(start.getPath(), files[i]);
                        if (f.isDirectory()) {
                                String p = f.getPath();
                                System.out.println(p);
                                find(p);
                        }
                }
        } catch (Exception e) {
                System.out.println(e);
        }
   }
}
```