1.00 Lecture 24

Streams 2

Reading for next time: Phidgets documentation

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`**

Reading and writing sensor data

```
import java.io.*;

public class SensorReading implements Serializable {
   private static final long serialVersionUID = 1L;
   private String ID;
   private int time;
   private double reading;    // 0-1000

   public SensorReading() {}
   public SensorReading(String iD, int time, double reading) {
        ID = iD;
        this.time = time;
        this.reading = reading;
   }

   public String getID() { return ID;}
   public void setID(String iD) {ID = iD;}
   public int getTime() {return time;}
   public void setTime(int time) {this.time = time;}
   public double getReading() {return reading;}
   public void setReading(double reading){this.reading= reading;}
   public String toString() {return (ID+"\t"+time+"\t"+ reading);
}  }
```

Sensor data in text files

```
import java.io.*;
public class SensorFile {
   public static void main(String[] args) {
        SensorReading[] list= new SensorReading[4];
        list[0]= new SensorReading("S1", 1, 50.0);
        list[1]= new SensorReading("S22", 2, 70.0);
        list[2]= new SensorReading("S308", 4, 90.0);
        list[3]= new SensorReading("S4", 7, 0.0);
        try {   // Write data to text file
                FileWriter f= new FileWriter("sensor.txt");
                BufferedWriter b= new BufferedWriter(f);
                PrintWriter out= new PrintWriter(b);
                writeData(list, out);
                out.close();
                // Read data from text file
                FileReader fin= new FileReader("sensor.txt");
                BufferedReader in= new BufferedReader(fin);
                SensorReading[] newList= readData(in);
                in.close();
                for (int i=0; i < newList.length; i++)
                        System.out.println(newList[i]);
        } catch(IOException e) { System.out.println(e);}
   }
```

Sensor data in text files, p.2

```
public static void writeData(SensorReading[] s, PrintWriter out)
  throws IOException {
     out.println(s.length);        // Number of readings
     for (int i= 0; i < s.length; i++) {
            String id= s[i].getID();
            int time= s[i].getTime();
            double reading= s[i].getReading();
            out.println(id + "," + time + "," + reading);
     }
}

public static SensorReading[] readData(BufferedReader in)
  throws IOException {
     int n= Integer.parseInt(in.readLine());  // Nbr of readings
     SensorReading[] sArr= new SensorReading[n];
     for (int i=0; i < n; i++) {
            sArr[i]= new SensorReading();
            String str = in.readLine();
            String [] parts = str.split(",");
            sArr[i].setID(parts[0]);
            sArr[i].setTime(Integer.parseInt(parts[1]));
            sArr[i].setReading(Double.parseDouble(parts[2])
     }
     return sArr;
} }
```

Exercise 1

- **Download and run SensorFile**
  - **Look at sensor.txt in Wordpad or other editor**
- **Questions:**
  - **Does it still work if you use just a FileReader, not a Buffered Reader? Remove it and see.**
  - **Does it work with just FileWriter, not a PrintWriter? Remove it and see.**
  - **What would change if we didn't have the number of sensor readings as the first line of the file?**

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	8	12	20
0	S1	1	50.0	
20	S22	2	70.0	
40	S308	4	90.0	
60	S4	7	0.0	
				80

The file is just a stream of bytes on disk:

0	8	12	20	28	32	40	48	52	60	68	72
S1	1	50.0	S22	2	70.0	S308	4	90.0	S4	7	0.0

We can access this file at any place within it if we treat it as a `RandomAccessFile`. 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.

Sensor data in binary files

```
import java.io.*;
public class RandomSensorFile {
   public static final int ID_SIZE = 4;
   public static final int RECORD_SIZE = ID_SIZE * 2 + 4 + 8;
   public static void main(String[] args) {
        SensorReading[] list= new SensorReading[4];
        list[0]= new SensorReading("S1  ", 1, 50.0);
        list[1]= new SensorReading("S22 ", 2, 70.0);
        list[2]= new SensorReading("S308", 4, 90.0);
        list[3]= new SensorReading("S4  ", 7, 0.0);
        try {   // Write data to binary file
            FileOutputStream f= new FileOutputStream("sensorRandom.dat");
            DataOutputStream out= new DataOutputStream(f);
            writeData(list, out);
            out.close();
            // Read data from binary file
            RandomAccessFile in=
                        new RandomAccessFile("sensorRandom.dat","r");
            SensorReading[] newList= readData(in);
            in.close();
            for (int i=0; i < newList.length; i++)
               System.out.println(newList[i]);
        } catch(IOException e) {
               System.out.println(e);  }  }
```

Sensor data in binary files, p.2

```
    public static void writeData(SensorReading[] s,
                DataOutputStream out) throws IOException {
        for (int i= 0; i < s.length; i++) {
                String id= s[i].getID();
                int time= s[i].getTime();
                double reading= s[i].getReading();
                out.writeChars(id);
                out.writeInt(time);
                out.writeDouble(reading);
        }
    }

    public static SensorReading[] readData(RandomAccessFile in)
      throws IOException {
        int n= (int) (in.length()/ RECORD_SIZE);
        SensorReading[] s= new SensorReading[n];
        for (int i= n-1; i >= 0; i--) {
                int j= (n-1) - i;      // Reverse sensor readings
                s[j]= new SensorReading();
                in.seek(i*RECORD_SIZE);
                readDataFields(in, s[j]);
        }
        return s;
    }
```

Sensor data in binary files, p.3

```
    public static void readDataFields(DataInput in,
                SensorReading s) throws IOException {
        StringBuffer b = new StringBuffer(ID_SIZE);
        for (int i = 0; i < ID_SIZE; i++) {
                char ch = in.readChar();
                b.append(ch);
        }
        s.setID(b.toString());
        s.setTime(in.readInt());
        s.setReading(in.readDouble());
    }
}
// Strings are immutable (implicitly final) in Java
// To manipulate a string, we use a StringBuffer, which we can
// then change to a String when we're done.
```

Exercise 2

- Download RandomSensorFile
- Why do we stack DataOutputStream on FileOutputStream?
- Run RandomSensorFile
  - Try to open studentRandom.txt in an editor
  - What do you see?
- Remove the trailing spaces in the Strings in the SensorReading constructor calls (list[0]-list[3])
  - Run RandomSensorFile. What happens, and why?

Comment on Exercise 2

- Random access files allow reading and writing
  - We can rearrange and modify the data in the binary file arbitrarily by using seek, read ("r"), write ("w")
  - RandomAccessFile has "r", "w" and "rw" options
- 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.
  - Can be used for high performance and to save space

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`

Sensor readings in object files

```
import java.io.*;
public class ObjectSensorFile {
   public static void main(String[] args) {
        SensorReading[] list= new SensorReading[5];
        list[0]= new SensorReading("S1", 1, 50.0);
        list[1]= new SensorReading("S22", 2, 70.0);
        list[2]= new SensorReading("S308", 4, 90.0);
        list[3]= new SensorReading("S4", 7, 0.0);
        list[4]= new MagSensor("M1", 10, 53.0, 24.0);
        try {
            FileOutputStream f= new FileOutputStream("sensorObject.dat");
            ObjectOutputStream out= new ObjectOutputStream(f);
            out.writeObject(list);
            out.close();
            FileInputStream fin= new FileInputStream("sensorObject.dat");
            ObjectInputStream in= new ObjectInputStream(fin);
            SensorReading[] newList= (SensorReading[]) in.readObject();
            in.close();
            for (int i=0; i < newList.length; i++)
               System.out.println(newList[i]);
        } catch(IOException e) { System.out.println(e); }
          catch(ClassNotFoundException e) { System.out.println(e); }
}  }  // Doesn't use SensorReading get(), set(). What about security?
```

Class MagSensor

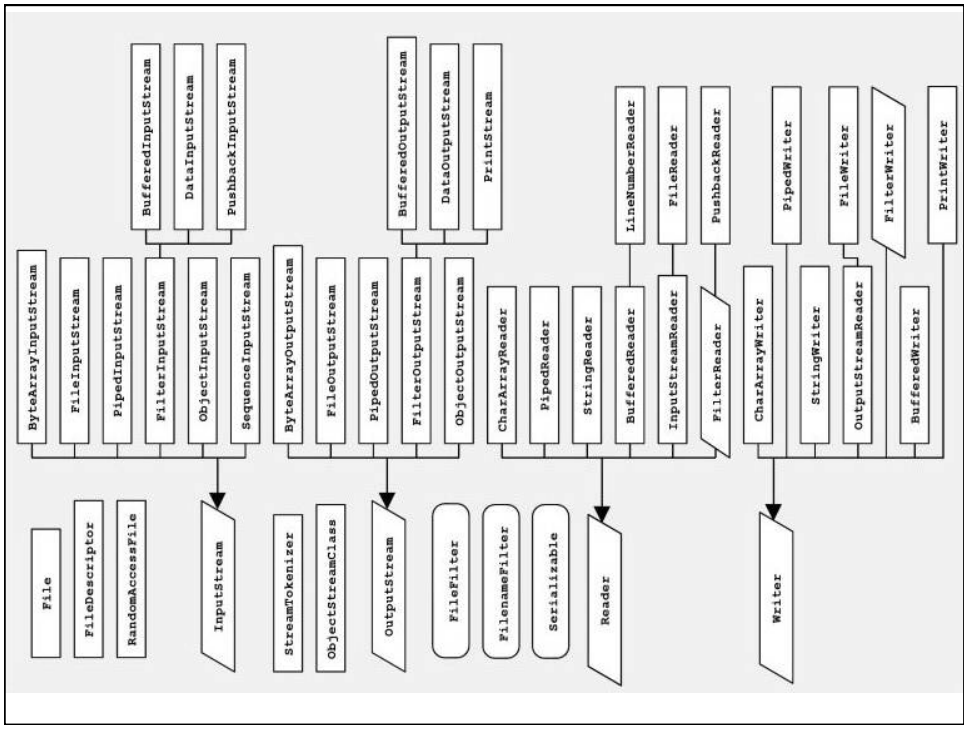
```
public class MagSensor extends SensorReading {
   private static final long serialVersionUID = 1L;
   private double magRead;

   public MagSensor(String iD, int time, double reading,
     double magRead) {
        super(iD, time, reading);
        this.magRead = magRead;
   }

   public String toString() {
        return super.toString() + "\t\t" + magRead;
   }
}
```

Object Streams

- Object streams preserve object structure
  - They are self-describing
  - 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 SensorReading, LightSensor, with one more field: int light
- In main():
  - Create a LightSensor s, "L11", time 3, reading 21, light 8
  - Dimension list to be size 6
  - Add s to the list
  - Write and read the list as before
- Could you handle multiple object types with a random access file?
- Could you handle multiple object types with a text file?



1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012