COMP 202 – File Access

CONTENTS:

- I/O streams
- Reading and writing text files

I/O Streams

- A stream is a sequence of bytes that flow from a source to a destination
- In a program, we read information from an input stream and write information to an output stream
- A program can manage multiple streams at a time
- The `java.io` package contains many classes that allow us to define various streams with specific characteristics

I/O Stream Categories

- The classes in the I/O package divide input and output streams into other categories
- An I/O stream is either a
  - *character stream*, which deals with **text** data
  - *byte stream*, which deal with byte (**binary**) data

Standard I/O

- There are three standard I/O streams
- The System class contains three object reference variables (in, out, err)
  - declared public and static (can be accessed via class name).
- `System.in`
  - standard input (typically keyboard)
  - we give `System.in` as input to `Scanner` constructor to read from keyboard
- `System.out`
  - standard output (typically a window on screen)
  - `println` is method of out output stream, thus to print to standard output we call `System.out.println`

The Standard Input Stream

- We've used the standard input stream to create a `Scanner` object to process input read interactively from the user:

```
Scanner scan = new Scanner (System.in);
```

- The `Scanner` object converts bytes from the stream into characters, and provides various methods to access those characters (by line, by word, by type, etc.)

Reading Text Files

- We can read a text file sequentially using the file as the input stream for our scanner object:
  - Assume you have a file: test.txt
  - Alternative 1:
    - Construct a `FileReader` object and use that as input for the `Scanner` constructor
    - `FileReader reader = new FileReader("test.txt");`
    - `Scanner scan = new Scanner(reader);`
  - Alternative 2:
    - Construct a `File` object and use that as input for the `Scanner` constructor
    - `Scanner scan = new Scanner(new File("test.txt"));`
- You can then use the `Scanner` operators (`next, nextLine, nextInt, …`) to read the file sequentially.

Reading Text Files

- Many things can go wrong with file access
  - many operations require *checked* exceptions
- All file related access must be enclosed in a try/catch block

```
try {
   FileWriter fout = new FileWriter("foo.txt");
   ...
} catch (java.io.IOException ioe) {
   System.out.println("Can't write to foo.txt");
}
```

- or the method must indicate that it can "throw" a `java.io.IOException`

```
public String readWord() throws IOException {
   FileWriter fin = new FileWriter("foo.txt");
   ...
}
```

Line Numbering

```
// reprint lines in input file and prefix them with line numbers
import java.io.*;
import java.util.Scanner;
public class LineNumberer {
   public static void main (String[] args)
   {
      int lineNumber = 1;  // line number initialized to 1
      try {

         Scanner fileScan = new Scanner (new File("test.txt"));
         // reprint every line with number prefixed
         while (fileScan.hasNextLine())
         {
            System.out.println("/* " + (lineNumber++)
                                 + " */" + fileScan.nextLine());
         }
      } // you must catch the exception
      catch (IOException ex)
      {
         System.out.println("Error processing file: " + ex);
      }
   }
}
```

Writing to a Text File

- To write output to a file, construct a `PrintWrite` object the given file name
  - `PrintWriter out = new PrintWriter("output.txt");`
  - If the output file already exists, it is emptied before the new data is written into it (overwrite)
  - if it doesn't exist, it will be created.
- Use the `print` and `println` methods to send numbers, objects an strings to a `PrintWriter` object
  - `out.print(29.95 + "\t");`
  - `out.println("Hello World");`
- When you are done writing to a file close the corresponding `PrintWriter`.
  - `out.close();`

Line Numberer 2:

```
// reprint lines in input file and prefix them with line numbers
import java.io.*;
import java.util.Scanner;
public class LineNumberer2 {
   public static void main (String[] args) throws IOException {
      int lineNumber = 1;  // line number initialized to 1
      String inputFile = "test.txt";      // default input
      String outputFile = "output.txt";   // default output

      // input and output could be given through command line
      if (args.length >= 1)
         inputFile = args[0];
      if (args.length >= 2)
         outputFile = args[1];

      Scanner fileScan = new Scanner (new File(inputFile));
      PrintWriter out = new PrintWriter(outputFile);
      // output every line with number prefixed
      while (fileScan.hasNextLine()) {
         out.println("/* " + lineNumber++ + " */" + fileScan.nextLine());
      }
      // close the file
      out.close();
   }
}
```

Processing structured text files

- Scanner
  - The usual delimiter for next of Scanner class is " " (whitespace)
    - other delimiters can be chosen (e.g., ";")
    - `Scanner.useDelimiter(String pattern)`
  - Useful for reading data separated by other delimiters than whitespace
    - actual pattern can be quite complex (but we won't cover it)
- Writing to a file
  - By using first a
    - `FileWriter` (provides only minimum support to write to a file)
    - and then a `BufferedWriter` (writes data to disk in chunks)
    - and then a `PrintWriter`
  - data is written in a more efficient way to the disk
  - this will speed up your program if you write large amounts of data

MyWorld.java

```
public class MyWorld {
   public static void main (String[] args)  {
      final int MAX = 200;
      Country[] myWorld = new Country[MAX];
      String line, name, fileName="countries.dat";
      int count = 0;  long population, area;

      try {
         Scanner fileScan = new Scanner (new File(fileName));
         while (fileScan.hasNext()) {
            line = fileScan.nextLine();
            Scanner lineScan = new Scanner(line).useDelimiter(";");
            try {
               name = lineScan.next();
               population = lineScan.nextLong();
               area = lineScan.nextLong();
               myWorld[count++] = new Country (name, population, area);
            } catch (NoSuchElementException exception) {
               System.out.println ("Error in input. Line ignored:");
               System.out.println (line);
            }
         }
         for (int i = 0; i < count; i++)
            System.out.println (myWorld[i]);
      } catch (FileNotFoundException exception) {
         System.out.println ("The file " + fileName + " was not found.");
      }
```

MyWorld.java

```
        // ....
        String outName = "world.dat";
        // write out all countries to outName
        try {
           for (int i = 0; i < count; i++)
              myWorld[i].addToFile(outName);
        } catch (IOException ioe) {
           System.out.println ("Can't write to " + outName + ": " +
                                ioe);
           System.out.println ("Giving up...");
        }
     }
  }
```

Country.java

```
public class Country {
   private String name;
   private long population; // number of people
   private long area;       // geographical area

   public Country (String countryName, long numPeople, long size) {
      name = countryName;
      population = numPeople;
      area = size;
   }
   public String toString() {
      return name + ": " + population + " people on " + area + " sqKms is "
             + (population / area) + " people per sqkm";
   }
   public void addToFile(String fileName) throws IOException {
      FileWriter fw = new FileWriter (fileName,true); // true for appending
      BufferedWriter bw = new BufferedWriter (fw);
      PrintWriter outFile = new PrintWriter (bw);

      outFile.println (name + ";" + population + ";" + area);

      outFile.close();
   }
}
```

Advanced Topics

- There are many other ways to read and write to files:
  - random access:
    - access arbitrary location of file directly
    - contrast to sequential access
  - read/write byte streams
    - *binary* data, more compressed
  - Object streams (*serialization*)
    - write out entire (aggregate) objects to a file
    - read in an entire (aggregate) object from a file
- We do not discuss these advanced topics in class but they are all very useful in specific situations. A good reference to study all kinds of IO
  - http://java.sun.com/docs/books/tutorial/essential/io/

When to use files…

- In any application that requires information to exist for a long time:
  - Experimental data
  - Music
  - Databases
- Computer memory is volatile, meaning that once the computer is shut off the information is gone.
- Information stored in files can remain available even after the computer is turned off.

Designing For Files

- Exception Checking
  - Input validation checking
  - Hardware problems checking
    - No floppy disk or storage device present
    - Sector error on diskette or device
- Input-Process-Save Methodology
  - Get input from user
  - Validate user input
  - Save it to a file

ProductCodes.java

```
import java.util.Scanner;
public class ProductCodes {
   public static void main (String[] args) {
      String code; char zone;
      int district, valid = 0, banned = 0;
      Scanner scan = new Scanner(System.in);
      System.out.print ("Enter product code (XXX to quit): ");
      code = scan.nextLine();

      while (!code.equals ("XXX")) {
         try {
            zone = code.charAt(9);
            district = Integer.parseInt(code.substring(3, 7));
            valid++;
            if (zone == 'R' && district > 2000)
               banned++;
         } catch (StringIndexOutOfBoundsException exception) {
            System.out.println ("Improper code length: " + code);
         } catch (NumberFormatException exception) {
            System.out.println ("District is not numeric: " + code);
         }
         System.out.print ("Enter product code (XXX to quit): ");
         code = scan.nextLine();
      }
      System.out.println ("# of valid codes entered: " + valid);
      System.out.println ("# of banned codes entered: " + banned);
   }
}
```