CS1007 lecture #22 notes

tue 23 apr 2002

- news
- command line interface and menu processing
- input and output, aka I/O
- StringTokenizer
- files
- reading: ch 8, especially 8.2-8.4

---

news.

- homework #6 due Thu May 2
- final exam:
  - Tue May 14 9am-12noon (AM class, section 002)
  - Thu May 16 1pm-4pm (PM class, section 001)

---

input and output.

- we've drawn a picture of input and output many times this semester:

input $\rightarrow$ CPU $\rightarrow$ output

- up to now, input has been from the keyboard and output has been to the screen
- today we will read input from "text files" and write output to "text files"

---

files.

- we'll talk about two kinds of files:
  - text files, like *.java files, *.html files, etc
  - binary files, like *.class files
- you can test the file type by entering:

  unix$ more <filename>

- and if your file is not a text file, you'll get back an error like this:

  ******** <filename>: Not a text file ********

java.io package.

- typically, there are three processing steps when using files:
  1. open
  2. read, write or update
  3. close
- we'll only talk about read and write in Java.
- in order to implement file I/O in Java, we need several classes from the java.io package:
  – FileReader
  – FileWriter
  – BufferedReader
  – BufferedWriter
  – PrintWriter

---

programs that work with data files.

- the simple model for programs that work with data files is to:
  1. open the data file for reading
  2. read the contents of the data file into program variables
  3. close the data file
  4. manipulate the values in the program variables
  5. open the data file for writing
  6. write the manipulated values to the data file
  7. close the data file
- we'll use the Inventory example in the textbook (listing 8.7) and expand on that

---

InvItem class (based on listing 8.8 from the textbook).

```java
public class InvItem {

    private String name;
    private int    units;
    private float  price;

public InvItem( String itemName, int numUnits, float cost ) {
    name = itemName;
    units = numUnits;
    price = cost;
} // end of InvItem constructor

public String getName() {
    return( name );
} // end of getName() method
```

```java
public int getUnits() {
    return( units );
} // end of getUnits() method

public float getPrice() {
    return( price );
} // end of getPrice() method

public void setPrice( float new_price ) {
    price = new_price;
} // end of setPrice() method

} // end of InvItem class
```

Inventory class (based on listings 8.7 and 8.9 from the textbook).

```java
import java.util.*;
import java.io.*;

public class Inventory {

    public static void main( String[] args ) {

        final int MAX = 100;
        InvItem[] items = new InvItem[MAX];
        StringTokenizer tokenizer;
        String line, name, filename="inventory.dat";
        int units, count=0;
        float price;
```

```java
        // read data from file into program variables
        try {
            FileReader fr = new FileReader( filename );
            BufferedReader infile = new BufferedReader( fr );
            line = infile.readLine();
            while( line != null ) {
                tokenizer = new StringTokenizer( line );
                name = tokenizer.nextToken();
                try {
                    units = Integer.parseInt( tokenizer.nextToken() );
                    price = Float.parseFloat( tokenizer.nextToken() );
                    items[count++] = new InvItem( name,units,price );
                }
                catch( NumberFormatException nfx ) {
                    System.out.println( "error in input; line ignored: " +
                                        line );
                }
                line = infile.readLine();
            } // end of while
            infile.close();
```

```java
        }
        catch( FileNotFoundException fnfx ) {
            System.out.println( "file not found: " + filename );
        }
        catch( IOException iox ) {
            System.out.println( iox );
        }
    }
```

```java
        // manipulate the values of the program variables:
        // increase the prices by 10%
        for ( int i=0; i<count; i++ ) {
            items[i].setPrice( items[i].getPrice()*1.10 );
        } // end for i

        // display the values of the program variables on the screen
        for ( int i=0; i<count; i++ ) {
            System.out.println( items[i].getName() +
                                " number of items = " +
                                items[i].getUnits() +
                                " price = $" + items[i].getPrice() );
        } // end for i
```

```
// write data from program variables into data file
try {
  FileWriter fw = new FileWriter( filename );
  PrintWriter outfile =
        new PrintWriter( new BufferedWriter( fw ) );
  for ( int i=0; i<count; i++ ) {
    outfile.println( items[i].getName() + " " +
                     items[i].getUnits() + " " +
                     items[i].getPrice() );
  } // end for i
  outfile.close();
}
catch( FileNotFoundException fnfx ) {
  System.out.println( "file not found: " + filename );
}
catch( IOException iox ) {
  System.out.println( iox );
}
} // end of main() method
} // end of Inventory class
```

StringTokenizer

- used to break up a string into "tokens", i.e. components
- each token is separated by a "delimiter"
- default delimiter is whitespace
- but you can set another value for delimiter
- primary method used: `public String nextToken();`

---

example run:

- data file (inventory.dat) =

```
Widget 14 3.35
Spoke 132 0.32
Wrap 58 1.92
Thing 28 4.17
```

- first time through the while loop in the main() method of Inventory:

line = "Widget 14 3.35"

name = "Widget"

unit $\to$ Integer.parseInt("14") = 14

price $\to$ Float.parseFloat("3.35") = 3.35

---

command-line interface and menu processing.

- we can draw a flow chart to handle typical database processing operations.
- these are:
  - add data
  - edit data
  - remove data
  - print data
- typically, a program begins by reading the data from a data file – for your homework, this will be a text data file – into variables inside the program (e.g., Vector or array).
- then the program will manipulate the data in the Vector(s)/array(s), until the user is ready to quit
- finally, the program will write the manipulated data back to the data file

menu processing flow chart.

- here's another flow chart:

read data file

display menu

get user's input

add → ask for data → read user's data

edit

remove → ask for key → find key → remove data

print

quit → write data file → exit program