CS1007 lecture #20 notes

tue 19 nov 2002

- news
- streams
- files
- `java.io` package
- exceptions
- `StringTokenizer`
- formatting output
- reading: ch 10-11

streams (1).

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

  input $\rightarrow$ $\boxed{\text{CPU}}$ $\rightarrow$ 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"
- input and output flow from and to *streams*
  - a *stream* is an ordered sequence of bytes
  - streams flow from a source to a destination
  - with input, the source is the keyboard and the destination is a program
  - with output, the source is a program and the destination is the screen

streams (2).

- thus there are two categories of streams:
  - *input streams*
  - *output streams*
- streams can also be subdivided based on their content:
  - *character streams* (i.e., text)
  - *byte streams* (i.e., binary data)
- or their usage:
  - *data streams* (e.g., String in memory, file on disk)
  - *processing streams* (manipulation of a data stream)

streams (3).

- in order to handle streams in Java, we need several classes from the `java.io package`:
- classes that handle *byte streams*
  - `InputStream` $\leftarrow$ `FileInputStream`
  - `OutputStream` $\leftarrow$ `FileOutputStream` $\leftarrow$ `PrintStream`
- classes that handle *character streams*
  - `Reader` $\leftarrow$ `BufferedReader`
  - `Writer` $\leftarrow$ `BufferedWriter`
- for example, in `java.lang.System`:
  - `System.in` is an `InputStream`
  - `System.out` is a `PrintStream`

example.

- `Keyboard.java`

files (1).

- 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 ********
```

files (2).

- 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`

files (3).

- 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

exception handling.

- example:

```
try {
  i = System.in.read();
}
catch ( IOException iox ) {
  System.out.println( "there was an error: " + iox );
}
```

- `try` clause contains code which may generate an exception, i.e., an error
- `catch` clause contains code to execute in case the error happens; i.e., where to go if the exception gets *caught*

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();`

formatting output.

- `java.text.DecimalFormat` class
- used to format decimal numbers
- construct an object that handles a format
- use that format to output decimal numbers
- methods include:
  - `DecimalFormat( String pattern );`
  - `void applyPattern( String pattern );`
  - `String format( double number );`
- formatting patterns include:
  - 0 used to indicate that a digit should be printed, or 0 if there is no digit in the number (i.e., leading and trailing zeros)
  - # used to indicate that if there is a digit in the number, then it should be printed; indicates rounding if used to the right of the decimal point
  - e.g., `DecimalFormat fmt = new DecimalFormat( "#.00" );`

example code.

- `InvItem.java`
- `Inventory.java`

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 $\rightarrow$ `Integer.parseInt( "14" ) = 14`

  price $\rightarrow$ `Float.parseFloat( "3.35" ) = 3.35`