File Reading Basics, Token-Based, Line-Based & Advanced File Processing

CSCI 161 – Introduction to Programming I

Professor Thomas Rogers

Overview

- Reading: Chapter 6 - File Processing
- Topics:
  - File Reading Basics
  - Token-Based Processing
  - Line-Based Processing
  - Advanced File Processing

File Reading Basics

- **File** - A collection of information that is stored on a computer and assigned a particular name.

- Files can be used for storing:
  - program code
  - compiled programs
  - audio, image, video
  - documents
  - etc.

File Reading Basics (continued)

- Common file extensions:

Extension	**Description**
.txt	text file
.java	Java source code file
.class	compiled Java bytecode file
.doc	Microsoft Word file
.xls	Microsoft Excel file
.pdf	Adobe Portable Document File
.mp3	audio file
.jpg	image file
.zip	compressed archive
.html	hypertext markup language file (web page)
.exe	executable file

File Reading Basics (continued)

- **File** object
  - import java.io.*

    ```
    import java.io.*; // for File
    ```

- **Important:** Must include the import statement above before you can use the file object.

File Reading Basics (continued)

- Many useful methods of the File object:

Method	Description
canRead()	Whether or not this file exists and can be read
delete()	Deletes the given file
exists()	Whether or not this file exists on the system
getAbsolutePath()	The full path where this file is located
getName()	The name of the file as a string without directory attached
isDirectory()	Whether this file represents a directory/folder on the system
isFile()	Whether this file represents a file (nonfolder) on the system
length()	The number of characters in this file (size of file)
renameTo()	Changes this file's name to the given file's name

File Reading Basics (continued)

- **Scanner** object reading from file
  - **CountWords** - Example that counts words in hamlet.txt file.
  - **Checked Exception** - An exception that must be caught or specifically declared in the header of the method that might generate it.
  - **throws Clause** - A declaration that a method will not attempt to handle a particular type of exception.
  - **Common Programming Error** - Reading beyond end-of-file. A *NoSuchElementException* will result if trying to read past the end of file (no more tokens lines). Moral know your file format use a conditional with *hasNext...()* if not sure.

Token-Based Processing

- Chapter 3 review - **nextInt()** - get next integer **nextDouble()** - get next double **next()** - get next token as a string
- **ShowSum1** example from book - This example is much like our Lab examples where the user is asked the number of items in the series.
- **ShowSum2** example from book - Reads from a file processing all doubles within (not a specific known number of items).
- **Common Programming Error** - Reading the wrong token. An *InputMismatchException* will result if trying to read the wrong type of token. Moral know your file format use a conditional with *hasNext...()* if not sure.
- **Common Programming Error** - Forgetting *new File(...)*. If you forget to include wrap the filename in a new File object constructor then the Scanner object takes the filename as a literal string and scans that.

Line-Based Processing

- **Line-Based Processing** - The practice of processing input line-by-line (i.e. reading in entire lines of input at a time). This practice usually requires additional processing of each line seperately.
  - **nextLine()** - Use *nextLine()* to get and *hasNextLine()* to check if another line is available.
  - **HoursWorked2** - Example that combines line reading and token scanning.

Advanced File Processing

File output and guaranteeing a file may be read

- **PrintStream** object - Used for file output
  - Declare a PrintStream object constructed with a new File object specifying the output file:

    ```
    PrintStream output = new PrintStream(new File("output.txt"));
    ```

  - Then instead of using *System.out* methods use the same like-named methods of the PrintStream class object

    ```
    output.println("Hello world!");
    ```

  - **HoursWorked3** - Example Reads hours from a file and outputs summary to the screen an an output file.

Advanced File Processing (continued)

- **Guaranteeing that Files can be read** - *canRead()* method
  - **CountWords2** - Example counts words in input file entered by user.
  - Note the use of **getInput** method for getting an input file name. That is a good *boilerplate* method.
  - **Boilerplate Code** - Code that tends to be the same from one program to another.

Advanced File Processing (continued)

- **File Paths and Directories:**
  - **File Path** - A description of a file's location on a computer starting with a drive (or volume) and including the path from the root directory to the directory where the file is stored.
  - **Current Directory (a.k.a. Working Directory)** - The directory that Java uses as the default when a program uses a simple file name (sans file path).

```
Scanner input = new Scanner(new File(
                 "C:/My Documents/MyData/data.txt"));
```

  - **Relative Path** - A path that is relative to the Current Directory typically starts with ./

```
Scanner input = new Scanner(new File("./data/hours.dat"));
```