Lesson 59: File I/O (W20D1)

Balboa High School

Michael Ferraro

January 11, 2016

Do Now

3–4min: Read over §§11.2–11.3 from PS #10, the `FileRewinder` and `Animals` exercises.

Aim

Students will learn the structure of text files and the basics of File I/O in Java.

File I/O

- **Input:** Reading from files

File I/O

- **Input:** Reading from files
- **Output:** Writing to files

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Text Files**

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Text Files**
    - contain human-readable characters (i.e., those that can be typed using a keyboard)

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Text Files**
    - contain human-readable characters (i.e., those that can be typed using a keyboard)
    - may be opened with a text editor

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Text Files**
    - contain human-readable characters (i.e., those that can be typed using a keyboard)
    - may be opened with a text editor
    - may be read/edited by programs and people

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Binary Files**

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Binary Files**
    - are not human-readable, though special editors can be used to interpret the content

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Binary Files**
    - are not human-readable, though special editors can be used to interpret the content
    - are typically read and used by programs

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
  - **Binary Files**
    - are not human-readable, though special editors can be used to interpret the content
    - are typically read and used by programs
    - e.g., MS Word documents are saved in a specific binary format

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
- Most programming languages provide mechanisms for these text file operations (and more for binary files):

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
- Most programming languages provide mechanisms for these text file operations (and more for binary files):
  - CREATE

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
- Most programming languages provide mechanisms for these text file operations (and more for binary files):
  - CREATE
  - OPEN for reading/writing/appending

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
- Most programming languages provide mechanisms for these text file operations (and more for binary files):
  - CREATE
  - OPEN for reading/writing/appending
  - CLOSE

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
- Most programming languages provide mechanisms for these text file operations (and more for binary files):
  - CREATE
  - OPEN for reading/writing/appending
  - CLOSE
  - READ (a character, a line, the whole file)

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
- Most programming languages provide mechanisms for these text file operations (and more for binary files):
  - CREATE
  - OPEN for reading/writing/appending
  - CLOSE
  - READ (a character, a line, the whole file)
  - WRITE (a character, a line, multiple lines)

File I/O

- **Input:** Reading from files
- **Output:** Writing to files
- Two types of files
- Most programming languages provide mechanisms for these text file operations (and more for binary files):
  - CREATE
  - OPEN for reading/writing/appending
  - CLOSE
  - READ (a character, a line, the whole file)
  - WRITE (a character, a line, multiple lines)
  - APPEND, or add to the end of a file

Text File I/O in Java

Litvin §§15.1–15.4 provide a fairly complete overview of working with text files (**highly recommended reading!**).

Composition of a Text File

```
Hello, this is my text file.<EOL>
<EOL>
Eating grapes makes me happy.<EOL>
I like apples, too.<EOL>
<EOF>
```

- `EOL`: End-of-line marker
  - `\n` on *nix systems
  - `\r` on Apple systems[1]
  - `\r\n` on Windows/DOS systems
- `EOF`: End-of-file marker; how a program can tell it has reached the end

[1] `\r` is *carriage return*

Text File Operations: Basics

- You cannot read and write to a file simultaneously! The JVM will ask the OS to open a file for reading OR writing, not both.
- If you write to a filename that doesn't exist, a new, empty file will be created.
- If you read from a non-existent file, an exception will be thrown.
- When you open a *file handle* for reading/writing, you must close the handle when you're finished.
- These concepts are *portable*[2] to many other languages!

[2] I.e., these ideas are valid for use with other languages, like BASIC, PERL, PHP, C, etc.

Example #1: Reading from a Text File

1. Create a new project: `FileIO`.
2. Import `Ex1FileReader.java` from here.
3. Download `blah.txt` from the same directory as above, and save in project `FileIO`'s working directory, `~/MOUNTED/apcs-locker/workspace2/FileIO` [3]
4. Try to fix the error! (You need to catch an exception.)
5. Once fixed, run the program.
6. Works? Now rename `blah.txt` to some other name and run again.

[3] The working directory is where Eclipse will look for files we try to open in case we don't specify a directory.

Example #2: Writing to a New Text File

1. Import `Ex2FileCreator.java` from here. **DO NOT RUN YET!**
2. Read through the source code. Note that the `PrintWriter` class supplies a `println()` method.
3. Use Nautilus to view the contents of the `FileIO` folder.
4. Run `Ex2FileCreator`. Was an output file generated? (See Nautilus window.)
5. Open up the CSV[4] file in a text editor, make changes, and save.
6. Re-run `Ex2FileCreator`. Open the CSV file. What do you notice?

continued on next slide →

[4] comma-separated values; very common way to export spreadsheet and database data to a text file for reading/loading elsewhere.

Example #2: Writing to a New Text File

7. Make the file read-only: Right-click on the file in Nautilus, *properties* → *permissions* tab → uncheck *write* for file owner.
8. What happens when you re-run the program?
9. Start LibreOffice Calc (≈MS Excel).
    1. Open the CSV file that was generated. Tell Calc that the text in the file is a set of comma-separated values [5]
    2. Select columns A & B, click `Insert`→`Chart`, and create an $x - y$ scatterplot.

[5] '\t', or *tab*, characters are also valid for separating columns of data!

Example #3: Appending to a Text File

- Appending to a file is comparable to *concatenating* to an existing `String`.
- Process is slightly different:
  1. Create a `Writer` object using a `FileWriter` constructor.
  2. Catch the `IOException` that the `FileWriter` constructor may throw.
  3. Create a `PrintWriter` object, sending the `Writer` you created to its constructor (no need to catch a `FileNotFoundException` this time).
  4. Use the `PrintWriter` object's methods — e.g., `println()` — to send data to the `Writer`, which in turns appends the data to the text file.
  5. `close()` the `PrintWriter`.

Example #3: Appending to a Text File

1. Import `Ex3FileAppender.java` from here. **DO NOT RUN YET!**
2. Examine its contents, taking note of how the process on the last slide is implemented.
3. View the contents of `output.csv` PRIOR to running the class.
4. Run `Ex3FileAppender`.
5. See how the contents of `output.csv` have changed. As before, have Calc graph the contents as an $x - y$ scatterplot.

Challenge: Prepending a File

Write a class that takes the data from `output.csv` and prepends rows to its data so that, when graphed, there's a symmetrical parabola (i.e., the domain is $-9 \leq x \leq 9$).

You should write the resulting data set to a different file.

If time allows. . .

Live demonstration of how another language handles File I/O: PERL

HW

- You should already be done with most of PS #10, §§1-10, inclusive
- Today's lesson and the textbook reading should allow you to complete the remaining sections.
- Be mindful of the due date and don't procrastinate!