Lab Exercise 3
CS 2334

September 10, 2015

Introduction

There are many reasons to import a file. Once a file is imported, it can be analyzed, it can be presented, it can be migrated into another application, and it can be sent to another person or computer. This is a useful skill that will be used for the rest of your academic and professional career as a computer scientist.

In this laboratory, we will import a file, analyze it, and then display the information to the user. We have provided a specification and partial implementation of two classes, as well as a text file of the correct output against which you can check your output. Your task is to complete the implementation of the classes according to the specification.

Learning Objectives

1. Read and understand method-level specifications
2. Read and understand previously written code
3. Read data from the file
4. Complete the implementation of a class containing multiple instance variables and methods
5. Use the information in the file to create an array of objects
6. Scan through an array of objects and display the items that match a given criterion

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Import the existing lab2 implementation into your eclipse workspace.

   (a) Download the lab3 implementation:
   `http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab3/lab3.zip`

   (b) In Eclipse, select *File/Import*

   (c) Select *General/Existing projects into workspace.* Click *Next*

   (d) Select *Select archive file.* Browse to the lab3.zip file. Click *Finish*

2. Carefully examine the code for the *Driver* and *CourseEvent* classes.

3. Take note of the *TODO* markers in the code. This is where you will be doing your work. Note that Eclipse highlights the "TODOs" along the right hand side of the java file editor.

Reading a File

Within the Driver class in the lab3.zip file, we have included the code needed to open a file for reading:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;


public class Driver {

    /**
    * @param args
    */
    public static void main(String[] args) throws IOException {

    BufferedReader br = new BufferedReader(new FileReader("schedule.csv"));

    //Do stuff with the BufferedReader

    br.close();

    }
}
```

A *BufferedReader* can take different input streams as a parameter. In Lab 1, an *InputStreamReader* was used in order to take input from *System.in*. In this lab, a *FileReader* will be used. A *FileReader* takes a file name as a parameter. Notice that the file name is a String and therefore must have quotes around it. This opens the file and turns it into a *FileReader* object that can be read by the *BufferedReader* object. The file schedule.csv has already been imported as part of your lab 3 project.

Automatically Generating Getters and Setters in Eclipse

Eclipse is able to generate the getters and setters for a class for you. While editing the class, the steps are:

- Declare all the instance variables
- Select *Source* on the toolbar
- Select *Generate Getters and Setters...* from the dropdown menu

From this menu, you can select the getters and setters you would like generated from the dropdown associated with each class variable. You can also select all getters, all setters or both. See below for specific instructions for this lab.

Lab 3

For this lab, you will be parsing a file that is used to generate our course schedule for the class web page. This file is already imported into your lab 3 workspace, and encodes the set of events that will happen during the semester, including lectures, labs and due dates. The file encodes this information in a table using the *comma separated values* (CSV) format. If you double click on this file from within Eclipse, it will attempt to open the file in a spreadsheet program, such as Excel. Alternatively, you can select the file and *open with* a text editor. This will bring up the raw file in Eclipse.

Each line of this file encodes exactly one event. Each event is described using seven distinct values that are separated by commas in the file. These are:

1. The index number of the lecture. This value counts lectures starting from the beginning of the semester. If the event is not a lecture, then this field will have some other value
2. The index number of the lab. Like the lecture counter, this counts the number of labs from the beginning of the semester
3. The date of the event
4. A description of the event
5. Readings to be done in preparation of the event
6. The assignment that is being given on the day of the event
7. Any items that are due on the day of the event

Your task for this lab is to load these events from the schedule.csv file, store these events in an Array and then scan the array for events that match certain criteria.

Classes

The *CourseEvent* class is responsible for representing a single event. This class is partially implemented for you already (including a couple complete "helper" methods). Here are the methods that need to be completed:

- *CourseEvent* constructor. Here you will need to take in a String that corresponds to one event and parse it to populate the class variables
- *isLecture()* This method indicates whether a *CourseEvent* object is a lecture. An event is a lecture if its *lectureNumber* variable is a positive value
- *isLab()* This method indicates whether the event is a lab. An event is a lab if its *labNumber* variable is a positive value
- *isDue()* This method indicates whether an event includes items that are due. You know that something is due if this String contains characters
- A full set of getters. Use Eclipse to generate these for you

The *Driver* class is responsible for handling the file I/O, constructing an ArrayList of all events and generating three different reports containing events that match certain criteria. This class is partially implemented. The main() method of this class:

- Reads in the file
- Creates a *CourseEvent* object with each line of the file
- Adds those objects to the *ArrayList<CourseEvent>*
- Displays the *CourseEvent* objects that are classes
- Displays the *CourseEvent* objects that are labs
- Displays the *CourseEvent* objects that are due dates

Final Steps

1. Generate Javadoc using Eclipse.
    - Select *Project/Generate Javadoc...*
    - Make sure that your project is selected, as are the Driver and CourseEvent classes
    - Select *Private* visibility
    - Use the default destination folder
    - Click *Finish*
2. Open the *lab3/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that both of your classes are listed and that all of your documented methods have the necessary documentation.
3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Friday, September 11th.
- Prepare your submission file:
    1. Select the project in the *Package Explorer* window.
    2. Right-click. Select *Export*
    3. Select *General/Archive File*
    4. Expand your project and verify that both the *src* and *doc* folders are selected
    5. Enter the archive file name: *lab3.zip* (note that you may want to browse to a different destination folder)
    6. Select *Save in zip format*
    7. Click *Finish*
- Submit your zip file to the lab3 folder on D2L.

References

- The API of the *BufferedReader* class can be found at:
  `http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Implementation: 35 points

Program formatting: 10 points

- (10) The program is properly formatted (including indentation, curly brace and semicolon locations).
- (5) There is one problem with program formatting.
- (0) The program is not properly formatted.

Data types and method calls: 15 points

- (15) The program is using proper data types and method calls.
- (10) There is one error in data type or method call selection.
- (5) There are two errors in data type or method call selection.
- (0) There are multiple errors in data type and method call selection.

Required Methods: 10 points

- (10) All of the required methods are implemented.
- (0) The required methods are not implemented.

Proper Execution: 30 points

Output: 15 points

- (15) The program passes all tests.
- (10) The program fails one test.
- (5) The program fails two tests.
- (0) The program fails three or more tests.

Execution: 15 points

- (15) The program executes with no errors.
- (8) The program executes, but there is one minor error.
- (0) The program does not execute.

Documentation and Submission: 35 points

Project Documentation: 5 points

(5) The java file contains all of the required documentation elements at the top of the file.

(3) The java file is missing one of the required documentation elements.

(2) The java file is missing two of the required documentation elements.

(0) The java file is missing more than two of the required documentation elements.

Method-Level Documentation: 10 points

(10) Every method contains all of the required documentation elements ahead of the method prototype.

(7) The method documentation is missing one of the required documentation elements.

(3) The method documentation is missing two of the required documentation elements.

(0) The method documentation is missing more than two of the required documentation elements.

Inline Documentation: 10 points

(10) Every method contains appropriate inline documentation.

(7) There is one missing or incorrect line of inline documentation.

(3) There are two missing or incorrect lines of inline documentation.

(0) There are more than two missing or incorrect lines of inline documentation.

Submission: 10 points

(10) The correct zip file name is used and has the correct contents.

(5) The correct zip file name is used, but one required component is missing.

(0) An incorrect zip file name is used or more than one required component is missing.