Project 5: College Info App

Due: Skeleton Code (ungraded) – Friday, October 4, 2013 by 11:59 PM (checks class, method names, etc.)
Completed Code – Monday, October 7, 2013 by 11:59 PM

Updated 1:02 p.m. Wed 10/02/2013 – See highlighted text on page 4.

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until Fri 4 Oct 2013 11:59 PM. You should do your best to get this done in lab on Wed or by the end of the Help session on Fri to take advantage of this prior to the Fri night deadline. You must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code to avoid a late penalty for the project. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one day grace period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline.

Files to submit to Web-CAT:

- DeptInfo.java
- CollegeInfo.java
- CollegeInfoApp.java

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines DeptInfo objects, the second class defines CollegeInfo objects, and the third, CollegeInfoApp, reads in a file name from the user then reads dept info data from the file, creates DeptInfo objects and stores them in a ArrayList, creates a CollegeInfo object with the ArrayList and prints the CollegeInfo object, and then prints summary information about the CollegeInfo object.

- **DeptInfo.java (assuming that you successfully created this class in Project 4, just copy it to your new Project 5 folder and go on to CollegeInfo.java. Otherwise, you will need to create DeptInfo.java as part of this project.)**

  Requirements: Create a DeptInfo class that stores the name, abbreviation, location, number of faculty, number of graduate teaching assistants, number of undergraduate students, number of graduate students, and amount of research funding. It also includes methods to set and get each of these fields, a toString method, and methods that calculate the total number of students and the department viability index.

  Design: The DeptInfo class has fields, a constructor, and methods as outlined below.

  (1) **Fields** (instance variables and constants): name, abbreviation, and location which are of type String; number of faculty, number of graduate teaching assistants, number of undergraduate students, number of graduate students, which are of type int; and research funding which is of type double. These instance variables should be private so that they are not directly accessible from outside of the DeptInfo class. The last three fields should be public constants named MIN_FACULTY, MIN_UGRAD, and MIN_GRAD of type int that are set to 10,

200, and 50 respectively. These constants will be used to determine the viability index.

(2) **Constructor**: Your DeptInfo class must contain a constructor that accepts eight parameters (see types of above) representing the name, abbreviation, location, number of faculty, number of graduate teaching assistants, number of undergraduate students, number of graduate students, and amount of research funding. The values for name, abbreviation, and location should be trimmed of leading and trailing spaces prior to setting the fields (hint: use trim method from the String class). Below are two examples of how the constructor could be used to create a DeptInfo object in interactions. Note that although String and numeric literals are used for the actual parameters (or arguments) in these examples, variables of the required type could have been used instead of the literals.

```
DeptInfo dept1 = new DeptInfo("Computer Science and Software Engineering",
                       "CSSE", "Shelby Center", 18, 50, 550, 140,
                       2500000.0);
DeptInfo dept2 = new DeptInfo("Information Technology",
                       "IT", "Haley Center", 42, 64, 1050, 240,
                       850000.0);
```

(3) **Methods**: **Usually** a class provides methods to access and modify each of its instance variables (known as get and set methods) along with any other required methods. The methods for DeptInfo are described below.

- `getName`: Accepts no parameters and returns a String representing the name.
- `setName`: Takes a String parameter and returns a boolean. If the string parameter (name) is null, then the method returns false and the name is not set. Otherwise, the "trimmed" String is set to the name field and the method returns true.
- `getAbbrev`: Accepts no parameters and returns a String representing the abbreviation.
- `setAbbrev`: Takes a String parameter and returns a boolean. If the string parameter is null, then the method returns false and the abbreviation is not set. Otherwise, the "trimmed" String is set to the abbreviation field and the method returns true.
- `getLocation`: Accepts no parameters and returns a String representing the location.
- `setLocation`: Takes a String parameter and returns a boolean. If the string parameter is null, then the method returns false and the location is not set. Otherwise, the "trimmed" String is set to the location field and the method returns true.
- `getNumFaculty`: Accepts no parameters and returns the int value from number of faculty field.
- `setNumFaculty`: Accepts an int parameter, sets the number of faculty field, and returns nothing.
- `getNumGradAssts`: Accepts no parameters and returns the int value from number of graduate assistants field.
- `setNumGradAssts`: Accepts an int parameter, sets the number of graduate assistants field, and returns nothing.
- `getNumUgrads`: Accepts no parameters and returns the int value from number of undergraduates field.

- `setNumUgrads`: Accepts an int parameter, sets the number of undergraduates field, and returns nothing.
- `getNumGrads`: Accepts no parameters and returns the int value from number of graduate students field.
- `setNumGrads`: Accepts an int parameter, sets the number of graduate students field, and returns nothing.
- `getResFunding`: Accepts no parameters and returns the double value of the research funding field.
- `setResFunding`: Accepts an double parameter, sets the research funding field, and returns nothing.
- `totalStudents`: Accepts no parameters, calculates the number of students in department and returns the int value.
- `avgFundingPerFaculty`: Accepts no parameters, calculates the average funding per faculty (i.e., research funding / number of faculty) and returns the double value.
- `viabilityIndex`: Accepts no parameters, calculates viability index using the constants from the DeptInfo class with following formula and returns a double:
  (# faculty / MIN_FACULTY) + (# ugrad stu / MIN_UGRAD) + (# grad stu / MIN_GRAD)
- `toString`: Returns a String containing the information about the DeptInfo object formatted as shown below. You will need to include decimal formatting for the numerical values and newline escape sequences to achieve the proper layout. The following methods should be used to compute appropriate values in the toString method: totalStudents(), avgFundingPerFaculty(), and viabilityIndex(). The results of printing two examples above, dept1 and dept2, are shown below.

```
Computer Science and Software Engineering (CSSE)
Location: Shelby Center
Faculty: 18   GTA: 50
Students: 550(UG)   140(G)   690(total)
Research Funding: $2,500,000.00   Avg/Fac: $138,888.89
Viability Index: 7.35

Information Technology (IT)
Location: Haley Center
Faculty: 42   GTA: 64
Students: 1,050(UG)   240(G)   1,290(total)
Research Funding: $850,000.00   Avg/Fac: $20,238.10
Viability Index: 14.25
```

Code and Test: As you implement your DeptInfo class, you should compile it and then test it using interactions. For example, as soon you have implemented and successfully compiled the constructor, you should create an instance of DeptInfo in interactions (see the examples above). Remember that when you have an instance on the workbench, you can unfold it to see its values. You can also open a canvas window and drag the instance from the Debug tab to the canvas window. After you have implemented and compiled one or more methods, create a DeptInfo object and invoke each of your methods on the object to make sure the methods are working as intended.

- **CollegeInfo.java**

  Requirements: Create CollegeInfo class that stores the name of the college and an ArrayList of DeptInfo objects. It also includes methods that return the name of the college, total faculty, total undergraduate students, total graduate students, department with the maximum number of undergraduate students, department with the maximum number of graduate students, total research funding, a research funding report, and a toString method that prints the name of the collage followed by each DeptInfo object in the ArrayList.

  Design: The CollegeInfo class has two fields, a constructor, and methods as outlined below.

  (1) Fields (or instance variables): (1) a String representing the name of the college and (2) an ArrayList of DeptInfo objects. These are the only fields (or instance variables) that this class should have. The ArrayList field should be initialized to a new ArrayList of DeptInfo objects.

  ```
  ... = new ArrayList<DeptInfo>();
  ```

  (2) Constructor: Your CollegeInfo class must contain a constructor that accepts parameter of type String representing the name of the college and a parameter of type ArrayList<DeptInfo> representing the list of DeptInfo objects. The parameters should be used to assign the fields described above.

  (3) Methods: The methods for CollegeInfo are described below.

  - `getName`: Returns a String representing the name of the College.
  - `totalFaculty`: Returns an int representing the total number of faculty in the college. If there are zero DeptInfo objects in the list, zero should be returned.
  - `totalUgrads`: Returns an int representing the total number of undergraduate students in the college. If there are zero DeptInfo objects in the list, zero should be returned.
  - `totalGrads`: Returns an int representing the total number of graduate students in the college. If there are zero DeptInfo objects in the list, zero should be returned.
  - `deptWithMaxUgrads`: Returns a String representing the name of the department followed by number of undergraduate students in parentheses for the DeptInfo object with the maximum number of undgraduate students. If there are zero DeptInfo objects in the list, "(0)" should be returned. See line 30 in example output.
  - `deptWithMaxGrads`: Returns a String representing the name of the department followed by number of graduate students in parentheses for the DeptInfo object with the maximum number of undgraduate students. If there are zero DeptInfo objects in the list, "(0)" should be returned. See line 31 in example output.
  - `totalResearchFunding`: Returns a double representing the total research funding for the college. If there are zero DeptInfo objects in the list, zero should be returned.

- `researchFundingReport:` Returns a String containing the department name and research funding for each DeptInfo object followed by the total research funding for the college (each on a separate line). If there are zero DeptInfo objects in the list, the String "No departments found" should be returned. See lines 33 through 36 in the example below. Note that each line has three leading spaces.
- `toString`: Returns a String containing the name of the college followed by the information in each of the DeptInfo objects in the list. In the process of creating the return result, this toString() method should call the toString() method for each DeptInfo object in the list. Be sure to include appropriate newline escape sequences. For an example, see lines 2 through 25 in the output below from CollegeInfoApp for the *college.dat* input file. [*The result should **not** include the summary items in lines 26 through 35 of the example.*]

Code and Test: Remember to import java.util.ArrayList. Each of the methods above requires that you use a loop to through the objects in the ArrayList. As you implement your CollegeInfo class, you should compile it and then test it using interactions.

- **CollegeInfoApp.java**

Requirements: Create a CollegeInfoApp class with a main method that reads in the name of the college and then reads department info data from the file, creates DeptInfo objects, stores them in an ArrayList, creates a CollegeInfo object with the name of the college and the ArrayList of DeptInfo objects, and then prints the CollegeInfo object followed summary information about the CollegeInfo object.

Design: The main method should prompt the user to enter a file name, then it should read in gthe file. The first record (or line) in the file contains the name of the college. This is followed by the department information data. As the data for each department is read in, a DeptInfo object should be created and stored in an ArrayList. After the file has been read in and the ArrayList created, the main method should create a CollegeInfo object with the name of the college and the ArrayList as parameters in the constructor. It should then print the CollegeInfo object followed summary information about the CollegeInfo object (i.e., total faculty, total undergraduate students, total graduate students, department (and number) with the most <u>undergraduate</u> students, department (and number) with the most <u>graduate</u> students, and the research funding report for the college). Below is output produced after reading in the sample input file (**college.dat**). Your program output should be formatted exactly as follows (see all 37 lines):

```
Line #  Program output
1       Enter file name: college.dat
2
3       College of Engineering
4
5       Computer Science and Software Engineering (CSSE)
6       Location: Shelby Center
7       Faculty: 18   GTA: 50
8       Students: 550(UG)   140(G)   690(total)
9       Research Funding: $4,500,000.00   Avg/Fac: $250,000.00
10      Viability Index: 7.35
11
```

```
Electrical and Computer Engineering (ECE)
Location: Broun Hall
Faculty: 30   GTA: 60
Students: 450(UG)   180(G)   630(total)
Research Funding: $6,500,000.00   Avg/Fac: $216,666.67
Viability Index: 8.85

Chemical Engineering (ChE)
Location: Ross Hall
Faculty: 22   GTA: 60
Students: 400(UG)   200(G)   600(total)
Research Funding: $8,500,000.00   Avg/Fac: $386,363.64
Viability Index: 8.2

----- Summary for College of Engineering -----
Total Faculty: 70
Total Undergraduate Students: 1400
Total Graduate Students: 520
Dept with Most Undergraduate Students: Computer Science and Software Engineering (550)
Dept with Most Graduate Students: Chemical Engineering (200)
Dept Research Funding:
   $4,500,000 Computer Science and Software Engineering
   $6,500,000 Electrical and Computer Engineering
   $8,500,000 Chemical Engineering
   Total: $19,500,000

```

Code: Remember to import java.util.ArrayList, java.util.Scanner, and java.io.File, and java.io.IOException prior to the class declaration. Your main method declaration should indicate that main `throws IOException`. After your program reads in the file name, it should read in the file using a Scanner object that was created on a file using the file name entered by the user.

```
... = new Scanner(new File(fileName));
```

A while loop should be used to read in the file. You can assume that the first line is the name of the college, then each set of eight lines contains the data from which a DeptInfo object can be created. After the name of the college has been read and assigned to a local variable, you should use a loop to read the rest of lines in the file so that each iteration reads eight lines. As each of the lines is read from the file, the respective local variables for dept name, dept abbreviation, dept location, number of faculty, number of graduate teaching assistants, number of undergraduate students, number of graduate students, and research funding should be assigned, after which the DeptInfo object should be created and added to an ArrayList. The loop should then read the next set of eight lines. After the file has been processed (i.e., the loop terminates), name of the college and the ArrayList should be used to create a CollegeInfo object. The CollegeInfo object should then be printed. Finally, the summary information is printed by calling the appropriate methods on the CollegeInfo object.

Test: You should test your program minimally by reading in the sample file (college.dat), which should produce the output above. Although your program may not use all of the methods in DeptInfo, you should ensure that all of your methods work according to the specification. You can either user interactions in jGRASP or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.