Due: Skeleton Code (ungraded) – Friday, October 11, 2013 by 11:59 PM (checks class, method names, etc.)
Completed Code (100%) – Monday, October 14, 2013 by 11:59 PM

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 11 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 late 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.

F**iles to submit to Web-CAT:**

- DeptInfo.java
- CollegeInfo.java
- CollegeInfoMenuApp.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 a CollegeInfo, and the third, CollegeInfoMenuApp, presents a menu to the user with seven options and implements these: read the input file and create a CollegeInfo object, print the CollegeInfo object, print a summary of the CollegeInfo object, add a DeptInfo object to the CollegeInfo object, delete a DeptInfo object from the CollegeInfo object, find a DeptInfo object in the CollegeInfo object, or quit the program. **[I strongly recommend that you create a new "Project06" folder and copy your Project05 files to it, rather than work in the same folder as Project 5 files.]**

- **DeptInfo.java (Assuming that you successfully created this class in Project 4 or Project 5, just copy DeptInfo.java to your new Project 6 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 –** extended from Project 5 by **adding the last five methods below. (Assuming that you successfully created this class in Project 5, just copy CollegeInfo.java to your new Project 6 folder and then add the indicated methods. Otherwise, you will need to create all of CollegeInfo.java as part of this project.)**

  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.*]

The following five methods are new in Project 6:

- `readFile`: Takes a String parameter representing the file name, reads in the file, storing the college name and creating an ArrayList of DeptInfo objects, uses college name and the ArrayList to create a CollegeInfo object, and then returns the CollegeInfo object. See note #1 under Important Considerations for the CoolgeInfoMenuApp class (last page) to see how this method should be called.
- `summary`: Takes no parameters and returns a String representing a summary of the CollegeInfo that includes the total faculty, total undergraduate students, and total graduate students, the department with most undergraduate students, the department with the most graduate students, and research funding report.
- `addDeptInfo`: Returns nothing but takes eight parameters (name, abbreviation, location, number of faculty, number of graduate teaching assistants, number of undergraduate students, number of graduate students, and amount of research funding) creates a new DeptInfo object and adds it to the list.
- `deleteDeptInfo`: Takes the dept abbreviation as a parameter and returns true if the corresponding DeptInfo is found in the CollegeInfo object and deleted; otherwise returns false. This method should ignore case during a comparison for the dept abbreviation.
- `findDeptInfo`: Takes a dept abbreviation as the parameters and returns a String representing the toString() for the corresponding DeptInfo object if found in the list; otherwise returns a String containing the dept abbreviation " not found". This method should ignore case during the comparisons for the dept abbreviation.

Code and Test: Remember to import java.util.ArrayList, java.util.Scanner, java.io.File, java.io.IOException. These classes will be needed in the readFile method which will require a throws clause for IOException. Some of the methods above require that you use a loop to go through the objects in the ArrayList. You may want to implement the class below in parallel with this one to facilitate testing. This is, after implement one to the methods above, you can implement the corresponding "case" in the menu described below in the CollegeInfoMenuApp class.

- **CollegeInfoListMenuApp.java** (new – replaces the previous CollegeInfoApp class)

Requirements: Create a CollegeInfoMenuApp class with a main method that presents the user with a menu with seven options each of which is implemented: (1) read the input file and create a CollegeInfo object, (2) print the CollegeInfo object, (3) print the summary for the CollegeInfo object, (4) add a DeptInfo object to the CollegeInfo object, (5) delete a DeptInfo object from the CollegeInfo object, (6) find a DeptInfo object in the list, or (7) quit the program.

Design: The main method should print a list of options with the action code and a short description followed by a line with just the action codes prompting the user to select an action. After the user enters an action code, the action is performed, including output if any. Then the line with just the action codes prompting the user to select an action is printed again to accept the next code. Note that the first action a user should normally perform is 'R' to read in the file and create a CollegeInfo object. The other actions can then be used to process the CollegeInfo object. This continues until the user enters 'Q' to quit. Note that your program should accept both uppercase and lowercase action codes.

Below is output produced after printing the action codes and short descriptions followed by the prompt with the action codes waiting for the user to make a selection.

Line #	Program output
1	`CollgeInfo System Menu`
2	`R - Read in File and Create CollegeInfo`
3	`P - Print CollegeInfo`
4	`S - Print Summary`
5	`A - Add DeptInfo Object`
6	`D - Delete DeptInfo Object`
7	`F - Find DeptInfo Object`
8	`Q - Quit`
9	`Enter Code [R, P, S, A, D, F, or Q]:`

Below shows the screen after the user entered 'r' and then (when prompted) the file name. Notice the output from this action was "File read in and CollegeInfo created". This is followed by the prompt with the action codes waiting for the user to make the next selection. You should use the college.dat file from Project 5 to test your program.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, or Q]: r`
2	`      File name: college.dat`
3	`      File read in and CollegeInfo object created`
4	
5	`Enter Code [R, P, S, A, D, F, or Q]:`

The result of the user selecting ‘p’ to print a report is shown below.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, or Q]: p`
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	
12	`Electrical and Computer Engineering (ECE)`
13	`Location: Broun Hall`
14	`Faculty: 30   GTA: 60`
15	`Students: 450(UG)   180(G)   630(total)`
16	`Research Funding: $6,500,000.00   Avg/Fac: $216,666.67`
17	`Viability Index: 8.85`
18	
19	`Chemical Engineering (ChE)`
20	`Location: Ross Hall`
21	`Faculty: 22   GTA: 60`
22	`Students: 400(UG)   200(G)   600(total)`
23	`Research Funding: $8,500,000.00   Avg/Fac: $386,363.64`
24	`Viability Index: 8.2`
25	
26	`Enter Code [R, P, S, A, D, F, or Q]:`

The result of the user selecting ‘s’ to print the summary for the list is shown below.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, or Q]: s`
2	
3	`Total Faculty: 70`
4	`Total Undergraduate Students: 1400`
5	`Total Graduate Students: 520`
6	`Dept with Most Undergraduate Students: Computer Science and`
7	`Software Engineering (550)`
8	`Dept with Most Graduate Students: Chemical Engineering (200)`
9	`Dept Research Funding:`
10	`   $4,500,000 Computer Science and Software Engineering`
11	`   $6,500,000 Electrical and Computer Engineering`
12	`   $8,500,000 Chemical Engineering`
13	`   Total: $19,500,000`
14	
15	`Enter Code [R, P, S, A, D, F, or Q]:`

The result of the user selecting 'a' to add a DeptInfo object is shown below. Note that after 'a' was entered, the user was prompted for dept name, abbreviation, location, number of faculty, number of graduate teaching assistants, number of undergraduate students, number of graduate students, and amount of research funding. Then after the DeptInfo object was added to the CollegeInfo object, the message "DeptInfo added" was printed. This is followed by the prompt for the next action.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, or Q]: a`
2	`      Dept Name: Mechanical Engineering`
3	`      Abbrev: ME`
4	`      Location: Shelby Center`
5	`      Number of Faculty: 26`
6	`      Number of TAs: 30`
7	`      Number of UGrads: 1000`
8	`      Number of Grads: 200`
9	`      Research Funding: 5450000`
10	`      DeptInfo added`
11	
12	`Enter Code [R, P, S, A, D, F, or Q]:`

Here is an example of the successful deletion of a DeptInfo object, followed by an attempt to delete a DeptInfo object that wasn't found.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, or Q]: d`
2	`      Dept Abbrev: ChE`
3	`      "ChE" deleted`
4	
5	`Enter Code [R, P, S, A, D, F, or Q]: d`
6	`      Dept Abbrev: ECE`
7	`      "ECE" deleted`
8	
9	`Enter Code [R, P, S, A, D, F, or Q]:`

Here is an example of and attempt to "find a DeptInfo object that wasn't found, followed by a successful "find".

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, or Q]: f`
2	`      Dept Abbrev: CSSE`
3	`Computer Science and Software Engineering (CSSE)`
4	`Location: Shelby Center`
5	`Faculty: 18   GTA: 50`
6	`Students: 550(UG)   140(G)   690(total)`
7	`Research Funding: $4,500,000.00   Avg/Fac: $250,000.00`

```
8       Viability Index: 7.35
9
10      Enter Code [R, P, S, A, D, F, or Q]: f
11            Dept Abbrev: AE
12            "AE" not found
13
14      Enter Code [R, P, S, A, D, F, or Q]:
15
```

Code and Test:
Important considerations: This class should import java.util.Scanner, java.util.ArrayList, and java.io.IOException. Carefully consider the following information as you develop this class.

1. At the beginning of your main method, you should declare and create an ArrayList of DeptInfo objects and then declare and create a CollegeInfo object using the college name and the ArrayList as the parameters in the constructor. This will be a CollegeInfo object that contains no DeptInfo objects. For example:

   ```
   ArrayList<DeptInfo> _________ = new ArrayList<DeptInfo>();
   CollegeInfo _________ = new CollegeInfo(______,________);
   ```

   The 'R' option in the menu should invoke the readFile method on your CollegeInfo object. This will return a new CollegeInfo object based on the data read from the file, and this new CollegeInfo object should replace (be assigned to) your original CollegeInfo object. Since the readFile method throws IOException, your main method need to do this as well.

2. **Very Important**: You should declare only one Scanner on System.in for your entire program and should be in the main method. That is, all input from the keyboard (System.in) must be done in your *main* method.

3. For the menu, your switch statement expression should evaluate to a char and each case should be a char; alternatively, your switch statement expression should evaluate to a String with a length of 1 and each case should be a String with a length of 1.

After printing the menu of actions and descriptions, you should have a *do-while* loop that prints the prompt with just the action codes followed by a switch statement that performs the indicated action. The *do-while* loop ends when the user enters 'q' to quit. The new methods that require you to search the list should be implemented with a *for-each* loop as appropriate. You should be able to test your program by exercising each of the action codes. After you implement the "Print CollegeInfo" option, you should be able to print the list after operations such as 'A' and 'D' to see if they worked. You may also want to run in debug mode with a breakpoint set at the switch statement so that you can step-in to your methods if something is not working. In conjunction with running the debugger, you should also create a canvas drag the items of interest (e.g., the Scanner on the file, your CollegeInfo object, etc.) onto the canvas and save it. As you play or step through your program, you'll be able to see the state of these objects change when the 'R', 'A', and 'D' options are selected.

Although your program may not use all of the methods in your DeptInfo and CollegeInfo classes, 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.