Project 4: DeptInfo Program

Due: Skeleton Code (ungraded) – Friday, September 20, 2013 by 11:59 PM (checks class, method names, etc.)
Completed Code – Monday, September 23, 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 the project due date but should try to do this by Friday (there is no late penalty since this is ungraded for this project). For an example of skeleton code, see loan.java in the Class Notes for this week (04_Writing_Classes\examples\method_stubs\loan.java). In order to avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code. 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.

Files to submit to Web-CAT:

- DeptInfo.java
- DeptInfoApp.java

Specifications

Overview: You will write a program this week that is composed of two classes, one that defines DeptInfo objects and the other, DeptInfoApp, which inputs data, creates a DeptInfo object, and then prints the object.

- **DeptInfo.java**

  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 city .
- `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:
  (# 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. You may find it useful to create a separate class with a main method that creates an instance of DeptInfo then prints is out. This would be similar to the class you will create in Part 2, except that in in Part 2 you will read in the values and then create the object.

- **DeptInfoApp.java**

Requirements: Create DeptInfoApp class with a main method that creates the two DeptInfo objects described above and prints the object.

Design: The main method should prompt the user to enter the department name, abbreviation, location, number of faculty, number of graduate teaching assistants, number of undergraduate students, number of graduate students, and amount of research funding. After the values have been read in, a DeptInfo object should be created and printed. Below is an example where the user has entered the values from the example in Part 1 above. Your program input/output should be **exactly** as follows (be sure to skip a line before you "print" the DeptInfo object):

Line #	Program input/output
1	`Dept: Computer Science and Software Engineering`
2	`Abbreviation: CSSE`
3	`Location: Shelby Center`
4	`Number of Faculty: 18`
5	`Number of GTA: 50`
6	`Number of Ugrad Students: 550`
7	`Number of Grad Students: 140`
8	`Research Funding: 2500000`
9	
10	`Computer Science and Software Engineering (CSSE)`
11	`Location: Shelby Center`
12	`Faculty: 18  GTA: 50`
13	`Students: 550(UG)  140(G)  690(total)`
14	`Research Funding: $2,500,000.00  Avg/Fac: $138,888.89`
15	`Viability Index: 7.35`

Code: Your program should use the nextLine method of the Scanner class to read user input. Note that this method returns the input as a String. When you need an int or double value, you can use the Integer.parseInt method to convert a String to an int or the Double.parseDouble method to convert a String to a double. For example: `Double.parseDouble(s1)`will return the double value represented by String s1.

Test: You should test several sets of data to make sure that your program is working correctly. Although your main method does not use all the methods in DeptInfo, you should ensure that all of your methods work according to the specification. You can use interactions in jGRASP or you can write another class and main method to exercise the methods. The canvas should also be helpful. Web-CAT will test all of the methods specified above for DeptInfo to determine your project grade.