Due: Monday, October 8, 2012 by 11:59 PM

Deliverables

The following project files must be submitted to the "graded" assignment by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You should plan to start on the project not later than Wednesday in lab. To ensure that your classes and methods are named correctly, you should make sure that your file is successfully submitted to the "ungraded" assignment in Web-CAT during lab on Wednesday. **Projects sent via e-mail past the deadline at 11:59 PM will not be accepted without a university-approved excuse.**

Files to submit to Web-CAT:

- StudentInvoiceListMenuApp.java
- StudentInvoiceList.java
- StudentInvoice.java

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines StudentInvoice objects, the second class defines a StudentInvoiceList, and the third, StudentInvoiceMenuApp, presents a menu to the user with eight options and implements these: read the input file and create a student invoice list, add a student invoice to the list, delete a student invoice from the list, set tuitionFees for a student invoice in the list, set scholarships for a student invoice in the list, find a student invoice in the list, 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.]**

- **StudentInvoice.java** (this class is **unchanged** version from Project 5)

  Requirements: Create StudentInvoice class that stores student name, number, tuition and fees (one item), and scholarships. It also includes methods to set and get name and number, and methods to adjust each of the cost items.

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

  (1) **Fields** (or instance variables): student name and number which are String objects, and tuitionFees and scholarhips which are of type double. These instance variables should be private so that they are not directly accessible from outside of the class, and they should be set to appropriate default values. These are the only instance variables this class should have.

  (2) **Constructor**: Your StudentInvoice class must contain a constructor that accepts four parameters representing the student name, number, tuition and fees (one item), and scholarships. Below is an example of how the constructor could be used in main to create a StudentInvoice object:

```
StudentInvoice student1 = new StudentInvoice(studentName, studentNumber,
                                             tuitionFees, scholarships);
```

The parameter for studentName and studentNumber should be checked for null and not set if null or if equal to an empty String after being trimmed of leading and trailing spaces.

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

- `getName`: Accepts no parameters and returns a String representing the student's name.
- `setName`: Takes a String parameter and returns a boolean. If the string parameter (the new student's name) is null or if it has a length of zero, then the method returns false and the name field is not set. Otherwise, the "trimmed" parameter value is set to the name field and the method returns true.
- `getStudentNumber`: Accepts no parameters and returns a String representing the student's number.
- `setStudentNumber`: Takes a String parameter and returns a boolean. If the string parameter (the new student's number) is null or if it has a length of zero, then the method returns false and the student number field is not set. Otherwise, the "trimmed" parameter value is set to the student number field and the method returns true.
- `getTuitionFees`: Accepts no parameters and returns a double representing the student's tuition and fees.
- `setTuitionFees`: Accepts a double parameter and returns a boolean. Sets the tutionFee field and returns true if the parameter is greater than or equal to zero; otherwise returns false without changing the field.
- `getScholarships`: Accepts no parameters and returns a double representing the student's scholarships.
- `setScholarships`: Accepts a double parameter and returns boolean. Sets the scholarships field and returns true if the parameter is greater than or equal to zero; otherwise returns false without changing the field.
- `adjustTuitionFees`: Accepts a double parameter and returns double. Adds the parameter to the tutionFee field if the result would be greater than or equal to zero; otherwise the tutionFee field is not changed. Returns the value of the tutionFee field.
- `adjustScholarships`: Accepts a double parameter and returns double. Adds the parameter to the scholarships field if the result would be greater than or equal to zero; otherwise the scholarships field is not changed. Returns the scholarships field.
- `toString`: Returns a String containing the information in the StudentInvoice object as shown below, including newline escape sequences and formatting for the dollar values. Note that no lines are skipped.

```
Name: Pat Smith
ID Number: 012345
Tuition & Fees: $4,300.00
Scholarships: $500.00
You owe: $3,800.00
```

Or, if the scholarship amount exceeds the tuition and fees:

```
Name: Pat Smith
ID Number: 012345
```

```
Tuition & Fees: $4,300.00
Scholarships: $5,000.00
Please pick up your check for: $700.00
```

Code and Test: As you implement your StudentInvoice 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 StudentInvoice in interactions (see the example above). Remember that when you have an instance on the workbench, you can unfold it to see its values. After you have implemented and compiled one or more methods, create a StudentInvoice object and invoke each of your methods on the object to make sure the methods is working as intended.

- **StudentInvoiceList.java** (extended from Project 5 by **adding the last seven methods below**)

Requirements: Create StudentInvoiceList class that stores an ArrayList of StudentInvoice objects. It includes methods that return the average of all tuition and fees in the list, the percentage of students with scholarships in the list, the average of the scholarship for those who have scholarships, and a toString method. These are the same as in Project 5. Additional methods include: add a student invoice to the list, delete a student invoice from the list, set the tuitionFees for an invoice in the list, set the scholarships for an invoice in the list, set and get tuitionFees for an invoice in the list, and set and get scholarships for an invoice in the list, find an invoice in the list.

Design: The StudentInvoiceList class has a field, a constructor, and methods as outlined below.

(1) Field (or instance variable): an ArrayList of StudentInvoice objects. This is the only field (or instance variable) that this class should have, and it should be initialized to a new ArrayList of StudentInvoice objects.

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

(2) Constructor: Your StudentInvoiceList class must contain a constructor that accepts a single parameter of type ArrayList<StudentInvoice> representing the list of StudentInvoice objects. The parameter should be used to assign the field described above.

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

- `avgTuitionFees`: Returns a double representing the average of all the tuitionFee fields of the StudentInvoice objects in the list. If there are zero StudentInvoice objects in the list, an average of zero should be returned.
- `percentScholarships`: Returns a double in the range [0,1] representing the percentage of StudentInvoice objects in the list that have a scholarship greater than zero. If there are no StudentInvoice objects in the list, a percentage of zero should be returned.
- `avgScholarships`: Returns a double representing the average of the non-zero scholarship fields of the StudentInvoice objects in the list. If there are no StudentInvoice objects in the list, an average of zero should be returned.

- `toString`: Returns a String containing the information in each of the StudentInvoice objects in the list as shown below. To create the return result, the toString() method should call the toString() method for each StudentInvoice object in the list. [*The result should **not** include any of the following: average tuition and fees, percentage of students with scholarships, and the average scholarship.*]

The following seven methods are new in Project 6:

- `addStudentInvoice`: Returns nothing but takes four parameters (name, number, tuitionFees, and scholarships) creates a new student invoice and adds to the list.
- `deleteStudentInvoice`: Takes a student number as a parameter and returns true if the corresponding student invoice is found in the list and deleted; otherwise returns false.
- `setStudentInvoiceTuitionFees`: Takes a student number and a double and returns true if the corresponding student invoice is found in the list and the tuitionFees is set; otherwise returns false.
- `getStudentInvoiceTuitionFees`: Takes a student number as a parameter and returns a double representing the tuitionFees for the corresponding student invoice if found in the list; otherwise returns -1.
- `setStudentInvoiceScholarships`: Takes a student number and a double and returns true if the corresponding student invoice is found in the list and the scholarships field is set; otherwise returns false.
- `getStudentInvoiceScholarships`: Takes a student number as a parameter and returns a double representing the scholarships for the corresponding student invoice if found in the list; otherwise returns -1.
- `findStudentInvoice`: Takes a student number as a parameter and returns a String representing the toString() for the corresponding student invoice if found in the list; otherwise returns the String " not found".

Code and Test: Many of the methods above require that you use a loop to read the objects in the ArrayList searcjon. As you implement your StudentInvoiceList class, you should compile it and then test it using interactions. Alternatively, you can 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 StudentInvoiceMenuApp class.

- **StudentInvoiceListMenuApp.java** (replaces the previous StudentInvoiceListApp class)

Requirements: Create a StudentInvoiceMenuApp class with a main method that presents the user with a menu with eight options each of which is implemented: (1) read the input file and create a student invoice list, (2) print the report, (3) add a student invoice to the list, (4) delete a student invoice from the list, (5) set tuitionFees for a student invoice in the list, (6) set scholarships for a student invoice in the list, (7) find a student invoice in the list, or (8) 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 the list of student invoices. The other actions can then be used to process the list. 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	StudentInvoice System Menu
2	R - Read in File
3	P - Print Report
4	A - Add An Invoice
5	D - Delete An Invoice
6	T - Set Tuition and Fees
7	S - Set Scholarships
8	F - Find An Invoice
9	Q - Quit
10	Enter Code [R, P, A, D, T, S, 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 StudentInvoiceList created". This is followed by the prompt with the action codes waiting for the user to make the next selection. You should use the invoice.dat file from Project 5 to test your program.

Line #	Program output
1	StudentInvoice System Menu
2	R - Read in File
3	P - Print Report
4	A - Add An Invoice
5	D - Delete An Invoice
6	T - Set Tuition and Fees
7	S - Set Scholarships
8	F - Find An Invoice
9	Q - Quit
10	Enter Code [R, P, A, D, T, S, F, or Q]: r
11	Enter file name: invoice.dat
12	file read in and student invoice list created
13	Enter Code [R, P, A, D, T, S, F, or Q]:

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

Line #	Program output

	`Enter Code [R, P, A, D, T, S, F, or Q]: p`
	
	`Name: Pat Smith`
	`ID Number: 0123`
	`Tuition & Fees: $12,000.00`
	`Scholarships: $0.00`
	`You owe: $12,000.00`
	
	`Name: Sam Jones`
	`ID Number: 0124`
	`Tuition & Fees: $11,000.00`
	`Scholarships: $2,500.00`
	`You owe: $8,500.00`
	
	`Name: Bee Bonnett`
	`ID Number: 0125`
	`Tuition & Fees: $10,000.00`
	`Scholarships: $12,500.00`
	`Please pick up your check for: $2,500.00`
	
	`Average Tuition and Fees: $11,000.00`
	`Percentage of Students with Scholarships: 66.67%`
	`Average Scholarship: $7,500.00`
	
	`Enter Code [R, P, A, D, T, S, F, or Q]:`

The result of the user selecting 'a' to add a student invoice is shown below. Note that after 'a' was entered, the user was prompted for name, number, tuition & fees, and scholarships. Then after the student invoice was added to the list, the message "student added" was printed. This is followed by the prompt for the next action.

Line #	Program output

	`Enter Code [R, P, A, D, T, S, F, or Q]: a`
	`Name: Sammi Joseph`
	`ID Number: 0127`
	`Tuition & Fees: 3800`
	`Scholarships: 500`
	`student added`
	`Enter Code [R, P, A, D, T, S, F, or Q]:`

Here is an example of deleting a student invoice which wasn't found, followed by a successful deletion.

Line #	Program output

	`Enter Code [R, P, A, D, T, S, F, or Q]: d`
	`Delete Student with ID Number: 9999`
	`9999 not found`
	`Enter Code [R, P, A, D, T, S, F, or Q]: d`
	`Delete Student with ID Number: 0123`
	`0123 deleted`
	`Enter Code [R, P, A, D, T, S, F, or Q]:`

Here is an example setting tuition & fees that failed followed by a successful action.

Line #	Program output

	`Enter Code [R, P, A, D, T, S, F, or Q]: t`
	`Set TuitionFees for Student with ID Number: 9999`
	`Enter Tuition: 2700`
	`9999 not found`
	`Enter Code [R, P, A, D, T, S, F, or Q]: t`
	`Set TuitionFees for Student with ID Number: 0123`
	`Enter Tuition: 2700`
	`0123 Tuition and Fees: $2,700.00`
	`Enter Code [R, P, A, D, T, S, F, or Q]:`

Here is an example setting scholarships that failed followed by a successful action.

Line #	Program output

	`Enter Code [R, P, A, D, T, S, F, or Q]: s`
	`Set Scholarships for Student with ID Number: 9999`
	`Enter Scholarships: 5000`
	`9999 not found`
	`Enter Code [R, P, A, D, T, S, F, or Q]: s`
	`Set Scholarships for Student with ID Number: 0123`
	`Enter Scholarships: 5000`
	`0123 Scholarships: $5,000.00`
	`Enter Code [R, P, A, D, T, S, F, or Q]:`

Here is an example of finding a student invoice in the list that failed followed by a successful find.

Line #	Program output
	Enter Code [R, P, A, D, T, S, F, or Q]: f Find Student with ID Number: 9999 9999 not found Enter Code [R, P, A, D, T, S, F, or Q]: f Find Student with ID Number: 0124 Name: Sam Jones ID Number: 0124 Tuition & Fees: $11,000.00 Scholarships: $2,500.00 You owe: $8,500.00 Enter Code [R, P, A, D, T, S, F, or Q]:

Code and Test: 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 actions 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. You may also want to run in debug mode with a breakpoint set at the switch statement. Although your program may not use all of the methods in StudentInvoice and StudentInvoiceList, 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.