Due Date

March 2, by 11:59pm

Submission

1. You must designate a submitter (one of the team members) to submit all required files to Canvas. The comment block on top of each Java file must include the names of all team members.
2. Your project with all Java files (6 Java classes) that allow the TAs or graders to run and grade, one copy per team. [25 points]
3. Personal Time logs. This is an individual assignment. [5 points]
4. Test design document, one copy per team. [5 points]
5. Java docs, one copy per team. You must check the Java docs after you generate them and make sure the class hierarchy is correct and your comments are successfully generated. [5 points]
6. Class diagram, one copy per team. [5 points]

Program Description

You will be developing a Tuition Manager that manages student information and calculates the tuition due for the students based on the tuition/fee schedule in the following table. For tuition purpose, there are in-state, out-of-state and international students. Credit hours beyond 15 count as 15 only. That is, all students pay up to 15 credit hours if the credit hours exceed 15. International students must enroll at least 9 credit hours.

Student Type	**In-State**	**Out-of-State**	**International**
Tuition	$433 per credit	$756 per credit	$945 per credit
University Fee (all student types)	$846/part time(<12credit hours) $1,441/full time(>= 12 credit hours)		
International Student Fee	N/A	N/A	$350

Different types of students may have different tuition remissions.

- In-state students may be rewarded various amounts of funds. **Part time students do not qualify.**
- Out-of-state full-time students from the tri-state area get the tuition discount of $200 per credit.
- International students who are exchange students only pay full-time University fee and International student fee.

```
Example 1: In-state, 9 credit hours; Tuition due = 433 * 9 + 846 = 4743.
Example 2: In-state, 17 credit hours with $1,000 funding; Tuition due = 15 * 433 + 1441 – 1000 = 6936.
Example 3 Out-of-state, 17 credit hours with discount; Tuition due = 15 * (756 – 200) + 1441 = 9781.
Example 4 International, 12 credit hours with exchange student status Tuition due = 1441 + 350 = 1791.
```

Program Requirement

1. This is a **group assignment**. You MUST work in pair in order to get the credit for this program. You MUST submit a runnable program to pass this course.
2. You MUST follow the software development ground rules.
3. You are required to log your times working on this project with the template provided. **You will lose up to 5 points** if the log is not submitted or incomplete. The time log is an individual assignment.
4. Each Java class must go in a separate file. **-2 points** if you put more than one Java class into a file.
5. You are NOT ALLOWED to use any **Java container classes** from the Java library classes. **-10 points** if you do.
6. You must define and use the abstract class `Student` below. And **you must use it as is.** You will **lose 10 points if you change it.** You may need to add some constants in the Student class; however, you need to make sure the constants defined in this class are common data for all subclasses. **-2 points** for any violation.

```
public abstract class Student implements Comparable {
   private   String fname;
   private   String lname;
   protected int    credit;

   public Student(String fname, String lname, int credit) {...} //constructor

   //must implement compareTo method because Student class implements the Comparable Interface
   //return 0 if fname and lname of the two students are equal,
   //return -1 if this fname and lname is < obj's, return 1 if this fname and lname is > obj's
   //Hint: use the compareToIgnoreCase methods of the String class to compare fname
   //and lname; compare the fname first, then lname; names are not case-sensitive;
   public int compareTo(Object obj)

   //return a string with fname, lname and credit hours; subclasses will be using this method.
   public String toString() {...}

   //compute the tuition due; concrete implementation is required in the subclasses.
   public abstract int tuitionDue() {...}
}
```

7. You MUST have the following subclasses that extend the abstract `Student` class:

```
public class Instate extends Student {
   private int funds;
   ...
}

public class International extends Student {
   private boolean exchange;
   ...
}

public class Outstate extends Student {
   private boolean tristate;
   ...
}
```

   You CANNOT add other data members to these class, except the constants required for the classes, and you CANNOT do I/O in this class. **-2 points** for each violation. You must write a **test bed main** for each of the subclasses above. **-2 points** for each violation. The test cases must be documented with a **test design document** similar to Program 1. You **will lose up to 5 points** if this is not done. All subclasses must have a concrete implementation for the abstract method `tuitionDue()`. In addition, all subclasses must override the `toString()` method of the Student class. `tuitionDue()` method calculates the tuition amount due for a variety of students; `toString()`method must call the `toString()` method in the superclass, add addition information specific to the subclass, and return the detail information of a student as a string.
8. You MUST implement an array-based growable `StudentList class`. StudentList must be a separate class and must provide `add(Student s)`, `remove(Student s)`, and `print()` public methods. You must use `compareTo()` method in this class **-2 points** if you don't. You CANNOT do I/O in this class, EXCEPT the `print()` methods, **-2 points** for each violation.
9. **Polymorphism** is required. An instance of `StudentList class` must hold a list of `Instate`, `Outstate` and `International` student objects. Your program should print the correct tuition amounts by calling the `tuitionDue()` method based on the actual student types in the array list. You CANNOT store tuition amount information in any classes. **-10 points** if you define tuition amount as a data member in any classes, or your container class cannot hold multiple types of students.

10. You must implement a `TuitionManager class` to handle the interaction with the user and the I/O (console input and output.) That is, this is **the interface class** that will process the commands and output on the console. This class should take command line input as follows.
    - **Add** a student to the list; the command line format is

      <command> <fname> <lname> <credit> <type-specific data>

      Where `<command>` could be I, O, or N;
        - I, is to add an in-state student to the student list; `<type-specific data>` is the dollar amount of the funding.
        - O, is to add an out-of-state student to the student list; `<type-specific data>` is a boolean value representing the tri-state status: T or F.
        - N, is to add an international student to the student list; `<type-specific data>` is a boolean value representing the exchange student status: T or F
        - `<fname>` and `<lname>` are the student's first name and last name, respectively.
        - `<credit>` is the number of credits the student is taking; your program should NOT take `0` or negative numbers, or a number that is less than `9` if it is an international student.

      For all the adds, you should check if a student with the same first and last name already in the list. If so, you must output an error message and do not add the student to the list.
    - **Remove** a student from the student list; the command line format is

      R <fname> <lname>

      For all the removes, only remove the student from the list if the fname and lname are equal.
    - **P**, without other input tokens needed, is to **print** the list of students and **the tuition amounts due** to the console output, in the following format.

      <student information> tuition due: $ ...

      Where `<student information>` is displayed by calling the toString() method of the actual student type.
    - **Q** is to terminate the program; your program should output "Program terminated".

    The `TuitionManager class` should continuously process the command line input until the command 'Q' is entered. The commands are case-sensitive, i.e., only valid uppercase letters are accepted. You can assume the first token of each line of input is always a single letter command.
11. Create a Class Diagram for Program 2 showing the relationships between classes. Each class must include the class name, data fields and methods showing appropriate modifiers and data types. You must use a drawing tool to create the diagram, **-2 points** if you don't. You can find free drawing tools online, such as draw.io.

Program Testing

1. You MUST create a test document and design the test cases **for the 3 subclasses**: Instate, Outstate, and International classes. The test document is worth 5 points.
2. You should always test the "invalid" input or operations, which may include
    - `0` or negative for the number of credits, or negative for the funding amount of an in-state students.
    - Invalid commands; lowercase commands or commands not supported.
    - Part time in-state students are not eligible for the funding, even if a positive amount is entered
    - Adding a student who is in the list, printing an empty list, removing a student from an empty list, or removing a student not in the list.
    - The number of credits for international students is less than 9.

3. Some valid test cases for adding the students are shown below for your reference.

Command Line Input	Expected tuition due
O May Anderson 17 F	$12,781
O Lauren Brown 17 T	$9,781
I Peter Parker 8 0	$4,310
I Wilson Long 8 1000	$4,310
O Simons Michael 8 F	$6,894
O Stiller Anderson 8 T	$6,894
I Peter Liang 12 0	$6,637
I John Young 12 1000	$5,637
O Good Man 12 F	$10,513
O John White 12 T	$8,113
N David Lee 12 F	$13,131
N Joe Kim 12 T	$1,791
I Michael Chamber 17 0	$7,936
I John Smith 17 1000	$6,936
N April Young 17 F	$15,966
N Mary Yang 17 T	$1,791