Project 2: Student Registration System

Due Date: Sunday, March 17th at 11:59pm.

Overview

In this project, you will be implementing a registration system. The objective of this second project is to implement a functioning system based on object-oriented programming concepts: encapsulation, inheritance and polymorphism, in the form of abstract classes and method overriding, will be exercised in this assignment.

Registration System Functionality and Hierarchy

You have been assigned the task of implementing a student registration system for some small college with the following basic functionality:

- Maintaining the course catalog.
- Registering students for classes.
- Dropping a class for a student.
- Posting a course grade for a student.

There are two types of students: - undergraduate students, and – graduate students. An undergraduate student must register a minimum of 6.0 credit hours before they can register a non-major course, while a graduate student can only register in major courses. Grade assignment is also different between a graduate and an undergraduate student. We therefore define our `Student` as a super class with `Undergraduate` and `Graduate` as its two subclasses, such that registration and grade assignments are specialized by the subclasses. The project also defines a `Course` class for maintaining course information and a `TranscriptEntry` class which basically keeps track of course information for a course attended by a student along with the student's grade in that course. Finally, the system defines a Registrar class which takes care of registration related tasks such as maintaining the course catalog, starting a new semester, enrolling students into classes and so on. The system implements the following inheritance structure:

`Student` : maintains basic student information and performs registration tasks and generates transcripts / course schedule for a student.
`|`
`+--- Undergraduate extends Student` : implements course registration and assign grades for an undergraduate student.
`|`
`+--- Graduate extends Student` : implements course registration and posting of grades for a graduate student.

`Course` : maintains basic course information.
`|`
`+--- TranscriptEntry extends Course` : defines a transcript entry for a student, and includes attended course information along with a student's grade in that course.

`Registrar` : offers courses and performs student registration.

Implementation/Classes

For the implementation of the project, please note the following:

- You may **not** use any classes from java.util (unless Scanner which is already included).
- You may **not** change the signature of any of the public methods.
- You may **not** define new public methods to the provided classes (you can, however, add as many private methods as you need).
- All data structures needed for this project are included in the provided class code templates, your focus should be on implementing the functionality of the system by coding the public methods.

In the project's package, you will find the following classes (*the provided template files contain further comments and additional details*):

(1) `Course` (Course.java): Maintains information about a course, and defines the following two public methods:

- `@override toString()`: returns a String of the course information in the following format:
  "INFS 612 Communication Systems"
- `fullString()` : returns the same result as the `toString()` method and adds to total credit hours for the course:
  "INFS 612 Communication Systems, (3) credit hours"

(2) `TranscriptEntry` (TranscriptEntry.java): extends Course and adds the following two public methods:

- `@override toString()`: overrides the method in its parent class and returns a student transcript for the course as the following string:
  "\tINFS 510 Database Systems, credits: 3, GRADE: A"
  And if the student does not have an assigned grade, then the string value should be:
  "\tINFS 510 Database Systems, credits: 3"

- `isActive`(): returns true if the student is currently enrolled in the course (maintained by this `transcriptEntry` object), returns false if it is a past course (more details are provided in TranscriptEntry.java).

(3) `Student` (Student.java): This is an *abstract* class (meaning that it cannot be instantiated) which includes two abstract methods: `approvedForClass()` and `setCourseGrade()`. It also contains the following public (concrete) methods:

- `public boolean registerAclass(Course course, String semester, int year)`: adds a course to the student and returns true. If the student is currently enrolled in the same course, registeration is not performed and the method return false.
- `public boolean dropAclass(String CourseCode)`: removes a course in which a student is currently enrolled and returns true. If the student is not currently enrolled in the course or if it's a previous course (has a grade assigned), returns false. Note that this

method must literally remove the course object by shifting elements of the arrays leftward such that all course objects always appear consecutive in the array. (replacing a course object with null is not sufficient)

- `public boolean obtainGrade(String CourseCode, int score)` : given the code, this method find the student's registration in the course, assigns a letter grade for the student and returns true. If the student is not currently enrolled in the course or if it's a past course, it returns false.
- `public String getClassList(String semester, int year)`: this method returns the student's course registration a specified semester. Returns null if no registration record is found for that semester.
- `public boolean equals(Object anotherStudent)` : returns either true or false.
- `@override public String toString()` : returns a string representation of a student in the following format: "Smith, John (G#0000000000)" (note the last name appears first).

(4) `Undergraduate` (Undergraduate.java): A subclass of `Student` which overrides the `toString()` method and implements the two abstract methods defined in its parent class:

- `protected boolean approvedForClass(Course course)`: invoked by the `registerAclass()` method in the super class Student, this method will check if the student is eligible for registering in a given course and returns true if so or false otherwise.
- `protected boolean setCourseGrade(TranscriptEntry entry, int score)`: invoked by the `obtainGrade()` in the parent class, this method will assign a letter grade to the student in the given course.
- `@override String toString()` : overrides the `toString()` method defined in its parent class, returns a String in the following format:
  "Smith, John (G#0000000000), Degree: B.S., Major: Computer Science"
  (can include a call to its parent's `toString()` to generate the first part of the string).

(5) `Graduate` (Graduate.java): same as `Undergraduate`.

(6) `Registrar` (Registrar.java): This is the main class which implements the required functionality of the registration system. It defines the following public methods:

- `public boolean addCourse(String code, String title, int hours)` : instantiates a new course object, adds it to the course catalog (an array of type Course) and returns true. If the course catalog is full, will return false.
- `public String getCourseCatalog()` : returns a string containing all courses offered at the college.
- `public boolean addStudent(String fname, String lname, long gnum, String major, String degree, String highSchool)`: instantiates an *undergraduate* student object, adds it to the student list which is maintained by the registrar (an array of type `Student`), and returns true. This method will return false if the maximum student capacity is reached (i.e.) a full students array).

- `public boolean addStudent(String fname, String lname, long gnum, String major, String degree, String uMajor, String degree)`: an overloaded method for adding a *graduate* student..
- `public boolean register(long gnum, String courseCode)` : returns false if a course (by the given `courseCode`) does not exist in the course catalog or if a student with the given gnum is not found in the students array. Otherwise, this method will invoke the r`egisterAClass()` on the student object to complete registration. If the student's registration is done successfully true is returned, false otherwise.
- `public boolean drop(long gnum, String courseCode)` : returns false if a course (represented by the given `courseCode`) does not exist in the course catalog or if a student with the given `gnum` is not found. Otherwise, this method will invoke the `dropAClass()`on the student object to complete registration. If the course drop is successful true is returned, false otherwise.
- `public boolean postGrade(long gnum, String courseCode, int score)` : returns false if a course (by the given `courseCode`) does not exist in the course catalog or if a student with the given `gnum` is not found. Otherwise, this method will invoke the `obtainGarde()` on the student object to post the course grade. If the grade update is successful true is returned, false otherwise.
- `public String getProgress(long gnum, String semester, int year)` : returns null if a student with the given `gnum` is not found. Otherwise, this method will invoke `getClassList()` and will return the class list for that student in the specified semester/year.
- `public static String getDeptName(String code)` : this is a static method which, given a course code, will return the department name to which that course belongs.

To implement the application logic and enable users to access the registration system, a driver program (RegistrationProgram.java) is provided in the project package which you may use for testing your application.

Honors Section

If you are in the honors section, you must complete this part and it is worth 20 points of the project grade. If you are not in the honors section, you are welcome to attempt this but you do not need to complete it and it is not worth any points if you do.

1) Define the `DynamicArray` class which represents a collection of Student objects in an array whose capacity can grow and shrink as needed.
2) Use `DynamicArray` to implement the array of student objects (name of the arrays is students) defined in the Registrar class. You must replace the standard array defined with your own array class.

Name your class DynamicArray.java and implement it so that it maintains a collection of `Student` objects in an array whose capacity can grow and shrink as needed. Your class should supports the usual operations of adding and removing objects, and must provide the following public methods:

- `boolean isEmpty()`: returns true if the array is empty, false otherwise.

- `boolean add(Student s)`: adds a student to the array.
- `boolean remove(Student s)`: removes the Student object from the array. Items are shifted left-ward to replace the removed array element.
- `int Size()`: returns the number of elements in the array.
- `int capacity()`: returns the actual size of the array.
- `Object [] toArray()`: returns an array containing all of the elements in the array.
- `void set(int index, Student s)`: replaces the element at the specified position in the array with the passed student object.
- `Student get(int index)`: returns the element at the specified position.

The capacity of a `DynamicArray` must change as items are added or removed and the following rules must be followed in your implementation:

- The default initial capacity is fixed to be 2.
- When you need to add items and there is not enough space, grow the array to double its capacity.
- When you delete an item and the size falls below 1/3 of the capacity, shrink the array to half its capacity.

Submission Instructions

Download the template project: https://mason.gmu.edu/~iavramo2/classes/cs211/s19/p2.rar. This is a zipped file which contains template code/classes for the project. Your task will be to fill in each of the public methods as described. The package also includes a tester class which you can use for checking your project. *These include sample test cases*. For grading, we may modify the set of test cases.

Submission instructions are as follows.

1) Let xxx be your lab section number, and let yyyyyyyy be your GMU userid. Create the directory xxx_yyyyyyyy_P2/
2) Place all your project's .java files in the directory you've just created.
3) Create the file ID.txt in the format shown below, containing your name, userid, G#, lecture section and lab section, and add it to the directory.
   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 004
   Lab section: 213
4) Compress the folder and its contents into a .zip file, and upload the file to Blackboard.