CCCS-301 Assignment #3 *Polymorphism, Interfaces and Abstract*

Due Date:
March 16, 2015 at 23:30 on myCourses

Question 1: Aggregated Objects (5 points)

Create a Java application from 3 java source files called Program.java, Student.java and Classroom.java.

In the source file Program.java is a single class called Program with a single static main method. This main method will have the only user interface. The other java files are not permitted to perform any I/O. In other words, only the main method is permitted to write to the screen or read from the keyboard.

Program.java is the only class permitted to instantiate objects. The other two java files are not permitted to instantiate anything.

Write a main method that asks the user for the size of a classroom and then the number of students they would like to add into that classroom. Next, instantiate a classroom object using the classroom size the user wanted. Then, using a loop, instantiate one Student object for each student the user wanted and populate its class variables by prompting the user for information. Put these student objects into the classroom. To verify that you did this correctly, your main method will then display all the student information from the classroom. Format this output in an easy to read way. The output must be from objects returned to your main method by the classroom object. Your program will then end.

In the source file Student.java is a single class called Student. Each student class stores the information for a single student. Everything in this class is private except for the API.

You may construct the private portions of your Student class in any way you like, but it can only contain code that we saw from class or obtained from the assigned chapters from the textbook. You can have any number of private methods. You may have as many private class variables as you like. The Student class stores only the student's name and GPA. They are both private.

The API is the only public portion of your program and it must be constructed exactly as specified: (a) a constructor called **`Student(String name, double gpa).`** (b) the accessors: **`public void setName(String name), public void setGPA(double GPA), public String getName(), public double getGPA().`** Method setName() will only set the name if the parameter "name" is not empty. Method setGPA() will only set the GPA when the parameter "gpa" satisfies the condition 0 <= GPA <= 4.

In the source file Classroom.java is a single class called Classroom. This class aggregates Student objects. Everything in this class is private except for the API. Classroom contains the private aggregator: **`private Student students[]`** and the public API : (a) the constructor: **`Classroom(int maxClassroomSize)`** which defines the size of the students[] array. (b) The accessors: **`public boolean add(Student aStudent)`** and **`public Student getStudent(int position)`**. The method add() takes a student object that was instantiated from the main method, finds the next available free space in the array students[] and puts the object at that place. If there is no more space in the array the method return false, otherwise it returns true. Method add() will also return false when the Student object passed as a parameter is null. A null object will not be added to the array students[]. The method getStudent() takes an integer as a parameter. This parameter is used to specify a location in the array. If the integer is a negative number or it is larger than the size of the array then the method returns null as an error message. If the parameter refers to a cell in the array students[] that is null then it returns null, otherwise it returns the reference in that cell of the array.

Submit the following Java files: Program.java, Student.java and Classroom.java. Only submit the Java files. Make sure your program runs to receive maximum grades.

Question 2: The Basic Container Class using Object (5 points)

A useful Java programming technique is called the "container". A container aggregates objects in a generic way. To be generic a container class needs to be able to hold any kind of information, no matter what Java type, and it must provide a common interface to the information it holds. Java has the generic class Object that is able to reference any other class in Java. All classes, by default, are children of the Object class.

In this question we want to use Polymorphism with the Object class to create a basic container implementation. You will be creating two classes for this question: ContainerVer1 and Tester1. Tester1 will contain the main program and the code that validates your container. The class ContainerVer1 is your first implementation of the container concept. Question 3 will ask you to improve your container.

Create the following container class:

```
class ContainerVer1
{
     private Object array[];
     ContainerVer1(int size) { … }
     public boolean add(Object item) { … }
     public Object get(int index) { … }
     public boolean delete(int index) { … }
     public int find(Object item) { … }
     public boolean isMember(Object item) { … }
}
```

The container uses an array of type Object. The array's size will be initialized by the constructor. Initialize the array to null to indicate empty, or unused, cells.

The method add() inserts an Object into the first available null space in the array. If it cannot find a null space it returns false, otherwise it returns true.

The method get() returns a reference to the Object in the array at the given index number, or it returns null if nothing is there or if the user provided an invalid integer.

The method delete() removes the Object from the array. The deleted cell is null once more. The method returns false if the index number was not valid, otherwise it returns true. Even if the cell is already null and nothing is there, a true is returned. The deleted Object is not returned. The array is not compacted.

The method find() returns the index number of the first cell in the array that contains the same Object passed in the parameter list. This means that the data in these two objects must be the same (the references do not need to be the same). A negative one is returned if this is not the case. Doing this with an Object class is not straightforward. So for this version1 container let us assume that we will only pass String objects to it.

The method isMember() MUST use find() in it's implementation, and returns true if the Object passed in its parameter exists in the container, otherwise it returns false.

You are permitted to create additional helper methods, if you like, but they MUST be all private. Make sure all your methods properly test the validity of the input parameters and the validity of the operations they are performing.

The Tester1 class contains your main method. We will use it to validate your container. Do the following: (a) instantiate a container of size 20 as a local variable (not a class variable). (b) Populate the container with strings input by the user. Construct a while-loop that repeats a maximum of 20 iterations or until the user enters the string DONE in capital letters. Entering the DONE will terminate the loop early. The word DONE is not put into the container. (c) Make sure to display appropriate prompts so the user knows what to do. (d) Now we will test your container: (d-1) Prompt the user for an integer and display what the get() method returns. (d-2) Ask the user for another integer and invoke the delete() method. (d-3) Then display all the strings in the container, the one we deleted should not be there, unless the user gave an invalid index number. (d-4) Ask the user for a String and then display what the find() and isMember() methods return. (d-5) Your program then prompts the user "Would you like to test the container again?". If the user enters Y or y or YES or yes or Yes, or any other capitalization of yes then the program goes back and repeats steps (d-1) to (d-5). If the user enters N or n or NO or no or any other capitalization of the work no the program terminates. If the user enters another word then an error message is displayed and the user is asked to provide a yes or no answer. If they continue to enter the wrong information they stay in this error loop indefinitely, until they do the correct input.

Question 3: The Full Container (6 points)

Interfaces are used in two important ways: as a technique to standardize a concept or/and a contract between two programmers. For example, the “container” idea is a useful concept. So useful that it should be standardized. In other words there should be only one way to use a container.

A “contract” between two programmers helps when sharing code. One person writes a class and then shares that class with someone. The shared class, because of how it was programmed, expects the programmer who uses the shared class to write their code in a particular way so that the shared class does not crash. A contract expressed though a Java interface can help ensure the programmer includes things in their code that the shared class was expecting.

We want to standardize our container class interface and upgrade it to store any object, not only Strings. Use code from question 2, above, to help you build the solution for this question. Create three classes: Tester2, FullContainer, and BankAccount.

Class FullContainer implements the interface CompareObjects. FullContainer must also inherit from ContainerVer1. You are permitted to overload or override methods if you find it is necessary. Class BankAccount implements the interface Transactions.

Class FullContainer has the same methods as seen in ContainerVer1 except that the methods works on any Object, not only Strings. In addition FullContainer implements the interface CompareObjects which has the following signatures: `public void duplicate(int index); and private bool equals(Object o);`
The method duplicate() will use the index to refer to a cell in the array. If the cell is not null copy the value in that call and use add() to insert the copy into the next available cell in the container, if any. This duplicated object is a newly instantiated object containing the same information as the original object. In our previous versions of the container we took advantage of the fact that the String class has the .equals() method. This helped us in the find() and isMember() methods. We need to guarantee that any object added to our container will have a .equals() method. This interface will help us make sure.

Class BankAccount implements the interface Transactions that has the following signatures: `public void deposit (double amount); public void withdrawal (double amount); public void getBalance(); and public static final double overdraftFee = 5.00;`
The method deposit() will only deposit values greater than zero. The method withdrawl() only deducts amounts greater than zero and less than the current balance in the account. The method getBalance() returns the current balance to the calling program. If the user attempts a withdrawal greater than the balance they are charged the overdraftFee. This may cause their balance to become negative.

Class Tester2 displays a simple text menu with the following options: (1) Create account, (2) Delete account, (3) Deposit, (4) Withdraw, (5) Get balance, (6) Duplicate account, (7) Quit. Tester2 stays in this menu until the user presses Quit.

Create account asks the user for the account number (a string) and the balance (a double). Instantiates the account and then adds it to the container, if possible.

Menu options Delete account and Duplicate account asks the user for the account information and then invokes the delete() or duplicate() method respectively to remove or duplicate the account.

Methods Deposit(), Withdrawal(), and getBalance() asks the user for the information they need and then updates the values of the respective object they refer.

Selecting Quit first displays the contents of the Container for our examination by displaying in a nice way the bank account values and their index position in the Container. Then, the program terminates.

Question 4: Domain Model (4 Points)

Create a domain model for question 3.

IMPORTANT instructions:

Submit everything on My Courses. Click *Assignments*, and then select *Assignment #3*. This is where you should upload your assignment files.

Understand that: The regular assignments box does accept late assignments up to 2 days and you should put your late assignments there. After that, assignments are no longer accepted. But if for some special reason you want your assignment to be graded even after the 2 day late period you can email it to the instructor. You then must email your instructor and convince him that it should be graded. If that is successfully done then the instructor emails a TA to grade that specific problem box.

Make sure that all your java source files have at least the following information at the top (beginning) of every file:

```
1.Name: ...
2.ID number: ...
3.Course number: ...
4.Assignment number: ...
5.Where you developed your program: (i.e. at home, in the
CS lab, in the Engineering lab ...)
```

myCourses permits you to submit your assignments until 30 minutes before midnight. There are 2 late submission days with a penalty of 5% per day. Assignments are not accepted after the 2 late days without a very good reason attached.

If you cannot upload the assignment then email it to me.

Marking Section

General Marking: Total 2 points
- Commenting – 1 point
- Indentation and style – 1 points

Question 1: Total Points 5
- +2 Program
- +1 Student
- +2 Classroom

Question 2: Total Points 5
- +3 ContainerVer1
- +2 Terster1

Question 3: Total Points 6
- +2 FullContainer
- +2 Interfaces
- +1 Tester2
- +1 BankAccount

Question 4: Total Points 4