Homework 4: Department

1 Objective

This project is the first time you’ll create your own objects, along with private data, methods, etc. This isn’t a complete working system with a user interface; it’s just a chance for you to create classes, then instantiate and test them. You’ll create two *supplier* classes that interact with each other (through a “has a” relationship).

2 Objects You’ll Create

Here are UML Class Diagrams for the objects you are to create. Pay attention to the diagram notation indicating whether methods are public (+) or private (-); ask questions if you need clarification. Understanding the model is of *critical* importance here.

|  |  |  |
| --- | --- | --- |
| **Student** |  | **Department** |
| *(you figure out the private data needed)* |  | (*you figure out the private data needed*) |
| **Constructors**  +Student(firstName : String, lastName : String, id : String)  +Student(firstName : String, lastName : String, id : String, csc110 : Boolean, csc142 : Boolean, csc143 : Boolean)  **Accessors**  +getFirstName() : String  +getLastName() : String  +getID() : String  +getCSC110() : Boolean  +getCSC142() : Boolean  +getCSC143() : Boolean  +getBalance() : Double  +toString() : String  **Mutators**  +settFirstName() : String  +setLastName() : String  +setID() : String  +setCSC110(csc110 : Boolean)  +setCSC142(csc142 : Boolean)  +setCSC143(csc143 : Boolean) |  | **Constructors**  +Department()  +Department(totalStud110 : Integer, totalStud142 : Integer, totalStud143 : Integer, totalBalance : Double)  **Accessors**  +getTotalCSC110 () : Integer  +getTotalCSC142 () : Integer  +getTotalCSC143 () : Integer  +getTotalStudents() : Integer  +getTotalBalance() : Double  +getTotalGr110() : Integer  +getTotalGr142() : Integer  +getTotalGr143() : Integer  +toString() : String  **Other Methods**  +addStudent(newStudent : Student) : void |

3 Constraints and Assumptions

* Create *no* static methods except for the test methods.
* Mark *each* instance variable and *each* method as either public or private; follow the UML where it gives guidance and make smart decisions where it doesn’t.
* Use exactly the method names shown. Look carefully at the parameter data types and the return data types; they give you clues.
* Department’s addStudent() method maintains the appropriate total students of each group and the total balance.
* Department’s getTotalGr… methods calculate how many groups need for each class (28 students in the class).
* The toString() method should do their best to summarize the state of the object instance in questions; include newlines in the strings to make the result displayable and attractive.
* Create a test method for each class that checks every constructor and every method except toString.
* One credit equals 5, and one credit costs $110.
* Create Constant Credit and credit cost
* One student can only take one class
* Check if a student take more than 1 class
* Getbalance = credit \* credit cost
* Throw an IllegalArgumentException if any of these preconditions are violated:

o A first name, last name, and ID can't be empty.

o Total students of each group can't be negative.

* ID 10 length Random
* Check for first name and last name like if there exist empty, special symbol or number
* To do getTotalCSC110 we need to addStudent
* Default csc142 is true
* getTotalCSCXXX is the count of students that are taking that course
* Everytime we can only add 1 student

4 Code Implementation

Follow the provided Course Style Guide.

* Don’t use magic numbers, create constants for credits, credit price and the number of students in one class.

5 Testing

The testing required here is significant. You may spend as much time writing your tests as you do writing the entire rest of the code. Your test code needs to test everything (or nearly everything), your object can do; don’t treat it lightly. Don’t forget to test constructors and preconditions. And don’t forget test code is *still code* and can have bugs, so be suspicious if everything passes miraculously the first time, i.e., test that you can induce failures and get the appropriate error messages shown.

6 Submitting Your Work

You’ll be creating two java classes (.java) files, create a jar-file and submit them as a single file. BlueJ can create .jar files; be sure and specify “include source” if you use this method.

7 Hints

Don’t duplicate code; avoid this wherever possible. As an example, don’t forget it’s legal for one constructor to call another constructor; this is good practice and helps avoid duplication and the creation of extra code paths to test.

8 Grading Matrix and Achievement Levels

|  |  |
| --- | --- |
| **Area** | **Pct** |
| Design, correctness, implementation | 90% |
| Documentation and style | 10% |
| **Total** | **100%** |

|  |  |
| --- | --- |
| **Achievement** | **Max Points** |
| Compile/runtime errors | 50% |
| Basic objects, no test methods | 70% |
| Test accessors and mutators | 80% |
| Test constructors | 90% |
| Test for preconditions, including exception handling | 100% |