CS 203 HW #6
Designing Classes: MFP Project

PART 6A: due Friday, Mar. 7, 10:00 pm
PART 6B: due Wednesday, Mar. 19, 10:00 pm

Each semester the ASUP solicits ideas for projects to be funded through the Major Project Fund and then decides which projects to fund. While I am confident that your ASUP representatives have a perfectly good process for managing this important task, in this assignment we will imagine that we have been put in change of creating a computer program to simulate an ASUP vote on the budget. You will design classes to help manage this process, as described below.

Overview

To complete this assignment, you will design and implement two classes, Project and Senator.

Your starter file contains a partially written Meeting class to simulate an ASUP budget vote and, not coincidentally, to help you test your code. Because we do not know how you will design your classes, however, parts of the program are left for you to complete.

Part A: Design

Advice on Designing Your Classes

Read the entire assignment (both parts) carefully and review **`Meeting.java`** before you begin designing. The requirements state what your classes need to be able to do, but they do not spell out how to do it. For example, you will have to decide what data types to use for representing the information associated with each class, so knowing how the information will be used later may help you decide how best to represent it.

Review Sections 5.1 and 5.2 in your textbook.

Design the public methods before the instance variables. First, decide what each method's name will be and what it needs to return or, if it doesn't return anything, what task it needs to accomplish. Next, write the pseudocode for each method. That should give you a pretty good idea of what parameters your method will need to do its job, and what instance variables you will need for saving information after the method returns, and what types will be best for each parameter and instance variable.

As you write your pseudocode, you may find that you will need instance variables, class constants, and private "helper" methods that are not mentioned explicitly in the Requirements. That's fine. In fact, it is expected. Define them and add them to your class.

Remember:

- All (non-constant) attributes and other instance variables in each class should have **`private`** access.
- "Helper" methods, i.e., methods that are intended to be used only by other methods in the same class, also should have **`private`** access. Some methods are too dangerous to let just anybody use them.

- Methods that are intended to be used by other classes, e.g., the Meeting class, should have **`public`** access.
- All the code you write should have complete and appropriate comments.

Requirements for the Project class

An instance of this class contains all the information about a single proposed project. The information that must be represented includes:

- The name of the project.
- The estimated cost.
- The project category: capital improvement (a project that builds something permanent at the University), service (a project that seeks to make the world better for some identifiable constituency), or social (a project that sponsors one or more social events for the benefit of students).
- An ID number for this project, which is assigned by your class when the project is created. Each project must have a different ID number. (Hint: a static variable could be used to keep track of the next unused project id.)
- The number of votes cast for this project.
- At least one other attribute of your choice.

The Project class should support the following actions (methods):

- Accessor (getter) methods for getting the information listed above.
- A mutator (setter) method to increment the number of votes cast for this project.
- At least one method for getting or setting a value based on an attribute that you chose.

In addition, the Project class should have:

- A constructor that has no parameters and sets each of the instance variables to some legal default value. (Java uses this constructor when initializing arrays of objects.)
- A constructor with one parameter for each attribute above *except* the project id number and the number of votes. This constructor should initialize the project's instance variables to the parameter values, but only after checking that each of them is a legal value for that variable. It is up to you to decide what constitutes a legal value for each instance variable. If any of the parameters are not legal, set that instance variable to its default value instead. The number of votes should always be set to 0. The constructor should also assign the project number.
- A **`toString`** method that returns a String representation of this Project. You decide what that should include.
- An **`equals`** method that returns true when its argument (another Project) is equivalent to this Project. Again, you decide what that means.

Don't forget to supply Javadoc comments for the class and for each method.

Requirements for the Senator class

An instance of this class contains information about a single senator, which includes the following:

- The senator's name
- The senator's constituency (what group the senator represents)
- The projects that the senator supports. The number of projects can vary by senator.
- At least one other attribute of your choice

The Senator class should support the following actions (methods):

- Accessor methods to get the value of the senator's name and constituency.
- Mutator methods for adding projects that the senator supports and for removing projects that the Senator no longer supports.
- At least one method for getting or setting a value based on the attribute-of-your-choice.
- A method for voting for a project. Senators vote only for projects that they support, and they can vote for a project only once. There must be some way for a program using your senator class to determine whether the senator has already voted for every project that they support.

In addition, the Senator class should have:

- A constructor that has no parameters and sets each of the instance variables to some legal default value. (Java uses this constructor in initializing arrays of objects.)
- A constructor with parameters for the Senator's name, constituency, the maximum number of projects that this Senator intends to support, and the additional attribute(s) that you designed. This constructor should initialize the project's instance variables to the parameter values, but only after checking that each of them is a legal value for that variable. It is up to you to decide what constitutes a legal value for each instance variable. If any of the parameters are not legal, set that instance variable to its default value instead.
- A **`toString`** method that returns a String representation of this Senator. You decide what that should include.
- An **`equals`** method that returns true when its argument (another Senator) is equivalent to this Senator. Again, you decide how to determine that.

Part A Deliverables and Grading

Capture your design as a skeleton class implementation for Project.java and Senator.java. Use BlueJ to create new Project and Senator classes as demonstrated in lecture.

Your skeleton classes should contain:

- Javadoc comments for the class, including a description of the class, your name, and the version date. (10%)
- Instance variables, class variables, and constants. (10%)
- Javadoc comments for each method and constructor, including a brief description of the method, parameters (if any), and return value (if any). (20%)
- Method stubs for each method and constructor, i. e., the method header line, opening and closing braces, and a return statement with a dummy value of the correct type[1]. For example, a method that returns an int might return 0, a method that returns a boolean might return false. (20%)
- Inside each method, your pseudocode for that method as comments. The pseudocode should be sufficiently detailed to allow us to see that your class will meet all the requirements listed above or implied by the way the class is used in **`Meeting.java`**. (20%)

Your skeleton class should compile (10%) and meet the usual code quality standards (10%).

Turning in this Assignment

- Be sure your name is in the comment header at the top of each of your .java files.

---

1. The sampleMethod() that BlueJ provides when you create a new class is a method stub.

- Compress your entire BlueJ project into a single .zip archive. Be sure your archive includes the directory itself so that, when unzipped, it creates that directory.
- Submit your .zip file via the associated "Turn it in Here" link on the course website.

Part B

Implement and test your classes, **one method at a time**. Start with the constructors. Use the Meeting class to your advantage by adding to it bit-by-bit as you write the other two classes. Each time you write a new method in Senator or Project, add some code to your Meeting class that uses it and test it immediately. Don't wait until all the methods are coded to begin testing.

Turning in this Assignment

- Be sure your name is in the comment header at the top of each of your .java files.
- Compress your entire BlueJ project into a single .zip archive. Be sure your archive includes the directory itself so that, when unzipped, it creates that directory.
- Submit your .zip file via the associated "Turn it in Here" link on the course website.