Computer Science 3716: Software Methodology

Final Project Submission

Group JN Student Society System

Samuel Ash, Lucas Critch, Osede Onodenalore, Shehzaib Ramay

Classes and Responsibilities

SOCIETY

Society is the major class in our system which is responsible for the delegation of many methods. The society object holds all of the data associated with a society.

STUDENT

Student is the object upon which societies are based. Societies are filled with students. Each student has information about themselves and various roles with respect to the society system.

MEMBERROLE

memberRole is a class which is associated with a student by way of composition. A student contains a memberRole (null or not) which means that they are a member of a society. A student can only execute any actions of a member through their memberRole.

BOARDMEMBERROLE

boardMemberRole is a class which is associated with a student by way of composition. A student contains a memberRole (null or not) which means that they are a board member of a society. A student can only execute any actions of a board member through their boardMemberRole.

PRESIDENTROLE

presidentRole is a class which is associated with a student by way of composition. A student contains a presidentRole (null or not) which means that they are a president of a society. A student can only execute any actions of a president through their presidentRole.

ELECTION

Election is a class which handles the tallying of any votes made and calculates the new president of any society that has an election.

MEETING

Meeting is an object which partly composes a society. A society can have as many meetings as desired.

EVENT

Event is an object which partly composes a society. A society can have as many events as desired.

Essential Features

With our system, we focused on implementing the features outlined by the client, with the following features from most to least essential:

CREATE A SOCIETY

A student can create a society. They are made to be the society president, and are therefore also made into a board members.

JOIN A SOCIETY

Any student (must be a student) can join any society in the system. A student cannot join a society if they are already a member.

LEAVE A SOCIETY

Any student who is a member of a society can choose to leave that society at any time.

CONDUCT AN ELECTION

Any society can have an election at any time (called by president). If no votes are counted, the president will not change. If there is a tie, the first person who reaches the highest number of votes wins. This can be handled differently in the future.

HAVE AN EVENT/MEETING

A society can create an event or meeting at any time (called by board members). Any events/ meetings created will be stored in the society information. These can be cancelled at any time.

Design Principles

DON'T REPEAT YOURSELF

DRY was carefully implemented. We are working with many classes which often make references to each other and therefore certain code which could have been repeated, but was not.

SINGLE RESPONSIBILITY PRINCIPLE

A member cannot appoint his/herself to be a board member, for example. Every class has a specific responsibility in the system, and no class is used for a purpose which is not appropriate in the system.

LISKOV SUBSTITUTION PRINCIPLE

Events behave exactly the same way as meetings and can be used in place of meetings. Events, however, have names, while meetings do not.

DELEGATION

Many functions called by memberRole, boardMemberRole, presidentRole, and Election use each other but are all delegated through Society first.

OPEN-CLOSED PRINCIPLE

For the most part, our classes are built to completion. They can be extended upon, but cannot or should not be modified. For example, student does not need any more method with regards to our system. Anything else which needs to be added should be added outside of this class.

Key Decisions

We decided early on in our project what code we would build upon, and what approach we would take to efficiently solving the task at hand.

As it turns out, we were able to efficiently combine the code from groups J and N to make a stable and efficient system. However, there were some significant decisions left to be made after this.

COMPOSITION

For the responsibility of members, board members, and the president of a society, we decided after much discussion that we were going to implement composition to efficiently carry out the task.

In essence, a Student has a memberRole, boardMemberRole, and presidentRole as instance variables, initially declares as null. When a student becomes a member, board member, or president of a society, these are no longer null. When these are not null, a student can use these to execute certain methods inside these classes, by executing

Student.getMemberRole(Society).methodInsideMemberRole()

This also throws a memberPermissionException, so that if a student is not a member of any society, or if they are not a member of the society passed to the method, it will throw an exception and not allow them to execute any method.

Known Bugs

While our code is complete and is known to work within simple test classes, there are some bugs with regards to saving our code in our user interface.

-*Events do not save* upon closing the window

-Currently an event can be created in the UI *regardless of whether the student is a board member or not*

-Upon closing the window, a student who *is recognized as a member* of a society *cannot leave that society*

-In the society panel (socPanel), the name of the president *does no display properly* (does not update)