**CS-273 Exam 2 (Fall 2019) 50 points**

***After you have completed the required tasks, please submit your project folder on WhitGit as Exam2.***

Imagine you are building a system to manage attendance at events.

The UML diagram of the **design** of the software I currently have is as follows:

**<Abstract Class >**

**Invite**

**PlanMtgInvite**

**<Abstract Class >**

**Event**

**PlanningMtg**

My software design **has two abstract base classes** called **Event** and **Invite**. **Event** defines the **interface** to the different types of events I can create with my software, i.e. it tells you what methods an Event will have, but not how it implements them. The **Event** interface is defined by the following **Abstract Data Type** (ADT):

|  |  |
| --- | --- |
| Function | Description |
| Invite \*add(string name) | Adds a person, with **name**, into the event, and returns an invite for that person. |
| void list() | Prints out all the people currently at the event. |

The concrete class **PlanningMtg** is derived from **Event**, and it implements the required methods defined inthe base class.

When a person is added to an Event, he/she is given an **Invite.** This is an interface for an abstract ticket that an event goer can use to leave the event at any time and return later, just in case a certain measure of decorum and / or secrecy is required. (You can, and are encouraged, to imagine why.) The **Invite** interface is defined by the following **ADT**:

|  |  |
| --- | --- |
| Function | Description |
| void leave() | Allows the party goer to leave the event. |

The concrete class **PlanMtgInvite** is derived from **Invite**, and it implements this interface.

All my events will contain event goers, and therefore I will need a **data structure** to keep track of them. Now, attendees will be coming and going frequently from my events, so I need a data structure that will allow me to add and delete arbitrary people from them **efficiently**. Therefore in the design of my **PlanningMtg** class, I use a **list** to contain all the party goers in it.

Adding a person to a **PlanningMtg** just involves adding the person’s name to the **list** called **room** in the **PlanningMtg** class. Listing the people in the **PlanningMtg** involves iterating through the **room** list to print out the names of each person currently present.

Remember a person can only leave an event with an **Invite**? Well, our **PlanMtgInvite** therefore needs to store an **iterator** pointing to the position where the person was added when he/she first entered. This **iterator** is determined in the **add** method, and stored in the **PlanMtgInvite** that is returned from that method. To implement the **leave** method in **PlanMtgInvite,** the invite needs to know the event to which the invite is referring. The **leave** method uses this knowledge to call the **PlanningMtg** **remove** method to actually delete the iterator from the **room** in the party.

The full UML diagram of our program is shown below:

**<Interface>**

**Event**

**<Interface>**

**Invite**

**+ virtual Invite\* add(string name)**

**+ virtual void list()**

**+ virtual void leave()**

**PlanningMtg**

**- list<string> room**

**+ Invite\* add(string name)**

**+ void list()**

**+ void remove(list<string>::iterator &me)**

**PlanMtgInvite**

**- Event \*this\_party**

**- list<string>::iterator me**

**+ void leave()**

**Your task is to do the following:**

1. Implement the design indicated by the UML diagrams and description, above.
2. Following the same design pattern for **PlanningMtg** and **PlanMtgInvite**, create a custom party type of your own invention. (Kudos to creativity.)

**EXTRA CREDIT:**

If you do all the above but use an STL set instead of a linked list as the underlying type for room, I will award 15 extra points for this exam! Hint: Take a look at a good C++ reference on the web and observe how the insert method for sets has been overloaded. What do the various overloaded insert methods return? How can you use that knowledge to obtain an iterator to party-goer names as you insert them? Another hint: if you take this approach, I’d recommend you extend the base class for events to include a virtual remove method and that you thereby commit to using sets for all sorts of events, not just planning meetings.

You have until end of day Monday, 11/18 to complete this exam. Good luck!