**CS-273 Exam 2 (Spring 2018) 100 points**

***Preliminary setup: Copy the Visual Studio project exam2.zip to your desktop. After you have completed the required tasks, please copy your project folder to github, to a project called Exam2. For the exam, you are allowed to look at your prior programming projects.***

I love parties! It is a great way to unwind, catch up with old friends, and make new ones. Therefore, I’ve decided to write a piece of software to create any kind of party I want, i.e. a dinner party, a birthday, a dance party, a garden party, etc.

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

**<Abstract Class >**

**PartyTicket**

**DinnerPartyTicket**

**<Abstract Class >**

**Party**

**DinnerParty**

**PartyFactory**

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

|  |  |
| --- | --- |
| Function | Description |
| PartyTicket \*add(string name) | Adds a person, with **name**, into the party, and returns a party ticket for that person. |
| void list() | Prints out all the people currently in the party. |

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

When a person is added to a party, he/she is given a **PartyTicket.** This is an interface for an abstract ticket that a party goer can use to leave the party at any time. After all, there might be homework to do. The **PartyTicket** interface is defined by the following **ADT**:

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

The concrete class **DinnerPartyTicket** is derived from **PartyTicket**, and it implements this interface.

My software also uses the **factory method design pattern** to create any kind of parties I want, so long as the party is derived from my **Party** abstract base class. The **factory** method is defined in the class **PartyFactory**.

All my parties will contain party goers, and therefore I will need a **data structure** to keep track of them. Now, party revelers will be coming and going frequently from my parties … I hold great parties after all! So I need a data structure that will allow me to **efficiently** add and delete arbitrary people from my parties. Therefore in the design of my **DinnerParty** class, I use a **list** to contain all the party goers in it.

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

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

The full UML diagram of our program is shown below:

**<Interface>**

**Party**

**<Interface>**

**PartyTicket**

**+ virtual PartyTicket add(string name)**

**+ virtual void list()**

**+ virtual void leave()**

**PartyFactory**

**+ Party \*factory()**

**DinnerParty**

**- list<string> room**

**+ PartyTicket add(string name)**

**+ void list()**

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

**DinnerPartyTicket**

**- Party \*this\_party**

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

**+ void leave()**

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

1. Complete the **add**, **list,** and **remove** methods in the **DinnerParty** class.
2. Following the same design pattern for **DinnerParty** and **DinnerPartyTicket**, create a **Birthday** and **BirthdayPartyTicket** to allow us to have a birthday party … when the weather gets warmer of course!
3. Update **PartyFactory::factory** to allow it be the only method to create a **BirthdayParty**.