----------------------------------------------------------------------------------------------------------------------------

REFLECTION

Name: Hashmeet Singh Saini

Seneca ID : hsaini28

Student No. : 153070214

Date : October 5th, 2022

I have done all the coding by myself and only copied the code

that my professor provided to complete my project milestones.

----------------------------------------------------------------------------------------------------------------------------

This week’s workshop was all about learning three different types of relationships between classes. These were Compositions, Aggregations and Associations. Out of these, Composition is the most common and involves complete ownership as the composer class has complete control of the lifetime of the component. On the other hand, Aggregations display a weaker relationship as the aggregatee class controls its own lifetime. Finally, there is an Association, the weakest of all and involves no ownership. Both classes in an association can use each other. In my code, the restaurant class exhibits a composition relationship with the reservation class as its component and the confirmationSender class exhibits an aggregation relationship with the reservation class acting as an aggregate.

1. The main difference between the implementation in composition and an aggregation relationship is in the process of construction, destruction, copying and assigning. First all you can see in line 18 of restaurant.h:

Reservation\*\* m\_reservations{};

The array of pointers m\_reservations is not const. This means that restaurant class has complete control over the elements of this array of pointers to reservation objects. On the other hand, in line 18 of confirmationSender.h:

const Reservation\*\* m\_csReservations{};

The array of pointers m\_csReservations is const, meaning that the class itself can’t modify the elements of the array because it doesn’t have control over the construction and destruction of the Reservation objects. The main difference lies between the constructors of both relationships. In composition, it is the composers’ responsibility to manage the construction of component class, that is why in line 15 of Restaurant.cpp:

Restaurant::Restaurant(const Reservation\* reservations[], size\_t cnt)

A parametric constructor has been defined that essentially creates a deep copy of the objects supplied through the first parameter. There is also a dual-layered dynamic allocation because not only the array has been dynamically allocated, but also the different pointers to objects inside it. On the other hand, ConfirmationSender only has an empty body constructor because the construction responsibility of aggregatee is not in its hand.

As discussed above that Composition has a dual-layered allocation whereas Aggregation has just a single, it also means that the destructors have to work differently too. As you can see from lines 70 to 76 of Restaurant.cpp:

for (size\_t i = 0; i < size(); i++) {

delete m\_reservations[i];

}

delete[] m\_reservations;

we first delete every single element in the array, then the array itself. On the flip side in ConfirmationSender the destructor only deletes the array itself and the job is done because the destructor of every Reservation is called itself by the Reservation class. There are also add and remove functions or operators defined for aggregation to manage the addition and removal of different object pointers to its class. There are also differences in the copy and move constructors discussed in the next question.

Essentially in Composition, the data held is a copy whereas in Aggregation the data held is an existing object stored somewhere.

1. The differences between the copy and move constructors of a composition and aggregation relationship are minute. Since compositions have complete ownership, they also control the copying and moving of the component class. When the component class is being processed in terms of pointers it becomes essential to pay attention to these operators. In restaurant.cpp you can see at line 32 copy constructor that the same dual layered destruction mentioned above is being followed to delete every pointer as an element first and then the array being copied to. Also the construction of every element looks like this:

m\_reservations[i] = new Reservation(\*reservations[i]);

The same deletion process is followed in the move constructor of the Restaurant class as well. On the contrary the simple deletion of the array pointer is sufficient in aggregation as seen in the confirmationSender class copy and move constructors.

delete[] m\_csReservations;