**Name: Syed Asad Abrar**

**Roll No: L164292**

**Section: F**

**Assignment 3**

**Q1 a)** We will use virtual keyword with Creature when writing the classes for Fish and Woman while writing their classes, this is known as virtual inheritance. This problem is known as diamond problem, virtual inheritance is used to solve it. Without virtual inheritance, Fish and Woman both inherit age from Creature, class Mermaid inherits both from Fish and Woman so will contain two copies of age variable. When we use virtual inheritance, only one instance of age will be present in Mermaid because Fish and Woman inherit virtually from Creature.

class Fish : virtual public Creature

{

//code goes here

}

class Woman : virtual public Creature

{

//code goes here

}

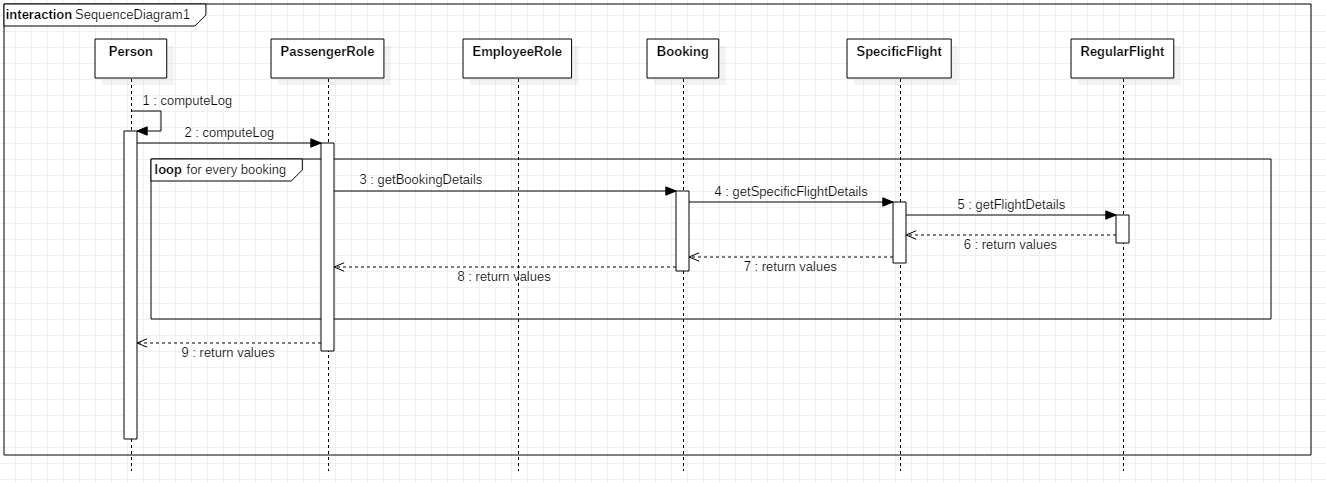
Class Mermaid : public Woman, public Fish

{

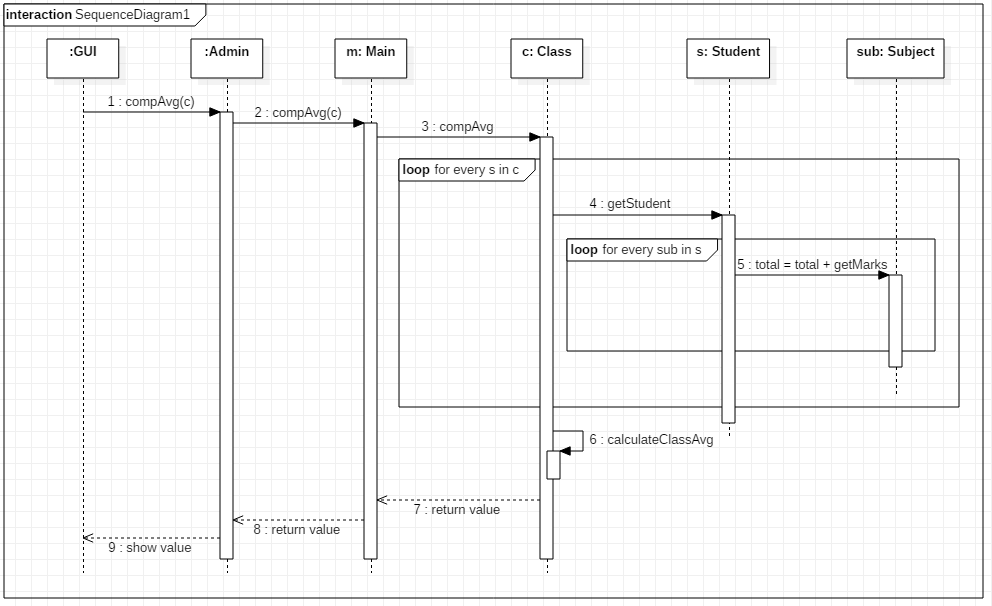
//code goes here

}

**Q1 b)** This code works when you use virtual inheritance.

**Q2)** This sequence diagram assumes that we are only interested in the persons having a passengerRole since employees do not have a seat number in the flight which is a requirement in the information which is to be returned by the computeLog function. The values returned from getFlightDetails is time and flightNumber, from getSpecificFlightDetails is values from getFlightDetails and date, from get Booking Details is values from previous function and seatNumber. These return values can be appended in a string or can be returned by the use of specifically made classes or can simply be printed. ****

**Q3)** getAllStudents() and getAllSubjects() are unnecessary functions.



The variable total is local and is present in the compAvg function of ‘Class’. calculateClassAvg is a ‘Class’ function, which divides the integer ‘total’ with the number of students. This value is then returned to main, which returns it to admin class, which prints it on gui.

**Q4)** These are a few drawbacks we face when a business layer isn’t used:

1. Heterogeneous environments/Business environments with rapidly changing rules and regulations are not suitable since the database server has to handle the business logic which slows down database performance.
2. Since client beholds most of the application logic, problems arise in controlling the software version and re-distributing of new versions.
3. Security wise this is complicated as users need to have separate login information for every SQL server.
4. Scalability: The 2-tier model lacks scalability as it supports only a limited number of users. When simultaneous client requests increases application performance degrades rapidly due to the fact that clients necessitate separate connections and CPU memory to proceed.
5. Minimal Logic Sharing: Since the application logic is coupled with the client it’s difficult to re-use logic dispersed among applications and tools.
6. Dispersion of Applications: Any change in an application should reflect all clients. If higher number of users exists in the system, it entails substantial administrative overhead.
7. Change of Database Structure: Most applications used for interaction is dependent on the database structure creating an issue when re-designing, as they are intimate with the prevailing structure.

Taken from <https://www.quora.com/What-are-the-advantages-and-disadvantages-of-architecture-1-tier-2-tier-3-tier-and-n-tier>

**Q5)** In proxy pattern, a class represents functionality of another class. We create object having original object to interface its functionality to outer world. Basically, the idea is that there would be a light weight class which would contain minimal information about the heavy weight class objects and each object of light weight class would have a reference to the heavy weight object. The light weight objects would be loaded into memory, heavy weight objects would be in storage. Only when we need a specific heavy weight object, we will load it in memory using the reference contained in the corresponding light weight class object. *The light weight class* is a proxy class to reduce memory footprint of *the heavy weight class* object loading.