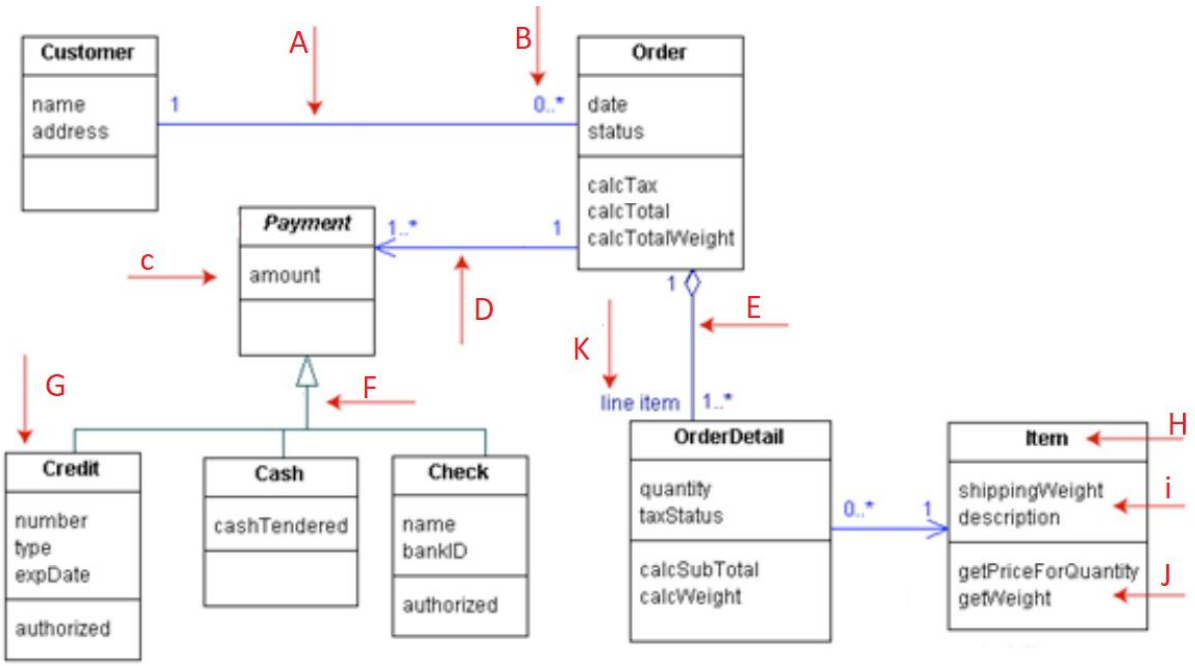
Fordeling av poeng:

1	1	1	1	1	1	1	1	1	1	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I	J	K								
1	1	1	1	1	1	1	1	1	1	1	4	6	6	6	6	6	6	6

10	10	10	11			
1	2	3	A	B	C	
UC	DM	Int				
5	5	5	2	2	2	

Karakter	%		Poeng	
A	90	100	70,2	78
B	80	89	62,4	69,42
C	60	80	46,8	62,4
D	50	60	39	46,8
E	35	46	27,3	35,88
F	0	34	0	26,52

1 [11 points] Identify each of the labeled components in this UML class diagram.



Answer:

A : Association (bi-directional)
B : Multiplicity
C : Superclass (generalization class or parent class)
D : Directional association (uni directional)
E : Aggregation
F : Generalization or inheritance

G : Subclass (specialization class or child class)
H : Class name
I : Attributes
J : Operations/Methods
K : Role

2. [4 points] In the previous class diagram, the association labeled E has a particular meaning in UML. Give a short description of that meaning, and how a developer should interpret that.
Answer:
This is NOT an own, parent-child relationship. It is not a composition, hence we do not tell the developer that OrderDetail should be deleted when Order is deleted. Many students have (1) argued from a view of composition, but that is neither correct nor good, because with that viewpoint they indicate that there is only two kind of association, composition and aggregation, but that is not correct. Further (2) they argue in a way that one could have just used a regular association instead. This is a “has-a” relationship, an aggregation, where Order has to have an OrderDetail to be “complete”. A library needs books to be a library, but books in general has no relation to a library, they can exist on their own. So the perspective is from the Order class, not the OrderDetail class.
If the student mention “own”, “is-a”, “partent-child” then they don’t understand this question.
It tells the developer that Order is not fully complete without at least one OrderDetail. The relation, from the point of view of Order, is stronger than a general relationship.
There is no points for telling that it is an aggregation in itself, that point was given in question 1E.

3. [4 points] What is the 1st law of software engineering, and why is that important?
Answer:
Software engineers are willing to learn the problem domain.
To solve a problem, you need to understand the problem.
The 2nd law, which is not the correct answer in this case, is that the software should be written for people first.

4. [6 Points] In light of your experience building a medium-sized program with a team, discuss briefly the key ideas in the "No Silver Bullet" article by Brooks. Be sure to discuss at least two of Brooks’s ideas.

Answer:
Brooks mentions:
Complexity; Conformity; Changeability; Invisibility or intangible; build vs buy; requirements refinement and rapid prototyping; incremental development – hence grow, not build, software; great designers.

5. [6 Points] List the stages of the software development lifecycle. Describe each stage in one phrase each.

Answer:
Requirements, Design, Implementation, Testing, Deployment and Maintenance.

6. [6 Points] Describe briefly what distinguishes the "agile" approaches to software development from more traditional approaches (like waterfall).

Answer:
Difficult to predict cost
Cross cutting concerns often come late
Heavily depend on refactoring
User centric focus
Architectural significant requirements may emerge late in the project.
Does not match many companies procurement models
Good if you have a stabile base
Good if the customer and domain is new and emerging
Good if you have a web based application that is already stable, and driven by internal forces – e.g finn.no, facebook.com etc.
Risk ownership is shifted from supplier to customer.

7. [6 Points] Choose three of these design principles. For each, either describe it briefly or give an example that illustrates it.

Single Responsibility Principle
Open-Closed Principle
Liskov's Substitution Principle
Dependency Inversion Principle
Interface Segregation Principle

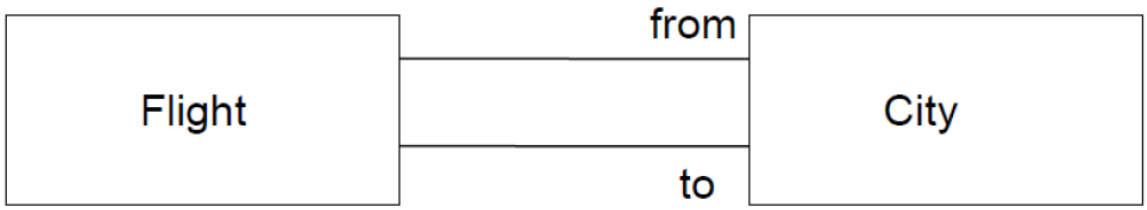
Answer:
SRP :
Every class/module should have only one reason to be changed the class/module should have only one responsibility.
If class/module "A" has two responsibilities, create new classes/module "B" and "C" to handle each responsibility in isolatio n, and then compose "A" out of "B" and "C".
Open-Close:
Every class should be open for extension (derivative classes), but closed for modification (fixed interfaces).
Put the system parts that are likely to change into implementations (i.e. concrete classes) and define interfaces around the parts that are unlikely to change (e.g. abstract base classes)
LSP:
Every implementation of an interface needs to fully comply with the requirements of this interface (requirements determined by its clients!)
Any algorithm that works on the interface, should continue to work for any substitute implementation.
Every subclass (or module) or derived class should be substitutable for their base or parent class.
DIP:
Instead of having concrete implementations communicate directly (and depend on each other), decouple them by formalizing their communication interface as an abstract interface based on the needs of the higher level class.

ISP:
A client should never be forced to implement an interface that it does not use, or client should not be forced to depend on methods they do not use --> segregate you interfaces.
Keep interfaces as small as possible, to avoid unnecessary dependencies --> only keep coherent methods in an interface.
Ideally, it should be possible to understand any part of the code in isolation, without needing to look up the rest of the system code.

8. [6 Points] When creating early UML use case diagram, UML has provided 3 stereotypes of classes that we have discussed in the course. What are they, and what are the purpose of each?

Answer:
Boundary (here I also accept interface, since we have used that term often in the lecture) – it is an outwards boundary between the system/component and its context.
Controller – the part of the system that is responsible for driving a use case.
Entity – responsible for holding information, like a database table.

9. [6 Points] Assuming you had this UML diagram, and you could autogenerate the code for this. Write the code for this UML Class diagram using C#.



Answer:

```
public class City{}

public class Flight
{
        City from;
        City to;

}
```

10. [15 points] For the following text:

1. Create a use case diagram

2. Create domain model

3. For two of the use cases related to a member, create the interaction diagram on level 2.

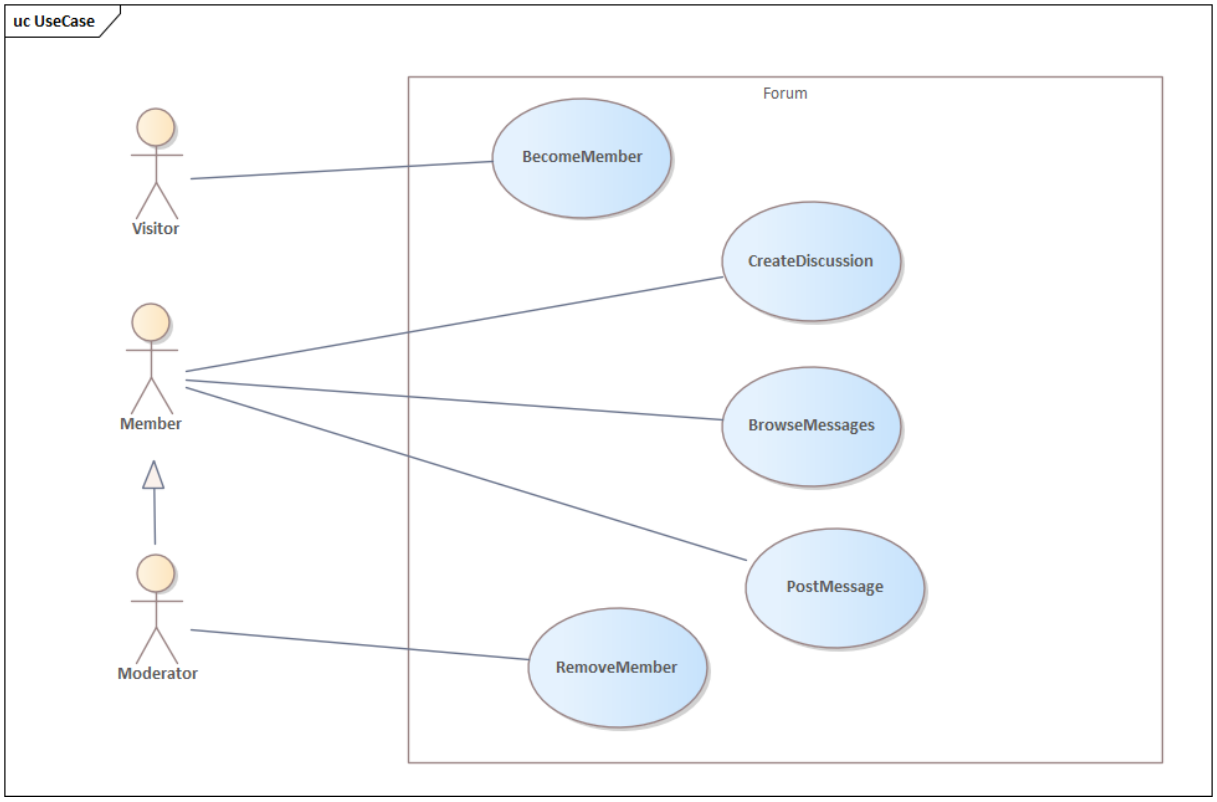
A forum is a web application that contains members and discussions. A discussion has a subject and contains messages. Each message has an author and content.

Visitors to the forum can join to become members. Members can create discussions, browse messages, and post messages to existing discussions. When a message is posted to a discussion, all members receive an email notification. A member has a name and an email address.

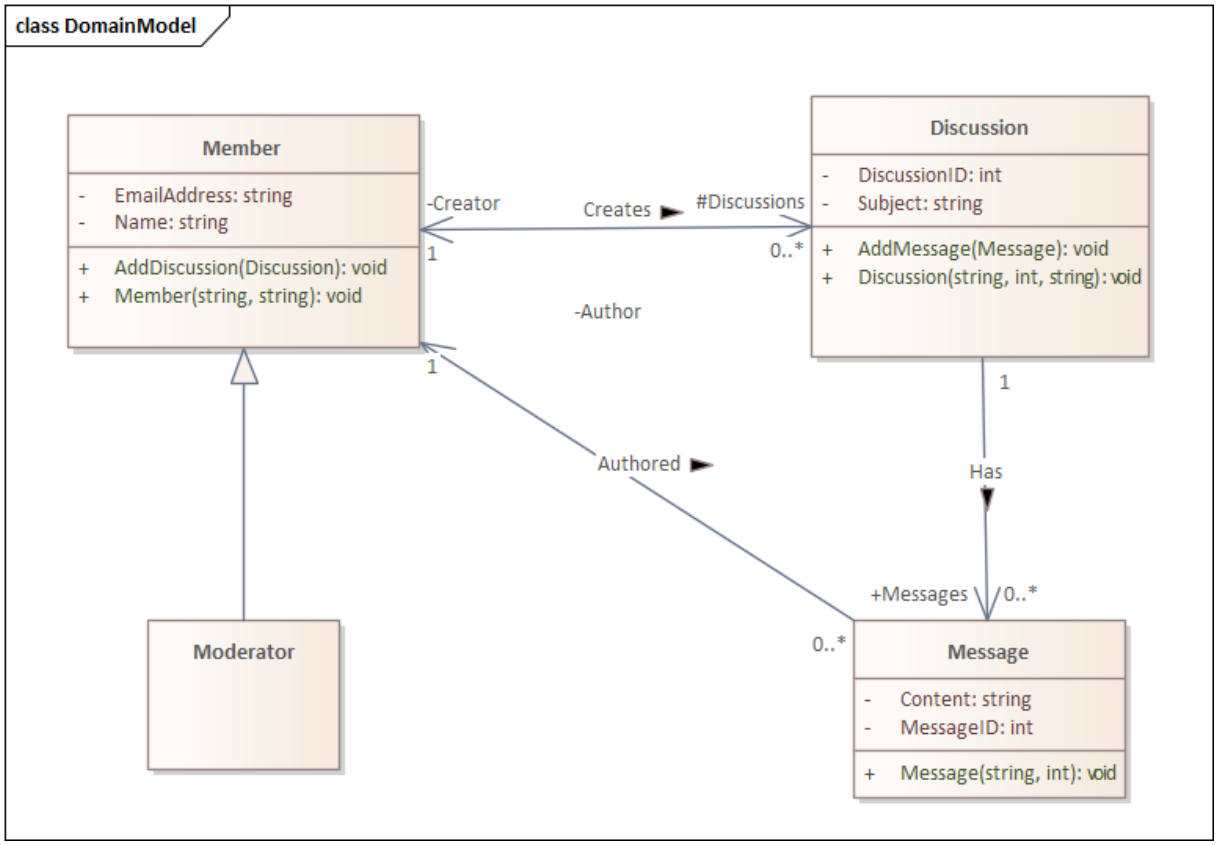
A moderator is a member who can remove other members from the forum.

Answer:

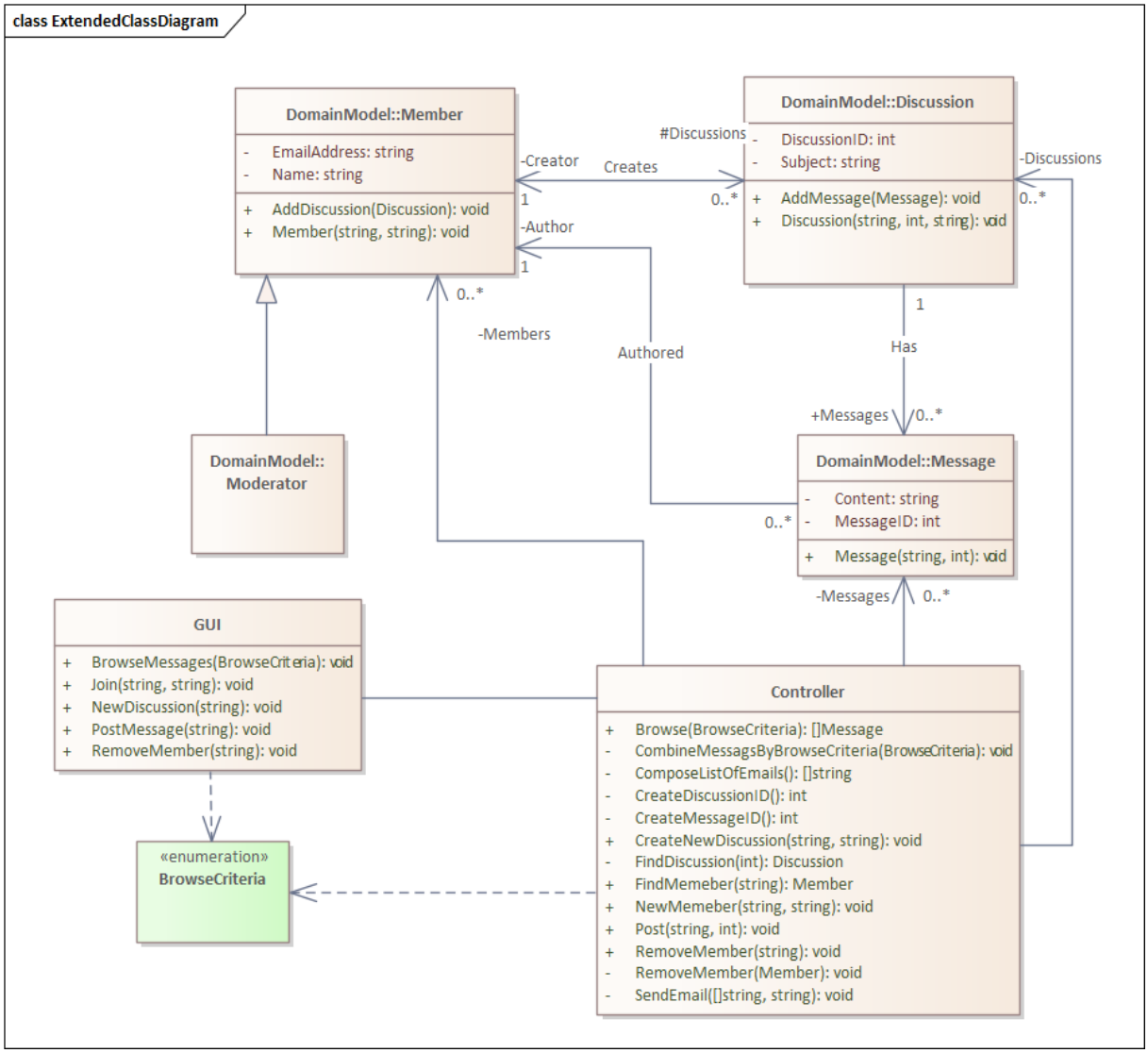
Use case



Domain model: (this domain model also includes methods, that is not a requirement)

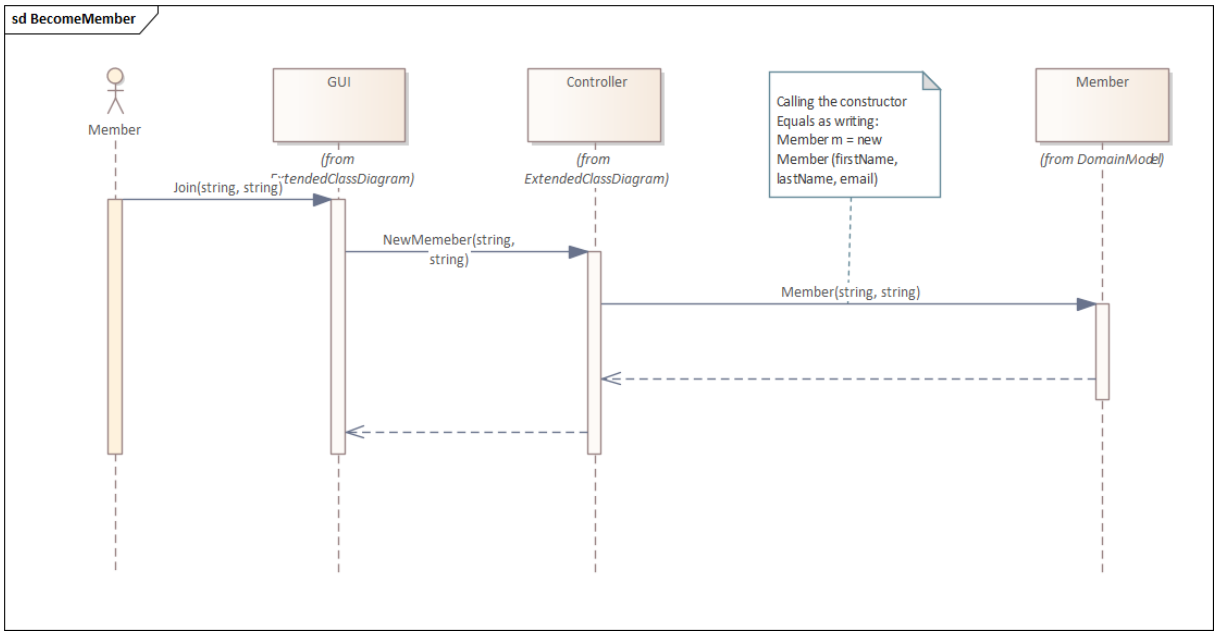


Extended class diagram, to better understand the interaction diagram. This diagram is not required.

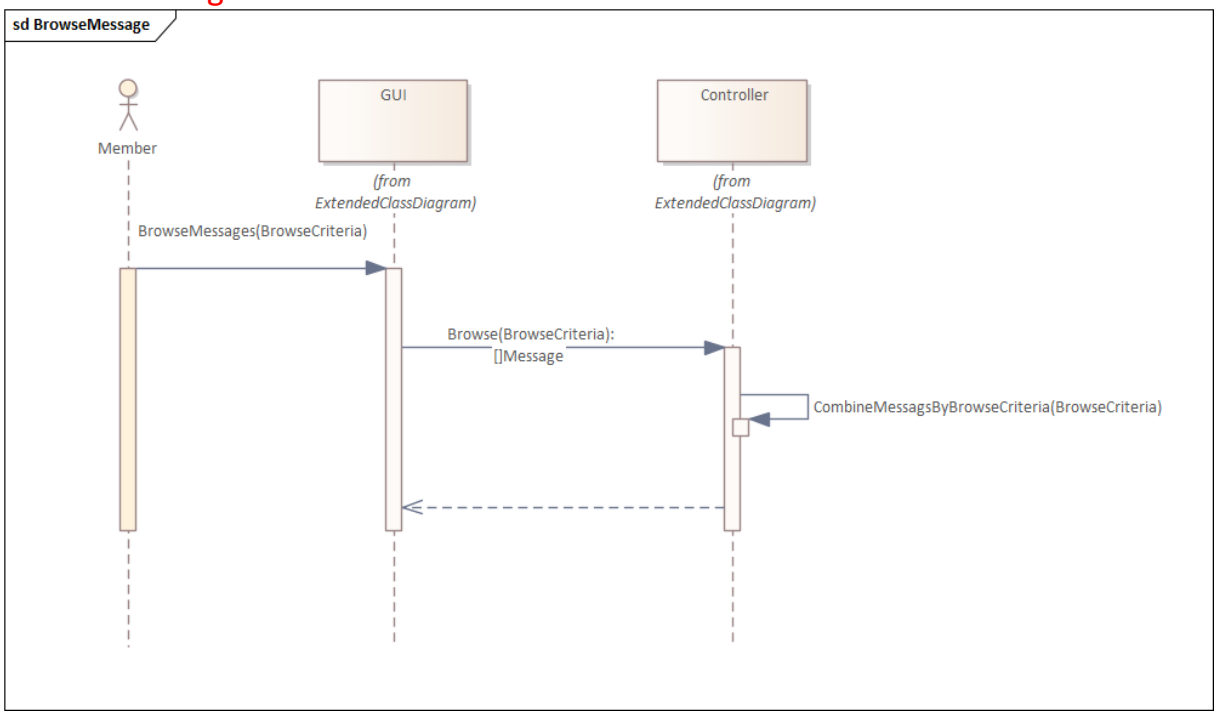


Interactions diagrams pr. use case:

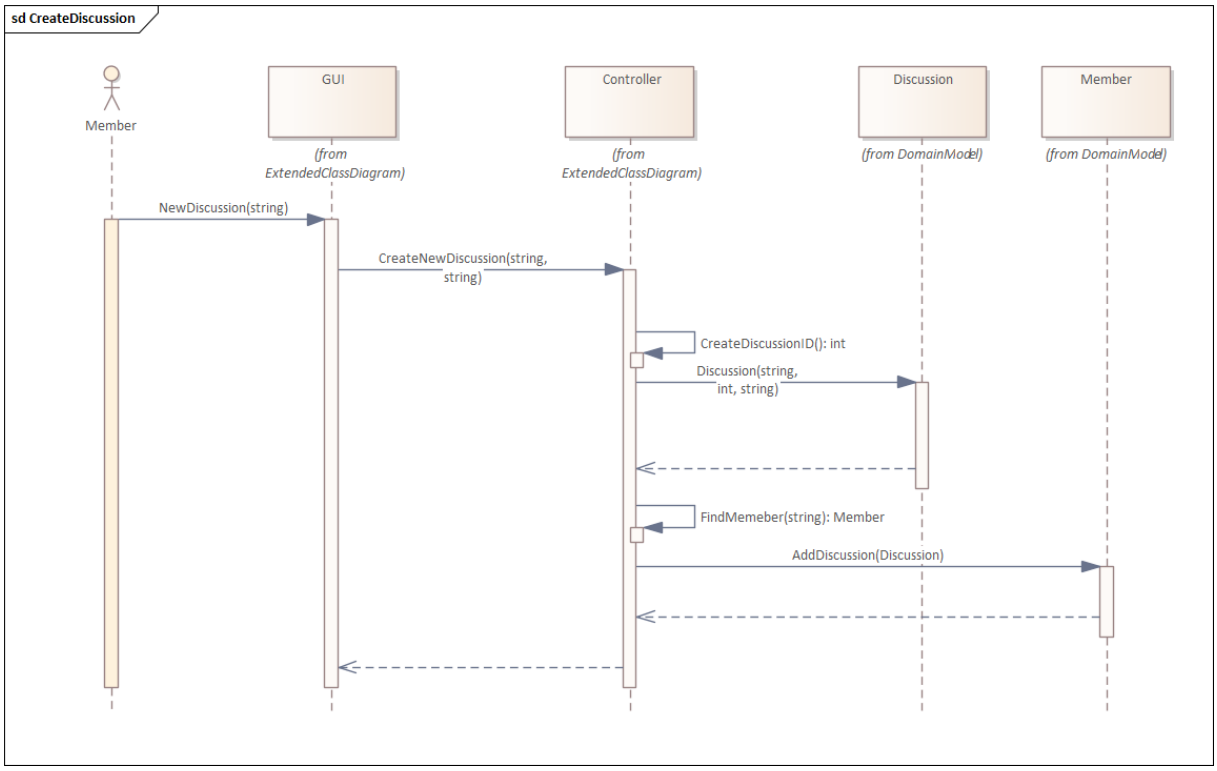
BecomeMember:



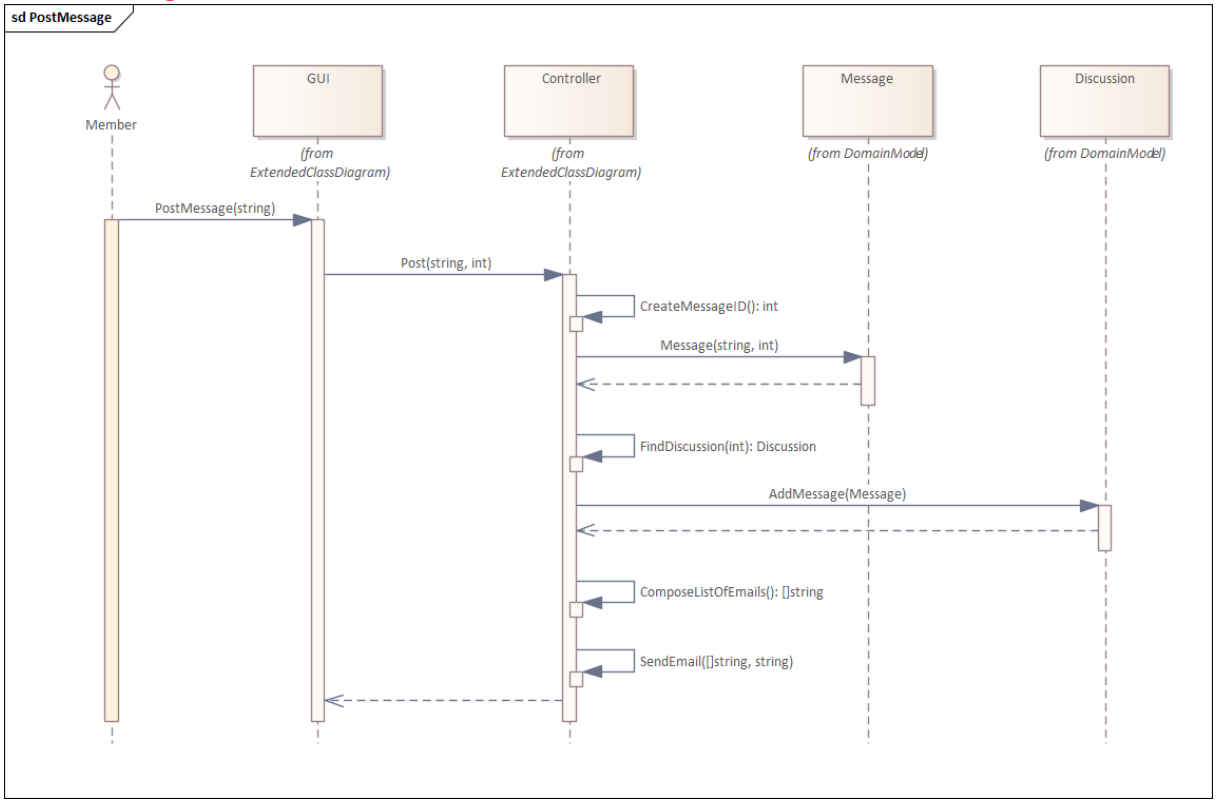
BrowseMessage:



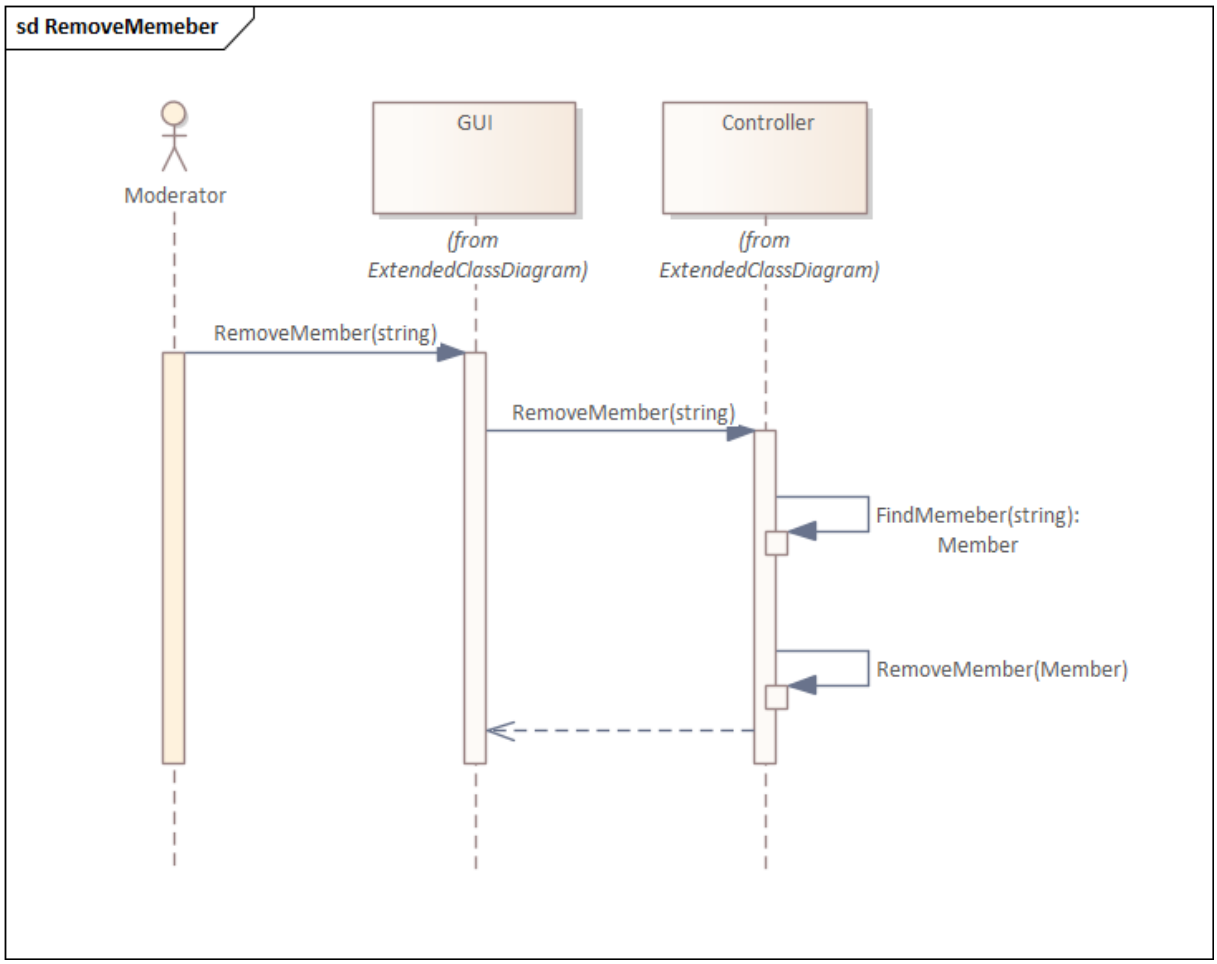
CreateDiscussion:



PostMessage:



RemoveMember:



11. [6 points] From the lab assignment for tax calculation. (a) What is one of the most important part of introducing the interface type, (b) in which part of the system was this interface placed, and (c) what was is prerequisite for this to work as intended.

Answer:

A) separating the implementation from the usage by hiding the implementation of the tax library. Decoupling, e.g. low coupling.

B) In the CalcuateTax library. In a separate library was also accepted.

C) That the Interface was public and all the methods in the interface is public. Using a factory (among others) was also accepted – since these concepts also was important.

Both question A and B was thoroughly discussed in class. And the project itself was often referred to in class. Question C was very difficult, but it was explained both in the assignment and in the lab. Further, it is not uncommon to have a difficult question to distinguish the good candidates (B's) from the really good

candidates (A’s).