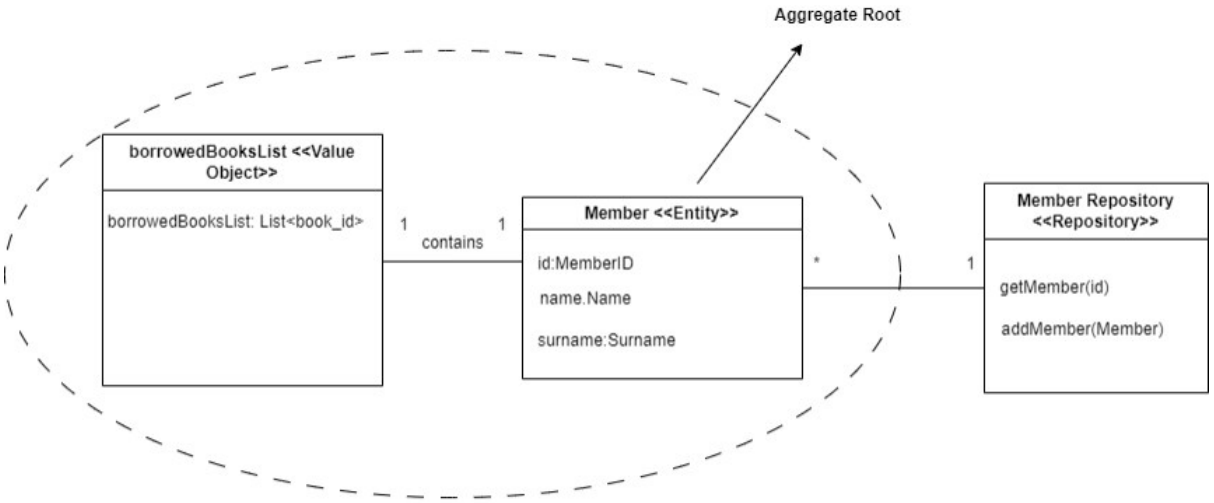
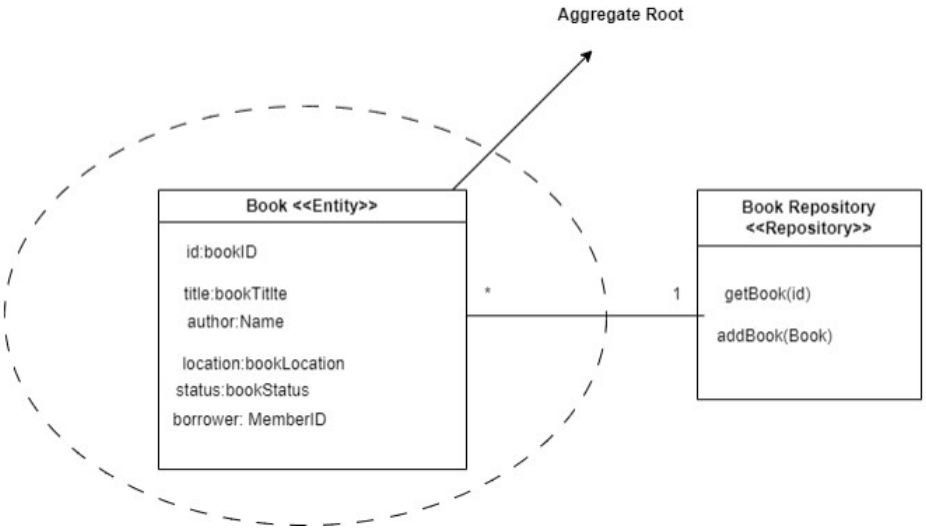
OBJECT ORIENTED ANALYSIS AND DESIGN HOMEWORK-4

Deniz Egemen Keneş – 05210000249
Ahmet Memiş – 05210000905

1)

Our domain model:





There are 2 aggregates in our domain model: the **Member aggregate** and the **Book aggregate**. **These aggregates communicate with each other through events**.
Inside the **Member** aggregate there are these value objects:
id, which holds members id,
name, which holds members' names,
surname, which holds members' surnames.
borrowedBooksList which holds the book id's of the books that the member has currently borrowed. Book ids can be appended to or removed from the list. The book id used for the list is referencing the book id value object from the **Book** entity.
All these fields are represented by value objects, not primitive types. By using value objects, we can validate the given values and represent the data in a more clear way.

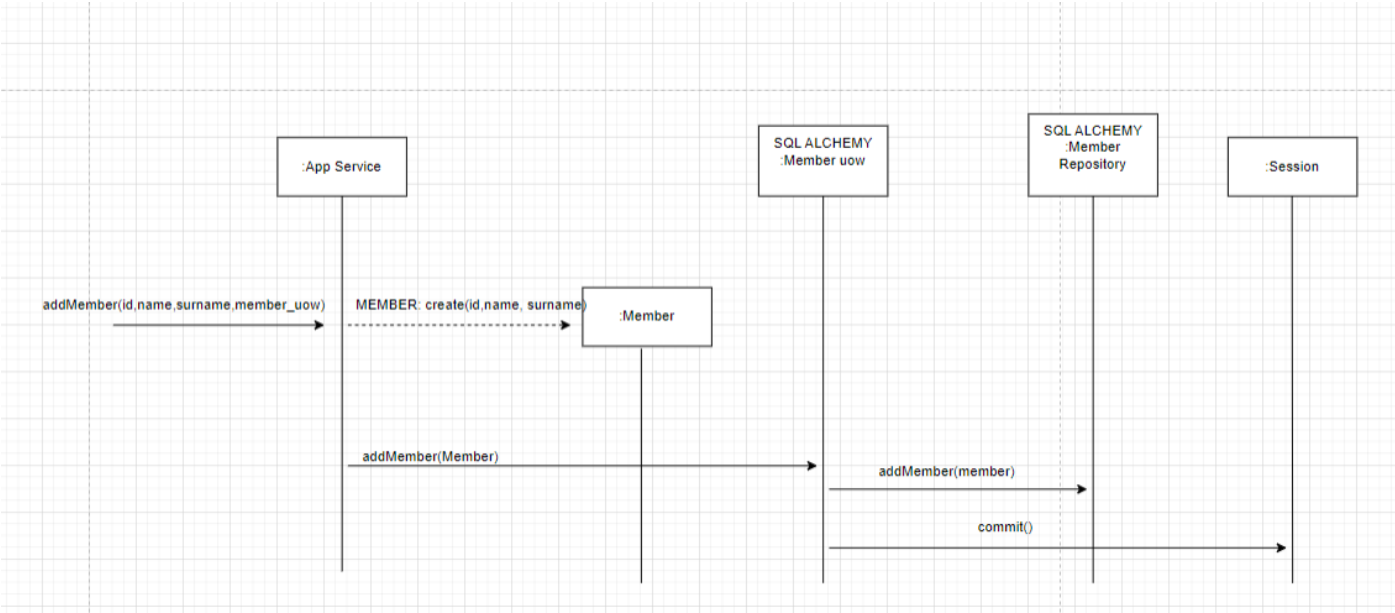
There is also the **member repository** which lets us use the database in an easier way by allowing us to use get and set methods, separating the database away from the domain model.

Inside the **Book** aggregate there are these value objects:
id, which holds the books id,
title, which holds the book title,
author, which holds the author's name,
location, which holds the physical location of the book inside the library,
status, which holds the current status of the book (whether or not it is borrowed, available etc.)
borrower, which holds the current borrower's member id (if it is not borrowed at the moment, it holds no member id). The member id used for the borrower value object is referencing the member id value object from the **Member** entity.
All these fields are represented by **value objects**, **not primitive types**. By using value objects, we can **validate the given values and represent the data in a more clear way**.

There is also the **book repository** which lets us use the database in an easier way by allowing us to use get and set methods, separating the database away from the domain model.

2)

<u>Use Case: Add a Member:</u>



Application service: We have assigned the **Controller** role to the application service because it is the first object beyond the UI layer that receives the inputs from the UI and coordinates a system operation (of adding a member by calling the **addMember** method.)

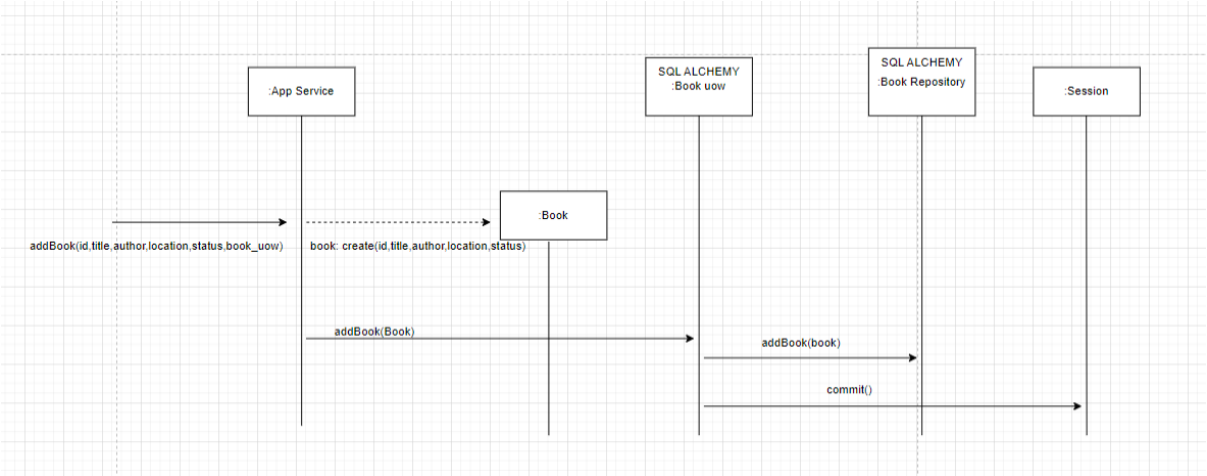
It is also a **Creator**, since it is the one who is creating the Member object. (by **creating** a new Member.) It is also an **Information Expert** because it knows about the repositories. Its main job is to control the overall flow of the program so it supports **high cohesion**.

Member: We have assigned the Information Expert role to the Member object. Because when being **created**, it has to validate and check its own input data based on the class requirements.

Member UoW: We have assigned the **Information Expert** role to the Member Unit of Work object because it is responsible for creating the transaction alongside the Member Repository. It also supports **high cohesion** because it forms a new layer of abstraction for the adding member use case by using the **addMember** method because it has only one responsibility for this use case.
It is coupled with the Member Repository because they work together to insert the new member into the database. There is **low coupling** between them.

Member Repository: We have assigned the role **Information Expert** to the Member Repository as Repositories are responsible for managing data access and manipulation, and they encapsulate the knowledge and logic related to interacting with a database.
It supports **high cohesion** because it inserts a layer of abstraction between the domain and the infrastructure. It is also **lowly coupled** with Member UoW as they are the ones responsible for properly inserting the Member into the database by using the repositories method **addMember**.

Use Case: Add a Book:



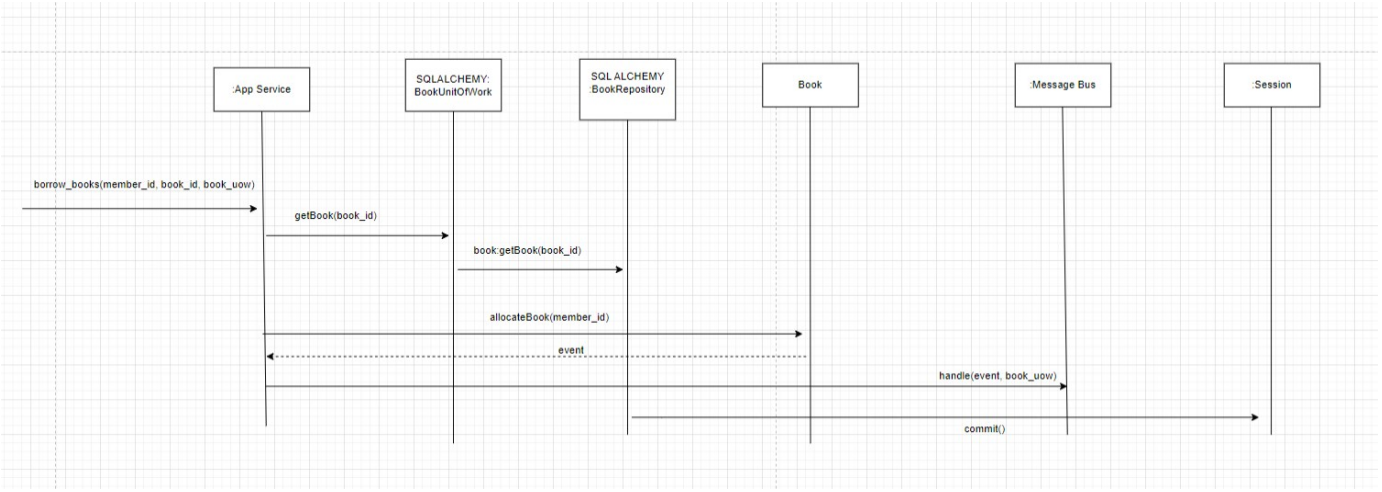
Application service: We have assigned the **Controller** role to the application service because it is the first object beyond the UI layer that receives the inputs from the UI and coordinates a system operation (of adding a book by using the **addBook** method.)
It is also a **Creator**, since it is the one who is creating the Book object.(By **creating** a new Book.) It is also an **Information Expert** because it knows about the repositories. Its main job is to control the overall flow of the program so it supports **high cohesion**.

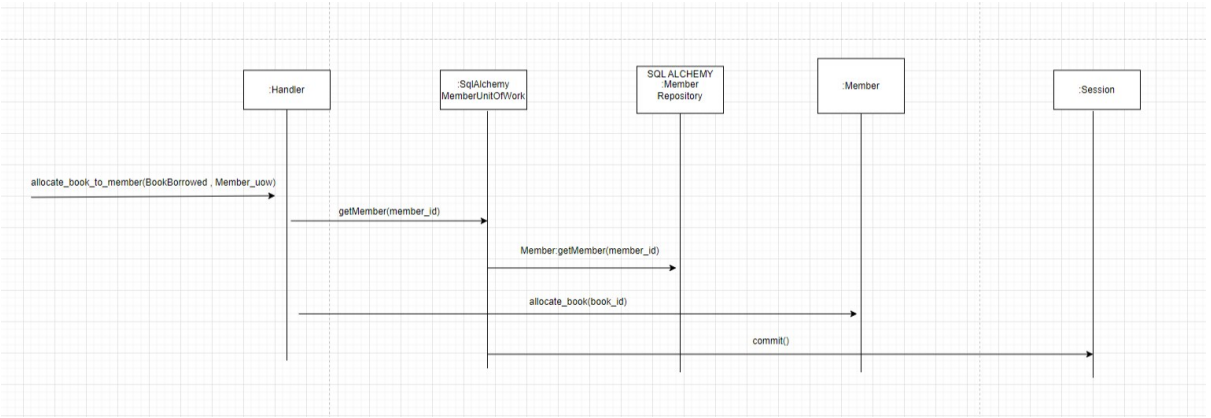
Book: We have assigned the **Information Expert** role to the Book object. Because when **being created**, it has to validate and check its own input fields based on the requirements.

Book UoW: We have assigned the **Information Expert** role to the **Book Unit of Work** object because it is responsible for creating the transaction alongside the **Book Repository**. It also supports **high cohesion** because it forms a new layer of abstraction between the repository and the object that is being added. It only has the **addBook** responsibility for this use case. It is coupled with the **Book Repository** because they work together to insert the new book into the database. There is **low coupling** between them.

Book Repository: Book Repository is designated as the **Information Expert**, acknowledging its responsibility for overseeing data access and manipulation while encapsulating the necessary knowledge and logic for interactions with the database. This assignment enhances **cohesion** by introducing a layer of abstraction between the domain and infrastructure, Its **low coupling** with the Member Unit of Work (UoW) is evident, as the **Book Repository** relies on the **Book UoW** to accurately insert the book data into the database by using the BookRepository method **addBook**.

<u>Use Case: Borrow Book:</u>





We have divided this use case's sequence diagram into two parts because the member updating process is separate from the book update process. It can be done asynchronously. That's why we decided to keep these two parts separate.

Use Case: Borrow Book (Part 1):

App Service: We have assigned the **Controller** role to the application service because it is the first object beyond the UI layer that receives the inputs from the UI and coordinates a system operation (of getting the book by using **getBook** method.)
App Service gets the event of the Book being borrowed/issued and sends this to the message bus with the method **handle()** with the **Book Uow.** It is also an **Information Expert** because it knows about the repositories. Its main job is to control the overall flow of the program so it supports **high cohesion**.

Book UOW: We have assigned the **Information Expert** role to the Book Unit of Work object because it is responsible for creating the transaction alongside the Book Repository. It also supports **high cohesion** because it forms a new layer of abstraction between the repository and the object that is being pulled from the database. It only has the **getBook** responsibility for this use case. It is coupled with the **Book Repository** because they work together to get the book from the database. There is **low coupling** between them.

Book Repository: Book Repository is designated as the **Information Expert**, acknowledging its responsibility for overseeing data access and manipulation while encapsulating the necessary knowledge and logic for interactions with the database. This assignment enhances **cohesion** by introducing a layer of abstraction between the domain and infrastructure, Its **low coupling** with the Book Unit of Work (UoW) is evident, as the **Book Repository** relies on the **Book UoW** to accurately get the book data from the database by using the BookRepository method **getBook**.

Book: We have assigned the **Information Expert** role to the Book object. Because when **being created**, it has to validate and check its own input fields based on the requirements. And it

Message Bus: By using a message bus object, we provide **high cohesion** and **low coupling** in our system.
High cohesion is provided because the message bus is only responsible for a certain number of specific functionalities.
Low coupling is provided because it is supposed to receive and send messages/events. It needs sources and destinations for this purpose.

Use Case: Borrow Book (Part 2):

Handler: We assigned the **Information Expert** role to the Handler object because it coordinates the flow of the member update part of the use case. It supports **High Cohesion** by creating a layer of abstraction for the member update part of the use case.

Member UoW: We have assigned the **Information Expert** role to the Member Unit of Work object because it is responsible for creating the transaction alongside the Member Repository. It also supports **high cohesion** because it forms a new layer of abstraction for

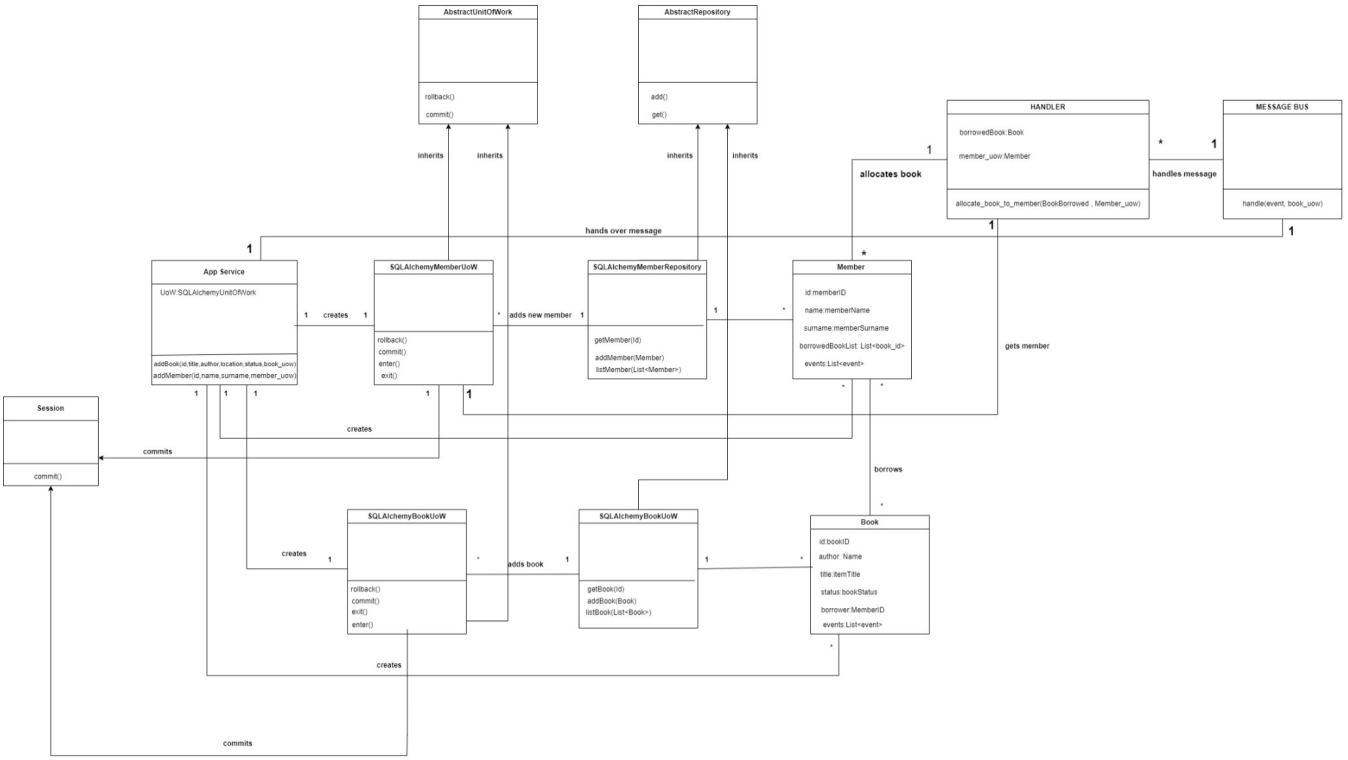
retrieving the member by using the **getMember** method because it has only one responsibility for this use case.
It is coupled with the **Member Repository** because they work together to retrieve a new member from the database. There is **low coupling** between them.

Member Repository: Member Repository is designated as the **Information Expert**, acknowledging its responsibility for overseeing data access and manipulation while encapsulating the necessary knowledge and logic for interactions with the database. This assignment enhances **cohesion** by introducing a layer of abstraction between the domain and infrastructure, Its **low coupling** with the Member Unit of Work (UoW) is evident, as the **Member Repository** relies on the **Book UoW** to accurately get the book data from the database by using the BookRepository method **getBook**.

Member: Member is designated as the **Information Expert** because it can perform its designated task of appending the borrowed books book id to the borrowedBookList object by using the **allocate_book** method.

3)

The class diagram:



4)

We have included the implementation in the file.