INFORMATICS AND MEDIA, UPPSALA UNIVERSITY

Project work - Library System

Object-Oriented Programming II

Mattias Nordlindh & Görkem Pacaci

Table of Contents

Introduction

It's time to put your programming skills to good use. You will create a library system that will handle books, loans, inventory, members etc. e.g. a complete library system. You will have a lot of freedom in how you will develop this application but you need to fulfil the requirements and guidelines stated for the library system described in this document.

The project is a little different than the previous labs, as it doesn't have step-by step instructions. This document contains the overall specifications of the project and you work on it according to your own plan.

Project Description

The library system has some requirements that the application must support, those are presented in the section "Requirements". Also there are technical requirements that you must follow while implementing your solution, these are described in the section "Technical Requirements".

Requirements

This section describes the requirements or operations that the library system must support.

High-level requirements for the features of the solution are:

- Keep a library catalogue (the list of books in the library) and the individual book copies of each book.
- Keep record of library members, and the books they have on loan.
- Keep a record of all loans, both current loans and loans previously returned.
- Keep track of which book copies are on loan and which are still available in the shelves.
- The GUI will need to stay updated at all times.
- The application should be stable i.e. not crash by user actions.

To be more precise, the software is required to support these actions:

- Book and Book Copies
    - List all the books in the library catalogue.
    - List all the books currently available in library (still have book copies not on loan).
    - List all books by a specific author.
    - Add new books to the library catalogue (a book needs to have an author).
    - Add new copies to a book in catalogue (when new copy of the same book has been bought).
- Loans

    - Record that a book goes on loan.
    - Record that a book has been returned from loan.
    - It should be clear in the GUI which books are on loan, which is returned and which are currently overdue.
    - List all loans by a specific member.
    - A single lending is for 15 days, no extensions allowed.
    - During return of a book, calculate if there is a fine and how much.
        - If a loan is past its due date, there is a 10kr per day fine.
- Members and Authors
    - Add a new member to the library.
    - Add a new author.

Code quality requirements

Your code should follow some quality standards:

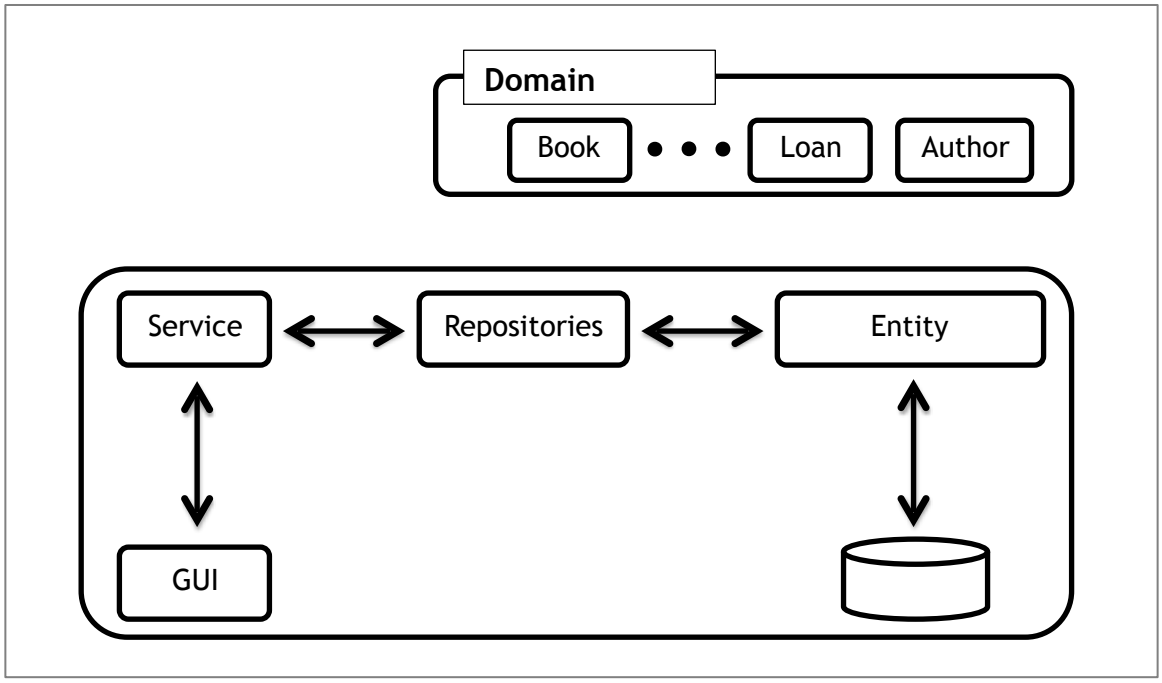
- Every type (class, interface, enum) has an XML comment giving its description
- Every public member (methods, properties, events) of a type has an XML comment giving its description and descriptions of its parameters (if applies).
- Naming standards are followed: https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/names-of-type-members
- General standards are followed:

  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

Technical Requirements

The application will be built using .NET and C#, the GUI will be built using Windows.Forms, the data storage should be a database and the data access should be done by using LINQ-to-Entities and Entity Framework. Do not use client-side filtering of data (via loops, etc). All filtering should be done by LINQ expressions.

The application should utilize different layers to separate concerns and lower the coupling between different parts of the application. The application should use both service and repository layer/pattern, the architectural design of the system should be as described in the conceptual model below.



Your GUI objects (forms) should be accessing the methods of your service objects, and the service objects should be accessing the methods of the repository. Finally, the repositories should query and save the objects to the database by using Entity Framework.

To summarize:

GUI: Forms, or form-based code that gets input from the user, uses the service classes to query/manipulate data. Subscribes to the Updated events on the services so when a change happens it refreshes the relevant parts of the GUI. The GUI classes never deal with the LibraryContext (EF) directly.

Services: Executes the queries requested by the GUI. Your LINQ code resides here in the services. A service does its job by interacting with the corresponding repository. When a service manipulates the data using a repository, it lets the GUI know by raising the Updated event. The service objects never deal with the LibraryContext directly.

Repositories: Fetches/writes objects from the database through the LibraryContext class. It responds to requests from the service layer and either fetches and returns data, or calls SaveChanges() on the LibraryContext to save the current changes.

Domain Objects

The following domain objects and their properties need to be defined in the application:

- Book
    - A unique Id (the ISBN could be used as the primary key if you feel like it, though there is some extra work involved to ensure that the ISBN number is unique for each book)
    - ISBN number (like ISBN 978-3-16-148410-0)
    - A Title
    - Description

    - A book should have one author
    - A book could have zero or more book copies
- Book Copy
    - Unique ID
    - A book copy is associated with a book
    - Condition (a number from 1 to 10, 1 being the worst)
- Loan
    - Unique ID
    - Time of loan
    - Due date
    - Time of return
    - A loan is associated with a specific book copy
    - A loan is associated with a specific member
- Member
    - A unique membership ID
    - A personal id (personnummer)
    - Name
    - Membership date
    - A member could have none or more loans
- Author
    - Unique ID
    - Name
    - An author can write several books

Repositories

A repository is a low-level component that resides very close to the data access, in the conceptual model above you find its place between the service-layer and the data access done by Entity Framework. A repository has the responsibility to have CRUD-operations (Create, Retrieve, Update and Delete) for a specific domain object. The repository-classes should connect to the database through Entity Framework using LINQ statements. All data operations should go through these repository-classes

You need to create a repository for each domain object in the application. For instance the BookRepository-class will have the responsibility of creating, retrieving, removing and

updating book-objects. Any service-class that needs access to or modify book-objects need to go through the BookRepository-class.

The generic interface described below defines CRUD-operations and must be implemented by all repository-classes in the application to ensure correct behavior. The type-parameter "T" is the type of the domain object the repository will work with and the type-parameter "Tid" is the type of the unique identifier (primary key) for that domain object. For example, the Member repository can be declared with Member for T and int for Tid, where int is the type of the ID property of a Member.

```
interface IRepository<T, Tid>
{
    void Add(T item);
    void Remove(T item);
    T Find(Tid id);
    void Edit(T item);
    IEnumerable<T> All();
}
```

Services

A service is a high-level component that will hold the majority of the business logic for the application. Services use repositories as a way of dealing with reading and persisting data, the services will not be dependent on what kind of data-storage the application is using, they only depend on the repository-classes.

As for the repositories, you need to define a service for each of the domain objects, for example you will need to create a BookService-class. All the service-classes should implement the IService-interface defined below.

```
interface IService
{
    event EventHandler Updated;
}
```

Your GUI will need to stay updated at all the time; this should be handled with the usage of events. All the service-classes expose an event named Updated (see IService-interface above) that will be raised when the service class change the underlying database e.g. for the BookService when a book is added, removed or edited. Let your GUI-objects (forms) listen to the Updated-event and whenever this event is raised, let the GUI-objects refresh all information associated with the data storage using the methods in the service-classes. This way, you do not have to manually update the GUI-objects every time you do a modification in the database. With the help of the event mechanism the GUI-objects will be able to update themselves whenever necessary. The GUI-objects should only update the necessary GUI-controls when an Update-event is raised; therefore it is not a valid solution to have a single method that updates all GUI-controls.

Note that, if you have multiple forms, you should avoid creating multiple instances of a single kind of service. For example, you should have just one instance of the BookService. The problem is, if you have multiple services, each will have its own Updated event. As a

result, while some forms are listening to one event, others will listen to another, so one will not hear about a Book object another added, and vice versa. To avoid this, have one instance of each service in the main form, and pass that instance to the other forms using a parameterized constructor.

The services are the correct place for high-level methods like MakeLoan(..), BooksWrittenByAuthor(..), LoansPassedTheirDueDates() etc. These kind of high-level methods make up for the core logic of the application and we are encapsulating them in the service-layer. Of course the services also need to implement some of the simple CRUD-operations like add, find, all etc, though this differs from service to service so implement only the ones that is needed.

Repository factory

We use the factory pattern to handle the instantiation of our repositories in the application. Below is simple implementation of a RepositoryFactory-class that we should use in the project and this class is an example of an implementation of the factory pattern.

Note that the RepositoryFactory has only a single LibraryContext object, and whenever it creates a new repository, it uses the same context.

```
// This class handles instantiation of the repositories.
public class RepositoryFactory
{
    ...

    // Retrieve a book repository instance.
    public BookRepository CreateBookRepository()
    {
        return new BookRepository(context);
    }

    // More repositories..
}
```

Entity Framework

Entity framework (EF) is an Object Relational Mapper (ORM) that makes it easier to store objects in relational databases. There are three ways of using EF: Database first, Model first or Code first. In this project work you should use the Code first approach. That means that you define regular classes in your solution and EF will afterwards generate a database with appropriate tables, relations, primary keys, required fields etc.

There is a separate document on the course page named “Entity Framework.pdf” that gives detailed instructions on how to setup and use EF in your project.

How to start

In the initial project, we’ve prepared a `vertical slice` of the program for you to observe. There is a listbox where some example books are shown, and a button where we edit the

title o a book as an example. You do not need to keep this listbox and the button in your final application, these are there just as an example of the layered architecture. Debug the code for this button and observe how GUI calls service, service calls repository, repository calls the database. Also look at the LibraryDbInit file to see how these initial books are created in the database.

Examination

The assignment needs to be accepted by one of the teachers of the course. It's not only the source code that makes up for a passing grade, the student also needs to be prepared to explain and answer questions in regards to his or her solution.

The assignment will be evaluated by these criteria:

- Fulfilment of all base and technical requirements of the specifications.
- How appropriately you use Object Oriented concepts.
- How you follow the different design principles discussed in the course.
- Usability and Visual efficiency of your UI forms and GUI design.
- Follow the code quality standards.

After gaining a pass on the assignment the student should hand in the entire solution at the course page. The name of the compressed-file should follow this model:

Surname_FirstName.zip e.g. Nordlindh_Mattias.zip

Or if you've worked as a pair:

Surname_Firstname_Surname2_Firstname2.zip

If you worked in pairs, one person uploads the project and the uploader MUST write the name of the second person in the file name, as well as a note.