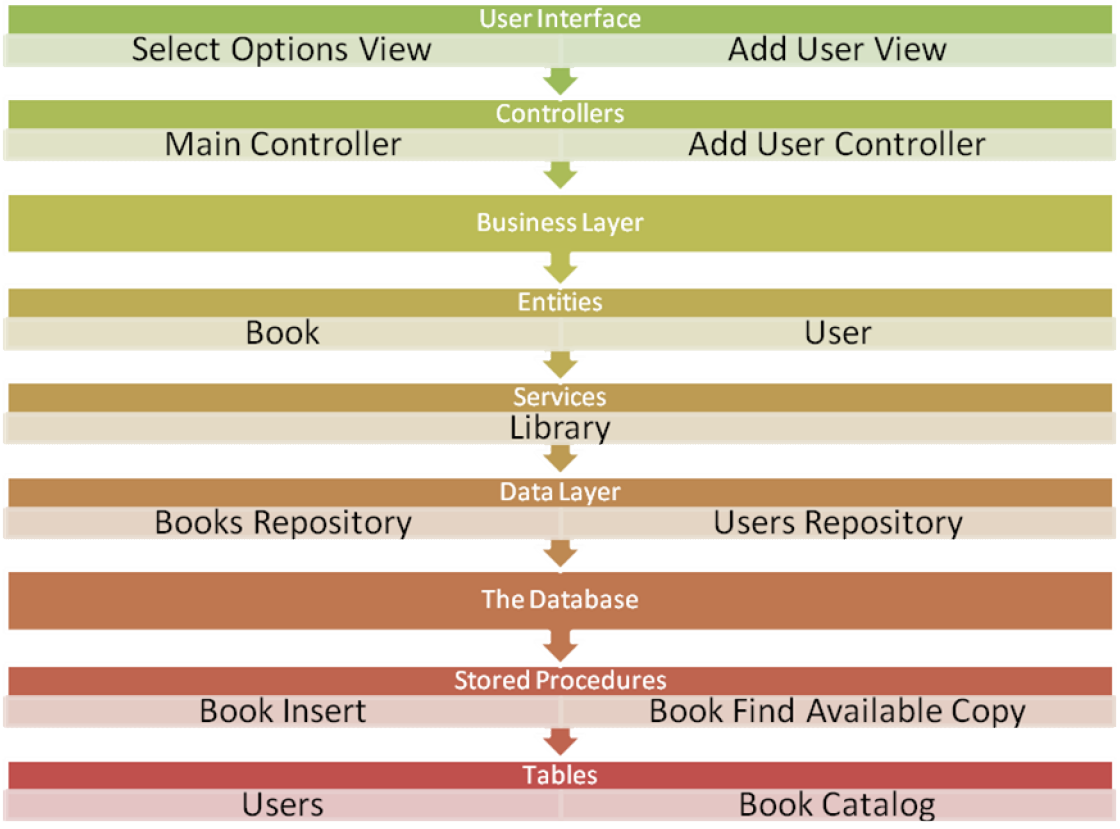
The Library Application

The book store application is a C# 2.0 client application, which aims to help The Librarian manage the book collections, users and checking books in and out of the library.

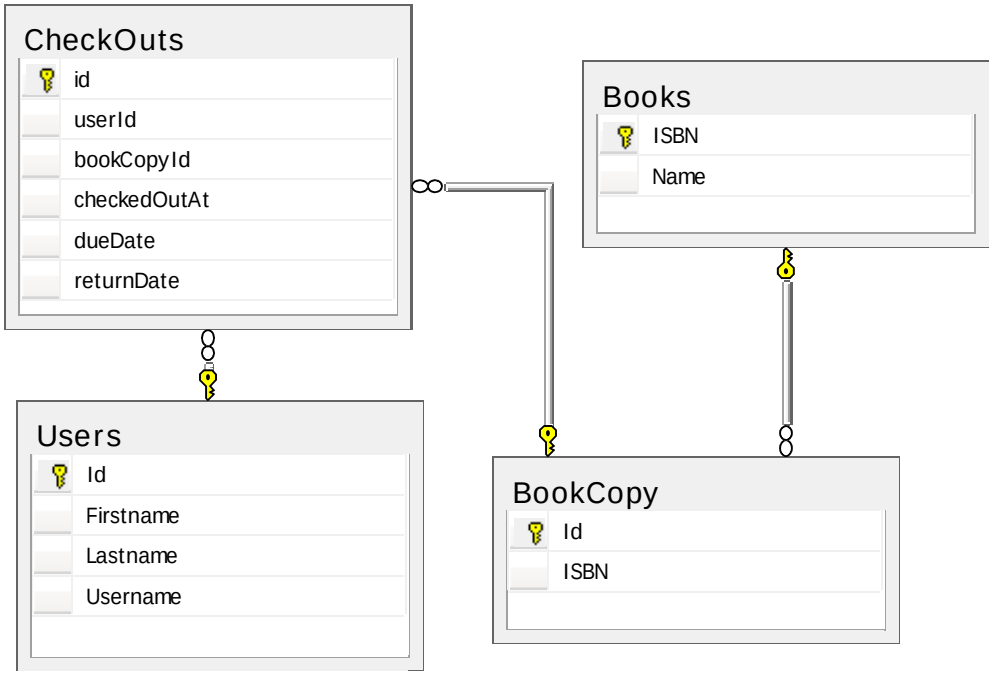
In recent lessons, we have built the data layer and some of the application layer. Since we have covered quite a bit of ground, I wanted to stop with an overview of what we have done so far.

First, let us take a look at our application as it stands today. In the diagram below you can see the application layers as it stand today. We will get to the implementation details shortly.



The database

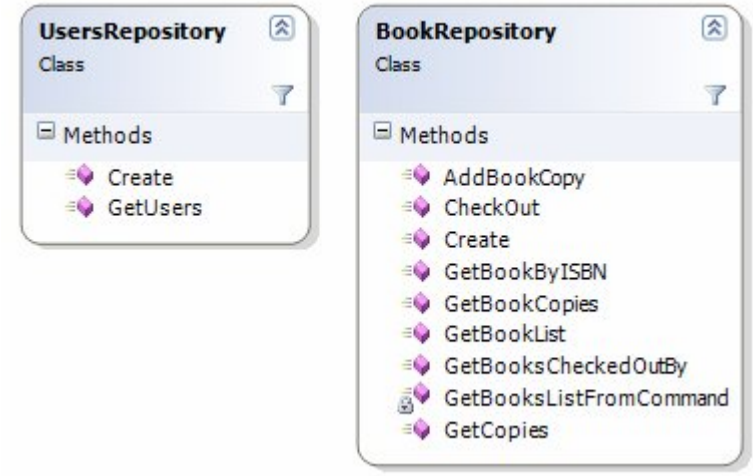
We have the following tables, which means that we need to use some joins in order to get some of the data (list of books names checked out by user, for instance):



The data layer

We have two classes that are responsible for talking to the database, the Users Repository and the Book Repository. The repositories will perform various operations on the database, such as adding / searching for information.

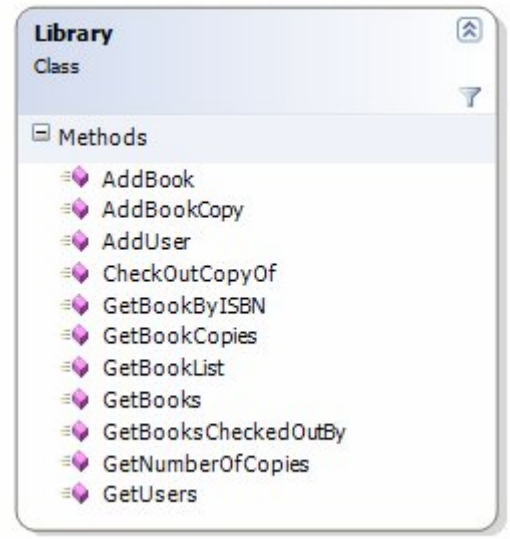
> *Note, the discussion of the With.Transaction pattern, used inside the repositories, is at the end of this document*



Note, that some methods, such as BookRepository.CheckOut do not have a fully working implementation.

The services

We actually have only a single service, the Library. The library is the class that we go in order to do operations in the application, it aggregate the various repositories and mostly forward the calls to the repositories.

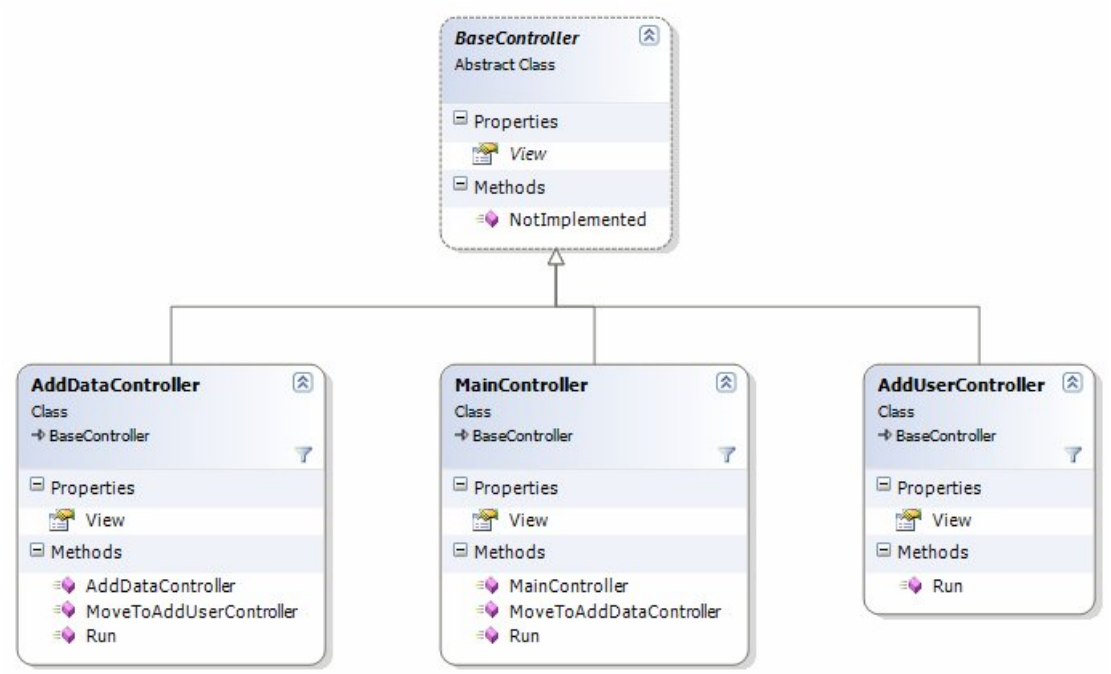


Note, in a real application, we would have the library do much more. For instance, the method Library.CheckOutCopyBook should find out if a copy is available on the library for the specified book, get its id, and then reserve it for the specified user.

The Controllers

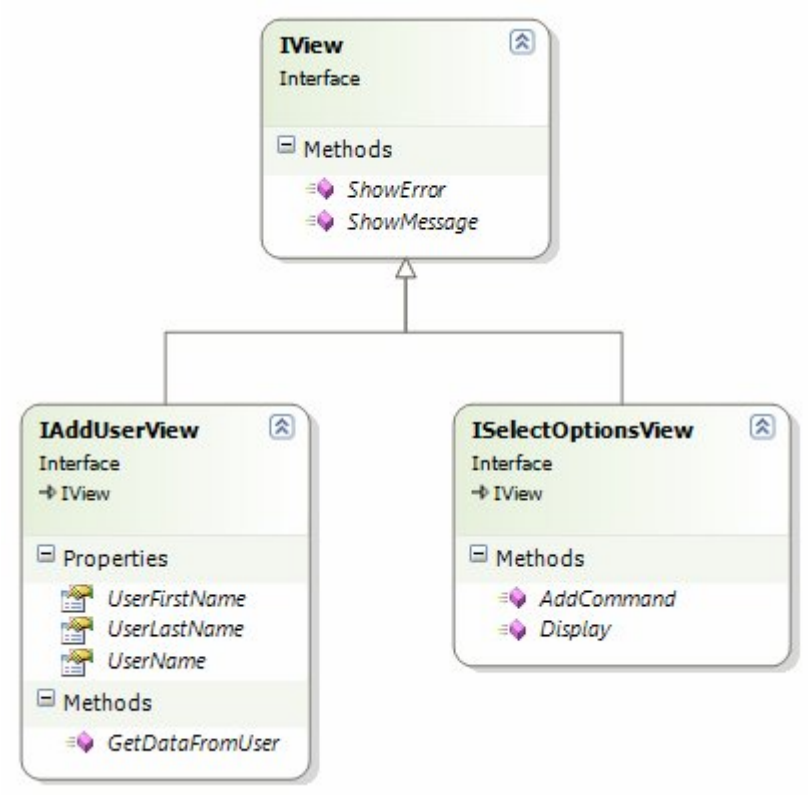
We have a hierarchy of controllers in the application, the most important one is the BaseController.NotImplemented method, which print "do your homework". Notice that the main controller will forward information to the AddDataController, and that the AddDataController will forward it to the AddUserController.

Each controller has a different view, and work independently with the view to build the relevant use case.



Note, we are going to perform some generics tricks today with the BaseController.View interface.

The User Interface



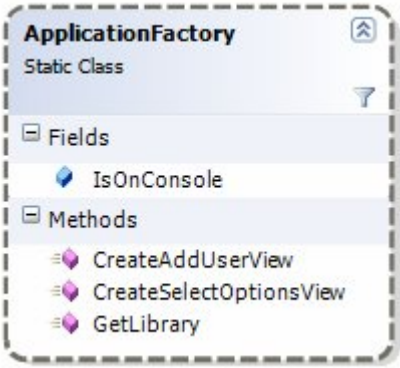
Here we have the views *interfaces*. We will discuss the implementation strategies in a bit. What we can see is that the IView interface exposes general view methods, and the ISelectOptionsView is a general menu-like view, which is used by several controllers, the IAddUserView is focused on a specific use case, and is here to show how to do a non generic view.

Using the IAddUserView is simply calling GetDataFromUser and then extracting the data from the views properties.

```
view.GetDataFromUser();
User user = new User(view.UserName, view.UserFirstName,
view.UserLastName);
ApplicationFactory.GetLibrary().AddUser(user);
```

The Application Factory

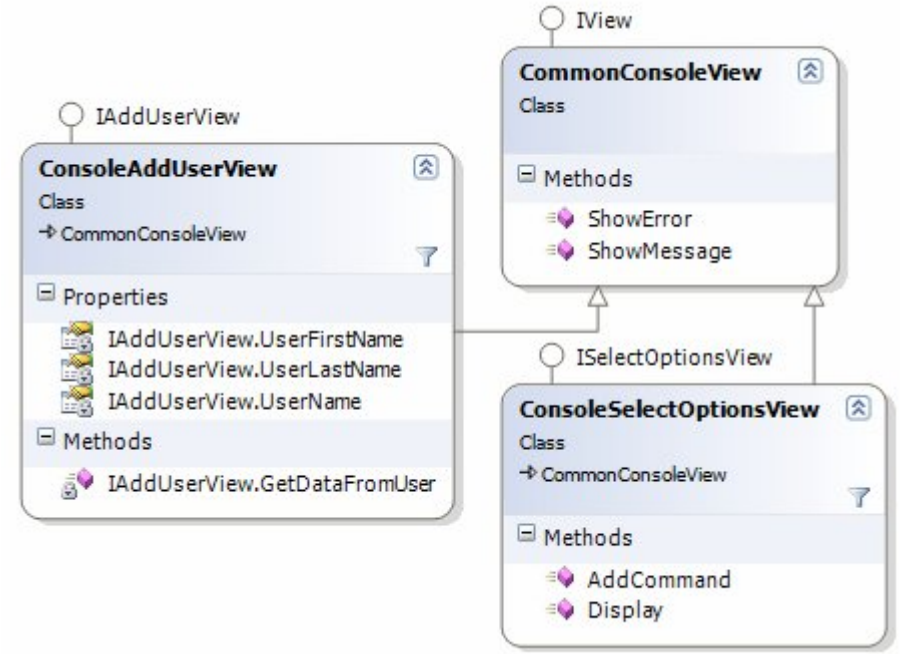
We don't create the Library or the views directly; we go through the Application Factory in order to do this. The reason behind that is that we would like to replace some of the implementations without affecting the rest of the application. (See next section)



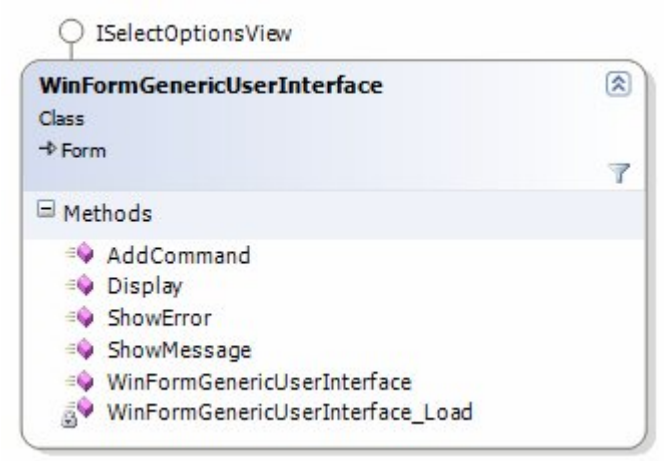
Replacing the UI layer

We have seen a short demo of how we can transparently replace the UI layer from console UI to WinForms UI without any code changes to the controllers.

Here is the console UI:

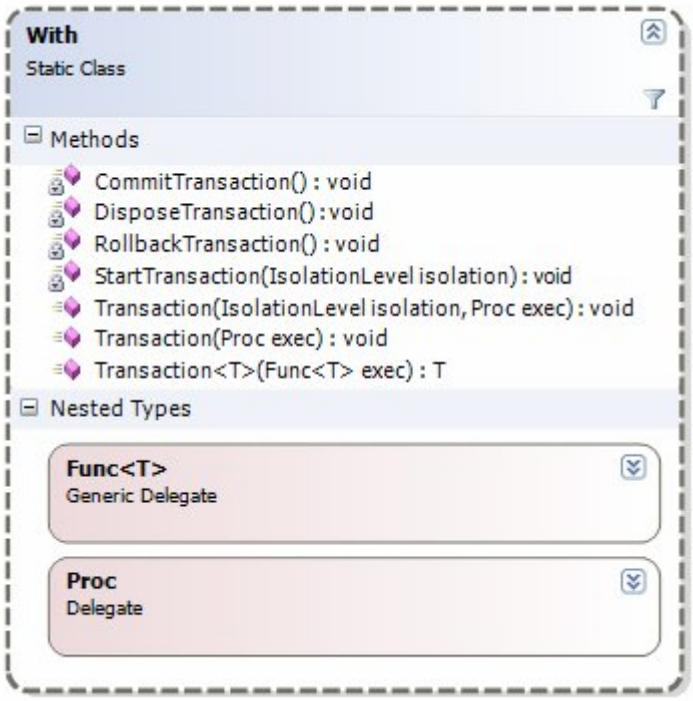


And here is the demo of the same UI using Win Forms, note that we don't have a base class yet, because we have created only a proof of concept of changing the UI in the meantime.



Utilities: With.Transaction

This approach allows us to reduce code duplication and repetitive code, as well as get more control on the way we handle transactions and connections.



Sample code:

```
public void AddBookCopy(Book book)
{
    With.Transaction(delegate(SqlCommand command)
    {
        command.CommandType = System.Data.CommandType.Text;
        command.CommandText = "insert into BookCopy (ISBN) values(@ISBN)";
        command.Parameters.AddWithValue("isbn", book.ISBN);
        command.ExecuteNonQuery();
    });
}
```

What we want the User Interface to look like

1) Add Data
    1) Add User
    2) Add Book
    3) Add Copy Of Existing Book
2) Checkout a book
3) Return Book
4) Search Book
    1) Update Details
5) Search User
    1) Update Details
6) Reports
    1) Books Checked Out By User
    2) Copies By Book
    3) Late Returns
    4) Fines