HW #8

Complete By: **Wednesday April 9th @ 9:00pm**

Policy: **Individual *or* teams of 2**

Submission: **electronic via Blackboard**

Assignment

The previous homework (HW7) focused on building a data-driven GUI application, accessing a Netflix database using SQL. If you look back on this application, it's probably a mess of GUI + C# + SQL code. How much of that code is redundant? How easily can you switch from one database file to another? Or imagine if you had to build a different client-side user interface, such as a console app or a web app. How easily could you extract and reuse the database access code?

In this homework exercise, you're going to redo your solution to HW7, taking a much more disciplined approach. We'll discuss the approach in class, but I'll also summarize here in the next section. We will provide the design, along with a skeleton code implementation. Your job will be to fill in the skeleton, using bits and pieces of code you have already written. Then you'll need to rewrite your GUI to use the new approach.

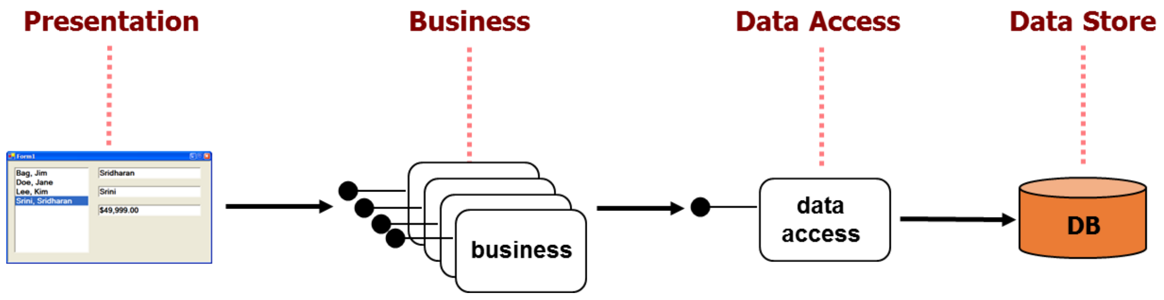
Our solution to HW7 is available on the course web page for your review and use. If you decide to use our GUI, note that a skeleton is provided for HW8, with all pieces in place and ready for modification. The solution to HW7 is provided if you need help with the SQL; if you want to use our solution in *this* assignment, start from the skeleton provided under "HW8" on the course web page.

Finally, notice that you can work alone on this assignment, or with a classmate. It's entirely up to you. If you are new to object-oriented programming, it would be smart to partner with someone who has OOP experience :-)

N-tier Design

Data-driven applications, whether on a phone, laptop, or a web server, are generally built from at least 2 layers of software: the **front-end user interface** that interacts with the user, and the **back-end data access** code that interacts with the data store (typically a database). Since most organizations offer a variety of apps — imagine a bank that has customer-facing software as well as employee-facing software — the standard design consists of four layers or **tiers**:

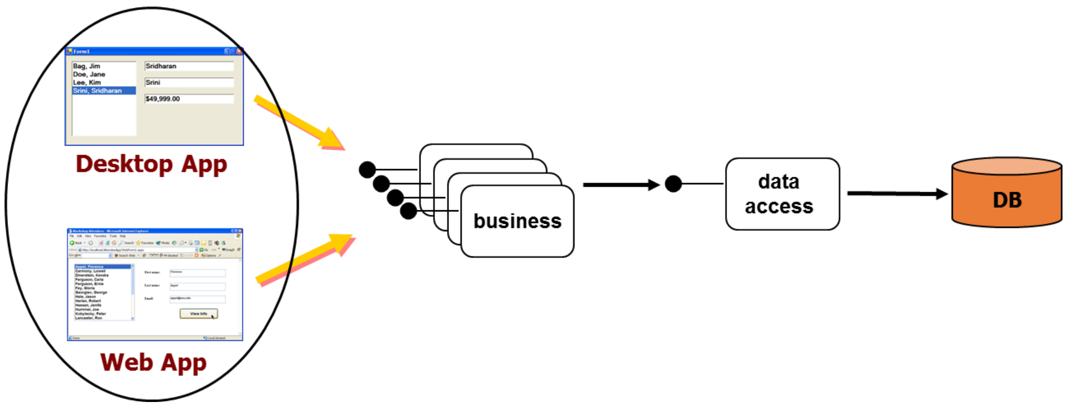
<< see diagram on top of next page >>



Here's a summary of what each tier does:

1.	**Presentation**:	interacts with the user
2.	**Business**:	supports the security, business rules, and data processing needed by this particular application. [*What's a business rule? Imagine a sales app; there would be rules about how to charge sales tax, who gets discounts, etc.*]
3.	**Data Access**:	interface between business tier and data store — the data access tier does not manage nor store data, it only facilitates access to the data
4.	**Data Store**:	actual data repository

This approach has numerous benefits. For example, if 2 different apps access the same data store, they can use the same Data Access Tier (DAT) to access the data store. As another example, suppose you need to support the same app on both a desktop and a mobile device. Then you should be able to reuse the Data Access and Business Tiers, and need only build 2 different Presentation Tiers:



That's the theory anyway :-) But most organizations practice this approach to application design.

Assignment Details

The assignment is to redo your solution to HW7 — the Netflix database application — using an N-tier design. Your GUI is the Presentation tier, and the Netflix Compact Edition database is the Data Store. The remaining two tiers — Business and Data Access — are being provided as skeletons. You *must* implement these designs as given. You are free to add to the given design, but at the very least you must fully implement what is given. On the course web page, under HW8, you'll find the following files:

1. **BusinessTierLogic**
2. **BusinessTierObjects**
3. **DataAccessTier**
4. **DatabaseApp.zip**

The first 3 are C# files representing the Business and Data Access Tiers you need to implement; the 4th file is our GUI from HW7 + the Business and Data Access Tier files.

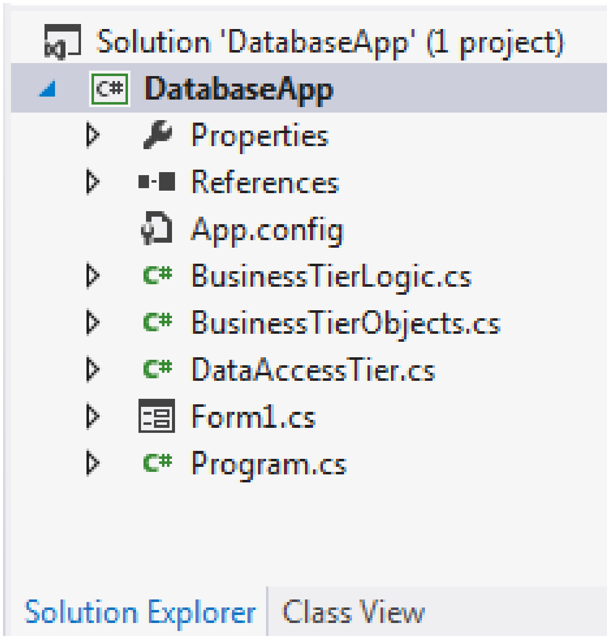
Getting Started…

The first step is to decide if you are going to work alone, or with a classmate. If you choose the latter, you need to email Sean and let him know. The second step is to decide if you will start from your own solution to HW7 (or your partner's), or if you will start from our provided HW8 Database App.

If you are starting from your own solution to HW7, then (a) make a copy of your HW7 project folder, (b) download files 1-3 mentioned above, (c) save these files into the same sub-folder as your other HW7 C# code, (d) open the project in Visual Studio, and (e) use the Project menu to *add* the existing files to your project.

If you are starting from out provided HW8 skeleton solution, then (a) download just file #4 mentioned above, (b) open the .zip file, (c) extract the folder "Database App" to your desktop, and (d) open in Visual Studio.

Once your project is open in Visual Studio and ready to go, your *Solution Explorer* window should look like this:



Notice the additional C# files: ***BusinessTierLogic***, ***BusinessTierObjects***, and ***DataAccessTier***.

Let's start with the *DataAccessTier*, which is the easiest to understand. This class is designed to execute SQL, passed as a string. There are 3 methods + a test method: the 3 methods correspond to the 3 types of queries we need: scalar, non-scalar, and action:

```
public bool TestConnection()
{
  //
  // TODO:
  //

  return false;
}

public object ExecuteScalarQuery(string sql)
{
  //
  // TODO:
  //

  return null;
}

public DataSet ExecuteNonScalarQuery(string sql)
{
  DataSet ds = new DataSet();

  //
  // TODO:
  //

  return ds;
}

public int ExecuteActionQuery(string sql)
{
  int rowsModified = 0;

  //
  // TODO:
  //

  return rowsModified;
}
```

The job of these methods is to open a connection to the database, perform the desired query, close the connection, and return the result. To call these methods, your Business Tier will need to create an instance of this class, and then call, like this:

```
DataAccessTier.Data  datatier;
datatier = new DataAccessTier.Data("Netflix-65K.sdf");

bool test = datatier.TestConnection();
```

The provided Business Tier code already creates an instance of the Data Access Tier; see the constructor in *BusinessTierLogic*.

The Business Tier is more complex, split across 2 files and containing a total of 7 classes. The functions you'll call from the GUI are contained in the file *BusinessTierLogic*, and the data transfer objects used to carry data back to the GUI are in the file *BusinessTierObjects*. Here are the 7 classes:

```
class Business
{ }

class Movie
{ }

class Review
{ }

class User
{ }

class Movies : List<Movie>    // inherits from / "is-a" list of Movie objects
{ }

class Reviews : List<Review>  // inherits from / "is-a" list of Review objects
{ }

class Users : List<User>      // inherits from / "is-a" list of User objects
{ }
```

For more details you'll need to look at the provided C# files. The first class — ***Business*** — is the only one you need to implement. The remaining classes can be used as given. To use the provided Business Tier, your GUI will need to create an instance of the *Business* class, and then call. For example:

```
BusinessTier.Business  businesstier;
businesstier = new BusinessTier.Business("Netflix-65K.sdf");

Movie m = businesstier.GetMovie(124);
```

Where to start? Read on…

Now What?

The idea is the GUI is written by calling the *Business Tier*. The *Business Tier* then creates the appropriate SQL, and executes this SQL by calling the *Data Access Tier*. The *Data Access Tier* executes the query against the database, and returns the results back to *the Business Tier*. The *Business Tier* then packages up the answer(s) in one or more data transfer objects — such as a Movie — and returns these objects back to the GUI. The GUI displays the results.

I would start by implementing the ***ExecuteScalarQuery*** method in the Data Access Tier. Then call this method from the ***GetMovie*** method in the Business Tier; the one that looks up the movie by MovieID. Then call this GetMovie method from the "Get Movie Name" button on the GUI.

Once that works, implement the ***ExecuteNonScalarQuery*** method in the Data Access Tier. Then call this method from the ***GetReviews*** method in the Business Tier; the one that looks up reviews by MovieID. Then call this GetReviews method from the "Get Reviews" button on the GUI.

Once you have those working, you should have a much better feeling for how the pieces fit together, and be able to complete the assignment. Recall that our solution to HW7 is provided if you need help with the SQL queries (or GUI programming).

Electronic Submission

If you are working with a classmate, make sure you have emailed Sean that you are working together. Then submit just one version of the program under one of your logins.

To submit, create a .zip file / compressed folder of your *entire* Visual Studio project folder. Then, using Blackboard, submit this .zip file / compressed under the assignment "HW8". We expect your C# code to be commented and readable, including a header comment at the top of your GUI code-behind file along the lines of

```
//
// N-tier Netflix database application.
//
// <<YOUR NAME(s) HERE>>
// U. of Illinois, Chicago
// CS341, Spring 2014
// Homework 8
//
```

You may submit as many times as you want before the due date, but we grade the last version submitted.

Policy

Late work is not accepted. In this assignment you may work individually, or with a classmate from CS 341. All work must be your own and that of your partner (if any).

Academic dishonesty is unacceptable, and all parties involved will be immediately subject to the official academic integrity review process. The University's policy is quite clear, and can be read here: http://www.uic.edu/depts/dos/studentconduct.html. In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, and allowing a tutor, TA, or another individual outside your team to write an answer for you.