# Introduction

## Problem

In this lab, you will expand your existing Windows Forms movie collection program to communicate with a database and properly handle errors.

## Solution

You will not be adding any new forms for this lab but you will be changing the behavior of the existing code.

Note: All screenshots are for demonstration purposes. You can make adjustments to meet your needs.

# Setting Up the Solution

As with previous labs, copy the entire solution (folder and all) into a new folder. This will allow you to reuse all your existing code but still keep it separate from previous lab assignments.

For this lab assignment you will need to connect to a SQL database. Visual Studio installs a local SQL database instance called LocalDB that you will use. The database is provided as a Visual Studio project that you can add directly to your solution.

1. Copy the MovieDatabase folder (extracted) to your solution folder.
2. In Solution Explorer right click the solution and select Add\Existing Project. Select the MovieDatabase project and click Open to add it to the solution.

# Simplifying IMovieDatabase Implementations

In this lab you will be creating another implementation of IMovieDatabase. To reduce the amount of code to be written and ensure a consistent implementation you will create a base implementation that enforces the rules of the interface and then update the existing implementation accordingly.

## Implementing the Base Class

1. Create a new abstract class called MovieDatabase that implements the IMovieDatabase interface. Note that it may be quicker to start by copying the existing implementation and then refactor it to be generic.
2. Ensure the abstract class is documented using the doctags.
3. Ensure the abstract class enforces all the existing rules.
4. Each public method should enforce the core rules and defer the actual implementation details to the derived types by defining a corresponding protected, abstract method. The following are suggestions. Others may be added as needed.
   1. Add => AddCore
   2. Get => GetCore
   3. GetAll => GetAlLCore
   4. Remove => RemoveCore
   5. Update => UpdateCore

## Update the MemoryMovieDatabase Class

Modify the existing implementation to use the new base class.

1. Change the class to derive from MovieDatabase instead of implementing IMovieDatabase.
2. For each method
   1. Rename the method to match the base classes abstract method.
   2. Change the accessibility to protected.
   3. Remove all the functionality that is already handled by the base implementation.

# Add Exceptions to the Movie Database

Now that exceptions are available, modify the MovieDatabase base class to throw exceptions when errors are detected.

1. Update the doctags to identify the errors that will be reported.
2. Update the existing implementation to throw exceptions based upon the rules defined by the interface.
3. Ensure the exception thrown is the correct type for the situation.
4. Ensure the exception thrown has an appropriate message.

Exceptions should be thrown based upon the following conditions.

## Add

* The movie being added is null.
* The movie is not valid as defined by IValidatableObject.
* A movie with the same name already exists.

## Get

* The ID is <= 0.

## Remove

* The ID is <= 0.

## Update

* The movie being updates is null.
* The movie ID does not exist.
* The movie is not valid as defined by IValidatableObject.
* A movie with the same name already exists (excluding the movie itself).

# SQL Database Implementation

The IMovieDatabase interface needs to be implemented for the SQL database. Create a new Class Library project in the solution called MovieLib.Data.Sql. The project will need a reference to the MovieLib project.

Create a new class (ex. SqlMovieDatabase) to implement IMovieDatabase via the new base class using ADO.NET to communicate with the database. Implement each of the methods following the rules of the interface. Ensure the class is properly documented using doctags.

Some guidelines about the implementation.

* Unlike the memory database implementation. Each call to this class will create new objects that are not tied to the version in the database. Therefore, it is not necessary to implement any sort of cloning.
* Errors can occur when communicating with a database. The implementation should, in general, not handle errors as that is the responsibility of the higher-level callers.
* The unique identifier of each movie is managed by the database. When adding a movie, the identifier will be calculated by the database. When retrieving a movie, it will be provided by the database. No calculations should be needed for this anymore.
* The new implementation should follow all the same exception rules as the memory implementation.
* Ensure the connection instance is properly cleaned up.

The following stored procedures can be used for the methods. Note the stored procedures will enforce most of the rules the code is enforcing but do not rely on that behavior.

|  |  |  |  |
| --- | --- | --- | --- |
| Stored Procedure | Parameters | Returns | Comments |
| AddMovie | @title – the title  @length – the length (as integer)  @isOwned – boolean flag  @description – the optional description | The Id of the new movie |  |
| GetMovie | @id – the ID of the movie | Id, Title, Description, Length, IsOwned | Returns nothing if the movie does not exist. |
| GetAllMovies |  | Id, Title, Description, Length, IsOwned | Returns back all movies. |
| RemoveMovie | @id – the ID of the movie |  |  |
| UpdateMovie | @id – the ID of the movie  @title – the title  @length – the length (as integer)  @isOwned – boolean flag  @description – the optional description |  |  |

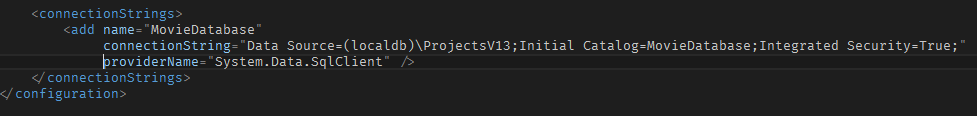
# Updating the Application

The main form will be updated to use the new implementation to manage the movies. Add a reference to the new MovieLib.Data.Sql project. Because it was already modified in a previous lab to use the interface, the only change that is needed is the call to create the instance of the field.

## Connecting the Database

The default connection string for the database project should use (localdb)\ProjectsV13. You can verify this by opening the project properties for the database project and going to the Debug tab. Under Target Connection String is the target database. If your connection string is different then it should still work correctly but you’ll have to mentally adjust any references to the connection string in the lab assignment.

To connect the application to the database project, do the following.

1. Copy the connection string from the database project (using the project’s properties if necessary).
2. Open the app.config file for the Windows project. If you do not have one yet then right click and add it (Application Configuration File).
3. Just before the ending </Configuration> element place the following code. the connection string should match what you copied earlier. the values are separated on multiple lines for readability purposes only. Case in this file matters so follow the casing exactly.  
     
   

## Creating the Instance

Unlike the previous implementations, the database implementation requires the connection string name (provided in the config file above). Since we do not want to use the database until we are actually showing the form, do the following.

1. Move the initialization of the field from the field initializer to the OnLoad method.
2. Pass the connection string (or name, depending upon how you implemented it) to the instance constructor so that it uses the correct database instance.

# Handling Errors

Now that the database will throw exceptions the UI needs to handle them so the application does not crash. When performing any call to the movie database field ensure that any exceptions are handled. Handled exceptions should be reported to the user. In most cases the user should remain in whatever UI they are currently at (i.e. the movie detail form).

# Requirements

Your program must meet all the following requirements.

* Compile cleanly with no warnings or errors.
* Meet all the points mentioned in the solution.
* Ensure each file has a file header indicating the course, your name and date.
* Solution, project and code should be uploaded to Github
* Submit lab in MyTCC by providing link to Github URL