Lab work - MovieService

Introduction

In the lab we will create an application that will handle movies and their categories. The lab will be focused around creating a service layer for retrieving movies and categories from some type of data storage. We will do two implementations of a movie-service that will work against different types of data storage. The first one will be an in-memory variant and the second will use a CSV-file as its data storage.

Concepts

In this lab the following concepts are discussed:

- Abstraction and abstraction layers
- Service layer
- LINQ (Language-Integrated Query)
- Lambda expressions
- Dependency injection

Instructions

Begin by download the already set up solution from the course page.

Create an in memory movie-service

Create a class “Movie”, this movie should have 5 properties:

Title	string
YearOfPublication	int
Rating	double
NumberOfUserVotes	int
Categories	ICollection<Category>

Create a class “Category”, let the class have 2 properties:

Name	string
Movies	ICollection <Movie>

Create another class, name it “InMemoryMovieService”. This is a class that will have the responsibility of retrieving Movies and their Categories from the data storage. In this class you will create **two** member variables as internal storage for holding movies and categories. For internal storage, you could use a collection class you consider appropriate. One example could be the List<T> class.

In the constructor of the InMemoryMovieService-class create a couple of Movies and Categories and add them to their corresponding internal storage. When a movie could have several categories and a category could have several movies some problems may occur in creating these hardcoded data-structures. A tip for making this easier is to declare a couple (3) movie variables and assign them new

Movie-objects, add values to all properties except the Categories-property. Then do the same for a couple (3) categories and don't assign the Movie-property. (In the constructor for both Movie and Category you should create and assign a new List to the Categories and Movies properties)

In the InMemoryMovieService create a method (see ConnectMovieAndCategory-method below) that adds a category-object to the collection of categories in a movie-object and also adds the movie-object to the collection of movies in the category-object. Implement the following method:

```
public void ConnectMovieAndCategory(Movie movie, Category category);
```

Now we have some movies and categories in an internal storage, implement the following methods in the InMemoryMovieService-class, they should query the internal storage using LINQ:

```
public IEnumerable<Movie> AllMovies();
public IEnumerable<Category> AllCategories();
public IEnumerable<Movie> MoviesFromCategory(string name);
public IEnumerable<Movie> MoviesFromYear(int year);
```

We have separated the logic of retrieving and filtering movies to the InMemoryMovieService-class. The classes consuming the service don't need to know about how the movies and categories are stored.

Create a simple GUI in the MovieServiceForm, let the form-class have a private variable:

```
InMemoryMovieService movieService = new InMemoryMovieService();
```

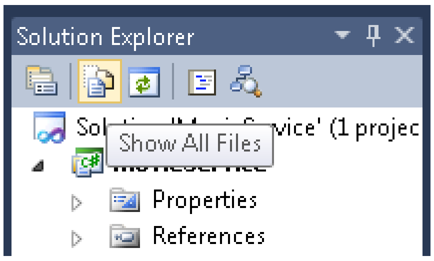
Use the service-instance to make the GUI display all movies and categories and have the functionality to retrieve movies from a specific category or year. Make the GUI as simple as possible, just enough to test that your methods are working correctly (Tip: build the GUI by using listboxes).

Introduce the interface IMovieService

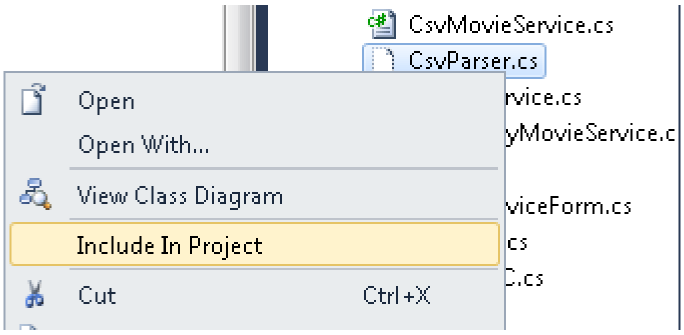
We have separated the concerns of storing and filtering movie data to the InMemoryMovieService-class but we still have a direct dependency to it from our form (MovieServiceForm). Create an interface called IMovieService and let it expose the four methods above. Make the InMemoryMovieService-class implement this interface (it already has the implemented methods so everything should work fine at this point). Now change the variable-declaration in the form from InMemoryMovieService to IMovieService. The form is still dependent on the actual concrete class (we will get to it) as it still instantiate it but now it can use any class that implements the IMovieService interface as its movie service (nothing will break because the interface exposes all methods that we have used from the actual service implementation).

Create the CsvMovieService

Ok, it's boring to work with such little data (assuming that you didn't have the stamina to create more than a couple movie- and category-objects). Let's create a service that will have a lot more data. Create a class named CsvMovieService and let this class implement the IMovieService-interface. This class will use a csv-file as the data storage but you will not be forced to struggle with parsing the csv-file yourself instead use the class CsvParser that is included in the project. To see it press the "Show all files"-button at the top of the Solution Explorer:



Then right-click the CsvParser.cs file and choose “Include In Project”:



In the constructor of the CsvMovieService use the CsvParser-class and retrieve the parsed data through the methods GetMovies and GetCategories (call the ParseCsv-method first!). Use some kind of internal data storage to hold the parsed data during the runtime of the application similar to the InMemoryMovieService. Also take a look at the ParseCsv-method and try to understand the parsing algorithm, the csv-file is located at path: project-folder/Files/Movies.csv.

Some rows from the csv-file:

```
Halloween: Resurrection;2002;3.9;13863;Horror,Thriller
Dog Day Afternoon;1975;8.1;65937;Crime,Drama
The Kids Are All Right;2010;7.3;26298;Comedy,Drama
Buffalo Soldiers;2001;6.9;14030;Comedy,Crime,Drama,Thriller,War
Shortbus;2006;6.6;15190;Comedy,Drama,Romance
The Forgotten;2004;5.7;28188;Drama,Mystery,Sci-Fi,Thriller
```

After reviewing the algorithm continue to implement all members of the IMovieService-interface. These implementations will be the same as the ones from the InMemoryMovieService so feel free to be inspired by those.

Now change the form-class to use the CsvMovieService implementation instead of the in-memory-service by this line:

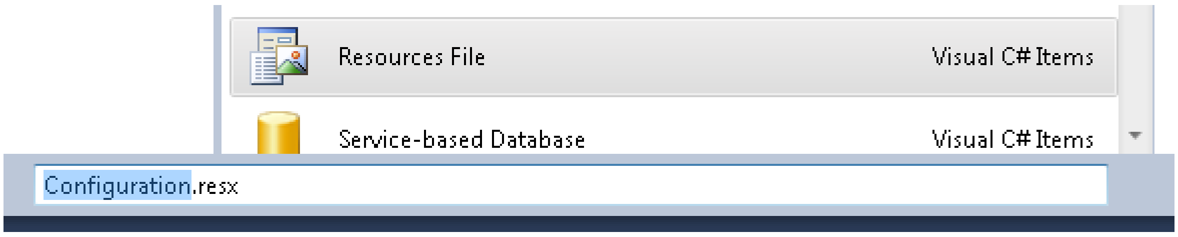
```
IMovieService movieService = new CsvMovieService();
```

Run your application and observe what happens.

We now have two different implementations of the IMovieService and we can use both from the form-class. But the assignment of the movieService-variable above shows we still have a direct dependency to the CsvMovieService even though we use an interface for declaring the variable. We could solve this by create a real simple dependency injection solution.

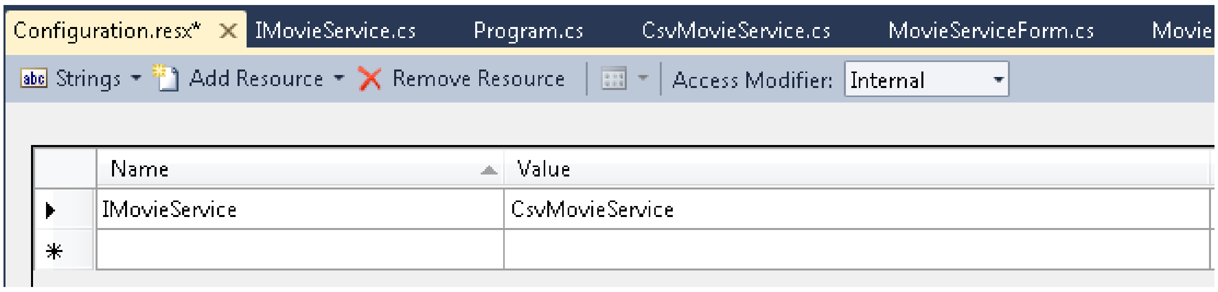
Include the SimpleDI.cs-file to the project in the same way as for the CsvParser.cs-file.

Add a resource-file to the project through Project -> Add New Item... Choose the Resources File and name it Configuration.resx:



Configuration.resx

A resource file consist of key-value-pairs, enter the following line in the Configuration.resx :



Name	Value
IMovieService	CsvMovieService

Resources are accessible by NameOfResourceFile.Name and will return the corresponding value to that name e.g. Configuration.IMovieService will return the string “CsvMovieService”.

Observe that in the static constructor to the SimpleDI-class the Configuration resource is used to register the current service (the one specified in the resource file). Change the assignment of the movieService-variable in the GUI-class to instead of always creating an instance of the CsvMovieService-class to instead use whatever IMovieService that is registered to the SimpleDI-class:

```
IMovieService movieService = SimpleDI.GetService();
```

The service assignment line above contains a micro example of a decoupling technique called ***'dependency injection'***. In this method, the direct reference to any of the service classes is replaced by a reference to an abstraction of the service instead. Thanks to the abstraction, a direct dependency is eliminated from the consuming class (the form-class) and this decreases the coupling between the concrete service-classes and the consuming class.

Now try to switch the name in the resource file between ”InMemoryMovieService” and “CsvMovieService” and see that without changing any code but only the content of the resource file that we can use different implementations of the movie service (don’t forget to build the application between the changes). Of course we can change between the different services during the runtime but this example tries to show explicitly that we don’t need to change the actual source code to get our program to behave differently.

Extend the CsvMovieService and IMovieService

We will ignore the InMemoryMovieService for the rest of the lab and you can exclude it from the project (right-click and choose "exclude from project"). You will get some compile errors that you need to fix by commenting/removing code. Now we just have one implementation of the IMovieService-interface and we will extend it with more functionality. Try figure out some different kinds of filtering, sorting, and aggregating that is possible and implement the methods by using LINQ and lambda-expressions. The important thing is to explore the different Standard Query Operators for LINQ and use sorting, projection, ordering and aggregation. If you don't come up with any here are some suggested methods:

```
IEnumerable<Movie> AllMovies();
IEnumerable<Category> AllCategories();
IEnumerable<Movie> MoviesFromCategory(string name);
IEnumerable<Movie> MoviesFromYear(int year);
IEnumerable<Movie> MoviesAfterYear(int year);
double AverageUserVotes();
int TotalUserVotes();
IEnumerable<Movie> TopMovies(int numberOfMovies);
IEnumerable<Movie> WorstMovies(int numberOfMovies);
IEnumerable<Movie> LessUserVotesThen(int votes);
IEnumerable<Movie> Top5UserVotes();
IEnumerable<Movie> FilterMovies(Func<Movie, bool> expression);
```

Now update your GUI to reflect some of your newly added methods.

Examination

Handing in is done at the PingPong course page, in the form of a compressed ZIP file containing the whole solution. The name of the compressed-file should follow this model:

Surname_FirstName.zip   e.g. Nordlindh_Mattias.zip

If you work in pairs both of you need to hand in the zip-file on the course page and also add a comment who in the comment field who you were working with.

Extra work

1. Create a paginated list for showing data in smaller chunks.
2. Change your service to not only have retrieve functionality but also to persist data. Define your own data storage structure for your data, like: text, csv, xml, database (will not work if you're working on the school clients because you don't have permission to create databases). Then define read- and store-procedures in the movie service so that it can persist movie data as well as retrieving it.