MbmStore – mandatory assignment 1

This is the first of two mandatory assignments in backend programming that focus on the development of a website for *MusicBookMovieStore*.

The web site must present *music-cd*, *book* and *movie* catalogues for customers and enable online shopping.

In this assignment, you are supposed to build an early prototype with the *basic domain classes*. In the assignment, you must demonstrate that you are able to write and instantiate classes and display data objects inside an ASP.NET MVC web application.

At this stage of the development process, data are stored as *non-persistent sample data* inside the program itself, and there will be no administration part with HTML forms for data input and maintenance of music-cd, book and movie data.

The layout and design are not important in this assignment. In assignment two, we'll focus more on design and add e-shop ("basket") functionally.

The default view of the website – that loads when you run the project – must have links to the webpages that are part of the solution.

Hand-in and deployment

You can do the assignment individually or in small groups of max 2-3 persons. You must upload the assignment to Canvas as a `.zip` file containing the whole solution or to GitHub (or similar), that enables me to download and run your project locally.

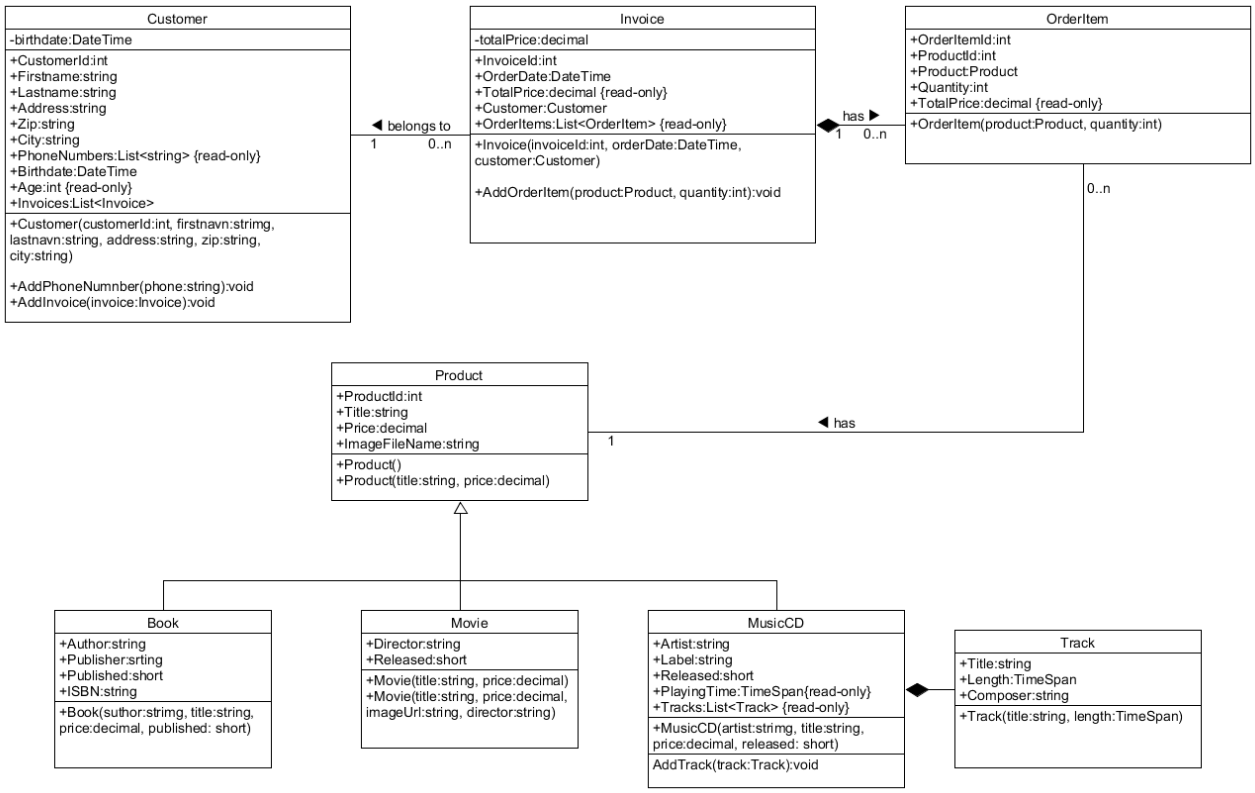
The assignment must be approved in order to be recommended for examination in *Backend Programming*.

Precondition

Before you start, be sure that you have completed *exercise 1-4* in lesson 2, and *exercise 1-5* in Lesson 3.

Exercise 1

Add the `Invoice` and `OrderItem` classes to the Models folder and program them as specified by the UML diagram below:



Review your classes in the Models folder to make sure that they reflect the UML diagram above.

Exercise 2

In all previous exercises during this course, we have instantiated our model classes inside controllers. Normally you will do it elsewhere. A common way of doing it is to store the data in a database and writing a *Data Access Layer* (DAL) component that handles access to the database.

The Repository Class

For this exercise however, we will create a new `Infrastructure` folder with a *static* `Repository` class that we will use to hold example data to display inside the application. We'll declare it as static because we have no requirement to store or retrieve data that is unique to an instance.

Create the `static Repository` class.

Declare a static `Products` property and instantiate it as a list of type `Product`, and likewise a `static Invoices` property that is instantiated it as a list of type `Invoice` and:

```
public static List<Product> Products = new List<Product>();
public static List<Invoice> Invoices = new List<Invoice>();
```

You should also create an empty constructor with no parameters.

Because the class is `static`, the constructor is not called through instantiation, and therefore the constructor in static classes does not have access modifiers (getters and setters). Consequently, it is declared as:

```
static Repository() { ... }
```

We now want to store our product objects as separate objects in the Products list. To do that, you must move all product objects instantiated in controllers into the `Repository` constructor, and you must add each product to the list of products, like this book object example:

```
// Book
Book myBook = new Book("Georg Martin", "With a Little Help from My Friends: The
                        Making of Sgt. Pepper", 1800M, 1995);
myBook2.Publisher = "Little Brown & Co";
myBook.ISBN = "0316547832";
myBook.ImageFileName = "The Making of Sgt. Pepper.jpg";
Products.Add(myBook);
```

The Catalogue Controller

When you have created all product objects inside the constructor of the repository class, you can fetch this list into the Catalogue controller by referencing the Products property of the `Repository` class inside the `Index` action method.

By saving the product list to a ViewBack property, you have access to the full products list in inside the view:

```
ViewBag.Products = Repository.Products;
```

The View

The next and final step is to display the full products list from inside the view grouped by categories.

You can now loop through the list and display products for each categories by selecting procucts by object type:

```
<h2>The Movies</h2>
@foreach (Product product in ViewBag.Products)
{
    if (product is Movie)
    {
          Movie movie = (Movie)product;

          <div>
              <strong>Title:</strong> @movie.Title<br />
              Director: @movie.Director<br>
              Price: @String.Format("{0:0.00}", movie.Price) <br /><br />
              <img src="/Content/Images/@movie.ImageFileName" alt="@movie.Title" />
          </div>

    }

     // ... and likewise for Book and MusicCD

}
```

Tip

Alternatively, you also can send seperate Book, MusicCD and Movie lists to the view by using LINQ (Language-

Integrated Query) inside the controller to save each product type in its own category list, as in this example of the book list:

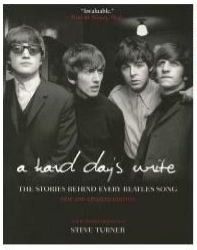
```
IEnumerable<Book> books = new List<Book>();
books = Repository.Products.OfType<Book>().ToList();
ViewBag.Books = books;
```

As `List<T>` is in the namespace `System.Collections.Generic` and the method `OfType<T>()` is a LINQ method, you need to include these namespaces:

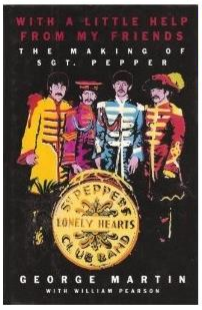
```
using System.Collections.Generic;
using System.Linq;
```

The view must generate an output similar to this display of products:

The Books



Title: A Hard Day's Write: The Stories Behind Every Beatles Song
Author: Steve Turner
Price: 150,00
Publisher: It Books (2005)
ISBN: 978-0060844097



Title: With a Little Help from My Friends: The Making of Sgt. Pepper
Author: Georg Martin
Price: 1800
Publisher: Little Brown & Co (1995)
ISBN: 0316547832

The Music CDs

Album: Abbey Road (Remastered)
Artist: Beatles
Price: 128,00
Publisher: EMI (2009)

Tracks:

1. Come Together (Lennon, McCartney) 4:20
2. Something (Harrison) 3:3
3. Maxwell's Silver Hammer (Lennon, McCartney) 3:29
4. Oh! Darling (Lennon, McCartney) 3:26
5. Octopus's Garden (Starkey) 2:51
6. I Want You (She's So Heavy) (Lennon, McCartney) 7:47
7. Here Comes The Sun (Harrison) 3:5
8. Because (Lennon, McCartney) 2:45
9. You Never Give Me Your Money (Lennon, McCartney) 4:2
10. Sun King (Lennon, McCartney) 2:26
11. Mean Mr. Mustard (Lennon, McCartney) 1:6
12. Polythene Pam (Lennon, McCartney) 1:12
13. She Came In Through The Bathroom Window (Lennon, McCartney) 1:57
14. Golden Slumbers (Lennon, McCartney) 1:31
15. Carry That Weight (Lennon, McCartney) 1:36
16. The End (Lennon, McCartney) 2:19
17. Her Majesty (Lennon, McCartney) 0:23

Total playing time: 47:18

The Movies

Title: Jungle Book
Director: Jon Favreau
Price: 160,50

Exercise 3

Now, we want customers to buy our products.

You'll do that by opening the `Repository` class and create a couple of `Customer` objects, and `Invoice` objects with `OrderItem` and `Customer` object references. Add these objects to the `Invoices` list.

To to that follow these steps:

1. Create (at least) two new `Customer` objects
2. Create (at least) two new `Invoice` objects
3. Create (at least) five new `Product` objects (or use the ones you already have)
4. Add two `OrderItem` objects to the first `Invoice` object
5. Add two `OrderItem` objects to the second `Invoice` object
6. Add each `Invoice` object to the `Invoices` list.

Exercise 4

Create an `Invoice` controller class and return the list of `Invoices` to a ViewBag property to the view that is called from the `Index` action method. Use razor code inside the view to generate a display similar to this (or at least with the same information):

Invoices

Customer	Product	Price
Tina Petterson	Forrest Gump	154,50
	With a Little Help from My Friends: The Making of Sgt. Pepper	180,00
Thomas Larsson	A Hard Day's Write: The Stories Behind Every Beatles Song	150,00
	Revolver (Remastered)	128,00

Exercise 5

Enhance the involices list with catagories and total amounts:

Invoices

Customer	Product	Price
Tina Petterson	Forrest Gump (Movie)	154,50
	With a Little Help from My Friends: The Making of Sgt. Pepper (Book)	180,00
	Total	**334,50**
Thomas Larsson	A Hard Day's Write: The Stories Behind Every Beatles Song (Book)	150,00
	Revolver (Remastered) (MusicCD)	128,00
	Total	**278,00**

Tip 1: In the Razor view you can declare variables by placing them inside a code block, like this:

```
@{decimal price = 0M;}
```

Tip 2: If you call the `ToString` method on an object, you'll get the namespace and the class name. To substract the *class name* from the string you can use the method `.Substring`. It has a overload with one parameter of type int that skips the number of characters you give from the beginning of the string and returns the remaining characters of the string.

You can use that for extracting the classname that corresponds to the category. In an upcomming lesson we'll create a new `Category` class for holding product categories.

Read more: http://www.dotnetperls.com/substring

/ Jes Arbov, 11 September 2020