Exercise 1, extending Product demo

Download `Lesson02_Startup.zip` from the Exercises folder. Extract and open the project in Visual Studio.

Open the website in a browser from inside Visual Studio (Ctrl+F5) and make the necessary corrections to the URL: http://localhost:[*portnumber*]/Exercise01. This should give you a browser window similar to this:

Open `Models/Product.cs` and inspect the Product class. Which fields and properties does it have? Which controllers? Open the `Controllers/Exercise01Controller.cs` controller. What happens inside the index controller? Explain how data is sent to the view? What is the purpose of the view (`Views/Exercise01/Index.cshtml`)?

Make the following modifications of the code:

1. Open the `Controllers/ Exercise01Controller.cs` file and create two new instances of the Product class with holds information of bin and knife products and
2. Open `Views/Exercise01 Index.cshtml` and update the view to display the two new products
3. Open the `Models/Product.cs` class file and change the Name property to a read-only property
4. Add a new private field of type string and with the name *manufacturer*
5. Add a property for this new field
6. Add *manufacturer* as a new parameter to the second constructor
7. When you now select Debug -> Build the website has a compile error, why?
8. Correct this error by calling the constructor with an extra argument.
9. Modify the view to display data from the new instances of Product.
10. Run the application and check that the new information is displayed correctly.
11. As the final step, you must change the `ImageUrl` property to an auto-implemented property.

Exercise 2, *a Person class*

Add a Person-class, which holds values of firstname, lastname, address, zip, city, and phone number.

1. Add member variables and properties (all data types should be strings)

2. Add a constructor with this signature:
   `Person(string firstnavn, string lastnavn, string address, string zip, string city)`

3. Add a new controller `Controllers/Exercise02Controller.cs` and create a couple of `Person` objects

4. Create a view and display address labels on a web page: [*firstname*] [*lastname*]
   [*address*
   ] [*zip*]
   [*city*]

   Tip: In order for the view to know of the Person class, you must include a reference to that class at the beginning of the view file:

   ```
   @using [YourProjectName].Models;
   ```

5. In order to write efficient and easily maintainable code, you must make it DRY (Don't Repeat Yourself). It is a problem in this case because you must repeat the mark-up for each person's label. Since ASP.NET MVC 3, you can solve this issue by declaring you own methods in Razor Views with the @helper syntax.

   Take a close look at this article: ASP.NET MVC 3 and the @helper syntax within Razor (http://weblogs.asp.net/scottgu/asp-net-mvc-3-and-the-helper-syntax-within-razor), and write a `RenderPerson` helper method that renders a person in accordance with the required layout.

   Tip: Use `Person` as parameter type in the `RenderPerson` method:

   ```
   @helper RenderPerson(Person person) {
       ...
   }
   ```

Phone numbers

1. Assign a phone number property to each person in the controller class and update the view to display a list with phone numbers beneath the address labels:

   [*firstname*] [*lastname*] ([*phone*])

2. Open the `Person.cs` file and change the data type of `Phone` from `string` to a `List` collection of type `string`. With this change, it is now possible to store more than one phone number for each person. To reflect this change rename the `phone` field and `Phone` property to `phoneNumbers` and `PhoneNumbers`.

3. Add a new method `AddPhone` with the header:

   ```
   public void addPhone(string phone)
   ```

4. Change the view to display a list of phone numbers for each person:

   ```
   <p>[firstname] [lastname] ([comma separated list of
   phone numbers])</p>
   ```

   Tip 1:
   To make it possible to add new phone numbers to the list you must first instantiate the List as an object. You can do it as you declare `phone` as a field:

   ```
   List<string> phone = new List<string>();
   ```

   Tip 2:
   To display a comma-separated list of strings you can use the `join` method of the `string` object:

   ```
   string.Join (", ", theList);
   ```

   The first parameter is the string that separates the items in the `List`, and the second parameter is the `List` you want to join as a single string.

Birthday and age

1. Open the Person class file and add a field `birthday` and a property `Birthday` of type `DateTime`.

2. Write a read-only property, `Age` that returns the age of the Person.

   Tip: To calculate the age of a person from birth date you can use this algorithm:

```
DateTime birthDate; // field

// code inside the get block of the Age
property DateTime now = DateTime.Now;
int age;

age = now.Year - birthDate.Year;

// calculate to see if the person hasn't had birthday yet
// subtract one year if that is not the case
if (now.Month < birthDate.Month || (now.Month == birthDate.Month
&& now.Day < birthDate.Day))

{
     age--;

}
```

3. Modify the `Birthday` property to ensure that only dates that calculate an age between 0 and 120 are accepted.

   Tip: Use the `set` block of the Birthday property to validate for a realistic age:

```
public DateTime Birthday
{
   set {
        if (expression)
        {
           throw new Exception("Age not accepted");
        }
        else
        {
            birthday = value;
        }
    }
    get { return birthday; }
}
```

4. Test the code by assigning a birth date to one or more `Person` objects.

5. Add code to the view and display the age of one of the `Person` objects,

   like: Peter Thompson is 24 years old

Automatic properties

1. In order to write short concise code you might consider changing some of the properties in the

`Person` class to *automatic* (auto-implemented) properties.

Opgave 3

Arbejd videre med din tre-lags model med en MVC klient

Resterende opgaver er valgfrie

Exercise 3

Create a new ASP.NET MVC Web Application project and give it a proper name like "musicstore", for example. Use the *Empty* project template.

1. Create a **`Product`** class (or copy paste the existing one) with these properties and equivalent member variables:
   - `Title (string)`
   - `Price (decimal)`
   - `ImageUrl (string)`

2. Create a **`Book`** class as an extension of `Product` with these extra properties and their equivalent member variables:
   - `Author (string)`
   - `Publisher (string)`
   - `Published (short)`
   - `ISBN (string)`

3. Create a **`MusicCD`** class which is also an extension of `Product`. It must have these properties and equivalent member variables:

   - `Artist (string)`
   - `Label (string)`
   - `Released (short)`
   - `Tracks (List<string>)`

   and method,

   - `AddTrack(string track)` – add an element to `List` of tracks

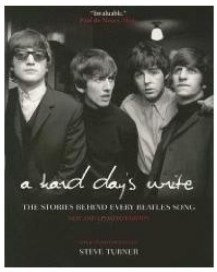
   Tip:
   In order to use the `Tracks` List you must first initialize it as a new `List` object of the type string:

   ```
   private List<string> tracks = new List<string>();
   ```

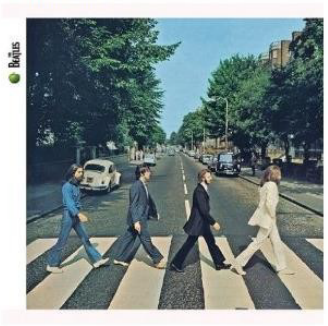
4. Create a new Controller class and name it `Exercise01`. Instantiate the two subclasses `Book` and `MusicCD` inside the `Index` action method and create an object of each type, `Book` and `CD` using something like Book book1 = new Book(); book1.Artist = …. Etc.

5. Create a view that displays the product details as HTML on a web page. The display must be similar to the screenshot below:

Title: A Hard Day's Write: The Stories Behind Every Beatles Song
Author: Steve Turner
Price: 150,00
Publisher: It Books (2005)
ISBN: 0060844094



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



Tracks:

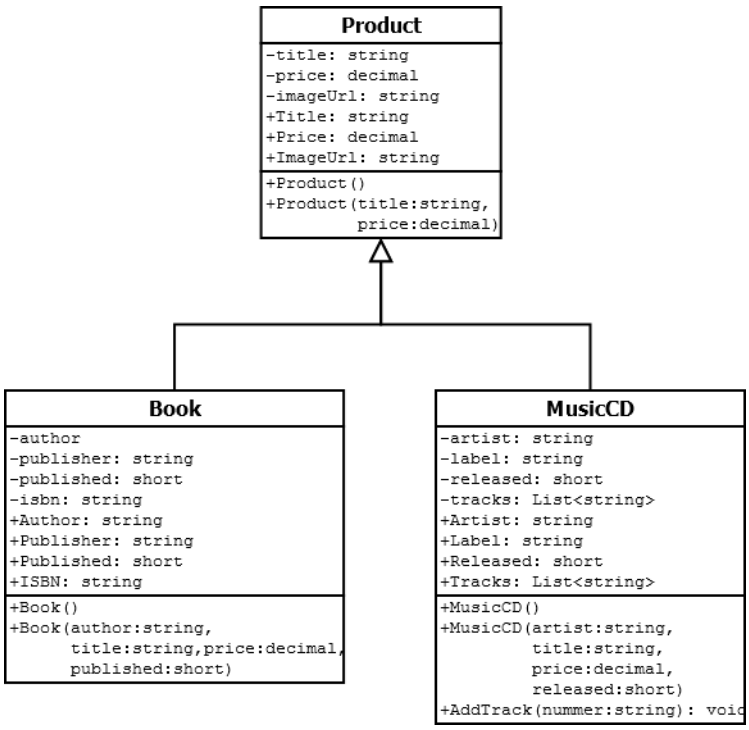
1. Come Together
2. Something
3. Maxwell's Silver Hammer
4. Oh! Darling
5. Octopus's Garden
6. I Want You (She's So Heavy)
7. Here Comes The Sun
8. Because
9. You Never Give Me Your Money
10. Sun King
11. Mean Mr. Mustard
12. Polythene Pam
13. She Came In Through The Bathroom Window
14. Golden Slumbers
15. Carry That Weight
16. The End
17. Her Majesty

Tip:
Use a foreach loop to iterate over the `Tracks` list as you display each track as an element in an ordered list using <li> tags inside the <ol> tag

Exercise 4

1. Create three constructors
   - `Product(string title, decimal price)` in base class and
   - `Book(string author, string title, decimal price, short published)` in Book class and
   - `MusicCD(string artist, string title, decimal price, short released)` in MusicCD class.

UML, version 1

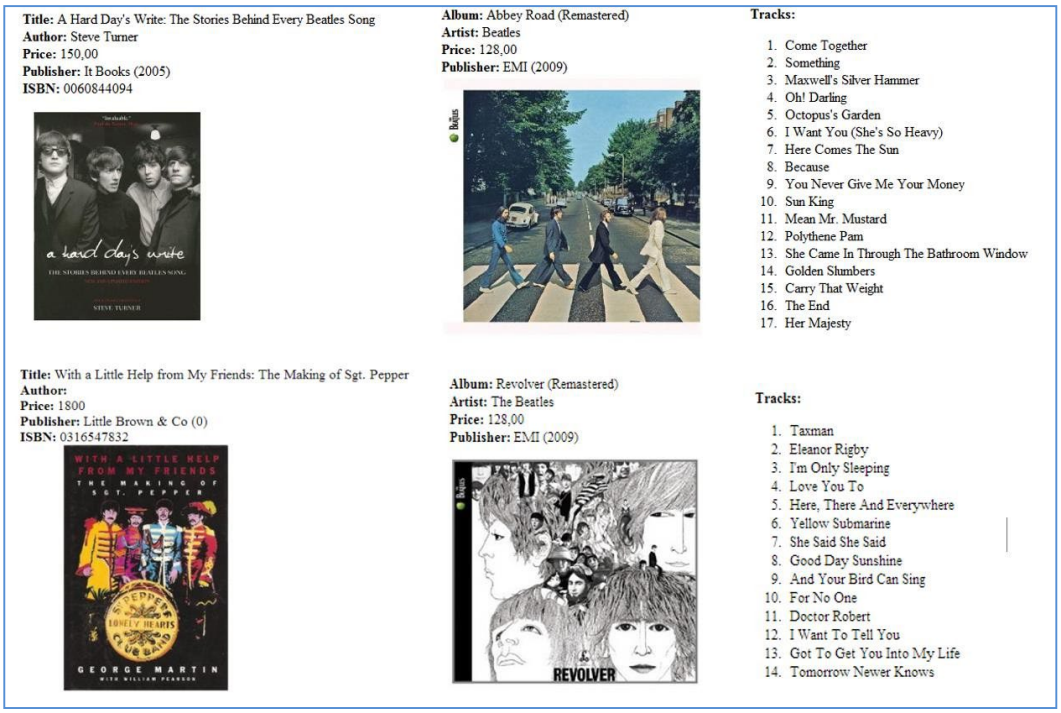


2. Add a new `Exercise02` controller, and create objects of the Book and MusicCD type using the new constructors. Create the new `Book` and `MusicCD` objects inside `Index` action method.

3. Add a View to the `Exercise02` controller and write two View Helper methods, `RenderBook` and `RenderCD` that display information about Books and CDs. Call the render methods inside the view whenever you want to display information about a book or cd.

   Tip: see Exercise 2 for further information on helper methods

Example of the Product class hierarchy in use



Exercise 5

Right now, the Tracks property only holds information about the name of tracks. Further information such as running time and composer is not assessable as long as we only use a simple string property to hold that information. We can solve that problem by creating a new class designed to hold detailed information about individual tracks.

1. Create class **`Track`** as a new file ("Track.cs") inside the Models folder. The class must contain these properties and equivalent member variables:

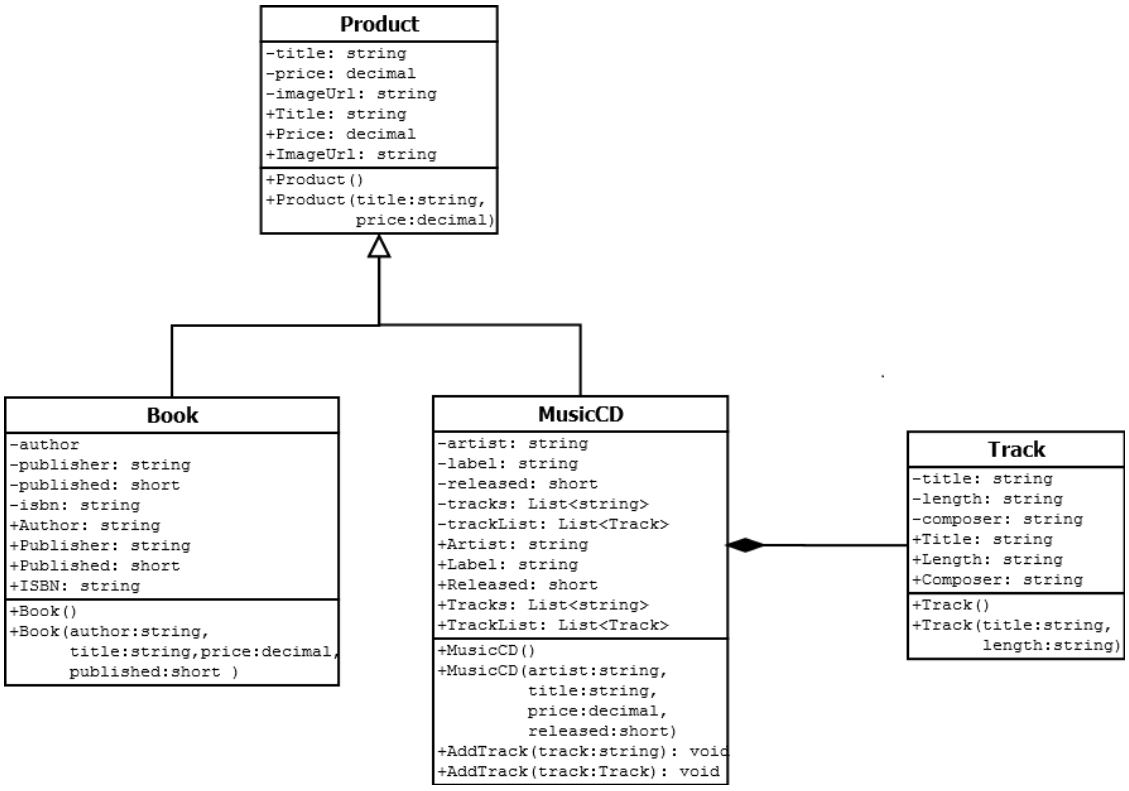
   - `Title (string)`
   - `Composer (string)`
   - `Length (string)`

2. Add a field `trackList` new property `TrackList` of data type List<Track> to the MusicCD class in order to hold instances of the new Track class:

   - `TrackList: List<Track>`

3. Create an overload method `addTrack` that adds track objects to the `trackList` list.

UML, version 2



4. Add a new `Exercise03` controller, and instantiate a new CD object and add several new `Track` objects to the `Tracks` List.

5. Create a View for the controller and display detailed information about tracks.

6. Test the controller and the view and make sure your webpage presentation looks somehow similar to this:

Album: Revolver (Remastered)
Artist: The Beatles
Price: 128,00
Publisher: EMI (2009)

Tracks:

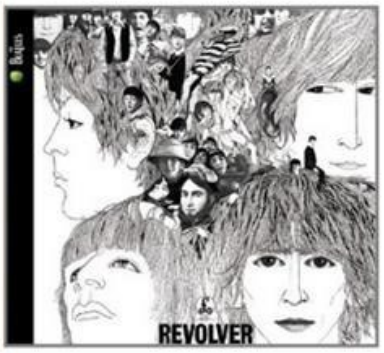
1. Taxman (Harrison) 2:28
2. Eleanor Rigby (Lennon, McCartney) 2:06
3. I'm Only Sleeping (Lennon, McCartney) 3:00
4. Love You To (Harrison) 2:59
5. Here, There And Everywhere 2:23
6. Yellow Submarine 2:38
7. She Said She Said 2:36
8. Good Day Sunshine 2:09
9. And Your Bird Can Sing 2:00
10. For No One 1:59
11. Doctor Robert 2:14
12. I Want To Tell You 2:27
13. Got To Get You Into My Life 2:29
14. Tomorrow Newer Knows 3:01

Opgave 6

1. Add a new method `GetPlayingTime` to the `MusicCD` class. `GetPlayingTime` must calculate the total playing time of a CD and return the result as a `TimeSpan` object:
   - `public TimeSpan GetPlayingTime() {…}`

2. Modify the view to display the result of the calculation:

Album: Revolver (Remastered)
Artist: The Beatles
Price: 128,00
Publisher: EMI (2009)



Tracks:

1. Taxman (Harrison) 2:28
2. Eleanor Rigby (Lennon, McCartney) 2:06
3. I'm Only Sleeping (Lennon, McCartney) 3:00
4. Love You To (Harrison) 2:59
5. Here, There And Everywhere (Harrison) 2:23
6. Yellow Submarine (Lennon, McCartney) 2:38
7. She Said She Said (Lennon, McCartney) 2:36
8. Good Day Sunshine (Lennon, McCartney) 2:09
9. And Your Bird Can Sing (Lennon, McCartney) 2:00
10. For No One (Lennon, McCartney) 1:59
11. Doctor Robert (Lennon, McCartney) 1:14
12. I Want To Tell You (Harrison) 2:27
13. Got To Get You Into My Life (Lennon, McCartney) 2:29
14. Tomorrow Newer Knows (Lennon, McCartney) 3:01

Total playing time: 33:29

Tip:
The `TimeSpan` `Struct` is part of the `System` namespace (you can see `Struct` as like a lightweight type of Class). You can use `Struct` to sum time intervals. It has a static method `Parse`, which returns a `TimeSpan` object from a time interval formatted as a string (hh:mm:ss):

```
TimeSpan s1 = TimeSpan.Parse("00:03:15");
TimeSpan s2 = TimeSpan.Parse("00:04:37");
TimeSpan s3 = TimeSpan.Parse("00:02:55");
```

You can calculate total time by adding `TimeSpan` objects:

```
TimeSpan sR = s1 + s2 + s3;
```