Lesson 3

Exercise 1 (Mandatory)

Open the MbmStore project.

Create these classes in the Models folder

1. A **`Product`** class with these auto-implemented properties:
   - `Title (string)`
   - `Price (decimal)`
   - `ImageUrl (string)`

   With two constructors:
   - `Product()`
   - `Product(string title, decimal price)`

2. A **`Book`** class as derived from `Product`with these extra auto-implemented properties:
   - `Author (string)`
   - `Publisher (string)`
   - `Published (short)`
   - `ISBN (string)`

   With two constructors:
   - `Book ()`
   - `Book(string author, string title, decimal price, short published)`

3. A **`MusicCD`** class which is also derived from `Product`. It must have this field:
   - `tracks (List<string>)`

   and these auto-implemented properties:
   - `Artist (string)`
   - `Label (string)`
   - `Released (short)`
   - `Tracks (List<string>)`

   With two constructors:
   - `MusicCD()`
   - `MusicCD(string artist, string title, decimal price, short released)`

   and one method,
   - `AddTrack(string track)` – add an element to `List` of tracks

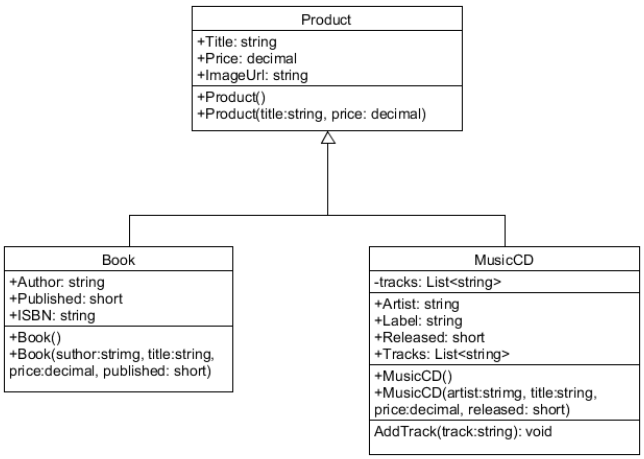
Tip:
If the base class has a constructor without parameters. It is called automatically when the derived class is instantiated. If however you need to call a constructor with parameters, you must explicitly initialize the constructor. You can do it with this syntax:

```
public MyDerivedClass (int x, string s ) : base (x, s)
{
        …
}
```

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>();
```

UML, version 1

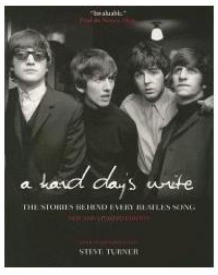


4. Create a new empty Controller and name it `Catalogue`. Instantiate the two subclasses `Book` and `MusicCD` inside the `Index` action method and create an object of each type, `Book` and `CD`.

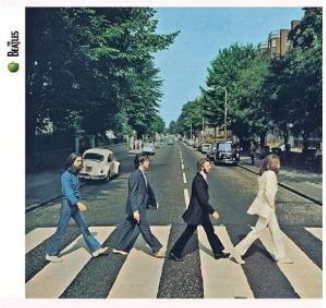
5. Create a view that displays the product details as HTML on a web page. The display must be similar to the screenshot below:

Example of the Product class hierarchy in use:

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:

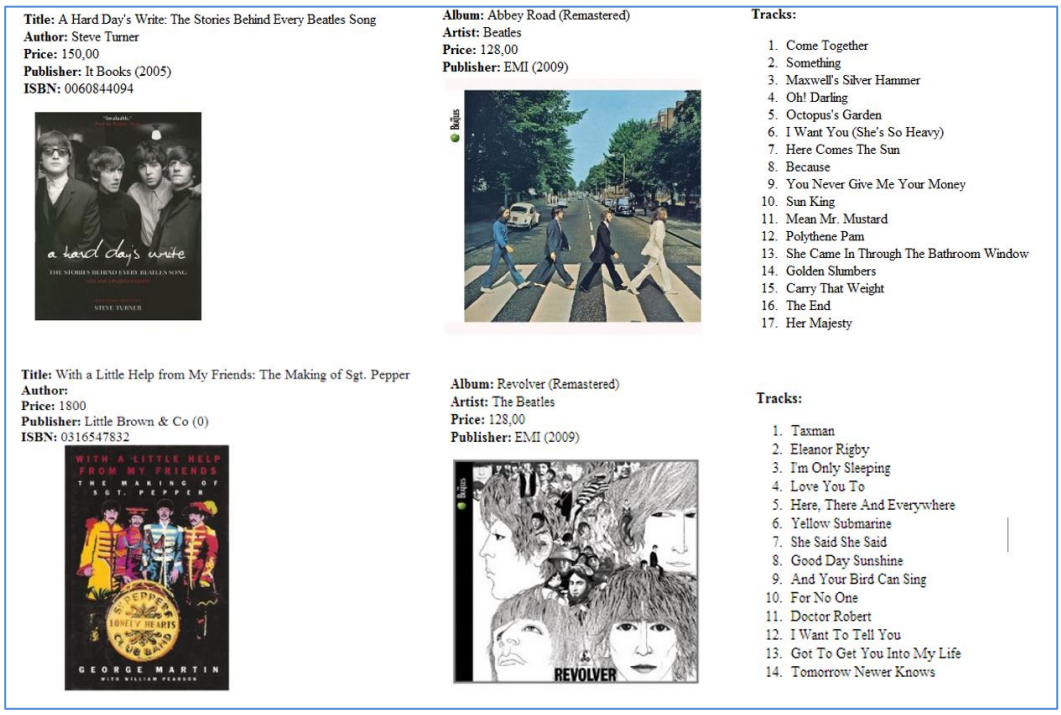
Use a foreach loop to iterate over the `Tracks` list as you display each track as an element in an unordered list.

Exercise 2 (Mandatory)

1. Write two View Helper methods inside the *Catalogue* view file named `RenderBook` and `RenderCD`. These methods must 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 Lesson 2, Exercise 2 for further information of how to do that

2. Instantiate one more Book and MusicCD object in the Index action method of the Catalogue class. Use the `RenderBook` and `RenderCD` methods inside the belonging view to display the information:



Exercise 3 (Mandatory)

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 a new class named **`Track`** by adding a new file ("Track.cs") to the Models folder. The class must have these properties and equivalent member variables:

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

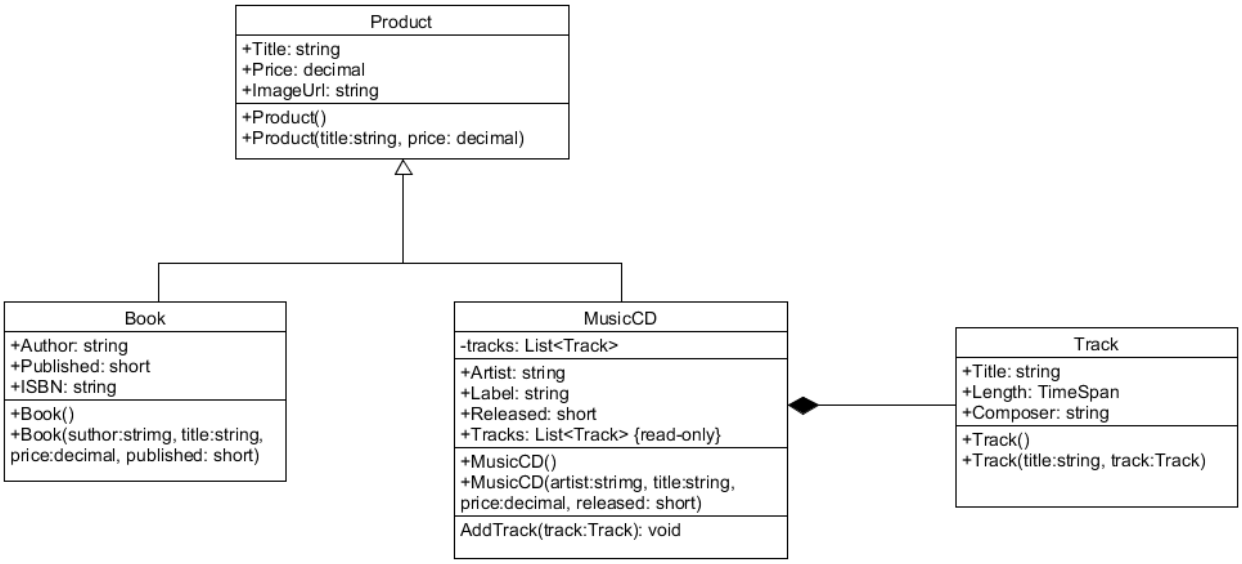
- `Length (TimeSpan)`

2. Open the MusicCD class and change the field `tracks` to type `List<Track>` and the `Tracks` property to a read-only property of type `List<Track>` in order to hold instances of the new `Track` class:

   - `Tracks: List<Track>`

3. Change the method `addTrack` method to add `Track` objects instead of strings to the `tracks` list.

UML, version 2



4. Change the `Catalogue` controller to create `Track` objects for each track, and add these new `Track` objects to the `Tracks` List of the `MusicCD` objects by calling the `AddTrack` method.

5. Modify the view to display detailed information about tracks.

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

Tip: If you just call the `TimeSpan` object inside the view it'll return hh:mm:ss (hours:minutes:seconds). If you want a nicer display, you can call the properties of the `TimeSpan` object:

```
<strong>Tracks:</strong>

<ol>

    @foreach (Track track in @cd.Tracks)
    {
        <li>@track.Title (@track.Composer)
                @track.Length.Minutes:@track.Length.Seconds </li>
    }

</ol>
```

Exercise 4 (Mandatory)

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: