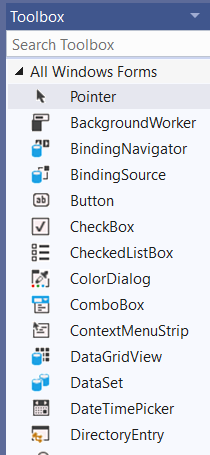
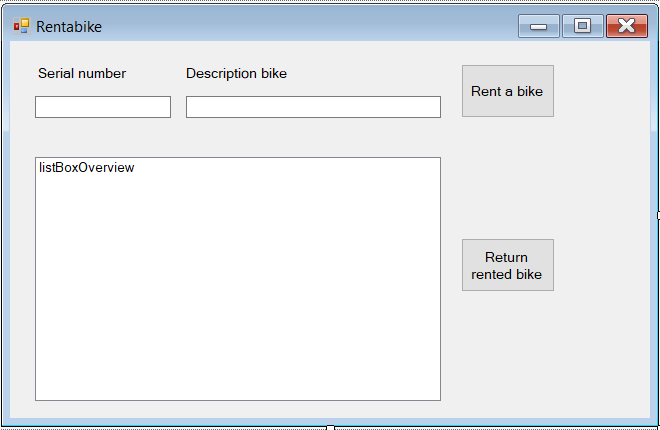
***Exercise introduction C#: Object Orientation***

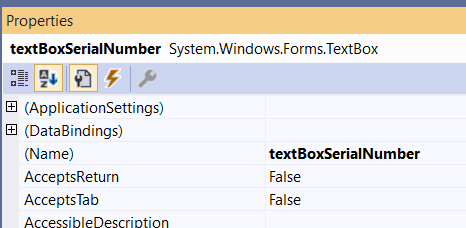
In this exercise you create an application for a bike rental service called “Rentabike”.

**Part 1: Designing the presentation layer**

Create a new Visual Studio C# Windows Forms App.

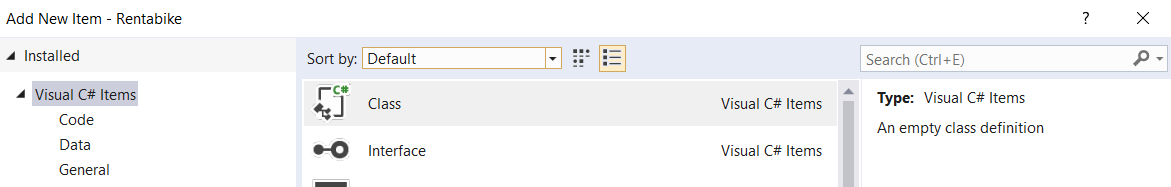
Use the Toolbox to design the given Form.

Give every control (label, textbox, button, listbox) you add to the form an appropriate name through the Properties window. The screenshot below shows that the internal name of the textbox used for the input of serial numbers is textBoxSerialNumber. For identifying controls camelCasing is the standard in .NET programming. Do not use underscores or other separators to differentiate words.  
  


You also use camelCasing for declaring your variables in code. However, for identifying public methods, classes, properties, … PascalCasing is the standard. You can consult the Microsoft naming conventions on <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines>

**Part 2: Creating the base class ‘Material’**

Add a new class ‘Material.cs’ to your solution. Therefore, right-click in solution explorer on your project and successively select the options ‘Add’, ‘New item’ and Class.  


Extend the class with the following features:

* Each object in the material class is characterised by a serial number and a description. Both properties are retrievable and adjustable.
* An object in the material class is constructed by the serial number and the description.
* The class contains a function “ReturnRentalPrice” which gives back a standard amount of 100 Euros.

**Part 3: Creating the subclass ‘Bike’**

Create a derived class Bike that inherits from the Material class. Foresee the following extra modalities:

* An object in the class Bike is, in addition to its serial number and its description, also characterized by a property called Code. This code is a number lock consisting of a random combination of 3 digits (100-999). The property Code is exclusively callable.
* A Bike object is just like an object from the class Material constructed by giving the serial number and the description of the bike. The number lock is assigned automatically as a random combination of 3 digits.
* The ReturnRentalPrice function returns an amount for a bike that is 20% higher than the rental price of a standard object in the Material class.
* Override the ToString() method so that whenever a Bike object is displayed its serial number, description, code and rental price are shown.

**Part 4: Creating the class ‘BikeRental’**

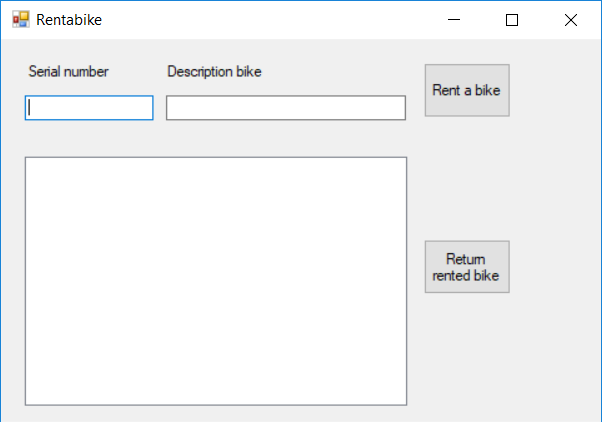
Finally, create a class BikeRental that meets the following requirements:

* Create a listproperty ‘OverviewRentals’ in the BikeRental class to store all the rented Bike objects. This list is only retrievable from outside the class.
* Write a method RegisterRent to fill this list with the loans. Make sure that when a Bike object is already in the list, it can’t be added twice. Check on serial number.
* Write a method DeregisterRent to remove a rented bike from the list.

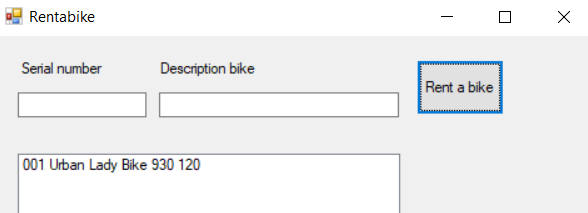
**Part 5: Writing the code-behind of the form**

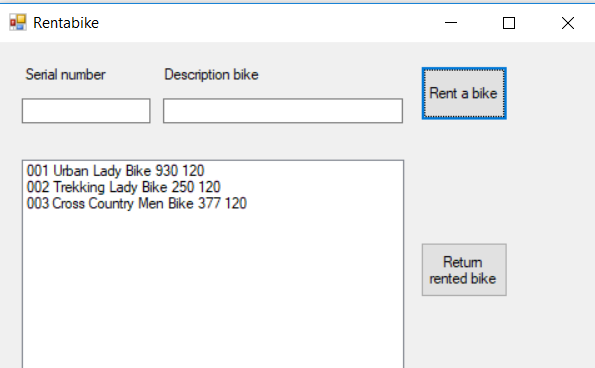
Test the classes, methods and properties you have built by writing the code behind the form designed in part 1.

When running the app, the form will look like this:



When you want to rent a bike, enter the serial number (e.g. 001) and the description of the bike (e.g. Urban Lady Bike). After clicking on "Rent a bike" all the data of the bike will be shown in the listbox. So also the random generated code (e.g. 930) and the rental price (120) will be displayed.



After registering 3 loans you get for example  


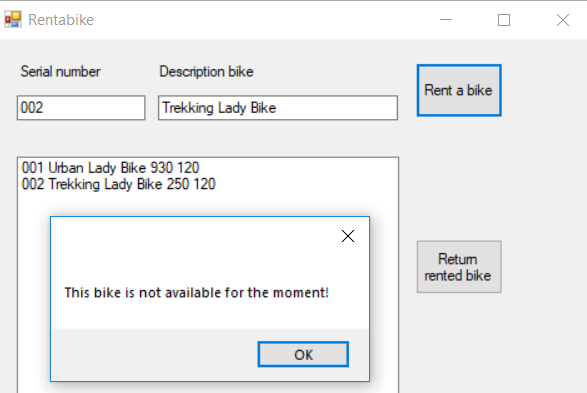
Test that you can’t rent a bike with the same serial number as one that’s already in the listbox.

Finally, code that when an item in the listbox is selected and you click on ‘Return rented bike’ the item is removed and the bike with this specific serial number can be rented again.

**Part 6: Extras**

Extend the class BikeRental with an exception that is thrown whenever you want to add a bike to the list which is already registered.

Respond in the form to this exception with a messagebox indicating that the bike is not available.



Instead of working with an exception, you can realise exactly the same behaviour by creating an event in the BikeRental class that is called whenever you want to add the same bike twice to the list.