Lab Exercise 3.1: C# Interfaces

In this exercise, you will create and use a few simple C# interfaces and play around with these.

Exercise 3.1-A:
Start a new C# console application project called ***Main*** (in a new MS Studio solution). Create a skeleton "Hello world!"-program that just writes on the console.

Exercise 3.1-B:
Add a new project (a class library) called ***DoStuff*** to the solution. Add an interface named ***IDoThings*** to DoStuff that contains these methods:

- void DoNothing()
- int DoSomething(int number)
- string DoSomethingElse(string input)

Exercise 3.1-C:
Add a class called ***DoHickey*** to DoStuff that implements IDoThings. It does not matter what the implementations of the methods actually do. Instantiatiate DoHickey in the Main project, and write some code that calls all 3 methods from IDoThings on the instance.

Exercise 3.1-D:
Here are two classes, GasEngine and MotorBike:

```
namespace Vehicles {

   public class GasEngine
   {
      private uint _currthrottle = 0;
      private uint _maxthrottle = 0;


      public GasEngine(uint maxthrottle)
      {
         _maxthrottle = maxthrottle;
      }

      public uint MaxThrottle // (*)
      {
         get { return _maxthrottle; }
      }

      public uint Throttle    // (*)
      {
         set { _currthrottle = value; }
         get { return _currthrottle;  }
      }
   }

   public class MotorBike
   {
      private GasEngine _engine = null;

      MotorBike(GasEngine engine)
      {
         _engine = engine;
      }

      void RunAtHalfSpeed()
      {
         _engine.Throttle = _engine.MaxThrottle / 2;
      }
   }
}
```

As you can see, a MotorBike object knows the specific class of its engine. What you must do is this:

1. Add a new class library project, ***Vehicles***, in your solution.

2. Create an interface, ***IEngine***, in Vehicles that contains methods with the same signature as those marked **(*)** in the GasEngine class.

3. Add GasEngine to Vehicles, and refactor the class so that it uses (i.e, is declared as implementing) IEngine.

4. Add a new class, ***DieselEngine***, to Vehicles that implements IEngine. Exactly what its methods do are not important, you can make DieselEngine behave exactly like GasEngine if you want.

5. Add MotorBike to Vehicles, and refactor the class so that all references to GasEngine are replaced with references to IEngine.

6. Add a few lines of code in your Main project, so that you:
    a. Create both GasEngine and DieselEngine instances.
    b. Create a couple of MotorBike instances, one that uses a GasEngine instance, and one that uses a DieselEngine instance.

So when you are done, so should have a MotorBike class that knows nothing about the specific *class* of its engine, and will work with any object that implements the IEngine interface.