Exercise 04: C# Interfaces 2 - A little more advanced

In this exercise, you will advance your use of interfaces a bit more and catch a glimpse of the power inherit in loosely coupled code – a benefit of coding to interfaces instead of implementations (concrete classes)

Exercise 1:
You are given the implementation of two classes, `GasEngine` and `MotorBike` (see below):

```
public class GasEngine
{
   private uint _curThrottle = 0;
   private uint _maxThrottle = 0;


   public GasEngine(uint maxThrottle)
   {
      _maxThrottle = maxThrottle;
   }

   public uint MaxThrottle
   {
      get { return _maxthrottle; }
   }

   public void SetThrottle(uint thr)
   {
      _curThrottle = thr;
   }

   public uint GetThrottle()
   {
      return _curThrottle;
   }

}
```

```
public class MotorBike
{
   private GasEngine _engine = null;

   MotorBike(GasEngine engine)
   {
      _engine = engine;
   }

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

As you can see, `MotorBike` knows the specific class of its engine, `GasEngine`. This is a rather high coupling and makes the installation of another type of engine difficult.

Our objective in this exercise is to change the implementation so that it becomes easier to install new types of engines. What you must do is this:

1. Create a Visual Studio 2013 solution `Interfaces-2` and add a C# class library `Vehicles` to this solution. Add the two classes `Motorbike` and `GasEngine` as given above to this solution.

2. Investigate `MotorBike` and identify the method(s) and properties of `GasEngine` that `MotorBike` uses

3. Create an interface `IEngine` that contains methods and properties with the same signature as the ones you identified in 2 above.

4. Refactor `GasEngine` so that it implements `IEngine`.

5. Refactor `MotorBike` so that it only depends on the `IEngine` interface, not the concrete `GasEngine` class.

6. Add a C# console project `Vehicle.Application` to the solution. Add a reference to `Vehicles` in this project.

7. In `Main()` in the `Vehicle.Application` project, instantiate a `Motorbike` object with an instance of the `GasEngine` class. Test it a bit.

8. Add a new class `DieselEngine` to `Vehicles` that implements `IEngine`.

9. In `Main()` in the `Vehicle.Application` project, also instantiate a `Motorbike` object with an instance of the `DieselEngine` class. Test it a bit.

Note how you can now add new engine types to the project and use them in the motorbike *without ever changing the motorbike implementation!* Ain't that neat? This is the very foundation upon which we will make unit testing sooo much easier in the future – and now you know how!