# Table of Contents

[Introduction 2](#_Toc535235910)

[Scenario 3](#_Toc535235911)

[Scope 4](#_Toc535235912)

[In Scope 4](#_Toc535235913)

[Non-Functional Requirements 4](#_Toc535235922)

[Form design 4](#_Toc535235924)

[Customer Details Form 4](#_Toc535235925)

[Login Form 5](#_Toc535235926)

[Gym Form 6](#_Toc535235927)

[Class Diagram 7](#_Toc535235928)

[Classes 8](#_Toc535235929)

[CalculatorEngine.cs 8](#_Toc535235930)

[Customer.cs 9](#_Toc535235931)

[EnteringDetails.cs 10](#_Toc535235932)

[ESCGym.cs 11](#_Toc535235933)

[Login.cs 12](#_Toc535235934)

[Utility.cs 13](#_Toc535235935)

[Validator.cs 14](#_Toc535235936)

[Properties 14](#_Toc535235937)

[Adding properties 14](#_Toc535235938)

[CalculatorEngine properties 14](#_Toc535235939)

[Customer properties 15](#_Toc535235940)

[Methods 16](#_Toc535235941)

[RoundValue : double 16](#_Toc535235942)

[GetBMICategory : string 17](#_Toc535235943)

[BMR\_HB : double 17](#_Toc535235944)

[Testing 18](#_Toc535235945)

[NUnit - Unit Tests 18](#_Toc535235946)

[CalculatorEngineTest.cs 18](#_Toc535235947)

[CustomerTest.cs 19](#_Toc535235948)

[UtilityTest.cs 21](#_Toc535235949)

[ValidatorTest.cs 21](#_Toc535235950)

[Review 22](#_Toc535235951)

[Richard’s Review 22](#_Toc535235952)

[What went well 22](#_Toc535235953)

[Development 22](#_Toc535235954)

[Database Schema 22](#_Toc535235955)

[QA Testing 22](#_Toc535235956)

[What didn't go well 22](#_Toc535235957)

[Project Management 22](#_Toc535235958)

[Development 22](#_Toc535235959)

[QA Testing 22](#_Toc535235960)

# Introduction

This is a report of my college assignment “East Surrey College GymApp” that I worked on at Infor.

ESC GymApp is a Windows Forms application written in C# using Visual Studio 2017. We were tasked to produce a piece of software using the contextual scenario below and follow all professional computing standards and good practice procedures. The finished solution consists of the frontend (GUI) and the backend (code).

# Scenario

You are asked to use your software design, development, testing and evaluation understanding and skills to produce a program that meets the client’s requirements.

East Surrey College gym has commissioned you as a software developer to write a program that will assess a gym member’s requirement to maintain their current weight accurately.

You need to create a program that will give the learners gym member information about:

• Their current basal metabolic rate (BMR)

• Their current body mass index (BMI)

• Their target BMI

• The number of kilocalories to maintain their current weight.

# Scope

## In Scope

## The program calculates and displays a gym member's basal metabolic rate (BMR)

## Calculates and display the gym member's body mass index (BMI)

## The number of kilocalories to maintain their current weight.

## The BMR calculation is given to 2 decimal places.

## The BMI calculation is given to 1 decimal place.

## The kilocalorie requirement output is shown rounded to a whole number.

## The program needs a "login" screen to access the application.

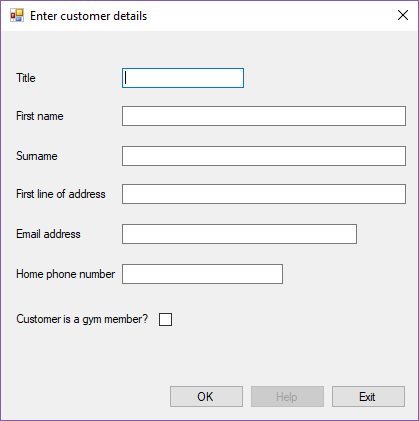
## A screen to collect customer details, if they're an existing gym member, name, address, email and telephone.

## Non-Functional Requirements

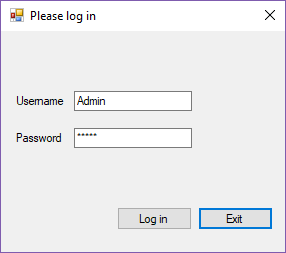
## Must be Windows Form application written in C# & Java

# Form design

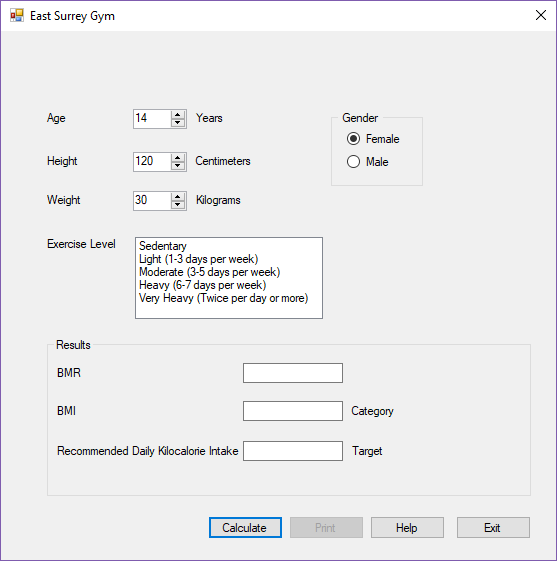
## Customer Details Form



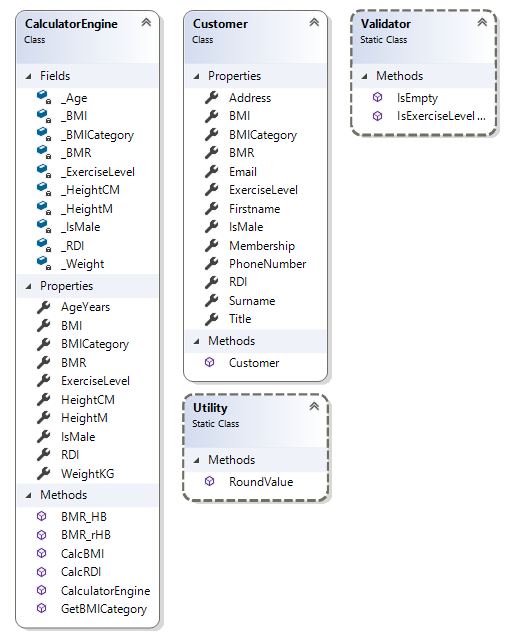
## Login Form



## Gym Form

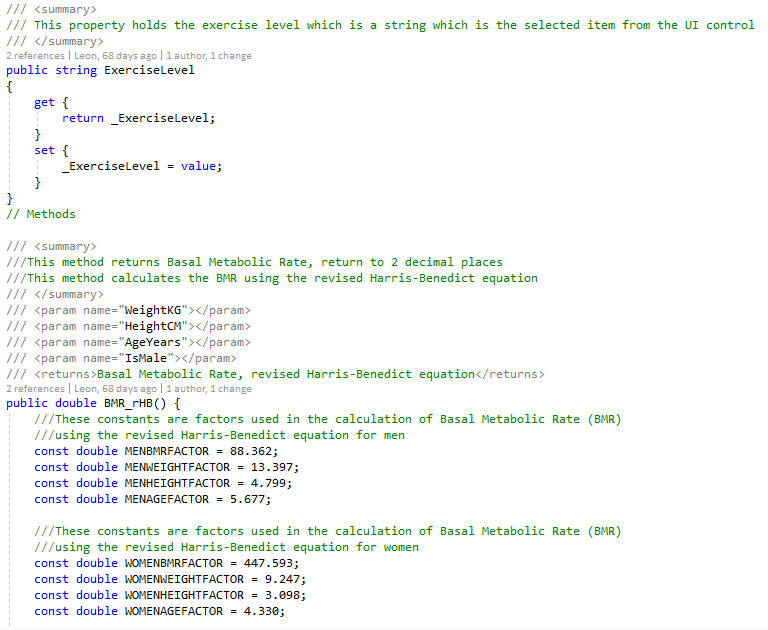


# Class Diagram

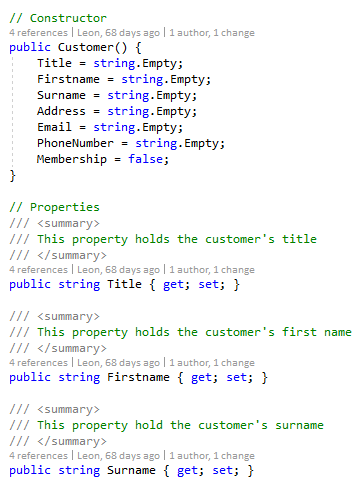


# Classes

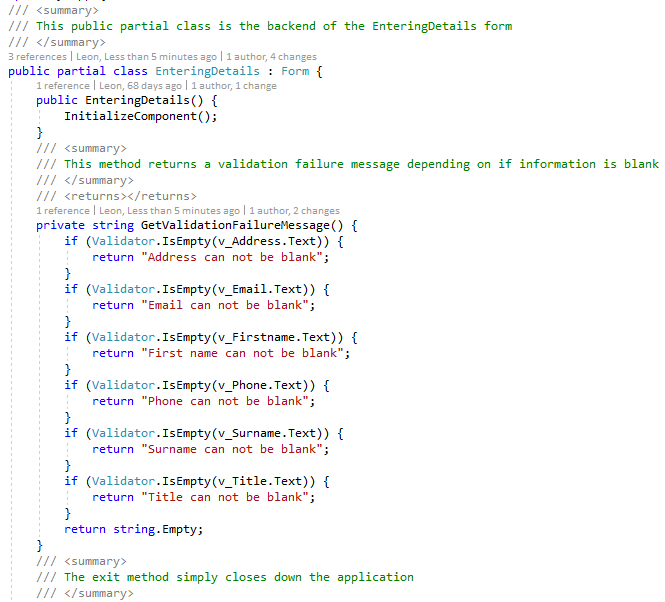
## CalculatorEngine.cs



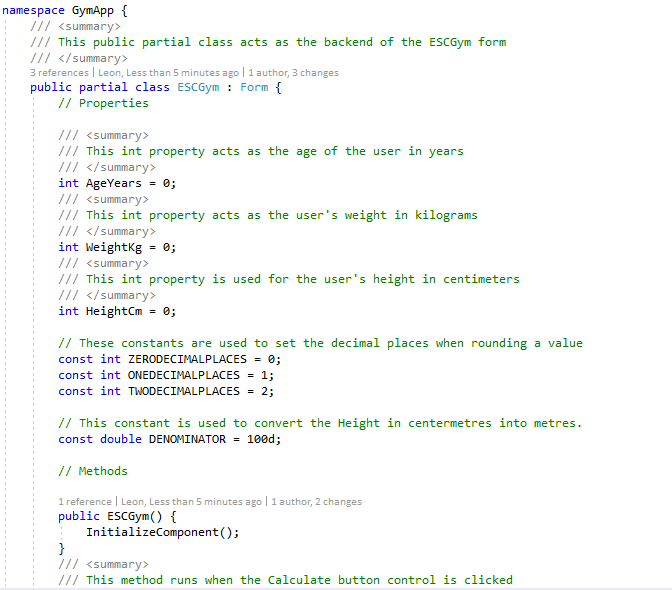
## Customer.cs



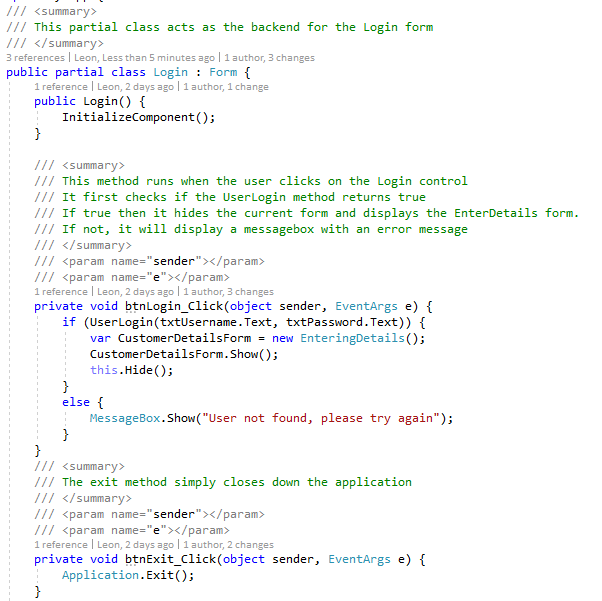
## EnteringDetails.cs



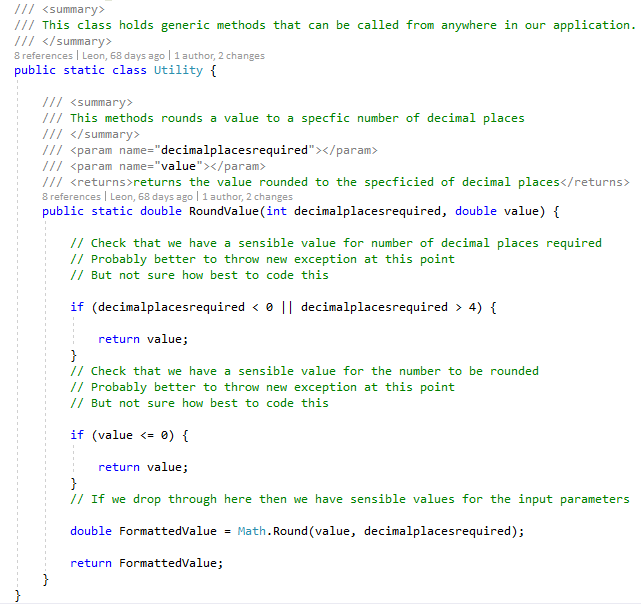
## ESCGym.cs



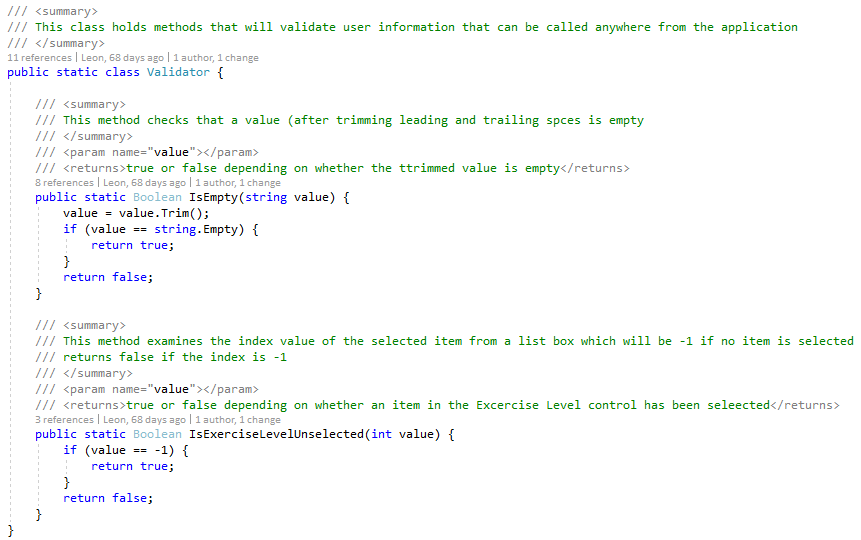
## Login.cs



## Utility.cs



## Validator.cs



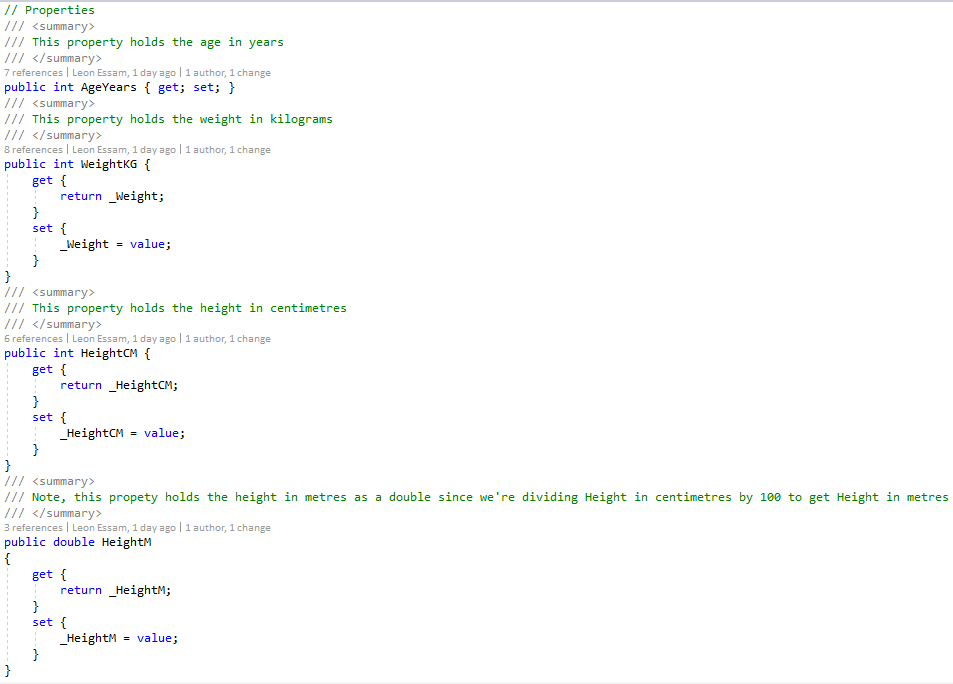
# Properties

## Adding properties

The first initial application didn't have any properties, instead it relied on public static variables in different classes to store and access the information. The revised application however now uses properties and fields in classes that must be instantiated. This is good because now multiple customers can be instantiated with different information instead. It is also useful for alter development like printing out a list of customers.

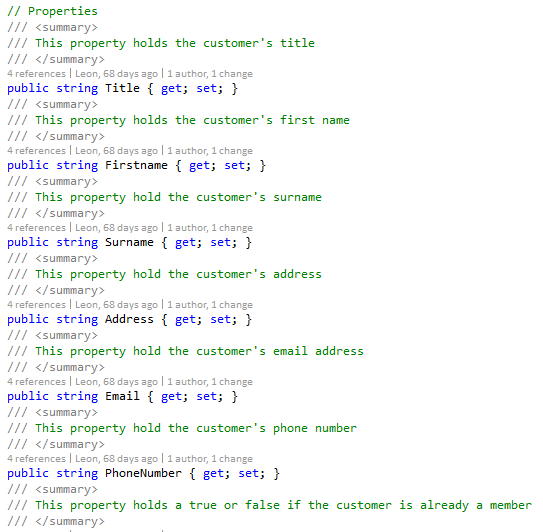
## CalculatorEngine properties

These properties are used in calculations of BMI, BMR and other information. Customer information is fed to this class which in turn is used in multiple methods and then assigned to new properties like BMI and BMR, which are used in different classes.



## Customer properties

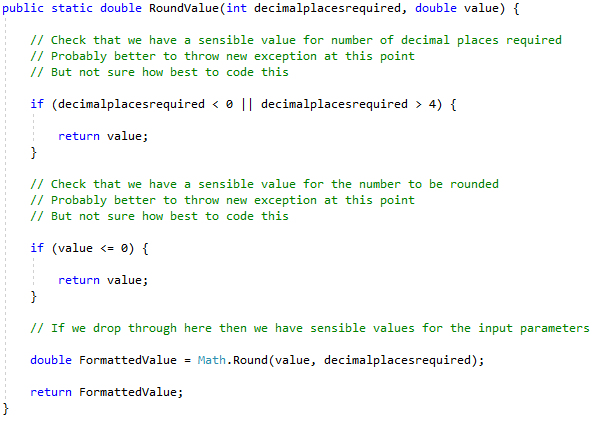
Customer properties include personal information like name, address and email. These are used to store BMI and other information in the main form.



# Methods

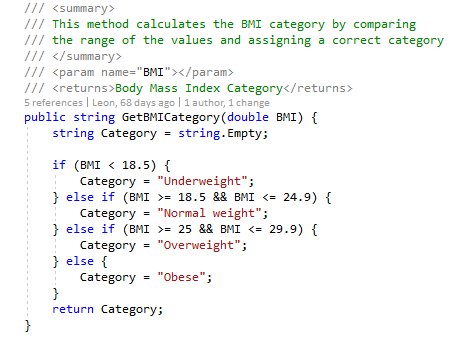
## RoundValue : double

This is a simple method in the Utility class, it takes a value and how many decimal places the value should be rounded to. I’ve done it this way as it makes maintaining easier and avoids hardcoding values.



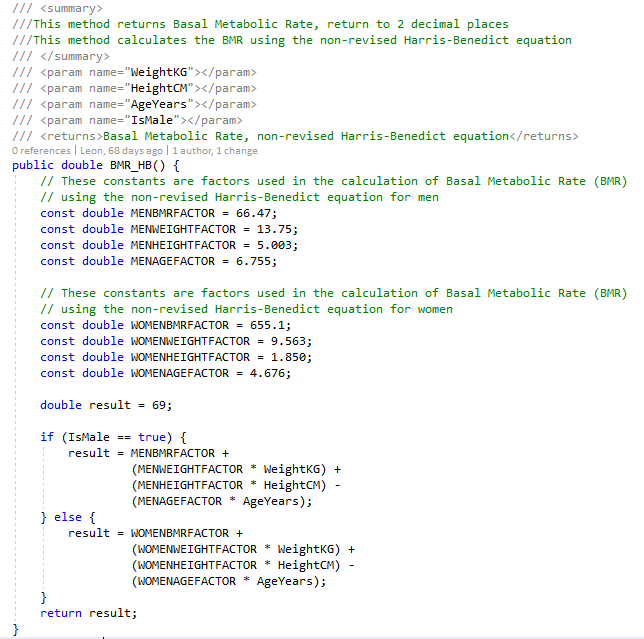
## GetBMICategory : string

This method checks the value of the BMI and returns a string accordingly for their current weight category. This method is easy to maintain since we know the BMI requirements from the scenario, we can use hardcoded values to return the correct BMI category.



## BMR\_HB : double

This method is similar for calculating the BMR, this method however uses the non-revised Harris-Benedict equation, so this allows the gym staff to now choose one of two equations.



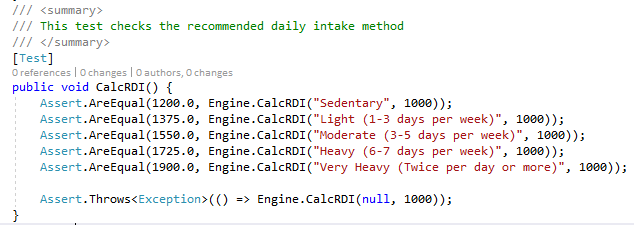
# Testing

## NUnit - Unit Tests

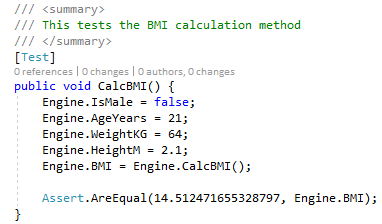
While working on my solution for the application, I started working with Unit Tests to make sure that the methods, properties and classes were all working as intended. This helped save time as I didn't have to worry about launching the program, navigate to what I wanted to check and see if it worked. A new class for unit tests that related to a single class, for example one class would be made for unit tests that all tested one class, then another class for a different set tests for a different class.

## CalculatorEngineTest.cs

CalcRDI is a test that makes sure that the 'CalcRDI' method functions correctly by returning the correct BMI category. First it checks all categories are returned with correct BMI values. Then it checks to see if the application throws an exception if the BMI is invalid. These tests work successfully.

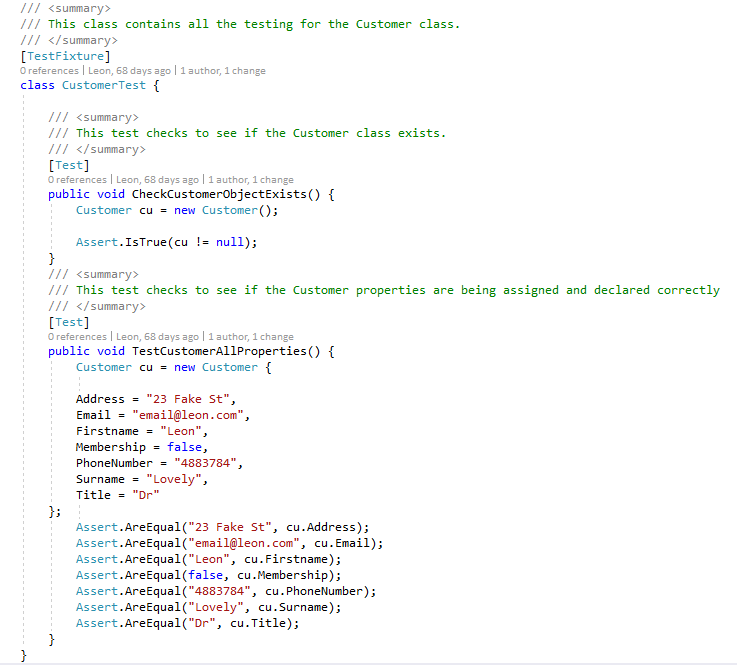


CalcBMI is a similar test in that it checks if the method 'CalcBMI' method functions correctly by returning by the correct BMI value. The test first feeds the CalculatorEngine class basic information then expects a set value that was calculated outside to find the correct value. This test works successfully.

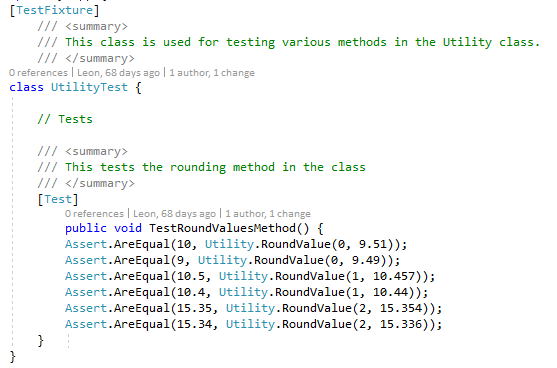


## CustomerTest.cs

This class contains two tests for the customer class, first it checks if the class exists, then if the properties are working accordingly.

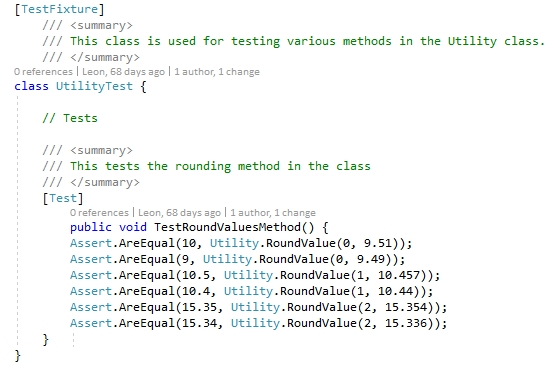


## UtilityTest.cs



## ValidatorTest.cs

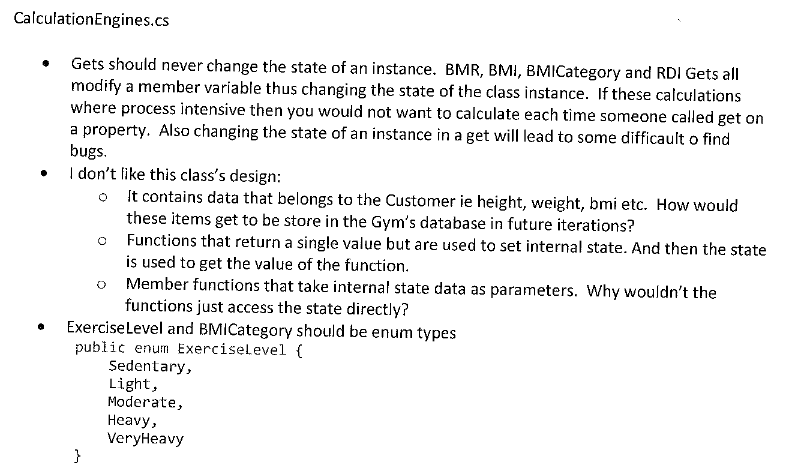
This class holds three tests for the Validator class, though one of them doesn't do anything. 'IsEmptyTestMethod' checks to see if the method outputs a correct true or false depending on if the string is empty. 'IsExerciseLevelUnselectedTestMethod' does something similar by checking that the number is in range.



# Review

## Richard’s Review

My initial application was reviewed by one of our mentors when I was happy with it. This is just an example but it the whole application was reviewed, and I was to change accordingly.



## What went well

### Development

I worked closely with our mentor during development of this program. This was good because we worked to build the program as more of a product delivered by a junior software engineer than as a college assignment. I was introduced to using objects and classes that hold their own appropriate properties and objects. This allowed me to have a much easier to read and maintain program instead of having everything in a main class.

I also used constructors and getters and setters for objects and classes, this made using the Calculator Engine much easier as well as manipulating unique customer objects.

Methods were always a much bigger aspect of the program as now methods were in different classes depending on their use. For example, the calculator class has a BMI calculator method while the utility class has a value rounding method.

Having my program reviewed allowed me to see how a professional developer would tackle the same problems that I encounter or a different method of achieving the same goal.

### QA Testing

This was the first time we used unit testing in our application. We had a small exercise from our C# mentor about how to use NUnit and write them as a professional developer, I found it very easy to implement into the gym application and now all methods, classes and properties are tested in various test classes.