|  |
| --- |
| Assignment Report |
| Sara Silva Student Number: 1669329Explaining Innovation Applied to the code and functions that have been testedJanuary 13, 2017 |

# Innovation that was applied to the code

**Lambda expressions in a query:**

In the code bellow I demonstrated how to use a lambda expression in a method-based query by using the Enumerable.Where standard query operator.

Using a lambda expression is a convenient way to write one-off methods that don't benefit from the extra work required to create a full method.

if (people.Contains(StudentName))

{

if (people.Where(i => i.Name == StudentName).FirstOrDefault() is Student)

{

Console.WriteLine("Student found");

}

else

{

Console.WriteLine("{0} is not a student",StudentName);

}

}

I used this code on case 3 of the switch statement which represents when the user chooses to find a student by name on the menu. (I have also used the same technique to look for a lecturer by name.)

The main use of the lambda expression on this case is to make sure that the name is the name of a student and not the name of a lecturer since the list contains both students and lecturers and therefore the contains method will return true if the variable name is in the list whether it belongs to a student or not. Since I implemented the ICollection interface I was able to use the Where method from Enumerable.

The same logic was used on case 4 of the switch statement which represents choice 4 on the menu where the user wishes to look for a lecturer by name:

if (people.Contains(LecturerName)) {

if (people.Where(i => i.Name == LecturerName).FirstOrDefault() is Lecturer)

{

Console.WriteLine("Lecturer found");

}

else

{

Console.WriteLine("{0} is not a lecturer", LecturerName);

}

}

Using a lambda expression along with the Where method assures that the name we find belongs to a lecturer in this case and not a student.

**Create unit tests in C# to test five functions in classes**

I created 8 test methods in 4 different UnitTest Classes. They test 6 functions.

The first Test class is the UnitTestCollege, which tests the College class for 2 different functions: searching student by name and searching lecturer by id. I created 2 test methods for each function and they test for whether the if statements are true or false by using: Assert.IsFalse and Assert.IsTrue. I had to import System.Linq since I am using Lambda expressions on my code and wanted to test for them.

The second Test class is the UnitTestLecturer, which tests the Lecturer class for 2 functions also:

This time I created only one Test method for each function since the first Test method is an [ExpectedException(typeof(ArgumentNullException))] and it only passes when it throws an exception. So I tested for an empty string array as the Subjects Taught and the test passed since it throws an exception. Then I tested my override getPaid method that I have in the Lecturer test which overrides the getPaid method from the Staff class by adding a bonus. So I tested for whether the expected amount was equal to the Lecturer’s Monthly Salary. The test failed because I am using decimal and could not get the exact precision so I decided to fix the code by rounding the result with Math.round. Then it passed.

The third Test class is the UnitTestStaff class which has only one Test Method: [ExpectedException(typeof(ArgumentOutOfRangeException))] which tests for a negative number as a Staff’s salary when creating a new object of the Staff class. This test throws an ArgumentOutOfRangeException and passes no problem.

The last class is the UnitTestStudent which tests also has only one Test Method:

[ExpectedException(typeof(ArgumentException))] which tests for creating a Student object with an invalid enum constant for the student status. This test throws an exception when a Student object is created with an invalid student status and passes.