CS 2026 – Spring 2009
Assignment #3
2/8/2009
Due: Friday 2/13/11:59 PM
(no late hand-in)

This assignment will exercise your knowledge of C# features such as reflection, exceptions, and LINQ that were all discussed last week.

In this assignment you will implement a `Querier` class that will take in a collection of objects and answer basic queries about them. A `Querier` object is initialized with a collection of items that it stores internally and then provides an interface to query them.

You will not know the type of items given to your `Querier` in advance. All you know is that each item will have one or more properties, and each property will be of type either: `int` (`System.Int32`), `bool` (`System.Boolean`), or `string` (`System.String`).

You will use reflection to manipulate the elements of your collection. Additionally, elements might have some methods that will change the value of some of their properties.

All the queries should be written in LINQ. Queries not written in LINQ (i.e. written with traditional loops) will not be accepted and will fail the assignment.

Methods

In all the following methods, properties names and values are passed in as strings. The same goes for functions to apply.

When dealing with passed-in property and function names you will need to use reflection to access those properties and functions. If the method also takes in a value for a property, you will have to use reflection to determine the type of the property and then you should convert the passed-in property value to the appropriate type and use that in your LINQ expression.

For example, if the type of the property turns out to be an integer, you should use its value in the LINQ query as follows: `int value = Int32.Parse(propertyValue);`

If a given property name does not correspond to a property of the elements in the collection then you will get an exception and you should catch that exception and return -1. Similarly, if the passed-in function name does not match a method of the elements of the collection, or the passed-in property value does not match the property type then you should catch the resulting exception and return -1.

Note: you should catch specific exceptions and not just the general exception type. This means that your code should not use a `catch(Exception e) { .. }` block but instead it should catch specific exceptions such as `catch(NullReferenceException nre) { .. }`

Your `Querier` class should implement the following methods

- `Querier(IEnumerable e)`
  - A constructor that takes in the collection of objects that will be operated on.
- `public int CountByProperty(string propertyName, string propertyValue)`
  - This method returns the number of elements in the collection that have the specified value for the specified property.
- `public int GroupByProperty(string propertyName)`
  - This method will take in the name of a property and print out to the console how many elements in the collection have the same value for this property.

    So if the passed-in property name is a car's manufacturing year, the console printout should show how many cars were made in each year. All you have to printout is a table with two columns that looks like this

    property value - - # of elements that have that value

    <u>The printout should be ordered in **ascending order** by the property value.</u>
    An example output will be something like this:

    ```
    2007 - - 10
    2008 - - 12
    2009 - - 6
    ```

    This method should return 1 if the given `propertyName` is for a valid property, -1 otherwise.
- `public int Transform(string functionName)`
  - This method applies the `functionName` for all the collections elements. As always, this function returns 1 if `functionName` is a valid method of the collection's elements, -1 otherwise.

    <u>You do not need to use a LINQ expression in this method.</u>
- `public int PairByProperty(string commonPropertyName, string uniquePropertyName, string printPropertyName)`
  - This method will take in a name of a property that multiple elements can have the same value of (`commonPropertyName`) and construct pairs of elements that share the same value of that property but differ on the value of the property `uniquePropertyName`.

    The pairs to be printed should be unique. This can be easily accomplished by making the "`where`" condition of your LINQ expression be that the value of the unique property of the first item of the pair should be less than that of the second item. Check the LinqStudents.cs example from the LINQ lecture.

    The printout should represent the elements of the pair by the value of the `printPropertyName` property and put the value of the unique property between parentheses separated by a comma from the value of the common property value.

For example, if the common property was that of a student's major, and the unique property was that of a student's id, and the print property was that of a student's name .. an example output should be:

```
Tom (100, CS) - - Dave (200, CS)
Alice (300, ECE) - - Mary (400, ECE)
```

Hints

- Consult the lecture slides and class examples for help with reflection and exception handling.
- You might find the following MSDN article on Reflection to be useful: http://msdn.microsoft.com/en-us/library/ms173183(VS.80).aspx
- You might find the following MSDN article on the Type class to be useful: http://msdn.microsoft.com/en-us/library/system.type(VS.80).aspx
- You might find the following MSDN article on the members of a Type object to be useful: http://msdn.microsoft.com/en-us/library/system.type_members(VS.80).aspx
- You might find the following MSDN article on the PropertyInfo class to be useful: http://msdn.microsoft.com/en-us/library/system.reflection.propertyinfo(VS.80).aspx
- You might find the following MSDN article on the members of a PropertyInfo object useful: http://msdn.microsoft.com/en-us/library/system.reflection.propertyinfo_members(VS.80).aspx
- You might find the following MSDN article on the MethodInfo class to be useful: http://msdn.microsoft.com/en-us/library/system.reflection.methodinfo(VS.80).aspx
- You might find the following MSDN article on the members of a MethodInfo object useful: http://msdn.microsoft.com/en-us/library/system.reflection.methodinfo_members(VS.80).aspx
- You may find yourself having to write three LINQ expressions and choosing to execute one of them based on the type of the property you are dealing with.

Why is this useful?

This assignment will exercise your skills in C# reflection and LINQ. LINQ is very useful when dealing with collections of data as you might imagine. Reflection is quite useful if you are building complex programs. So it is very useful to learn how to use these two functionalities of the C# language.

Academic Integrity Reminder

Remember that you may have general discussions about how to approach this problem with your peers, but you should work on the final solution by yourself alone. If you are stuck or are having trouble, you may email me or talk to me after class on Monday or during my office hour on Wednesday.

Good Luck!