Задания к модулю Expressions and IQueryable

# Expression Tree

# Задание 1.

Создайте класс-трансформатор на основе ExpressionVisitor, выполняющий следующие 2 вида преобразований дерева выражений:

* Замену выражений вида <переменная> + 1 / <переменная> - 1 на операции инкремента и декремента
* Замену параметров, входящих в lambda-выражение, на константы (в качестве параметров такого преобразования передавать:
  + Исходное выражение
  + Список пар <имя параметра: значение для замены>

Для контроля полученное дерево выводить в консоль или смотреть результат под отладчиком, использую ExpressionTree Visualizer, а также компилировать его и вызывать полученный метод.

## Задание 2.

Используя возможность конструировать Expression Tree и выполнять его код, создайте собственный механизм маппинга (копирующего поля (свойства) одного класса в другой).

Приблизительный интерфейс и пример использования приведен ниже (MapperGenerator – фабрика мапперов, Mapper – класс маппинга). Обратите внимание, что в данном примере создается только новый экземпляр конечного класса, но сами данные не копируются.

public class Mapper<TSource, TDestination>  
{  
    Func<TSource, TDestination> mapFunction;  
    internal Mapper(Func<TSource, TDestination> func)  
    {  
        mapFunction = func;  
    }  
    public TDestination Map(TSource source)  
    {  
        return mapFunction(source);  
    }  
}  
public class MappingGenerator  
{  
    public Mapper<TSource, TDestination> Generate<TSource, TDestination>()  
    {  
        var sourceParam = Expression.Parameter(typeof(TSource));  
        var mapFunction =   
            Expression.Lambda<Func<TSource, TDestination>>(  
            Expression.New(typeof(TDestination)),  
            sourceParam  
            );  
   
        return new Mapper<TSource, TDestination>(mapFunction.Compile());  
    }  
}  
public class Foo { }  
public class Bar { }  
   
[TestMethod]  
public void TestMethod3()  
{  
    var mapGenerator = new MappingGenerator();  
    var mapper = mapGenerator.Generate<Foo, Bar>();  
   
    var res = mapper.Map(new Foo());  
}

# IQueryable

## Задание 1.

Доработайте приведенный на лекции LINQ провайдер.

В частности, требуется добавить следующее:

* Снять текущее ограничение на порядок операндов выражения. Должны быть допустимы:
  + <имя фильтруемого поля> == <константа> (сейчас доступен только этот)
  + <константа> == <имя фильтруемого поля>
* Добавить поддержку операций включения (т.е. не точное совпадение со строкой, а частичное). При этом в LINQ-нотации они должны выглядеть как обращение к методам класса string: StartsWith, EndsWith, Contains, а точнее

|  |  |
| --- | --- |
| Выражение | Транслируется в запрос |
| Where(e => e.workstation.StartsWith("EPRUIZHW006")) | workstation:(EPRUIZHW006\*) |
| Where(e => e.workstation.EndsWith("IZHW0060")) | workstation:(\*IZHW0060) |
| Where(e => e.workstation.Contains("IZHW006")) | workstation:(\*IZHW006\*) |

* Добавить поддержку оператора AND (потребует доработки также самого E3SQueryClient). Организацию оператора AND в запросе к E3S смотрите [на странице документации](https://kb.epam.com/display/E3S/E3S+public+REST+for+data) (раздел FTS Request Syntax)