№ 8 Регулярные выражения, атрибуты валидайии, XML, меню, панели инструментов, строки состояния

Задание

К предыдущей ЛР № 7, добавить на форму меню с пунктами:

1) «Поиск» (подменю указанными в вариантах) В поиске, кроме поиска на полное соответствие, реализовать поиск по на основе регулярных выражений (диапазон, наличие букв на определенных позициях, число повторений символов и т.п.). Результаты поисковых запросов можно выводить в отельное окно. Сделайте отдельное окно для конструирования поисковых запросов (в том числе и по нескольким критериям).
2) «Сортировка по» (году, фамилии, специальности и т.п.) Для поиска, сортировки и модификаций используйте LINQ to XML.
3) «Сохранить» результаты поиска и сортировок в отдельных xml. Используйте сериализацию.
4) «О программе». При выборе пункта меню «О программе» должно выводиться окно сообщений с версией и ФИО разработчика.

При валидации вводимых данных используйте функционал в виде атрибутов из пространства имен System.ComponentModel.DataAnnotations и классов ValidationResult, Validator и ValidationContext. Используйте атрибуты RegularExpression, Range, свойство ErrorMessage и т.д. Создайте свой атрибут валидации.

Добавить панель инструментов с кнопками дублирующими команды «поиска», «сортировки», «очистить», «удалить», «вперед», «назад». Добавить возможность скрывать и закреплять панель инструментов.

Добавить строку состояния с тестовыми сообщениями о текущем количестве объектов и выполняемом действии, текущей датой и временем.

Вариант	Задание
1, 9	Поиск по: ФИО (по шаблону), специальности, курсу, среднему баллу (>n , диапазон). Сортировка по стажу работы и курсу.
2, 10	Поиск по: номеру, ФИО, балансу, типу вклада. Сортировка по типу вклада и дате открытия счета
3, 11	Поиск по: лектору, семестру и курсу. Сортировка по количеству лекций и виду контроля.

4, 12	Поиск по: издательству, году издания, диапазону страниц. Сортировка по автору(ам), дате поступления.
5, 13	Поиск по: классу, году, континенту. Сортировка по куратору, дате поступления
6, 14	Поиск по производителю и модели процессора. Сортировка по частоте работы процессора, размеру ОЗУ.
7, 15	Поиск по: авиакомпании, типу, количеству мест, грузоподъемности. Сортировка по ФИО командира, затем пилота, дате последнего тех. обслуживания
8, 16	Поиск по: названию, типу, диапазону цены. Сортировка по дате производства, стране производителя, затем по названию

Краткие теоретические сведения

Информацию о регулярных выражениях можно найти: на MSDN
Албахари_СправочникC#_2014.pdf глава 26 стр. 981

Регулярные выражения

Регулярные выражения – это язык для описания текста и внесения в него изменений. Регулярное выражение применяется к строке. Результатом применения является фрагмент строки, либо новая строка, либо группы подстрок, либо логический результат – в зависимости от того, какая операция выполняется.

Регулярные выражения очень мощный и в то же время простой механизм обработки текстовой информации. На данный момент наиболее полно они реализованы в язык Perl, хотя возникли гораздо раньше.

Для работы с регулярными выражениями в C# существует класс System.Text.RegularExpressions.Regex. Многие методы этого класса также существуют в двух версиях – статической и экземпляра.

У класса два конструктора с одним параметром строкового типа, определяющего правило обработки и с двумя – второй в этом случае задает параметры регулярного выражения (аналог опций в Perl).

В регулярных выражениях существует понятие «метосимвол», это аналог управляющей последовательности в строке. Если нам необходим символ как он есть, то перед ним ставится обратная наклонная черта. Некоторые символы наоборот начинаются с обратной наклонной черты. Именно по этому для РВ с C# лучше использовать дословные строки.

() – определение группы

| - задание перечисления

{n, m} – предназначены для обозначения кратности. В общем виде количество больше или равно n, но меньше или равное m. {2, 7}

Существуют частные случаи: {n,} – не менее, {,m} – не более, ровно – {n}, а также метасимволы

* аналогичен {0,}

+ аналогичен {1,}

? аналогичен {0,1}

^ - начало строки

$ - конец строки

[] – обозначение класса символов

[abcxyz] – обозначачет любой из символов класса,

[a-cx-y], [a-z], – можно использовать интервалы

\d –числовой символ аналог[0-9]

\D – нечисловой символ (Регулярные выражения регистрозависимы)

\w – алфавитно-цифровой символ или знак подчеркивания аналог [a-zA-Z_]

\W – не алфавитно-цифровой символ или знак подчеркивания

\s – пробел

\S – не пробел

```
Regex r1 = new Regex(@" |, |,");
string []a3 = r1.Split(s5);
foreach(string s in a3)
    Console.WriteLine(s);
```

Метод Split PB аналогичен строковому, но можно задать разделители состоящие более чем из одного символа (строки).

Для поиска в строке существуют два метода Match – ищет первое вхождение и Matches – ищет все вхождения. Первый возвращает объект класса Match, второй - коллекцию MatchCollection. Для выделения групп, как уже говорилось, используются круглые скобки

То что в круглых скобках и будет заносится в коллекцию MatchCollection.

```
Regex r2 = new Regex(@"(\w+)");
MatchCollection mc = r2.Matches(s5);
foreach(Match m in mc)
    Console.WriteLine(m.Value);
```

С помощью регулярных выражений можно проверять строку на соответствие какому-либо формату.