Практическая работа №3

Лямбда-выражения

Лямбда-выражения представляют упрощенную запись анонимных методов. Лямбда-выражения позволяют создать емкие лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы.

Лямбда-выражения имеют следующий синтаксис: слева от лямбда-оператора `=>` определяется список параметров, а справа блок выражений, использующий эти параметры:

```
(список_параметров) => выражение
```

Посмотрим как использовать лямбда-выражения в свойствах:

```
//Свойство в данном случае будет только для чтения
public int RowCount => _items.GetLength(0);
```

При обращении к свойству `RowCount` код вызовет метод `GetLength` у массива и вернет количество строк в нем. Так же лямбда-выражения можно использовать и в полных свойствах:

```
public T this[int rowIndex, int columnIndex]
{
    get => _items[rowIndex, columnIndex];//Аналог return _items[rowIndex,
columnIndex];
    set => _items[rowIndex, columnIndex] = value;
}
```

Default-значение

Выражение default value создает значение по умолчанию для типа.

Начиная с C# 7.1, можно использовать литерал `default` для создания значения по умолчанию для типа, если компилятор может вывести тип выражения. Литеральное выражение `default` создает то же значение, что и выражение `default(T)`, где `T` является выведенным типом. Литерал `default` можно использовать в любом из следующих случаев:

- при назначении или инициализации переменной;
- в объявлении значения по умолчанию для необязательного параметра метода;
- в вызове метода для предоставления значения аргумента;
- в инструкции `return` или в качестве выражения в элементе в тексте выражения.

Ниже приведен пример применения литерала `default`.

```
int number = default;//0
double floatNumber = default;//0.0
string text = default;//null
Random random = default//null
```

Сериализация и десереализация

Ранее мы посмотрели, как сохранять информацию в текстовые файлы. Но нередко подобных механизмов оказывается недостаточно особенно для сохранения сложных объектов. С этой проблемой призван справится механизм сериализации. **Сериализация** представляет процесс преобразования какого-либо объекта в поток байтов. После преобразования мы можем этот поток байтов или записать на диск или сохранить его временно в памяти. А при необходимости можно выполнить обратный процесс - **десериализацию**, то есть получить из потока байтов ранее сохраненный объект.

Формат сериализации

Хотя сериализация представляет собой преобразование объекта в некоторый набор байтов, но в действительности только бинарным форматом она не ограничивается. Итак, в .NET можно использовать следующие форматы:

- бинарный формат - класс `BinaryFormatter`
- SOAP - класс `SoapFormatter`
- xml - класс `XmlSerializer`
- JSON - класс `DataContractJsonSerializer`

В классе `BinaryFormatter` для сериализации будет использоваться метод `Serialize`, который в качестве параметров принимает поток, куда помещает сериализованные данные (например, бинарный файл), и объект, который надо сериализовать. А для десериализации будет применяться метод `Deserialize`, который в качестве параметра принимает поток с сериализованными данными.

Пример сериализации двухмерного массива:

```
//При указания пути необходимо указывать и расширения файла
string fullPath = @".\object.matrix";
T[,] data = new T[3, 3];
BinaryFormatter formatter = new();
//Режим работы с файлом указыается - создание
using (FileStream stream = new(fullPath, FileMode.Create))
{
    formatter.Serialize(stream, data);
}
```

Пример десериализации двухмерного массива:

```
//При указания пути необходимо указывать и расширения файла
string fullPath = @".\object.matrix";
T[,] data;
BinaryFormatter formatter = new();
//Режим работы с файлом указыается - открытие
using (FileStream stream = new(fullPath, FileMode.Open))
{
    data = _formatter.Deserialize(stream) as T[,];
}
```

Самостоятельная работа в домашних условиях

1. Изучить формулировку задания.
2. Разработать спецификации модулей.
3. Составить паспорта модулей (функций) производящих вычисления по заданию.

Практическое занятие в учебном классе

1. Разработать библиотеку, содержащую класс для работы с двухмерным массивом.

   Реализовать методы:

   - Устанавливающий всем элементам массива значение типа по-умолчанию - `void DefaultInit()`;
   - Сохраняющие элементы двухмерного массива в файл через сериализацию данных - `void Save(string path)`, где `path` - путь сохранения файла;
   - Загружающий элементы двухмерного массива из файла через десериализацию данных - `void Load(string path)`, где `path` - путь сохранения файла;

   Реализовать индексатора для доступа к элемента массива - `T this[int rowIndex, int columnIndex]`

   Реализовать свойство для хранение расширения файла при сохранение - `string Extension`, доступное только для чтения вне класса

   Реализовать класс обобщенным - `Matrix<T>`

   Название библиотеки **LibMatrix**. Название класса - **Matrix**.

2. Разработать библиотеку, содержащую вычислительные модули (функции) программы для решения задачи по варианту задания.

   Название библиотеки **Lib_x**, где **x** – номер варианта задания. Название класса - **ExtensionMatrix**.

3. Использовать для отображения информации элемент **DataGrid**.
4. Использовать элемент меню - **Menu**.
5. Добавить иконку в заголовок программы и исполняемый файл. Заполнить заголовок программы.
6. Предусмотреть в программе две кнопки «Выход» и «О программе», где вывести ФИО разработчика, номер работы и формулировку задания.
7. В каждом задание необходимо реализовать 2 метода.

   Первый должен генерировать массив с числами, в качестве параметром принимать количество чисел, минимальное и максимальное возможное значение. Диапазон чисел сделать необязательными параметрами. Сделать его методом расширения.

   Второй выполнить вычисления на созданным массивом. Сделать его методом расширения.
8. Назвать переменные, параметры, методы и другие элементы согласно стандартам оформления кода C#.
9. Составить Xml комментарии.

Варианты заданий (Базовый уровень)

1. Найти сумму чисел > 5. Результат вывести на экран.
2. Найти произведение чисел. Результат вывести на экран.
3. Найти разницу чисел. Результат вывести на экран.
4. Вычислить для чисел > 0 функцию x . Результат обработки каждого числа вывести на экран.
5. Найти произведение чисел < 3. Результат вывести на экран.
6. Найти сумму чисел < 15. Результат вывести на экран.

7. Вычислить для чисел < 0 функцию x^2. Результат обработки каждого числа вывести на экран.
8. Вычислить косинус (cos) суммы чисел < 3. Результат вывести на экран.
9. Найти произведение чисел. Результат вывести на экран.
10. Вычислить для чисел > 0 функцию x . Результат обработки каждого числа вывести на экран.
11. Найти разницу чисел. Результат вывести на экран.
12. Найти сумму чисел >15. Результат вывести на экран.
13. Найти произведение чисел > 2. Результат вывести на экран.
14. Найти сумму чисел < 8. Результат вывести на экран.
15. Вычислить функцию sin суммы чисел > 3. Результат вывести на экран.

Пример использования расширения

```
//Обобщенный класс
public class Matrix<T>
{
    //Поля
    private T[,] _items;
    //Конструктор
    public Matrix(int rowCount, int columnCount, string extension = ".matrix")
    {
        _items = new T[rowCount, columnCount];
        Extension = extension;
    }

    public string Extension { get; private set; }
}
```