**Занятие № 11**

**Дата выполнения работы:** 02.05.2023

# **Тема работы:** «Делегаты, события, потоки выполнения»

**Ход работы**

**Задание 1**

1. Используя делегаты (множественные) и события промоделируйте

ситуации, приведенные в таблице ниже. Можете добавить новые типы

(классы), если существующих недостаточно. При реализации методов везде

где возможно использовать лямбда-выражения.

2. Создайте пять методов пользовательской обработки строки

(например, удаление знаков препинания, добавление символов, замена на

заглавные, удаление лишних пробелов и т.п.). Используя стандартные типы

делегатов (Action, Func) организуйте алгоритм последовательной обработки

строки написанными вами методами.

Создать класс Пользователь с событиями Переместить (можно

задать смещение) и Сжать (коэффициент сжатия). В main

создать некоторое количество объектов разного типа. Часть

объектов подписать на одно событие, часть на два (часть можете

не подписывать). Проверить состояния объектов после

наступления событий.

**Листинг программы:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace App

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

var processor = new StringProcessor();

processor.Register(StringActions.RemovePunctuation);

processor.Register(StringActions.AddCharacters);

processor.Register(StringActions.ToUpperCase);

processor.Register(StringActions.RemoveExtraSpaces);

processor.Register(StringActions.Reverse);

var result = processor.Process("Hello, World!");

// Output: !!!DLROW OLLEh

}

private void button1\_Click(object sender, EventArgs e)

{

var user1 = new User();

var user2 = new User();

var user3 = new User();

var user4 = new User();

user1.OnMove += (x, y) => textBox1.Text = ($"User 1 moved to ({x}, {y})");

user1.OnCompress += ratio => textBox2.Text = ($"User 1 compressed by {ratio}");

user2.OnMove += (x, y) => textBox3.Text = ($"User 2 moved to ({x}, {y})");

user3.OnCompress += ratio => textBox4.Text = ($"User 3 compressed by {ratio}");

user4.OnMove += (x, y) => textBox5.Text = ($"User 4 moved to ({x}, {y})");

user4.OnCompress += ratio => textBox6.Text = ($"User 4 compressed by {ratio}");

user1.Move(10, 20); // Output: User 1 moved to (10, 20)

user1.Compress(0.5); // Output: User 1 compressed by 0.5

user2.Move(-5, 10); // Output: User 2 moved to (-5, 10)

user3.Compress(0.8); // Output: User 3 compressed by 0.8

user4.Move(15, -30); // Output: User 4 moved to (15, -30)

user4.Compress(0.2); // Output: User 4 compressed by 0.2

}

}

public class StringProcessor

{

public delegate string StringAction(string input);

private event StringAction OnProcess;

public void Register(StringAction action)

{

OnProcess += action;

}

public void Unregister(StringAction action)

{

OnProcess -= action;

}

public string Process(string input)

{

if (OnProcess == null)

{

return input;

}

foreach (var action in OnProcess.GetInvocationList())

{

var stringAction = (StringAction)action;

input = stringAction(input);

}

return input;

}

}

public static class StringActions

{

public static string RemovePunctuation(string input)

{

return new string(input.Where(c => !char.IsPunctuation(c)).ToArray());

}

public static string AddCharacters(string input)

{

return input + "!!!";

}

public static string ToUpperCase(string input)

{

return input.ToUpper();

}

public static string RemoveExtraSpaces(string input)

{

return string.Join(" ", input.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries));

}

public static string Reverse(string input)

{

return new string(input.Reverse().ToArray());

}

}

public class User

{

public event Action<int, int> OnMove;

public event Action<double> OnCompress;

public void Move(int x, int y)

{

OnMove?.Invoke(x, y);

}

public void Compress(double ratio)

{

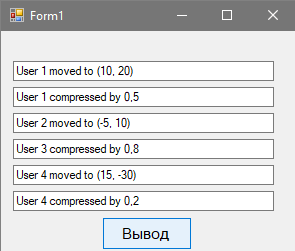
OnCompress?.Invoke(ratio);

}

}

}

**Результат:**

****

**Вывод:** После прохождения темы "Делегаты" в C# я понял, что делегаты представляют собой мощный механизм для передачи методов между объектами, что упрощает написание гибкого и расширяемого кода. Я научился создавать и использовать делегаты, а также создавать события и подписываться на них. Я также ознакомился со стандартными типами делегатов Action и Func, которые можно использовать для передачи методов без параметров и с параметрами соответственно. В целом, использование делегатов в C# позволяет создавать более эффективный и гибкий код, что является важным инструментом для разработки приложений на данном языке программирования.