**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

«Работа с делегатами и рефлексией»

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студентка группы ИУ5-33 |  | преподаватель каф. ИУ5 |
| Беспалова У.А. |  | Гапанюк Ю.Е. |
| Подпись и дата:  22.11.2018 |  | Подпись и дата: |

Москва, 2018 г.

1. Содержание задания

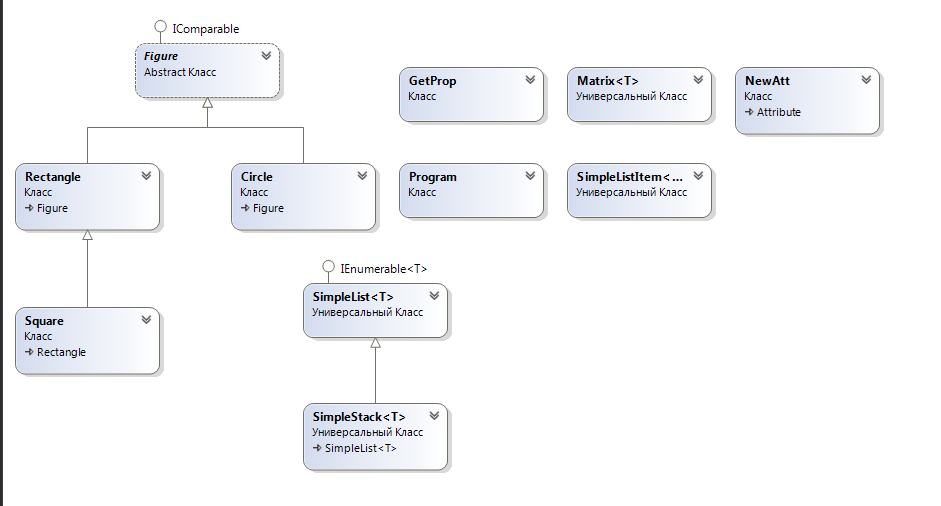
**Часть 1. Разработать программу, использующую делегаты.**

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, в качестве одного из входным параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:
   1. метод, разработанный в пункте 3;
   2. лямбда-выражение.
5. Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

**Часть 2. Разработать программу, реализующую работу с рефлексией.**

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.
6. Вызовите один из методов класса с использованием рефлексии.

Диаграмма классов



Текст программы

Часть 1

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace лаба\_6

{

class Program

{

delegate string Del1 (int a, string c); //объявление делегата

//Func<string, int, string> Del2; //использование обобщенного делегата

static string Met1(int a, string str) //метод, соотв 1 делегату

{

string result = a.ToString() + ". " + str;

return (result);

}

static string Met2(string str, int a) //метод, соотв 2 делегату

{

string result = a.ToString() + ". " + str;

return (result);

}

static void HiMethod1 (int n1, Del1 Met1) //превращает строку в строку-с-циферкой-впереди

{

Console.WriteLine("Введите строку: ");

string str1 = Console.ReadLine();

Console.WriteLine(Met1(n1, str1));//вызвали функцию с точкой

}

static void HiMethod2(int n1, Func<string, int, string> Met2)

{

Console.WriteLine("Введите строку: ");

string str1 = Console.ReadLine();

Console.WriteLine(Met2(str1, n1));

}

static void Main(string[] args)

{

Console.WriteLine("Метод с делегатом");

HiMethod1(1, Met1); //передали метод, ставящий точку

Console.WriteLine("Метод с лямбда-выражением");

HiMethod1(2, //второй вариант с лямбда-выражением, ставящим скобочку

(b, str2) => (b.ToString() + ") " + str2) //сокращенный вариант: что дано и что вернуть

);

Console.WriteLine("Метод с обобщённым делегатом");

HiMethod2(3, Met2);

Console.WriteLine("Метод с обобщенным делегатом и лямбда-выражением");

HiMethod2(4,

(str2, b) => (b.ToString() + ") " + str2)

);

Console.ReadLine();

}

}

}

Часть 22 (реализация на основе Лабораторной работы №3)

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Reflection;

namespace лаба3

{

class Program

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle(3, 4); //создали объект класса Прямоугольник размером 3на4 и так далее

Circle circle = new Circle(3);

Square sq = new Square(4);

rect.Print(); //вывели ответ

circle.Print();

sq.Print();

ArrayList arr = new ArrayList(); //пустая коллекция

arr.Add(rect);

arr.Add(circle);

arr.Add(sq);

arr.Sort(); //отсортировали

Console.WriteLine("\n\nОтсортированный ArrayList:\n");

foreach (Figure ivan in arr) //все элементы имеют тип Figure

{

Console.WriteLine(ivan);

}

List<Figure> list = new List<Figure>();

list.Add(rect);

list.Add(circle);

list.Add(sq);

list.Sort();

Console.WriteLine("\n\nОтсортированный List:\n");

foreach (Figure ivan in list)

{

Console.WriteLine(ivan);

}

Console.WriteLine("\n\nМатрица:\n");

Matrix<Figure> cube = new Matrix<Figure>(2, 2, 2, null); //создали кубик с ребром 2 ячейки

cube[0, 0, 1] = rect;

cube[0, 1, 1] = sq;

cube[1, 1, 1] = circle;

Console.WriteLine(cube.ToString());

SimpleList<Figure> sl = new SimpleList<Figure>//создали коллекцию

{

rect, circle, sq

};

sl.Sort();

Console.WriteLine("\nОтсортированный SimpleList\n");

foreach (var i in sl)

{

Console.WriteLine(i.ToString());

}

SimpleStack<Figure> st = new SimpleStack<Figure>();

st.Push(rect);

st.Push(circle);

st.Push(sq);

Console.WriteLine("\nSimpleStack\n");

foreach (var s in st)

{

Console.WriteLine(s.ToString());

}

Figure P = st.Pop(); //чтение с удалением

Console.WriteLine("\nУдалено " + P.ToString() + "\n");

foreach (var s in st)

{

Console.WriteLine(s.ToString());

}

//---------------------------------------------------

//---------------------------------------------------

Console.WriteLine("\n\n\n\nРабота с рефлексией");

Type t = typeof(Rectangle);

Console.WriteLine("\n\nТип " + t.FullName + " унаследован от " + t.BaseType.FullName);

Console.WriteLine("Пространство имён: " + t.Namespace);

Console.WriteLine("Находится в сборке " + t.AssemblyQualifiedName);

Console.WriteLine("\nКонструкторы:");

foreach (var i in t.GetConstructors())

{

Console.WriteLine(i);

}

Console.WriteLine("\nМетоды:");

foreach (var i in t.GetMethods())

{

Console.WriteLine(i);

}

Console.WriteLine("\nСвойства:");

foreach (var i in t.GetProperties())

{

Console.WriteLine(i);

}

Console.WriteLine("\nПоля данных:");

foreach (var i in t.GetFields())

{

Console.WriteLine(i);

}

t = typeof(Figure); //перешли к классу фигуры

Console.WriteLine("\nСвойства, помеченные атрибутом:");

//проверяем, какие свойства снабжены атрибутом, и выводим для них строку из Descript

foreach (var i in t.GetProperties())

{

object attOb;

if (GetProp.GetPropertyAttribute(i, typeof(NewAtt), out attOb))

{

NewAtt attr = attOb as NewAtt; //приведение к типу

Console.WriteLine(i.Name + " - " + attr.Descript);

}

}

t = typeof(Rectangle);

Console.WriteLine("\nВызов метода:"); //использование рефлексии

Rectangle ff = (Rectangle)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] {2, 3});

//создали объект (прямоугольник) со сторонами 2 и 3, без сюрпризов

object[] parameters = new object[] { };//параметры вызова метода (нет параметров)

t.InvokeMember("Print", BindingFlags.InvokeMethod, null, ff, parameters); //посчитали площадь

Console.ReadLine();

}

}

}

Класс Atr

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]

class NewAtt:Attribute

{

public NewAtt() { } //конструкторы

public NewAtt(string DescPar)

{

Descript = DescPar;

}

public string Descript { get; set; }

}

Класс GetProp

class GetProp

{

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attType, out object attribute) //проверка

//получает информацию о проверяемом свойстве, тип атрибута

{

bool Res = false;

attribute = null;

var isAttribute = checkType.GetCustomAttributes(attType, false); //возвращает найденные элементы

if (isAttribute.Length > 0)//если их больше 0

{

Res = true;

attribute = isAttribute[0]; //в выходной параметр записываем 1 элемент

}

return Res;

}

}

Класс Figure

abstract class Figure : IComparable //геометрическая фигура (+реализует интерфейс сравнения)

{

string \_Type;//название типа

[NewAtt (Descript = "помеченное свойство")]//пометили атрибутом

public string Type //свойство

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

public virtual double Area() { return 0; } //заготовка для площади

public override string ToString() //вывод ответа

{

return ("Площадь " + this.Type + "а = " + this.Area().ToString());

}

interface IPrint //заготовка для интерфейса

{

void Print();

}

public int CompareTo(object o)

{

Figure c = (Figure)o;//приводим тип

if (c != null)

{

return this.Area().CompareTo(c.Area());

}

else

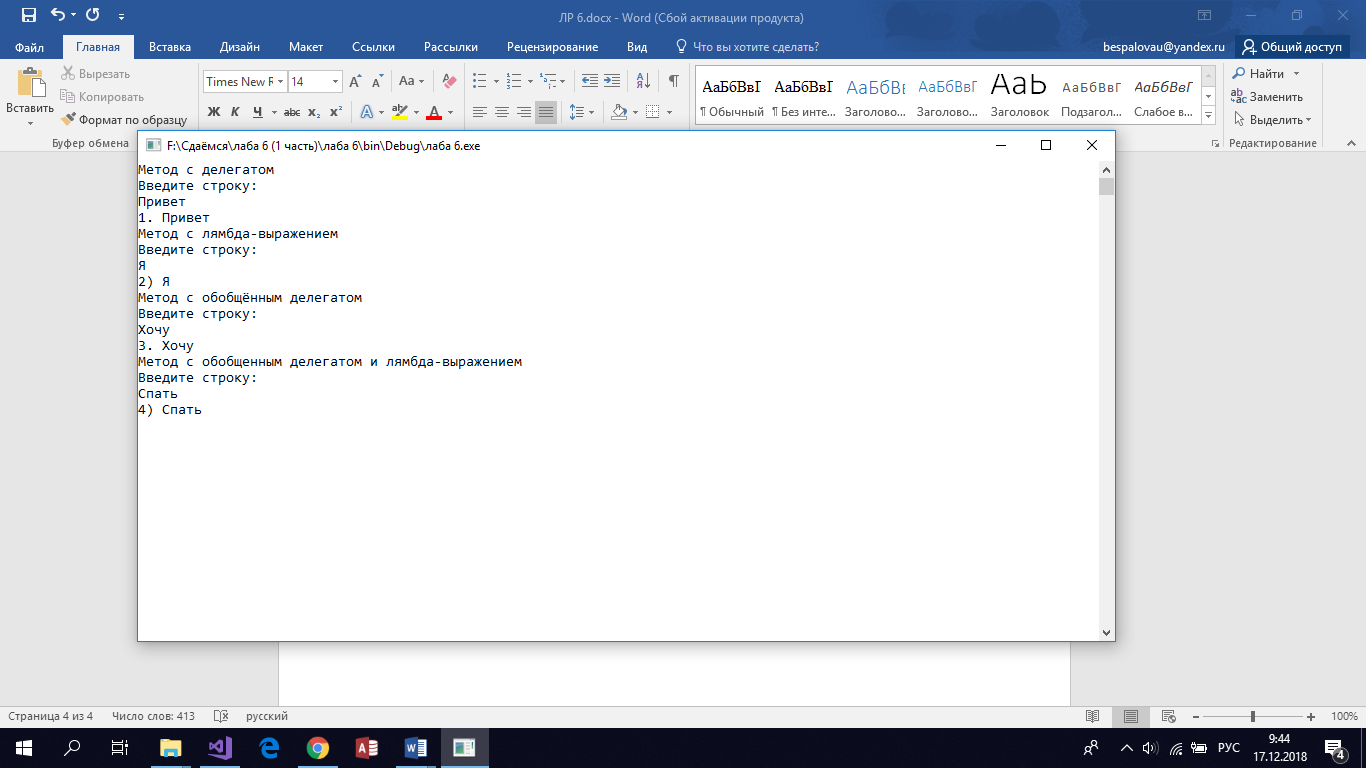
throw new Exception("Не удалось сравнить два объекта"); //ругаемся, если не удалось привести тип

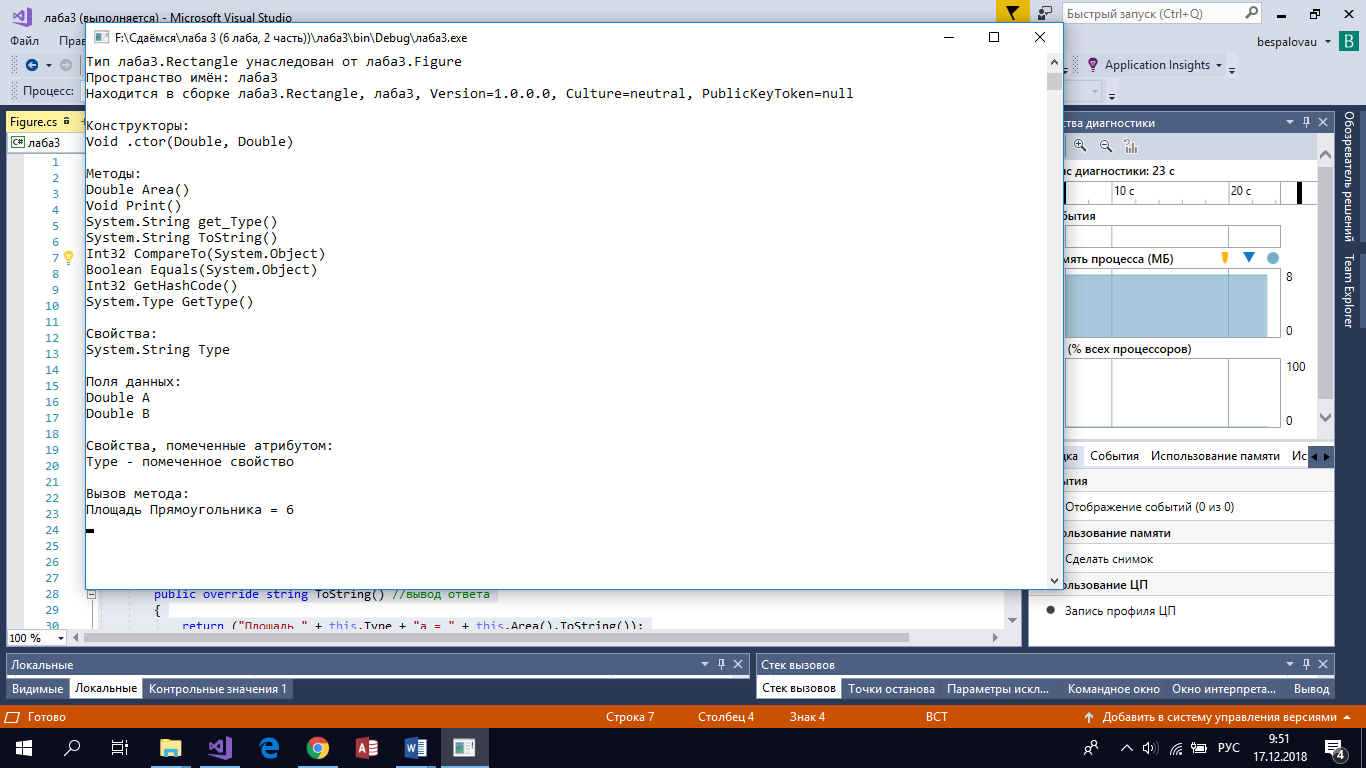
}

}

Экранные формы с примерами работы программы

Часть 1



Часть 2