**13. События**

Задание №1. Создать 3 метода, подсчитывающих периметр, площадь квадрата, и выводящий сторону треугольника. Создать делегат. В программе вызывать все три метода при помощи делегата.

Листинг программы:

using System.ComponentModel.Design;

Console.Write("Enter side equilateral triangle: ");

double side = double.Parse(Console.ReadLine());

TrianglCalc tc = SideWithdrawal;

tc += Square;

tc+=Perimeter;

Console.WriteLine($"Result of delegate execution: {tc(side)}");

const int three = 3;

static double Perimeter(double side)

{return side \* three;}

static double Square(double side)

{double height = Math.Sqrt(3) / 3 \* side;

return 0.5 \* side \* height;}

static double SideWithdrawal(double side)

{return side;}

delegate double TrianglCalc(double side);

Таблица 13.1 – Входные и выходные данных задание №1

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Сторона = 3 | Результат = 9 |

Анализ результатов:



Рисунок 13.1 – Результат работы программы из задания №1

Задание №2. На основе своей программы создать метод, в котором делегат будет параметром.

Листинг программы:

using System.ComponentModel.Design;

Console.Write("Enter side equilateral triangle: ");

double side = double.Parse(Console.ReadLine());

TrianglCalc tc = SideWithdrawal;

tc += Square;

tc += Perimeter;

ParamTrianglCalc(side, tc);

void ParamTrianglCalc(double side, TrianglCalc tc) => Console.WriteLine($"Result of delegate execution: {tc(side)}");

const int three = 3;

static double Perimeter(double side)=> side \* three;

static double Square(double side)

{ double height = Math.Sqrt(3) / 3 \* side;

return 0.5 \* side \* height;}

static double SideWithdrawal(double side) => side;

delegate double TrianglCalc(double side);

Таблица 13.2 – Входные и выходные данных задание №2

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Сторона = 4 | Результат = 12 |

Анализ результатов:



Рисунок 13.2 – Результат работы программы из задания №2

Задание №3. Для класса MyInfo содержащего поле name (Ваше имя) создать событие выводящее оповещение в случае изменения значения поля name.

Листинг программы:

internal class Subscriber

{ public string Name { get; set; }

public Subscriber(string name)

{ Name = name;}

public void Show(string message)

{ Console.WriteLine($"{Name} see {message}");}}

public delegate void Delegate(string message);

public event Delegate NameСhange = delegate { };

private string \_name;

public string Name

{ get => \_name;

set

{ if (\_name != value)

{ \_name = value;

NameСhange($"Имя было изменено на {\_name}");

return; }

NameСhange($"Имя не изменилось"); } }

using \_3;

//1

MyInfo info = new MyInfo("Света");

Subscriber subscriberVasia = new Subscriber("Vasia");

//2 Подписаться на событие

info.NameСhange += Name\_NameСhange;

info.NameСhange += subscriberVasia.Show;

//3 Вызов метода вызывающий событие

while (true)

{ Console.Write("Enter name: ");

string name = Console.ReadLine();

//name.Change(n);

info.Name = name;}

void Name\_NameСhange(string message)

{ Console.WriteLine(message);}

Таблица 13.3 – Входные и выходные данных задание №3

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Имя:Света | Имя не изменилось |

Анализ результатов:

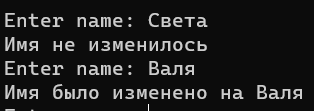


Рисунок 13.3 – Результат работы программы из задания №3

Задание №4. Создать класс, в нем делегат и соответствующее ему событие. Создать два класса-наблюдателя, в методах которых будет описана их реакция на событие. Добавить 2 обработчика к событию из первого класса и один из второго. Вывести результат на экран. Удалить один обработчик события и вывести результат на экран.

Листинг программы:

internal class Blogger

{ delegate void PublicPhoto(Blogger blogger);

event PublicPhoto? pb;

private string \_name;

public string Name { get => \_name; }

public Blogger(string name)

{ \_name = name;

} public void Attach(IObserver observer)

{ pb += observer.Update;

observer.Subscribing(this);

} public void Detach(IObserver observer)

{ pb -= observer.Update;

observer.Unsubscribing(this); } internal interface IObserver

{ void Update(Blogger blogger);

void Subscribing(Blogger blogger);

void Unsubscribing(Blogger blogger); }

using \_4;

Blogger blogger = new Blogger("Pokrov");

Subscriber1 subscriber1 = new Subscriber1("Sveta");

Subscriber2 subscriber2 = new Subscriber2("Valya");

blogger.Attach(subscriber1);

blogger.Detach(subscriber2);

internal class Subscriber1: IObserver

{ private string \_name;

public Subscriber1(string name)

{ \_name = name; }

public void Subscribing(Blogger blogger) => Console.WriteLine($"Подписчик {\_name} подписался на блогера: {blogger.Name} ");

public void Unsubscribing(Blogger blogger) => Console.WriteLine($"Подписчик {\_name} отписался от блогера: {blogger.Name}");

public void Update(Blogger blogger)

{ Console.WriteLine($"Подписчик {\_name} был уведомлен о публикации фото "); }

Таблица 13.4 – Входные и выходные данных задание №4

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  | Blogger Pokrov post a photo  Подписчик Sveta подписался на блогера: Pokrov  Подписчик Valya подписался на блогера: Pokrov |

Анализ результатов:

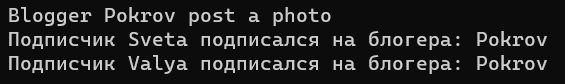


Рисунок 13.4 – Результат работы программы из задания №4