Язык программирования C#

C# Programming Language

Лекция 6. Методы. Конструкторы

Class

Класс–это конструкция языка, состоящая из ключевого слова class, идентификатора (имени) и тела.

Класс может содержать в своем теле: поля, методы, свойства и события.

Поля определяют состояние, а методы поведение будущего объекта.

```
classMyClass
{
    public int field; // Поле
    public void Method()// Метод
    {
        Console.WriteLine(field);
    }
}
```

Очень важно, **класс –**
это логическая абстракция.

Физическое представление класса в оперативной памяти появится лишь после того, как будет создан **объект этого класса**

Методы и переменные,
составляющие класс,
принято **называть членами класса**.

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В C# используется спецификация класса для построения объектов , которые являются **экземплярами класса**.

Класс, по существу, представляет собой ряд схематических описаний способа построения объекта. При этом очень важно подчеркнуть, что класс является логической абстракцией.

Метод –это конструкция языка, которая определяет (описывает) порядок выполнения некоторых действий

Методы представляют собой подпрограммы, которые манипулируют данными, определенными в классе, а во многих случаях они предоставляют доступ к этим данным. Как правило, другие части программы взаимодействуют с классом посредством его методов.

Метод состоит из одного или нескольких операторов. В грамотно написанном коде C# каждый метод выполняет только одну функцию. У каждого метода имеется свое имя, по которому он вызывается. Следует, однако, иметь в виду, что идентификатор Main() зарезервирован для метода, с которого начинается выполнение программы.

· Метод — это именованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо.

· Метод — это функция или процедура, выполняющая одну задачу.

· О функциях и процедурах. В некоторых языках программирования (например, в Паскале) функции и процедуры (подпрограммы, не возвращающие значения) чётко разграничены синтаксисом языка.

В языке C#, — процедуры являются частным случаем (подмножеством) функций, возвращающими значение типа void — пустое значение.

3)Если метод принимает аргументы – обязательно указать их тип и имя, если нет – оставить аргументные скобки () пустыми.

4)Если метод имеет возвращаемое значение, обязательно в теле метода должно присутствовать ключевое слово return. Тип возвращаемого значения метода должен соответствовать типу значения, указанному после ключевого слова return.

Для вызова метода необходимо:

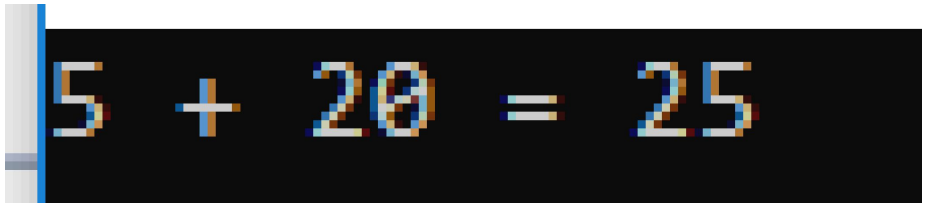
1)Написать имя метода.

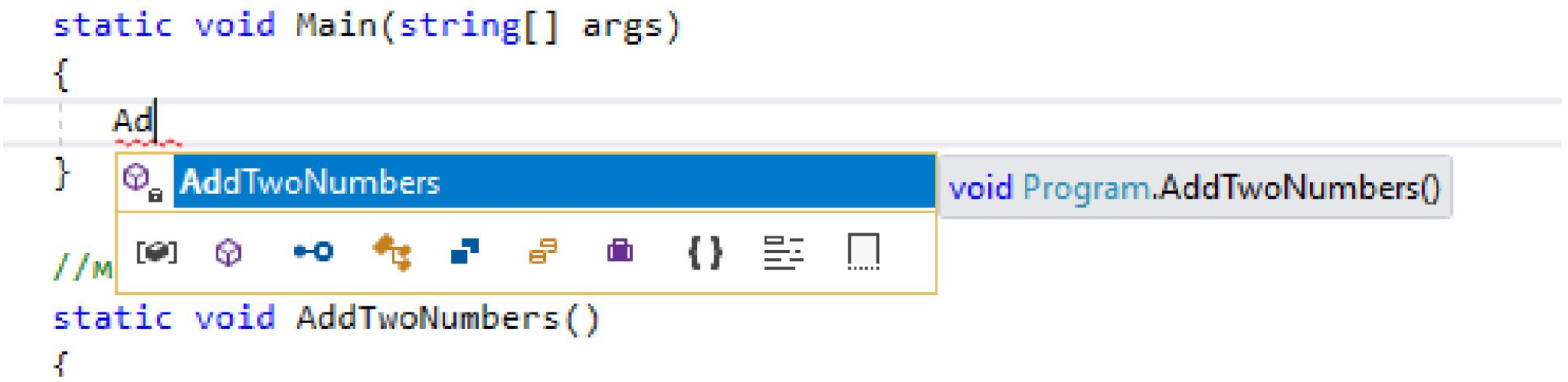
2)Обязательно указать после имени аргументные скобки(), если метод принимает какие-то аргументы, передать соответствующее количество аргументов соответствующего типа.

Сигнатура и Тело

```
namespace HelloApp
{
   class Program
   {
      static void Main(string[] args)
      {
         SayHello();
         SayGoodbye();

         Console.ReadKey();
      }

      static void SayHello()
      {
         Console.WriteLine("Hello");
      }
      static void SayGoodbye()
      {
         Console.WriteLine("GoodBye");
      }
   }
}
```

```
namespace MethodBasics
{
    class Program
    {
        static void Main(string[] args)
        {
            AddTwoNumbers();
        }

        //метод для сложения двух чисел
        static void AddTwoNumbers()
        {
            int x = 5;
            int y = 20;
            int result = x + y;
            Console.WriteLine("{0} + {1} = {2}", x, y, result);
            Console.ReadKey();
        }
    }
}
```

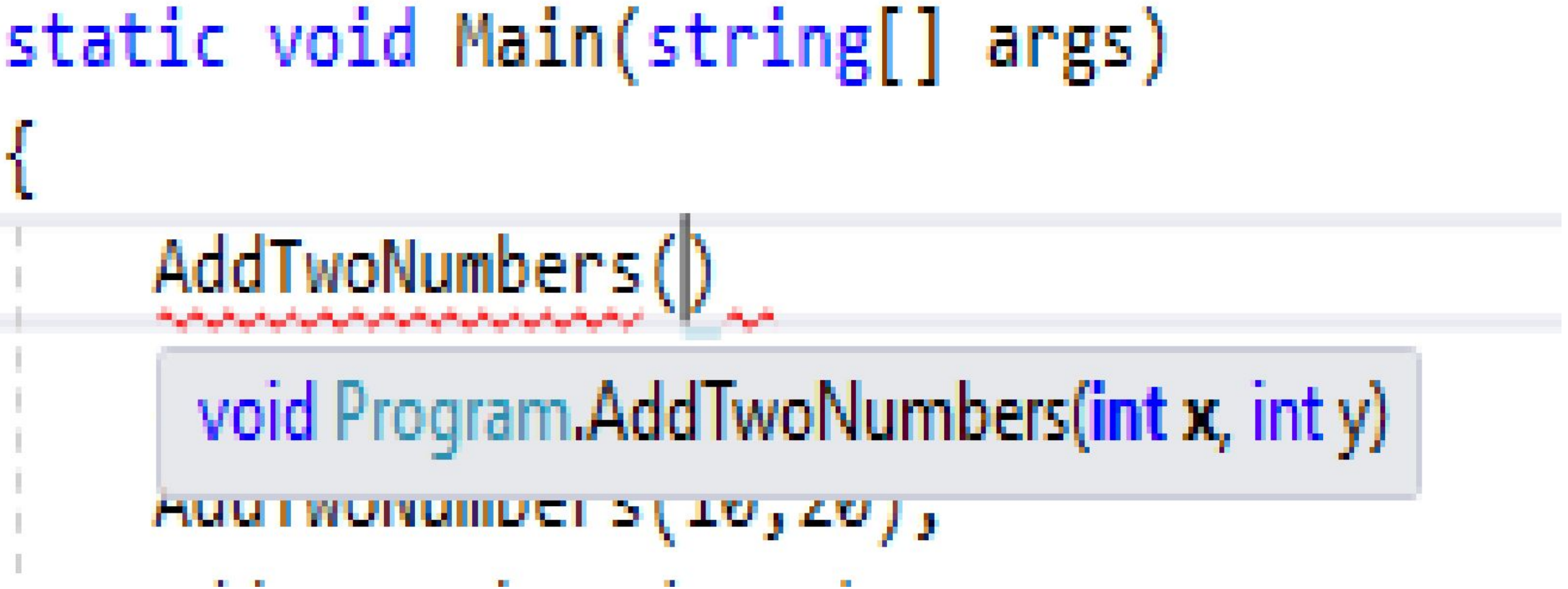


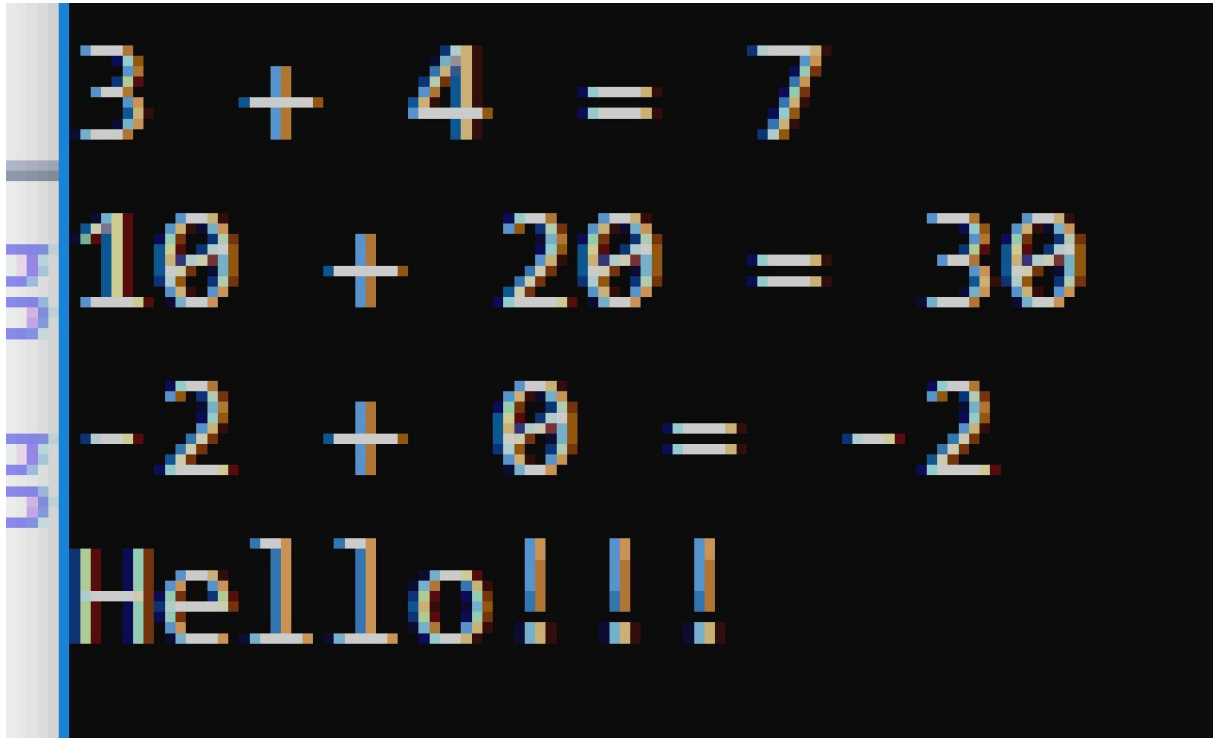



```
static void Main(string[] args)
    {
        AddTwoNumbers();
    }
```

```
 //Параметры методов
class Program
    {
        static void Main(string[] args)
        {
            AddTwoNumbers(3, 4);
            AddTwoNumbers(10,20);
            AddTwoNumbers(-2,0);
            TypeMessage("Hello");
            Console.ReadKey();
        }
```

```
//метод для сложения двух чисел
        static void AddTwoNumbers(int x, int y)
        {
            int result = x + y;
            Console.WriteLine("{0} + {1} = {2}", x, y, result);
        }

        //метод выводит на консоль сообщение
        static void TypeMessage(string message)
        {
            message = message + "!!!";
            Console.WriteLine(message);
        }
```


static void Main(string[] args)
{
AddTwoNumbers()
void Program.AddTwoNumbers(int x, int y)


3 + 4 = 7
10 + 20 = 30
-2 + 0 = -2
Hello!!!

```
 static void Main(string[] args)
 {
     AddTwoNumbers(3, 4);
     AddTwoNumbers(10,20);
     AddTwoNumbers(-2,0);

//метод для сложения двух чисел
static void AddTwoNumbers(string x, int y)
{
    int result = x + y;
    Console.Write                                    );
}

//метод выводит н
static void TypeMessage(string message)
```

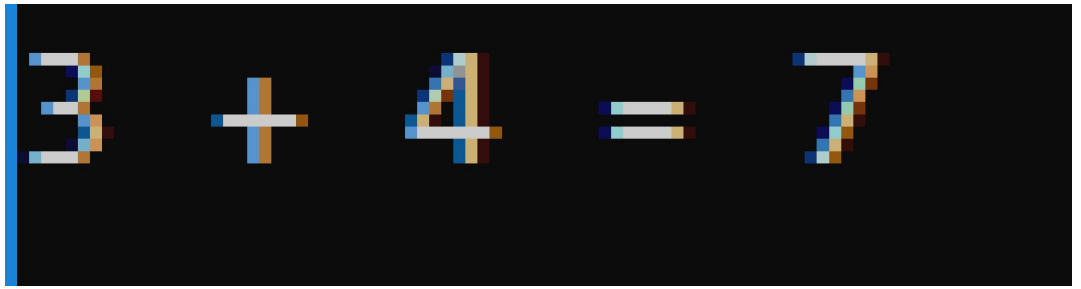
(parameter) string x

Cannot implicitly convert type 'string' to 'int'

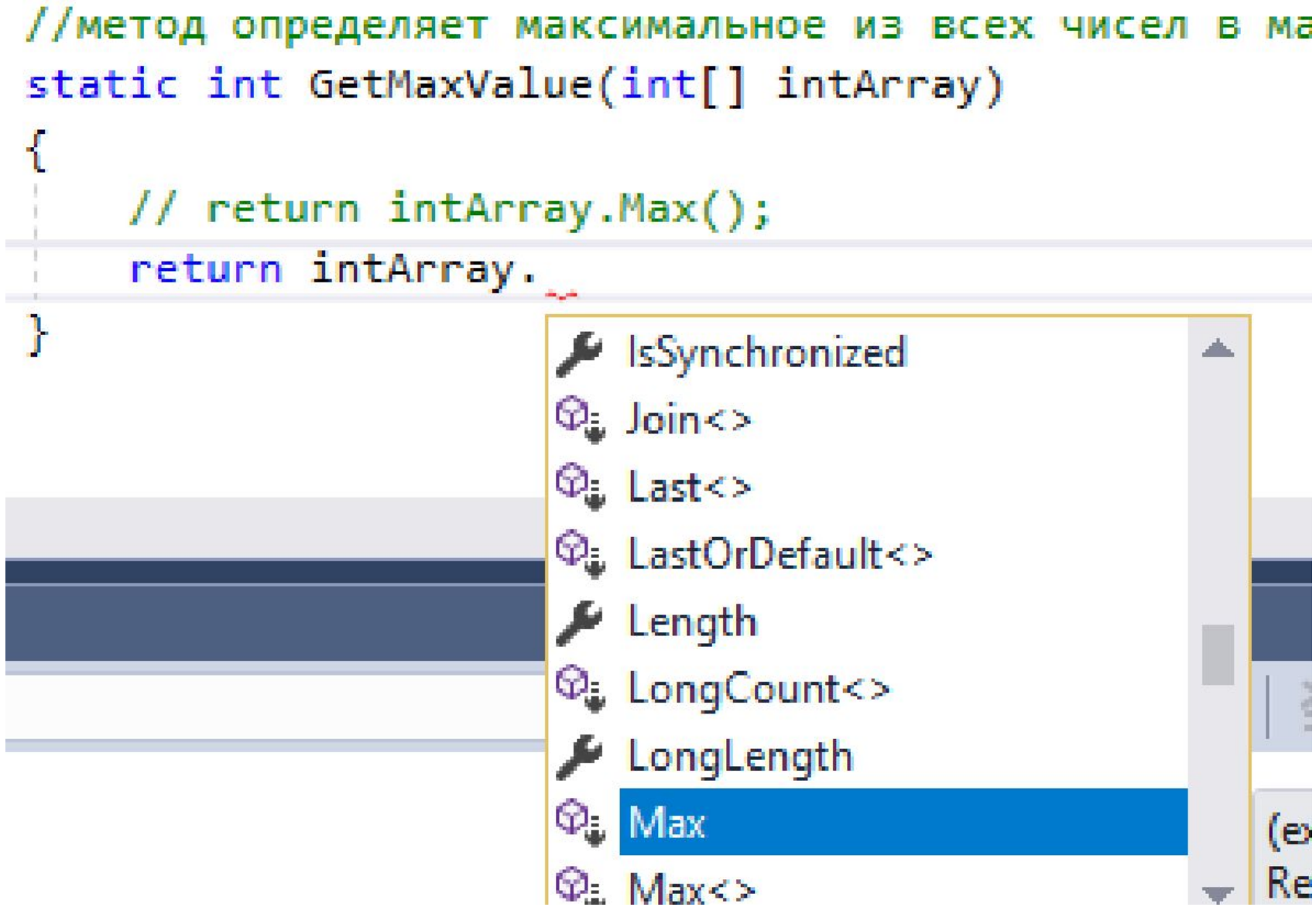
```
        AddTwoNumbers(3, 4);
        AddTwoNumbers(10,20);
        AddTwoNumbers(-2,0);
        // TypeMessage("Hello");
        TypeMessage()
    }

    AddTwoNumbers(-2,0);
    TypeMessage("Hello");
```

void Program.TypeMessage(string message)

```
class Program
  {
    static void Main(string[] args)
    {
      int methodResult = AddTwoNumbers(3, 4);
      System.Console.ReadKey();
    }
    //метод для сложения двух чисел
    static int AddTwoNumbers(int x, int y)
    {
      int result = x + y;
      System.Console.WriteLine("{0} + {1} = {2}", x, y, result);
      return result;
    }
```



```
 static void Main(string[] args)
        {

            int[] intArray = new[] { 222, -5, 0, 0, 15, 1000, -2, 211 };
            Console.WriteLine(GetMaxValue(intArray));
        }

        //метод определяет максимальное из всех чисел в массиве
        static int GetMaxValue(int[] intArray)
        {
            return intArray.Max();
        }
    }
}
```


//метод определяет максимальное из всех чисел в ма
static int GetMaxValue(int[] intArray)
{
// return intArray.Max();
return intArray.
}
IsSynchronized
Join<>
Last<>
LastOrDefault<>
Length
LongCount<>
LongLength
Max
Max<>

```
class Program
{
    static void Main(string[] args)
    {
        int a = 25;
        int b = 35;
        int result = Sum(a, b);
        Console.WriteLine(result);  // 60

        result = Sum(b, 45);
        Console.WriteLine(result);  // 80
```

```
result = Sum(a + b + 12, 18); // "a + b + 12" представляет значение параметра x
        Console.WriteLine(result);  // 90

        Console.ReadKey();
    }
    static int Sum(int x, int y)
    {
        return x + y;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Display("Tom", 24); // Name: Tom  Age: 24

        Console.ReadKey();
    }
    static void Display(string name, int age)
    {
        Console.WriteLine($"Name: {name}  Age: {age}");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        int a;
        int b = 9;
        Sum(a, b);  // Ошибка - переменной a не присвоено значение

        Console.ReadKey();
    }
    static int Sum(int x, int y)
    {
        return x + y;
    }
}
```

Конструкторы

В приведенных выше примерах программ переменные экземпляра каждого объекта типа Building приходилось инициализировать вручную, используя, в частности, следующую последовательность операторов.

```
house.Occupants = 4;
house.Area = 2500;
house.Floors = 2;
```

Такой прием обычно не применяется в профессионально написанном коде C#. Кроме того, он чреват ошибками (вы можете просто забыть инициализировать одно из полей). Нужно воспользоваться конструктором.

Конструктор инициализирует объект при его создании. У конструктора такое же имя, как и у его класса, а с точки зрения синтаксиса он подобен методу. Но у конструкторов нет возвращаемого типа, указываемого явно. Ниже приведена общая форма конструктора.

```
доступ имя_класса{список_параметров) {
  // тело конструктора
}
```

Как правило, конструктор используется для задания первоначальных значений переменных экземпляра, определенных в классе, или же для выполнения любых других установочных процедур, которые требуются для создания полностью сформированного объекта.

Кроме того, доступ обычно представляет собой модификатор доступа типа public, поскольку конструкторы зачастую вызываются в классе. А список_параметров может быть как пустым, так и состоящим из одного или более указываемых параметров.

У всех классов имеются конструкторы, независимо от того, определите вы их или нет, поскольку в C# автоматически предоставляется конструктор, используемый по умолчанию и инициализирующий все переменные экземпляра их значениями по умолчанию.

Для большинства типов данных значением по умолчанию является нулевое, для типа bool — значение false, а для ссылочных типов — пустое значение.

Но как только вы определите свой собственный конструктор, то конструктор по умолчанию больше не используется.

// Простой конструктор.

```
using System;
class MyClass {
  public int x;
  public MyClass() {
    x = 10;
  }
}
```

```
class ConsDemo {
  static void Main() {

    MyClass t1 = new MyClass(); //конструктор
MyClass() вызывается для объекта t1, присваивая
переменной его экземпляра t1.x значение 10.
    MyClass t2 = new MyClass();
    Console.WriteLine(tl.x + " " + t2.x);
  }
}
```

// Параметризированный конструктор.

```
using System;
class MyClass {
 public int x;
  public MyClass(int i) {
    x = i;
  }
}
```

```
class ParmConsDemo {
  static void Main() {
    MyClass tl = new MyClass(10);
    MyClass t2 = new MyClass(88);
    Console.WriteLine(tl.x + " " + t2.x);
  }
}
```

В данном варианте конструктора MyClass() определен параметр i, с помощью которого инициализируется переменная экземпляра x. Поэтому при выполнении следующей строки кода:

MyClass t1 = new MyClass(10);

параметру i передается значение, которое затем присваивается переменной x.

```
// Добавить конструктор в класс Building.
using System;
class Building {
  public int Floors;    //    количество этажей
  public int Area;    //    общая площадь здания
  public int Occupants;    //    количество жильцов

  // Параметризированный конструктор для класса Building,
  public Building(int f, int a, int o) {
    Floors = f;
    Area = a;
    Occupants = o;
  }
```

```
// Возвратить площадь на одного человека,
  public int AreaPerPerson() {
    return Area / Occupants;
  }

 // Возвратить максимальное количество человек, занимающих здание,
  // исходя из заданной минимальной площади на одного человека.
  public int MaxOccupant(int minArea) {
    return Area / minArea;
  }
}
```

// Использовать параметризированный конструктор класса Building,

```
class BuildingDemo {
  static void Main() {
    Building house = new Building(2, 2500, 4); //
```

конструктору Building() передаются значения 2, 2500 и 4 при создании нового объекта. Следовательно, в копиях переменных экземпляра Floors, Area и Occupants объекта house будут храниться значения 2, 2500 и 4 соответственно.

```
    Building office = new Building(3, 4200, 25);
```

Повторение:
Для преобразования текста, находящегося, например, в поле редактирования, в целое число нужно использовать функцию Convert.ToInt32(), в дробное число — Convert.ToDouble()

```
a = Convert.ToInt32(textBox1.Text);
```

Повторение:

Преобразовать численное значение в строку позволяет метод ToString().

label1.Text = "summand1 + summand2 = " + sum.ToString();

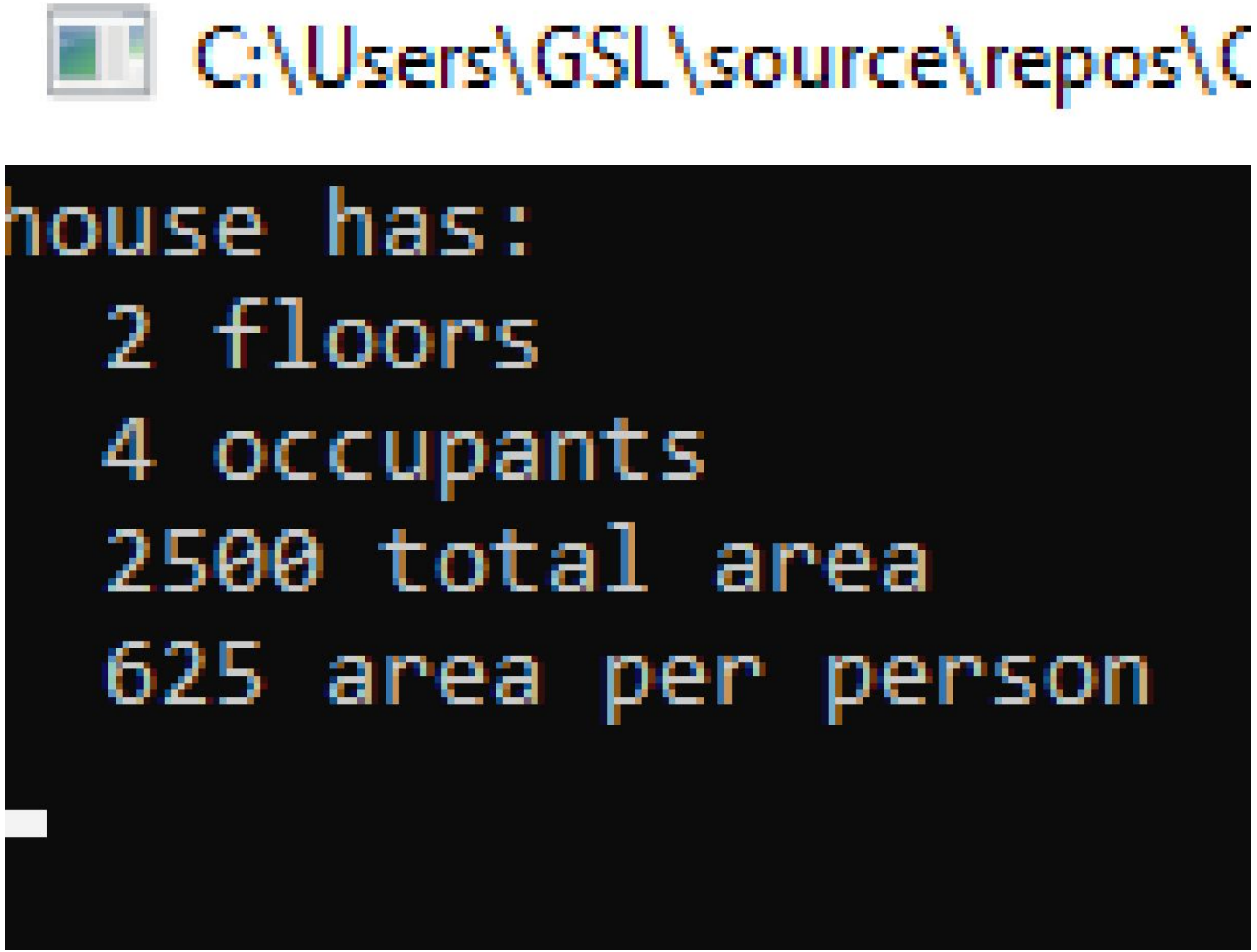
MessageBox.Show("sum="+sum.ToString());

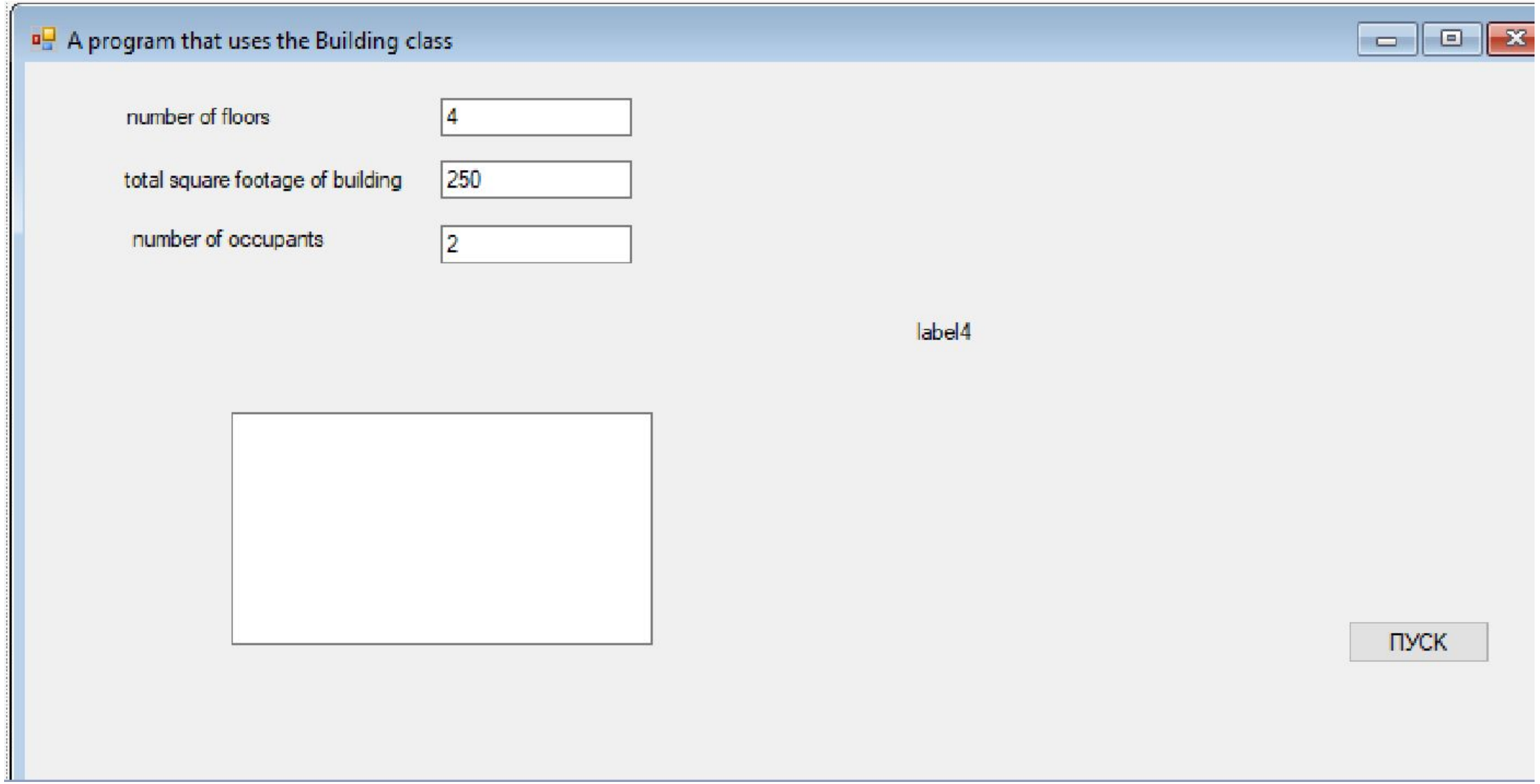
listing 1

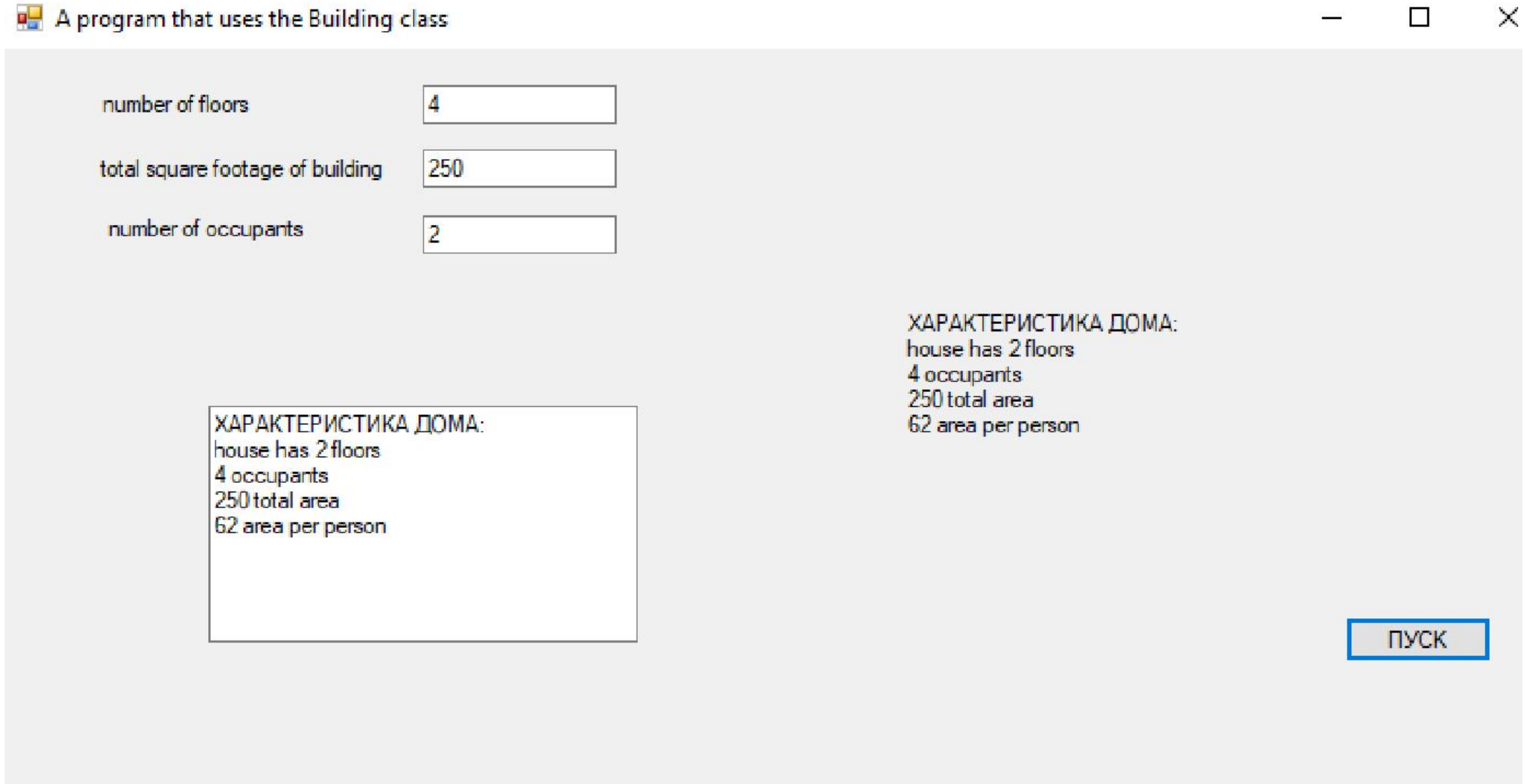
```
// A program that uses the Building class.

using System;

class Building {
  public int Floors;    // number of floors
  public int Area;      // total square footage of building
  public int Occupants; // number of occupants
}
```

```
// This class declares an object of type Building.
class BuildingDemo {
  static void Main() {
    Building house = new Building(); // create a Building object
    int areaPP; // area per person

    // Assign values to fields in house.
    house.Occupants = 4;
    house.Area = 2500;
    house.Floors = 2;
```

```
// Compute the area per person.
    areaPP = house.Area / house.Occupants;

    Console.WriteLine("house has:\n  " +
                      house.Floors + " floors\n  " +
                      house.Occupants + " occupants\n  " +
                      house.Area + " total area\n  " +
                      areaPP + " area per person");
  }
}
```


C:\Users\GSL\source\repos\(
house has:
2 floors
4 occupants
2500 total area
625 area per person


A program that uses the Building class
number of floors
4
total square footage of building
250
number of occupants
2
label4
ПУСК


A program that uses the Building class
number of floors
4
total square footage of building
250
number of occupants
2
ХАРАКТЕРИСТИКА ДОМА:
house has 2 floors
4 occupants
250 total area
62 area per person
ХАРАКТЕРИСТИКА ДОМА:
house has 2 floors
4 occupants
250 total area
62 area per person
ПУСК

```
using System.Windows.Forms;

namespace WindowsFormsApp31
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        class Building
        {
            public int Floors;    // number of floors
            public int Area;      // total square footage of building
            public int Occupants; // number of occupants
        }
```

```
 private void button1_Click(object sender, EventArgs e)
        {

            Building house = new Building(); // create a
Building object
            int areaPP; // area per person

            house.Occupants =
Convert.ToInt32(textBox2.Text);
            house.Area = Convert.ToInt32(textBox3.Text);
            house.Floors =
Convert.ToInt32(textBox4.Text);
            areaPP = house.Area / house.Occupants;
```

```
 string str;
        str= "ХАРАКТЕРИСТИКА ДОМА:" +
Environment.NewLine+ "house has "
            + house.Floors.ToString() + " floors " +
Environment.NewLine
            + house.Occupants.ToString() + " occupants " +
Environment.NewLine
            + house.Area.ToString() + " total area" +
Environment.NewLine
            + areaPP.ToString() + " area per person";

  textBox1.Text = str;
  label4.Text = str;
```

```
//Можно так:
//textBox1.Text = "ХАРАКТЕРИСТИКА ДОМА:" +
Environment.NewLine;
//textBox1.Text = textBox1.Text + "house has " +
//          house.Floors.ToString() + " floors " +
Environment.NewLine;
//// лучше так textBox1.Text+=, а не textBox1.Text =
textBox1.Text+...
//textBox1.Text += house.Occupants.ToString() + "
occupants " + Environment.NewLine;
//textBox1.Text += house.Area.ToString() + " total area" +
Environment.NewLine;
//textBox1.Text += areaPP.ToString() + " area per person";
```