Карагандинский технический университет имени Абылкаса Сагинова
Кафедра «Кибербезопасность и искусственный интеллект»

СЛАЙД-ЛЕКЦИЯ
Тема: «Введение в классы, объекты и методы»

Дисциплина: Программирование на C#

Специальность: ВТиПО

Автор: Ст.преп. Сницарь Лилия Ринатовна

ПЛАН ЛЕКЦИИ

1. Введение в классы.
2. Создание объектов.
3. Методы.
4. Конструкторы.
5. Сборка «мусора» и деструкторы.
6. Ключевое слово this.

1. Введение в классы

Класс – это шаблон, который определяет форму объекта. Он задает как данные, так и код, который оперирует этими данными. C# использует спецификацию класса для создания объекта.

Объекты – это экземпляры класса.

Таким образом, **класс** – это множество намерений (планов), определяющих, как должен быть построен объект.

Важно четко понимать следующее: класс – это логическая абстракция.

О ее реализации нет смысла говорить до тех пор, пока не создан объект класса, и в памяти не появилось физическое его представление.

Методы и переменные, составляющие класс, называются членами класса.

1. Введение в классы

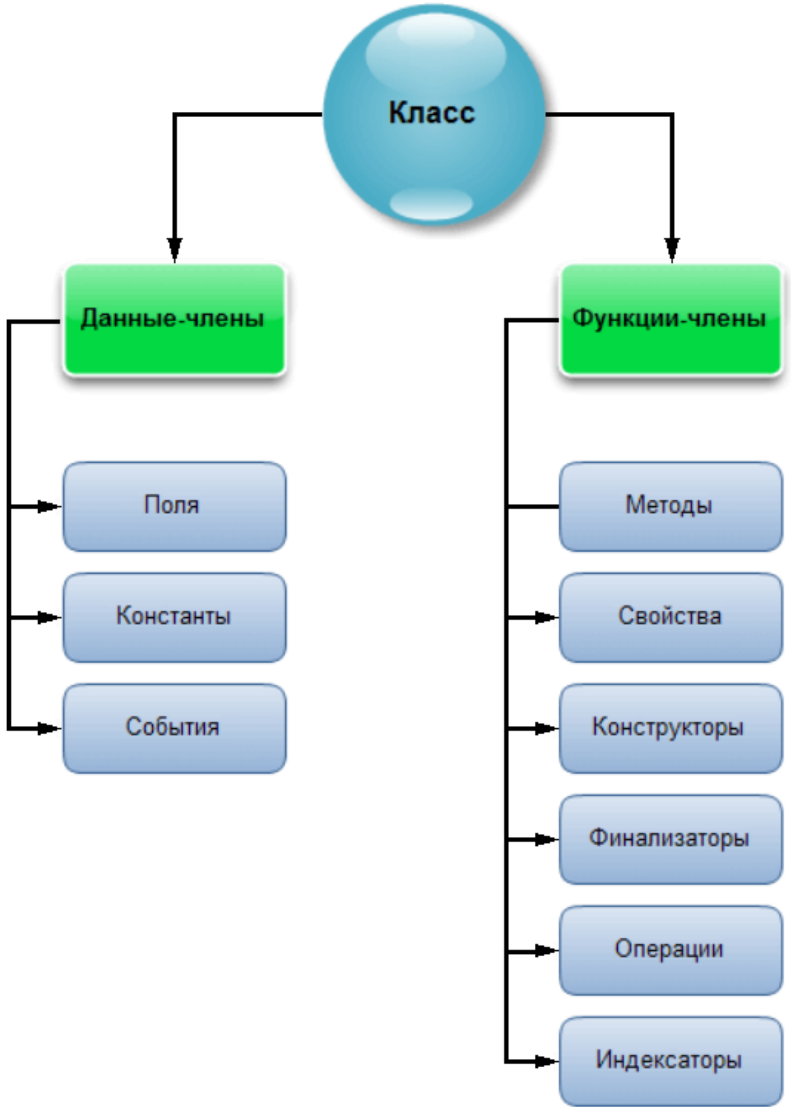
Общая форма определения класса

При определении класса объявляются данные, которые он содержит, а также код, оперирующий этими данными.

Если самые простые классы могут содержать только код или только данные, то большинство настоящих классов содержит и то и другое.

Данные содержатся в ***членах-данных (переменные экземпляров, поля)***, определяемых классом, а *код* – в ***функциях-членах (методах, члены-методы)***.

1. Введение в классы



1. Введение в классы

Данные-члены

Данные-члены – это те члены, которые содержат данные класса – поля, константы, события. Данные-члены могут быть статическими (static).

Член класса является членом экземпляра, если только он не объявлен явно как static.

Рассмотрим виды этих данных:

Поля (field) – это любые переменные, ассоциированные с классом.

```
public class MyClass
{
    public int myField; // Поле класса
}
```

1. Введение в классы

Константы могут быть ассоциированы с классом тем же способом, что и переменные. Константа объявляется с помощью ключевого слова `const`. Если она объявлена как `public`, то в этом случае становится доступной извне класса.

```
public class ConstantsClass
{
    public const int MaxValue = 100; // Публичная
                                     // константа
}
```

1. Введение в классы

События – это члены класса, позволяющие объекту уведомлять вызывающий код о том, что случилось нечто достойное упоминания, например, изменение свойства класса либо некоторое взаимодействие с пользователем. Клиент может иметь код, известный как обработчик событий, реагирующий на них.

```
public class Button
{
      public event EventHandler Click; // Объявление события
      protected virtual void On Click(EventArgs e) // Метод,
                                  //вызывающий событие ButtonPressed
    {
        Click?.Invoke(this, e);
    }
}
```

1. Введение в классы

Функции-члены

Функции-члены – это члены, которые обеспечивают некоторую функциональность для манипулирования данными класса.

Они включают методы, свойства, конструкторы, финализаторы, операции и индексаторы.

1. Введение в классы

Методы (method) – это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора static.

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

1. Введение в классы

Свойства (property) – это наборы функций, которые могут быть доступны таким же способом, как общедоступные поля класса.

В C# используется специальный синтаксис для реализации чтения (Get) и записи (Set) свойств для классов, поэтому писать собственные методы с именами, начинающимися на Set и Get, не понадобится.

```
public class Person
{
    private string _name; // поле для хранения имени
    public string Name // Свойство для доступа к имени
    {
        get { return _name; } // возвращает значение поля
        set { _name = value; } // устанавливает значение поля
    }
}
```

1. Введение в классы

Конструкторы (constructor) – это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами классов, которым они принадлежат, и они не имеют типа возврата.

Конструкторы полезны для инициализации полей класса.

```
public class Person
{
      public string Name { get; set; }
      public Person(string name) // Конструктор класса
                                 // Person
    {
        Name = name;
    }
}
```

1. Введение в классы

Финализаторы (finalizer) – вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды.

```
public class MyClass
{
	~MyClass() // Финализатор
	{
		//Освобождение ресурсов (например, закрытие файла)
	}
}
```

2. Создание объектов

В C# предусмотрен специальный класс `object`, который неявно считается базовым классом для всех остальных классов и типов, включая и типы значений.

Иными словами, все остальные типы являются производными от `object`. Это значит, что переменная ссылочного типа `object` может ссылаться на объект любого другого типа.

Кроме того, переменная типа `object` может ссылаться на любой массив, поскольку в C# массивы реализуются как объекты.

2. Создание объектов

Методы класса System.Object()

Метод	Описание
ToString()	Возвращает символьную строку, содержащую описание объекта, для которого он вызывается. Кроме того, метод ToString() автоматически вызывается при выводе содержимого объекта с помощью метода WriteLine(). Этот метод переопределяется во многих классах, что позволяет приспосабливать описание к конкретным типам объектов, создаваемых в этих классах.
GetHashCode()	Используется, когда объект помещается в структуру данных, известную как карта (map), которая также называется хеш-таблицей или словарем. Применяется классами, которые манипулируют этими структурами, чтобы определить, куда именно в структуру должен быть помещен объект. Если вы намерены использовать свой класс как ключ словаря, то должны переопределить GetHashCode(). Существуют достаточно строгие требования относительно того, как нужно реализовывать перегрузку.

2. Создание объектов

Метод	Описание
Equals()	Метод Equals(object) определяет, ссылается ли вызывающий объект на тот же самый объект, что и объект, указываемый в качестве аргумента этого метода. То есть он проверяет, являются ли оба объекта одинаковыми. Метод возвращает true, если сравниваемые объекты одинаковые, в противном случае - false.
Finalize()	Является деструктором и вызывается при очистке ресурсов, занятых объектом. По умолчанию этот метод ничего не делает. Переопределять этот метод нужно, если ваш объект владеет неуправляемыми ресурсами, которые нужно освободить при его уничтожении. Во всех остальных случаях переопределять этот метод не нужно.
GetType()	Возвращает экземпляр класса, унаследованный от System.Type. Может предоставить большой объем информации о классе, в том числе базовый тип, методы, поля и т.д.
Clone()	Создает копию объекта и возвращает ссылку на эту копию.

2. Создание объектов

```
class Program
    {
        static void Main(string[] args)
        {
            var m = Environment.Version;
            Console.WriteLine("Тип m: "+ m.GetType());
            string s = m.ToString();
            Console.WriteLine("Моя версия .NET Framework: " + s);
            Version v = (Version)m.Clone();
            Console.WriteLine("Значение переменной v: "+v);
            Console.ReadLine();
        }
    }
```

```
Тип m: System.Version
Моя версия .NET Framework: 4.0.30319.18063
Значение переменной v: 4.0.30319.18063
```

2. Создание объектов

```
static void Main(string[] args)
{
    var myOS = Environment.OSVersion;
    object[] myArr = { "Строка", 120, 0.345m, 2.34f, myOS, 'Z' };

    foreach (object obj in myArr)
        Console.WriteLine("Элемент \"{0}\" его тип - {1}", obj, obj.GetType());

    Console.ReadLine();
}
```

```
Элемент "Строка" его тип - System.String
Элемент "120" его тип - System.Int32
Элемент "0,345" его тип - System.Decimal
Элемент "2,34" его тип - System.Single
Элемент "Microsoft Windows NT 10.0.19045.0" его тип - System.OperatingSystem
Элемент "Z" его тип - System.Char
```

2. Создание объектов

```
Building house = new Building();
```

Это объявление выполняет две функции.

Во-первых, оно объявляет переменную с именем `house` классового типа `Building`. Но эта переменная не определяет объект, а может лишь ссылаться на него.

Во-вторых, рассматриваемое объявление создает реальную физическую копию объекта и присваивает переменной `house` ссылку на этот объект.

Это делается с помощью оператора `new`. Таким образом, после выполнения приведенной выше строки кода переменная `house` будет *ссылаться* на объект типа `Building`.

2. Создание объектов

Переменные ссылочного типа и присваивание

В операции присваивания переменные ссылочного типа действуют иначе, чем переменные типа значения, например типа int.

Когда одна переменная типа значения присваивается другой, ситуация оказывается довольно простой.

Переменная, находящаяся в левой части оператора присваивания, получает копию значения переменной, находящейся в правой части этого оператора.

2. Создание объектов

Когда же одна переменная ссылки на объект присваивается другой, то ситуация несколько усложняется, поскольку такое присваивание приводит к тому, что переменная, находящаяся в левой части оператора присваивания, ссылается на тот же самый объект, на который ссылается переменная, находящаяся в правой части этого оператора.

Сам же объект не копируется. В силу этого отличия присваивание переменных ссылочного типа может привести к несколько неожиданным результатам.

2. Создание объектов

```
// Создаем объект типа autoCar и присваиваем его переменной Car1
      autoCar Car1 = new autoCar();
// Присваиваем переменной Car2 ссылку на тот же объект, что и Car1
      autoCar Car2 = Car1;
//Устанавливаем значение свойства marka объекта, на который ссылается Car1
      Car1.marka = "Renault";
// Выводим значение свойства marka объекта, на который ссылается Car1, Car2
      Console.WriteLine(Car1.marka);
      Console.WriteLine(Car2.marka);
```

2. Создание объектов

Инициализаторы объектов

Инициализаторы объектов предоставляют способ создания объекта и инициализации его полей и свойств. Если используются инициализаторы объектов, то вместо обычного вызова конструктора класса указываются имена полей или свойств, инициализируемых первоначально задаваемым значением.

Следовательно, синтаксис инициализатора объекта предоставляет альтернативу явному вызову конструктора класса. Синтаксис инициализатора объекта используется главным образом при создании анонимных типов в LINQ-выражениях. Но поскольку инициализаторы объектов можно, а иногда и нужно использовать в именованном классе, то ниже представлены основные положения об инициализации объектов.

2. Создание объектов

Ниже приведена общая форма синтаксиса инициализации объектов:

```
new имя_класса {имя = выражение, имя = выражение, ...}
```

где имя обозначает имя поля или свойства, т.е. доступного члена класса, на который указывает имя_класса.

А выражение обозначает инициализирующее выражение, тип которого, конечно, должен соответствовать типу поля или свойства.

3. Методы

Методы – это процедуры (подпрограммы), которые манипулируют данными, определенными в классе, и во многих случаях обеспечивают доступ к этим данным.

Обычно различные части программы взаимодействуют с классом посредством его методов.

Любой метод содержит одну или несколько инструкций. В хорошей C#-программе один метод выполняет только одну задачу.

Каждый метод имеет имя, и именно это имя используется для его вызова. В общем случае методу можно присвоить любое имя.

Важно помнить, что имя `Main()` зарезервировано для метода, с которого начинается выполнение программы. Кроме того, в качестве имен методов нельзя использовать ключевые слова C#.

3. Методы

В C# определение метода состоит из любых модификаторов (таких как спецификация доступности), типа возвращаемого значения, за которым следует имя метода, затем список аргументов в круглых скобках и далее – тело метода в фигурных скобках:

```
доступ тип_возврата имя(список_параметров)
{
   // тело метода
}
```

3. Методы

Возврат из метода и возврат значения

Возврат из метода может произойти при двух условиях. Во-первых, когда встречается фигурная скобка, закрывающая тело метода. И во-вторых, когда выполняется оператор return.

Имеются две формы оператора **`return`**: одна – для методов типа **`void`** (*возврат из метода*), т.е. тех методов, которые не возвращают значения, а другая – для методов, возвращающих конкретные значения (*возврат значения*).

3. Методы

```
class MyMathOperation
{
    public double r;
    public string s;
    ссылка: 1
    public double sqrCircle()// Возвращает площадь круга
    {return Math.PI * r * r;}
    ссылка: 1
    public double longCircle()// Возвращает длину окружности
    {return 2 * Math.PI * r;}
    ссылка: 1
    public void writeResult()
    {
        Console.WriteLine("Вычислить площадь или длину? s/l:");
        s = Console.ReadLine();
        s = s.ToLower();
        if (s == "s")
        {
            Console.WriteLine("Площадь круга равна {0:#.###}", sqrCircle());
            return;
        }
        else if (s == "l")
        {
            Console.WriteLine("Длина окружности равна {0:#.##}", longCircle());
            return;
        }
        else
        {
            Console.WriteLine("Вы ввели не тот символ");
        }
    }
}
```

```
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        Console.WriteLine("Введите радиус: ");
        string radius = Console.ReadLine();

        MyMathOperation newOperation = new MyMathOperation { r = double.Parse(radius) };
        newOperation.writeResult();

        Console.ReadLine();
    }
}
```

```
Введите радиус:
7
Вычислить площадь или длину? s/l:
s
Площадь круга равна 153,938
```

3. Методы

Использование параметров

При вызове метода ему можно передать одно или несколько значений.

Значение, передаваемое методу, называется **аргументом**.

А переменная, получающая аргумент, называется *формальным параметром*, или просто **параметром**.

Параметры объявляются в скобках после имени метода. Синтаксис объявления параметров такой же, как и у переменных. А областью действия параметров является тело метода.

За исключением особых случаев передачи аргументов методу, параметры действуют так же, как и любые другие переменные.

4. Конструкторы

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:

Формат записи конструктора:

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

4. Конструкторы

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.

4. Конструкторы

Все классы имеют конструкторы независимо от того, определите вы их или нет, поскольку C# автоматически предоставляет конструктор по умолчанию, который инициализирует все переменные-члены, имеющие тип значений, нулями, а переменные-члены ссылочного типа – `null`-значениями.

Но если вы определите собственный конструктор, конструктор по умолчанию больше не используется.

Конструктор также может принимать один или несколько параметров. В конструктор параметры вводятся таким же образом, как и в метод. Для этого достаточно объявить их в скобках после имени конструктора.

4. Конструкторы

```
class myClass
{
    public string Name;
    public byte Age;
    // Создаем параметрический конструктор
    ссылка: 1
    public myClass(string s, byte b)
    {
        Name = s;
        Age = b;
    }
    ссылка: 1
    public void reWrite()
    {
        Console.WriteLine("Имя: {0}\nВозраст: {1}", Name, Age);
    }
}
Ссылок: 0
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        myClass ex = new myClass("Alexandr", 26);
        ex.reWrite();
    }
}
```

5. Сборка мусора и деструкторы

При использовании оператора `new` свободная память для создаваемых объектов динамически распределяется из доступной буферной области оперативной памяти.

Разумеется, оперативная память не бесконечна, и поэтому свободно доступная память рано или поздно исчерпывается.

Это может привести к неудачному выполнению оператора `new` из-за нехватки свободной памяти для создания требуемого объекта.

5. Сборка мусора и деструкторы

Система «***сборки мусора***» в C# освобождает память от лишних объектов автоматически, действуя незаметно и без всякого вмешательства со стороны программиста.

«Сборка мусора» происходит следующим образом. Если ссылки на объект отсутствуют, то такой объект считается ненужным, и занимаемая им память в итоге освобождается и накапливается.

Эта утилизированная память может быть затем распределена для других объектов.

5. Сборка мусора и деструкторы

Деструкторы

В языке C# имеется возможность определить метод, который будет вызываться непосредственно перед окончательным уничтожением объекта системой «сборки мусора».

Такой метод называется деструктором и может использоваться в ряде особых случаев, чтобы гарантировать четкое окончание срока действия объекта. Например, деструктор может быть использован для гарантированного освобождения системного ресурса, задействованного освобождаемым объектом.

Общая форма деструктора:

```
~имя_класса()
{
    // код деструктора
}
```

5. Сборка мусора и деструкторы

```
class myClass
{
    int k;
    //конструктор класса, инициализирующий поле k значением i.
    Ссылок: 2
    public myClass(int i)
    {
        k = i;
    }
    // Деструктор
    Ссылок: 0
    ~myClass()
    {
        Console.WriteLine("Объект {0} уничтожен", k);
    }
    // Метод создающий и тут же уничтожающий объект
    ссылка: 1
    public void objectGenerator(int i)
    {
        myClass ob = new myClass(i);
    }
}
Ссылок: 0
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        int i = 1;
        myClass obj = new myClass(0);
        //цикл, в котором вызывается метод objectGenerator множество раз
        for (; i < 120000; i++)
        {
            obj.objectGenerator(i);
        }
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("\nКонец");
        Console.ReadLine();
    }
}
```

```
Объект 20174 уничтожен
Объект 20173 уничтожен
Объект 20172 уничтожен
Объект 20171 уничтожен
Объект 20170 уничтожен
Объект 20169 уничтожен
Объект 20168 уничтожен
Объект 20167 уничтожен
Объект 20166 уничтожен
Объект 20165 уничтожен
Объект 20164 уничтожен
Объект 20163 уничтожен
Объект 20162 уничтожен
Объект 20161 уничтожен
Объект 20160 уничтожен
Объект 20159 уничтожен
Объект 20158 уничтожен
Объект 20157 уничтожен
Объект 20156 уничтожен
Объект 20155 уничтожен
Объект 20154 уничтожен
Объект 20153 уничтожен
Объект 20152 уничтожен
Объект 20151 уничтожен
Объект 20151 уничтожен
```

6. Ключевое слово this

В языке C# имеется ключевое слово **`this`**, которое обеспечивает доступ к текущему экземпляру класса.

Одно из возможных применений ключевого слова `this` состоит в том, чтобы разрешать неоднозначность контекста, которая может возникнуть, когда входящий параметр назван так же, как поле данных данного типа.

Разумеется, в идеале необходимо просто придерживаться соглашения об именовании, которое не может привести к такой неоднозначности.

6. Ключевое слово this

```
class MyClass
{
    public char ch;
    // 2 метода использующие входной параметр ch, при
    // этом во втором методе используется ключевое слово this
    ссылка: 1
    public void Method1(char ch)
    {
        ch = ch;
    }
    ссылка: 1
    public void Method2(char ch)
    {
        this.ch = ch;
    }
}
Ссылок: 0
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        char myCH = 'A';
        Console.WriteLine("Исходный символ {0}", myCH);

        MyClass obj = new MyClass();

        obj.Method1(myCH);
        Console.WriteLine("Использование метода без ключевого слова this: {0}", obj.ch);
        obj.Method2(myCH);
        Console.WriteLine("Использование метода с ключевым словом this: {0}", obj.ch);
        Console.ReadLine();
    }
}
```

```
Исходный символ A
Использование метода без ключевого слова this:
Использование метода с ключевым словом this: A
```

6. Ключевое слово this

Цепочка вызова конструкторов

Другое применение ключевого слова `this` состоит в проектировании класса, использующего технику под названием ***сцепление конструкторов*** или ***цепочка конструкторов*** (*constructor chaining*).

Этот шаблон проектирования полезен, когда имеется класс, определяющий несколько конструкторов. Учитывая тот факт, что конструкторы часто проверяют входящие аргументы на соблюдение различных бизнес-правил, возникает необходимость в избыточной логике проверки достоверности внутри множества конструкторов.

6. Ключевое слово this

```
class MyClass
{
    public byte Age;
    public string Name;
    Ссылок: 0
    public MyClass() { }
    // Создаем цепочку конструкторов
    Ссылок: 0
    public MyClass(string name) : this(name, 0)
    {
        Console.WriteLine("Поле Name");
    }
    ссылка: 1
    public MyClass(byte age) : this("", age)
    {
        Console.WriteLine("Поле age");
    }
    // Главный конструктор
    Ссылок: 2
    public MyClass(string name, byte age)
    {
        if (age > 25)
            age = 25;

        Name = name;
        Age = age;
    }
}
```

```
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        MyClass obj = new MyClass(26);
        obj.Name = "Alexandr";

        Console.WriteLine("Имя - {0}; Возраст - {1}", obj.Name, obj.Age);
        Console.ReadLine();
    }
}
```

```
Поле age
Имя - Alexandr; Возраст - 25
```

Контрольные вопросы

1. Что такое класс в C#?
2. Как создаётся объект класса?
3. Что такое метод класса?
4. Для чего нужен конструктор?
5. Что делает сборщик мусора?
6. Для чего используется this?

Список литературы

1. Вагнер Б. C# Эффективное программирование М.: ЛОРИ, 2013. - 320 с.

2. Ишкова Э. А. Самоучитель C#. Начало программирования М.: Наука и техника, 2013. - 496 с.

3. Подбельский В. В. Язык C#. Базовый курс М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

4. Троелсен Э. Язык программирования C# 2010 и платформа .NET4, М.: Москва: Огни, 2016. - 238 с.

ЛЕКЦИЯ ОКОНЧЕНА,
СПАСИБО ЗА ВНИМАНИЕ!