

# ПЯВУ. Лекция 13.

Элементы ООП. Рекурсия.

А.М. Задорожный

# Контрольные вопросы

1. Что такое ООП?
2. Чему соответствуют понятия **Класс** и **Объект** в функциональном программировании?
3. Где объявляются новые классы в C#?
4. Что означает слово **Public** в ООП?
5. Что такое Метод?
6. Где может располагаться в программе определение метода?
7. Что такое формальные параметры? Для чего они служат?
8. Что такое фактические параметры? Для чего они служат?

# План лекции

- Поля данных и свойства
- Рекурсия
  - Понятие рекурсии
  - Пример для задачи “Ханойские башни”
- Вопросы для повторения

# Поля данных

## Класс гистограммы.

```
public class Histogram { // Левая и правая граница
    public double LeftEdge;
    public double RightEdge;
    public int [] Data; // Массив
    public Histogram(double leftEdge, double rightEdge, int N)
    {
        ...
    }
    ...
}
```

**LeftEdge, RightEdge и Data** – поля данных

# Недостатки полей данных

Если поле данных объекта можно читать (оно доступно), то его можно и изменить.

Программно защитить или контролировать изменение поля нельзя!

*Как следствие, полям можно придать недопустимые значения, например, для гистограммы можно указать левую границу больше правой.*

# Свойства

Этих недостатков лишены Свойства.

```
public class Histogram {  
    double _leftEdge;  
    double _rightEdge;  
    int [] Data;  
  
    public double LeftEdge  
    {  
        get {return _leftEdge;}  
        set {_leftEdge = value;}  
    }  
  
    ...  
}
```

# Свойства

Используются свойства так же как и поля данных.

```
Histogram h = new Histogram();
```

```
h.LeftEdge = 0;
```

```
h.RightEdge = 100;
```

```
Console.WriteLine("{0} – {1}", h.LeftEdge, h.RightEdge);
```

Их можно задавать (присваивать) и считывать.

# Свойства

Варианты свойств. Только для чтения.

```
public class Histogram {  
    double _leftEdge;  
    double _rightEdge;  
    int [] Data;  
  
    public double LeftEdge  
    {  
        get {return _leftEdge;}  
        //set {_leftEdge = value;}  
    }  
  
    ...  
}
```



# Свойства

Варианты свойств. Проверка при присваивании.

```
public class Histogram {  
    double _leftEdge;  
    double _rightEdge;  
    int [] Data;  
  
    public double LeftEdge  
    {  
        get {return _leftEdge;}  
        set {if(value > 0) _leftEdge = value;}  
    }  
  
    ...  
}
```

В set проверяем значение value и, если оно допустимо, выполняем присваивание.

# Свойства

C# облегчает использование свойств ! Можно не задавать поля данных.

```
public class Histogram {  
    public double LeftEdge {get; set;};  
  
    public double RightEdge {get; set;};  
  
    int [] Data;  
  
    //public double LeftEdge  
    //{  
    //    get {return _leftEdge;}  
    //    set {if(value > 0) _leftEdge = value;}  
    //}  
  
    ...  
}
```

# Свойства.

## Заключительные замечания

1. Последнее упрощение позволяет требовать применения свойств вместо полей данных без исключений.
1. Мы не смогли заменить массив на свойства, но C# это позволяет!
1. Обратите внимание на элемент нотации (системы имен):  
Свойства как `public` начинаются с большой буквы, а поля, как `private` с маленькой, да еще со знаком подчеркивания.

# Рекурсия

**Рекурсия** – ссылка на себя

Вычисление факториала в цикле:

```
Double F(int N)
{
    f = 1;
    for(int i = 1; i <= N; i++)
        f *= i;
    return f;
}
```

# Рекурсия

**Рекурсия** – ссылка на себя

Вычисление факториала рекурсивно:

```
Double F(int N)
{
    if(N == 0) return 1;
    return N * F(N-1);
}
```

# Рекурсия Структура

1. Всегда имеются простейшие случаи, которые вычисляются явно!

```
if(N == 0) return 1;
```

1. Более сложные случаи вычисляются со ссылкой на более простые.

```
return N * F(N-1);
```

# Рекурсия.

## Как происходит вычисление

Алгоритм вызывает сам себя, но с более простыми значениями параметров.

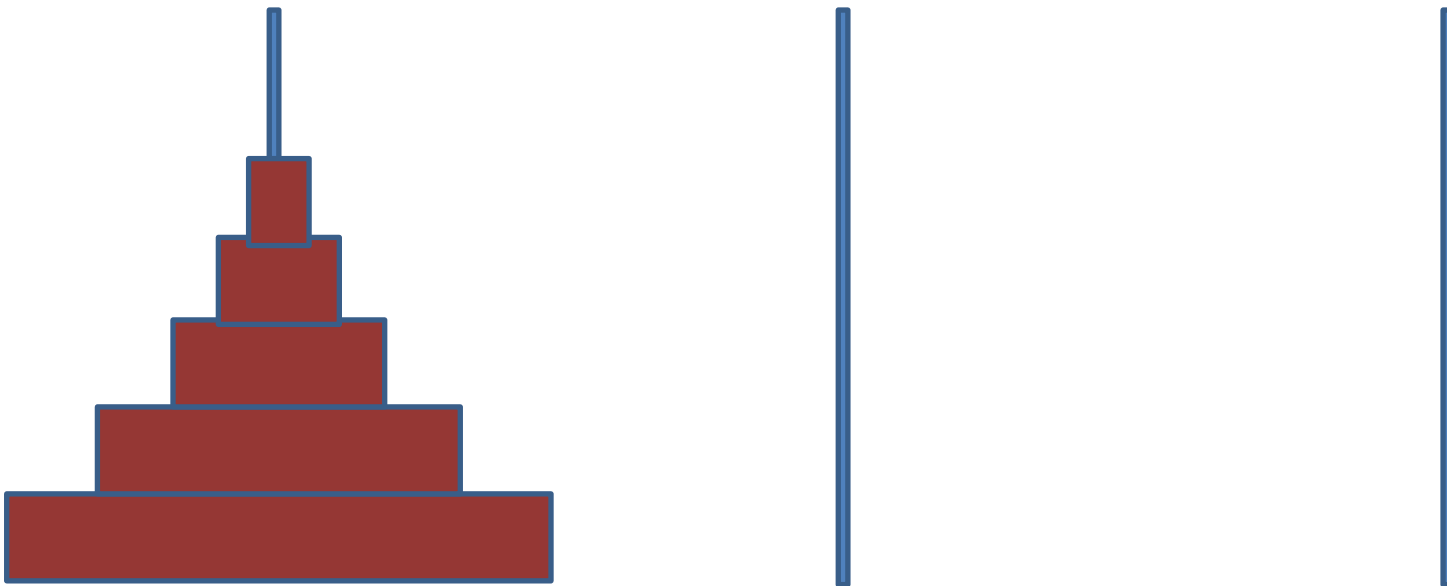
Это позволяют сделать методы! Без методов рекурсия теряет свои преимущества.

**Почему?**

Когда наступает простой случай, начинается последовательный возврат значений и он завершается решением исходной задачи.

# Рекурсия. “Ханойские башни”

Рекурсия помогает решать задачи.





# Рекурсия. “Ханойские башни”

**Рекурсия** помогает решать задачи.

**Задача.** Переместить  $N$  дисков с пирамиды  $i$  на пирамиду  $k$ .

Если  $N = 1$ , то задача тривиальна.

Допустим, мы умеем перемещать  $N-1$  дисков с любой пирамиды на любую.

# Рекурсия. “Ханойские башни”

1. Если  $N = 1$ , то переместить  $s$   $i$  на  $k$   
Иначе:
2. Переместить  $N-1$  дисков  $s$   $i$  на пирамиду  $j$
3. Переместить последний диск  $s$   $i$  на  $k$
4. Переместить  $N-1$  дисков  $s$   $j$  на пирамиду  $k$

Имеются все элементы рекурсии!

# “Ханойские башни”. Реализация.

Как представить пирамиды и диски?

```
int N = 12;           //Высота пирамид
```

```
int [,] p = new int[3, N];
```

Номера пирамид : 0, 1 и 2.

Диаметр диска задаем числом в массиве!

```
for(int i = 0; i < N; i++)
```

```
    p[0, i] = 2*(N - i) + 1;
```

```
//Что за формула  $2*(N - i) + 1$ ?
```

```
//Где низ, а где верх?
```

# “Ханойские башни”. Реализация.

```
/// <summary>  
/// Перемещает М дисков с пирамиды i на пирамиду j для системы башен p  
/// </summary>  
/// <param name="p">Ханойские башни</param>  
/// <param name="i">Исходная пирамида</param>  
/// <param name="j">Пирамида назначения</param>  
/// <param name="M">Количество дисков для перемещения</param>
```

```
void Move (int[,] p, int i, int j, int M)
```

```
{  
  
}
```

# “Ханойские башни”. Реализация.

```
void Move (int[,] p, int i, int j, int M)
{
    if(M == 1) Переместить 1 с i на j
    else
    {
        Переместить M-1 с i на k
        Переместить 1 с i на j
        Переместить M-1 с k на j
    }
}
```

1. Как найти k, если известно i и j?
2. Как узнать сколько дисков на пирамидке?

# “Ханойские башни”.

## Реализация.

```
/// <summary>
/// Вычисляет высоту пирамиды i для системы башен p
/// </summary>
/// <param name="p">Ханойские башни</param>
/// <param name="i">Пирамида для которой вычисляется высота</param>
/// <returns>Высоту пирамиды i</returns>
int Height (int[,] p, int i)
{
    int N = p.GetLength(1);
    int j = N;
    for( ; j > 0; j--)
        if(p[i, j-1] > 0) return j;
    return 0;
}
```

Находим положение первого сверху диска.

# “Ханойские башни”.

## Реализация.

```
/// <summary>
/// Перемещает 1 диск с пирамиды i на пирамиду j для системы
башен p
/// </summary>
/// <param name="p">Ханойские башни</param>
/// <param name="i">Исходная пирамида</param>
/// <param name="j">Пирамида назначения</param>
void Move1 (int[,] p, int i, int j)
{
    int hi = Height(p, i);
    int hj = Height(p, j);
    p[j, hj] = p[i, hi-1];
    p[i, hi-1] = 0;
}
```

# “Ханойские башни”. Реализация.

```
void Move (int[,] p, int i, int j, int M)
{
    if(M == 1) Move1(p, i, j);
    else
    {
        Move(p, i, 3-i-j, M-1);
        Move1(p, i, j);
        Move(p, 3-i-j, j, M-1);
    }
}
```



# Заключительные замечания

1. Рекурсия – полезный инструмент. Существенно упрощает некоторые задачи.
1. Декомпозиция – хорошая методика решения задач.
1. Нам пришлось:
  1. Продумать как будут представлены в программе пирамидки
  2. Как организовать рекурсию
  3. Как определить “высоту” пирамиды

Каждая задача оказалась простой, а в целом, мы решили довольно сложную задачу.

# Рекурсия

## Заключительные замечания

Факториал можно вычислить рекурсивно или в цикле.

Любую рекурсию можно заменить циклическим вычислением!

Алгоритм замены рекурсии циклическим вычислением может оказаться довольно сложным.

Рекурсия медленнее чем цикл! Злоупотреблять не следует!

# Контрольные вопросы и упражнения

1. Почему в ООП рекомендуется использовать свойства, а не поля данных?
1. Что такое рекурсия?
1. Реализуйте метод поиска решения уравнения на отрезке с использованием рекурсии.

# Вопросы для повторения

1. Объясните синтаксис оператора `for`
1. Какова область видимости счетчика, объявленного в `for`?
1. Что такое оператор цикла с предусловием/постусловием? Классифицируйте каждый из операторов цикла в C#.
1. Какие дополнительные операторы позволяют управлять исполнением циклов?