ПЯВУ. Лекция 12.

Элементы ООП.

А.М. Задорожный

Контрольные вопросы

1. Где можно и где нельзя использовать каждую конкретную переменную, например double x? (Где она определена, видна?)

2. Что такое гистограмма в смысле программирования?

3. Что такое **Конструктор**? В чем его особенности в C#?
4. Как **создать объект** в C#?
5. Как **вызвать метод** для конкретного объекта в C#?
6. Как **вызвать статический метод** в C#?

План лекции

1. Еще об ООП
    1. Сравнение с функциональным подходом
    2. “Эластичность” ООП

2. Класс точки. Алгоритм выяснения находится ли точка внутри полигона.

3. Статические и обычные методы
    1. Фильтры
    2. Пример класса сортировок.
    3. Сортировка выбором, вставкой и шейкер.

ООП и организация программы

Функциональный подход:

Переменная, операции применимые к типу

(int i: +, -, *, /, %, ++, --, +=, -=, …)

ООП:

Левая граница
Правая граница
Данные гистограммы

Конструктор
Hist
MeanValue

ект:

ые гистограммы

Методы работы

Гистограмма. Функциональный подход

Гистограмма в функциональном подходе

```
{
    Random r = new Random();

    int [] hist = new int [10];
    for(int i = 0; i < 1000; i++)
        Hist(10*r.NextDouble(), hist, 0, 10);
    WriteHist(hist);
    Console.WriteLine(MeanValue(0,10, hist));
}
```

Гистограмма. ООП подход

Гистограмма в ООП

```
{
    Random r = new Random();

    Histogram h = new Histogram (0, 10, 10);
    for(int i = 0; i < 1000; i++)
        h.Hist(10*r.NextDouble());
    h.Write();
    Console.WriteLine(h.MeanValue());
}
```

ООП. Эластичность I

Эластичность – простота изменения программы при изменении требований.

Изменить диапазон чисел с [0, 10], до [-5, 5] и увеличить количество каналов до 25.

```
int [] hist = new int [25];
for(int i = 0; i < 1000; i++)
    Hist(100*r.NextDouble()-10, hist, -5, 5);
WriteHist(hist);
Console.WriteLine(MeanValue(-5, 5, hist));
```

Понадобилось 5 изменений в 3-х строках кода!

ООП. Эластичность I

Эластичность – простота изменения программы при изменении требований.

Изменить диапазон чисел с [0, 10], до [-5, 5] и увеличить количество каналов до 25.

```
Histogram h = new Histogram (-5, 5, 25);
for(int i = 0; i < 1000; i++)
    h.Hist(10*r.NextDouble());
h.Write();
Console.WriteLine(h.MeanValue());
```

Понадобилось 3 изменения в 1-ой строке кода!

ООП. Эластичность II

Изменяться могут не только параметры задачи, изменяться могут требования.

Связать с каждой гистограммой заголовок, при выводе гистограммы выводить заголовок и диапазон, в котором строилась гистограмма.

ООП. Эластичность II

Добавим данные в гистограмму (заголовок)

```
public class Histogram
{
        public double LeftEdge;
        public double RightEdge;
        public int [] Data;         // Массив
        public string Title;

        public Histogram(string title, double leftEdge, double rightEdge, int N)
        {
                Title = title;
                LeftEdge = leftEdge;
                RightEdge = rightEdge;
                Data = new int[N];
        }

        public void Hist(double x){   …   }
        public double MeanValue () {  …  }
        public double Write() {  …  }
}
```

ООП. Эластичность II

Доработаем метод Write

```
public class Histogram
{
    …
    public double Write() {
        Console.WriteLine(“Гистограмма ‘{0}’”, Title);
        Console.WriteLine(“Диапазон [{0}, {1}]”,
                          LeftEdge, RightEdge);
        Console.Write(h[0]);
        for(int i = 1; i < h.Length; i++)
            Console.Write(“, {0}”, h[i]);
        Console.WriteLine();
    }
}
```

ООП. Эластичность II

Использование гистограммы:

```
Histogram h = new Histogram (“Проверка Random”, -5, 5, 25);

for(int i = 0; i < 1000; i++)
    h.Hist(10*r.NextDouble());
h.Write();
Console.WriteLine(h.MeanValue());
```

ООП. Выводы

На примере гистограммы:

1. ООП позволяет строить понятия более сложные чем заложенные в язык примитивы: числа, строки
2. Вызовы методов стали короче (легче изменять программу, если вызовов много)
3. Потенциально меньше ошибок (нельзя спутать или изменить границы, изменить массив результатов)
4. Потенциально ясно, что чем больше гистограмм, тем приведенные выше факторы будут влиять все больше.
5. Не нужно помнить, какие методы есть у класса (Visual Studio сама подскажет)
6. Легче развивать функции гистограммы. В основном меняются методы и данные класса, а в программе, которая использует этот инструмент изменений мало.

Вопросы для обсуждения

1. В чем преимущества ООП?
   a) Как изменится гистограмма, если нужно для каждой гистограммы учитывать и выводить сколько данных оказалось вне диапазона (левее или правее) гистограммы?

   b) Как ООП влияет на применение (использование) разработанной функциональности программы, например, построение гистограмм?

   c) Как ООП влияет на развитие функциональности программы, например, улучшение метода вывода гистограммы?

   d) Как ООП влияет на изучение разработанной функциональности?

Контрольные вопросы

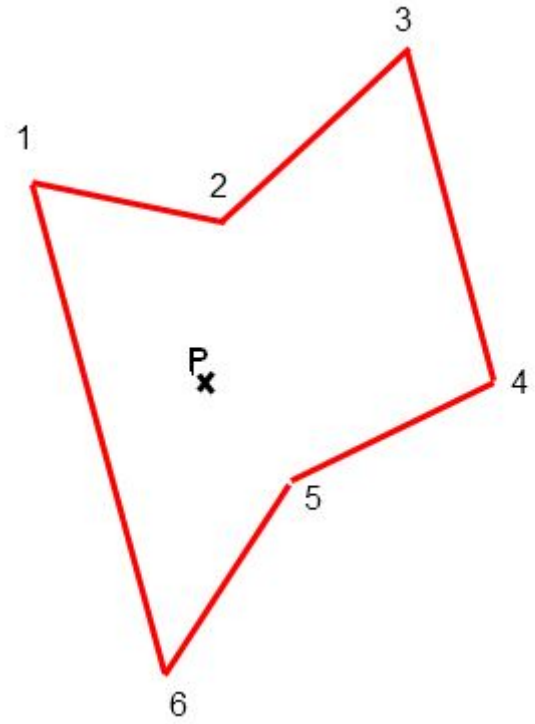
1. Где нужно объявить переменную, которая будет использоваться только в одном методе?

2. Где нужно объявить переменную, которая будет использоваться в нескольких методах класса?

3. Как следует поступить программисту, если ему нужно разработать алгоритм, который будет использовать более сложные конструкции, чем отдельные числа или строки, например, точки на плоскости или в пространстве, комплексные числа, описания студентов, описания состояния компьютерной игры?

Геометрия

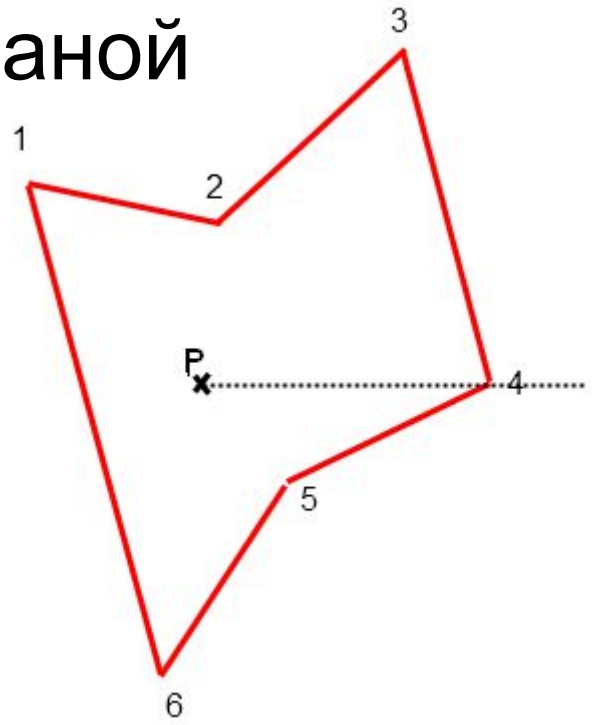
Лежит ли точка внутри?



Зачем могло бы применяться?

Геометрия

Посчитать число пересечений луча из точки Px с ломаной

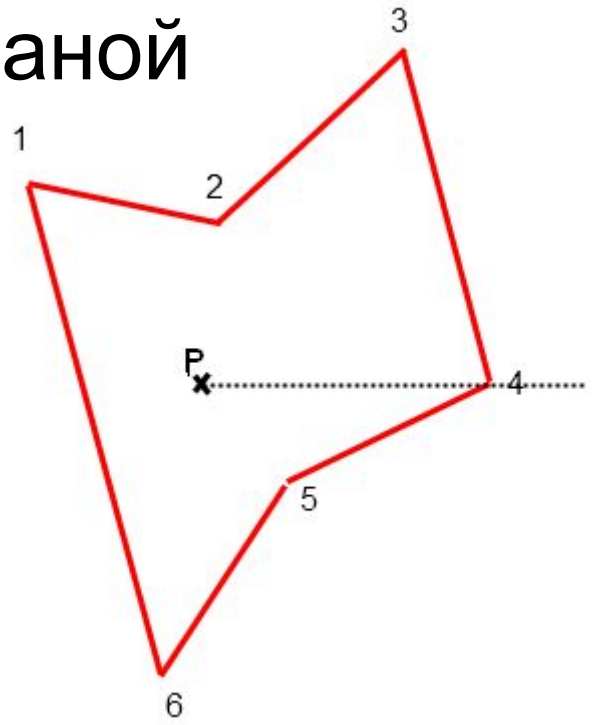


Представление полигона

```
class Point
{
    public double x;
    public double y;
}
Point [] poly; …
// Что бы получился замкнутый полигон
первая и // последняя точка должны быть
соединены
```

Геометрия

Посчитать число пересечений луча из точки Px с ломаной



Алгоритм (подсчет пересечений)

- $$x = \frac{(Bx - Ax)(y - Ay)}{(By - Ay)} + Ax$$

Для *y == Ay x == Ax*

Для *y == By x == Bx*

x – координата пересечения луча и отрезка!

Действия оптимизированы для вычисления количества пересечений!

Алгоритм (подсчет пересечений)

```
bool IsPointInsidePolygon(Point[] poly, Point p)
{  bool c = false;
   for (int i = 0, j = poly.Length - 1; i < poly.Length; j = i++)
   {  if ((((poly[i].y <= p.y) && (p.y < poly[j].y))
           || ((poly[j].y <= p.y) && (p.y < poly[i].y)))
       && (p.x < (poly[j].x - poly[i].x) * (p.y - poly[i].y) /
      (poly[j].y - poly[i].y) + poly[i].x))
           c = !c;
   }
return c;
}
```

Упражнения

1. Доказать, что при выполнении условия, луч пересекает отрезок ломаной!

2. Когда IsPointInsidePolygon не может быть выполнен?

3. Доработайте метод IsPointInsidePolygon так, что бы он безошибочно работал для всех полигонов (имея ввиду ответ на предыдущий вопрос).

ООП. Статические методы

Вызов обычного метода (метода объекта)

```
Histogram h = new Histogram (“Проверка Random”, -5, 5, 25);
…
h.Write();
```

Левая граница
Правая граница
Данные гистограммы

Конструктор
Hist
MeanValue

с объектом (h) метод Write получил и все
!

ООП. Статические методы

Вызов статического метода (метода класса)

double [] d = …. // За полнили массив данными

double [] fd = Filter. RunningAvrgFilter(d, N);

Статический метод не применяется к объекту!

V

Он не может использовать данные объекта!

Класс сортировок

```
public static class Sorter
{
	public static void BubbleSort(double [] a)
	{
		for(int j = 1; j < a.Length; j++) {
			bool sorted = true;
			for(int i = 0; i < a.Length - j; i++)
			 if(a[i]>a[i + 1]) {
				double	x = a[i];
				a[i] = a[i + 1];
				a[i + 1] = x;
				sorted = false;
			 }
			 if(sorted)  break;
		}
	}
}
```

Сортировка выбором

Иногда оказывается, что обмен переменных значениями (перестановка элементов в массива) довольно дорогая операция.

В таких случаях лучше применять “сортировку выбором”.

Пусть имеется массив длиной N. Тогда алгоритм может быть описан так:

1. Номер текущего элемента j = 0;
2. Для j от 0 до N-1
   a) Находим номер наименьшего элемента i, начиная с j;
   b) Обмениваем значениями элементы с номерами i и j;
   c) Увеличиваем j на 1 и переходим к 2.

по окончании цикла массив отсортирован в порядке возрастания.

Класс сортировок

```
public static class Sorter
{
	public static void BubbleSort(double [] a) {…}
	public static void SelectionSort(double [] a)
	{
		for(int j = 0; j < a.Length-1; j++) {
		  int m = j;
		  for(int i = j; i < a.Length; i++)
				if(a[i]<a[m])  m = i;
			if(j != m){
				double    x = a[j];
				a[j] = a[m];
				a[m] = x;
			}
		}
	}
}
```

Сортировка вставкой

Метод сортировки вставками полезен, когда нужно строить отсортированную последовательность поступающих данных.

Идея метода заключается в том, что рассматриваем отсортированную часть массива и новый (неотсортированный) элемент.

В отсортированной части массива:

- находим место для нового элемента,
- большие элементы сдвигаем вправо и
- помещаем новый элемент на нужное место.

Таким образом, отсортированная часть массива увеличивается.

Класс сортировок

```
public static class Sorter
{
    public static void BubbleSort(double [] a) {…}
    public static void SelectionSort(double [] a)
    public static void InsertionSort(double [] a)
    {
        for(int j = 1; j < a.Length; j++) {
          double x = a[j];
          int i = j;
          for(; i > 0 && x < a[i-1]; i--)
                a[i] = a[i-1];
            a[i] = x;
        }
    }
}
```

Сортировка “шейкер”

Мы улучшали сортировку пузырьком, останавливая ее, если по ходу итерации не было ни одной перестановки.

Ее можно “оптимизировать” дополнительно, имея ввиду, что если, например, в начале массива не было ни одной перестановки, то их не будет и в последующем.

Это и есть идея ShakerSort (CocktailSort). Здесь проход по массиву выполняется сначала слева-направо, потом в обратном порядке. И каждый раз запоминается положение последней перестановки.

Класс сортировок

```
public static class Sorter
{
    public static void BubbleSort(double [] a) {…}
    public static void SelectionSort(double [] a) {…}
    public static void InsertionSort(double [] a) {…}

    public static void ShakerSort(double [] a) {…}
    {



        …


    }
}
```

Класс сортировок Шейкер

```
public static void ShakerSort(double [] a)
{
      double left = 0, right = a.Length – 1, c;
      do {
                  c = 0;
                  for (int j = left; j < right; j++)
                     if (a[j] > a[j + 1]) {
                        double x = a[j]; a[j] = a[j + 1]; a[j + 1] = x;
                        c = j;
                     }
                  right = c;
                  for (int j = right; j > left; j--)
                     if (a[j - 1] > a[j]) {
            double x = a[j]; a[j] = a[j - 1]; a[j - 1] = x;
                              c = j;
                     }
                  left = c;
      } while (c != 0);
}
```

Класс сортировок
Применение

```
Double [] d = new Double[15];

Random r = new Random();
for(int i = 0; i < d.Length; i++)
    d[i] = r.Next(25);

Sorter.BubbleSort(a);
```

или

```
Sorter.SelectionSort(a);
```

или

```
Sorter. ShakerSort(a);
```

Контрольные вопросы

1. Зачем применяются статические методы?
2. Почему все методы сортировки объявлены как static?
3. Зачем может применяться сортировка “выбором”?
4. Когда применение алгоритма сортировки вставкой наиболее естественно?
5. В чем идея метода сортировки Shaker? Чем он отличается от улучшенного метода сортировки пузырьком?

Вопросы для повторения

1. Что определяет Тип данных?
2. Что такое массив?
3. Сколько операция сравнения определено для чисел в C#?
4. Сколько операций сравнения определено для строк в C#?
5. Сколько в C# определено булевских операций?
6. Какие операции можно выполнять над булевскими величинами?