Объектно-Ориентированное Программирование

Лекция 2

А.М. Задорожный

2016

Содержание

1. Эластичность ООП программ
2. Управление доступом и инкапсуляция
3. Свойства и поля данных в ООП

Вопросы для повтоения

1. Что такое ООП?
2. Что принесло применение ООП в примере с гистограммой?
3. Чему соответствуют понятия **Класс** и **Объект** в функциональном программировании?
4. Что такое **Конструктор**? В чем его особенности в C#?
5. Как **создать объект** в C#?
6. Как **вызвать метод** для конкретного объекта в C#?

ООП и организация программы

Функциональный подход:

Переменная, операции применимые к типу

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

ООП:

Левая граница
Правая граница
Данные гистограммы

Конструктор
Hist

MeanValue

…m:

…ые гистограммы

…ы работы

Использование ООП Гистограммы

Применяем гистограмму

```
{
    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

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

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

```
public class Histogram
{
        public double LeftEdge 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

Доработаем метод Write

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

ООП. Эластичность

Использование гистограммы:

```
Histogram h = new Histogram (“Вес ТС на дорогах”, 0.5, 10, 20);

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. **Классы объединяют в одном понятии данные и методы**. Вызовы методов стали короче (легче изменять программу, если вызовов много)

3. **Проще изучать**. Не нужно помнить, какие методы есть у класса (Visual Studio сама подскажет)

4. **Эластичность**. Легче развивать функции гистограммы. В основном меняются методы и данные класса, а в программе, которая использует этот инструмент изменений мало.

5. Потенциально меньше ошибок (нельзя спутать или изменить границы, изменить массив результатов). Потенциально ясно, что чем больше гистограмм, тем приведенные выше факторы будут влиять все больше.

Вопросы для обсуждения

Преимущества ООП.

a) Как изменится класс гистограмм, если нужно для каждой гистограммы учитывать и выводить сколько данных оказалось вне диапазона (левее или правее) гистограммы?
b) Что пришлось бы сделать без ООП?

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

d) Как ООП влияет на изучение разработанной функциональности?

Вопросы для обсуждения

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, матрицы, описания студентов, описания состояния компьютерной игры, …?

ООП и управление доступом

Классы позволяют управлять доступом к данным и методам.

```
class Histogramm
{
    public double LeftEdge;
    public double RightEdge;
    public int[] Data; // Массив
    public string Title;
    public Histogramm(string title, double leftEdge, double rightEdge, int N) …
    public void Hist(double x) …
    public double MeanValue() …
    public void Write() …
}
```

ООП и управление доступом

Именно слово **public** позволяет обращаться к этим данным и методам вне методов класса.

```
{
    Histogram h = new Histogram (“Вес ТС на дорогах”, 0.5, 10, 20);


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

ООП и управление доступом

Кроме слова **public** можно указывать **private** – доступно только для частного использования.

В объекте Histogramm недопустимо изменять массив **Data**, **левую** и **правую границы** в процессе использования. Их можно безболезненно изменить на **private**.

ООП и управление доступом

Кроме слова **public** можно указывать **private** – доступно только для частного использования.

В объекте Histogramm недопустимо изменять массив **Data**, **левую** и **правую границы** в процессе использования. Их можно безболезненно изменить на **private**.

```
class Histogramm
 {
        private double LeftEdge;
        private double RightEdge;
        private int[] Data; // Массив
        private string Title;
…
}
```

ООП и управление доступом

Кроме слов **public** и **private** имеются и другие (**protected**, **protected internal** и **internal**). *Не рассматриваются.*

Если не указывать никакого квалификатора, то по умолчанию это эквивалентно **internal** (почти **private**). Это заставляет программиста уделять больше внимания тому, какой доступ разрешить к члену класса. Если программист забыл указать, то будет **internal** и доступ будет ограничен.

В некоторых случаях и такого регулирования оказывается недостаточным. Как, например, сделать, что бы открыть величины для чтения, но закрыть их для изменения?

ООП и управление доступом

Определение!

Возможность в ООП:

- объединять в одном понятии данные и методы работы с ними
- а так же возможность управлять доступом к данным и методам

называется **Инкапсуляцией**!

Контрольные вопросы

1. Что означает термин “Управление доступом” применительно к ООП?
2. Как осуществляется управление доступом к членам класса (данным и методам)?
3. Что означает квалификатор **public**?
4. Что означает квалификатор **private**?
5. Какой доступ будет разрешен, если не указывать никакого квалификатора доступа?
6. Что означает понятие “инкапсуляция”?

ООП и свойства

Свойства – это “обертки” для данных объекта, которые позволяют контролировать доступ к этим данным.

Превратить данные в свойства в C# легко. Поэтому рекомендуется никогда не использовать поля данных, а использовать ТОЛЬКО свойства.

```
class Histogram
{
    public double LeftEdge {get; set;}
    public double RightEdge {get; set;}
    private int[] Data; // Массив
    public string Title {get; set;}
…
}
```

ООП и свойства

Слова **get** и **set** указывают на способы доступа к свойству. Например, если не указать **set**, то это свойство нельзя будет изменить!

Если перед **set** указать квалификатор **private**, то изменять свойство можно будет только из методов класса!

```
class Histogram
 {
      public double LeftEdge {get; private set;}
      public double RightEdge {get; private set;}
      private int[] Data; // Массив
      public string Title {get; private set;}
…
}
```

ООП и свойства

Массив **Data** остался **private** и не превращен в свойство. *Это не случайно, свойства с индексами объявляются несколько иначе и мы не будем нагромождать такие детали в рамках курса.*

```
class Histogram
 {
      public double LeftEdge {get; private set;}
      public double RightEdge {get; private set;}
      private int[] Data; // Массив
      public string Title {get; private set;}
…
}
```

Но при этом, мы скрываем от пользователя класса важную информацию – количество каналов гистограммы.

ООП и свойства

Скрывая массив **Data** мы скрываем от пользователя класса важную информацию – количество каналов гистограммы.

Свойства позволяют решить и эту проблему. Для этого определим **Вычисляемое свойство**.

Вычисляемое – значит за ним нет непосредственно поля данных, а его значение получается по некоторой формуле.

```
class Histogram
 {
      public double LeftEdge {get; private set;}
      public double RightEdge {get; private set;}
      private int[] Data; // Массив
      public int NChannels { get  {
                                  return Data.Length;
                            }
                  }
      public string Title {get; private set;}
…
}
```

Контрольные вопросы

1. Что такое Свойства в ООП?
2. Зачем они введены?
3. Как поле данных превратить в Свойство в C#?
4. Как определить свойство, которое нельзя изменять?
5. Как определить свойство, которое нельзя изменять вне методов класса?
6. Как создать вычисляемое свойство – свойство, которое не соответствует какому-нибудь полю данных?

Контрольные вопросы

Что означают следующие термины:

1. Класс, объект?
2. Конструктор?
3. Поле данных?
4. Метод?
5. Квалификатор доступа?
6. Свойство?
7. Вычисляемое свойство?
8. Эластичность кода программы?
9. Инкапсуляция?