Основы программирования

Состав класса

Свойства

Свойства

- Специальные методы доступа. Как правило, имя свойства совпадает с именем поля, но составляется по нотации Паскаля.
- Предоставляют простой механизм чтения, записи или вычисления значения закрытого поля класса или структуры.
- Свойство ничего не хранит, оно выступает в роли посредника между внешним кодом и переменной.
- Содержат специальные методы доступа к полям объекта – методы `get` и `set`, предназначенные для различных проверок
- Хотя, по своему назначению свойство является методом доступа к полю, но оно может и не соответствовать полю, а имитировать его наличие в объекте.

Синтаксис свойства

```
[атрибуты] [модификаторы] тип_возврата Имя_свойства
{
    [атрибуты] [модификаторы] get
    {
        код доступа для чтения поля;
    }
     [атрибуты] [модификаторы] set
    {
        код доступа для записи поля;
    }
}
```

...

- В блоке `get` мы возвращаем значение поля, а в блоке `set` устанавливаем.
- Атрибуты и модификаторы свойства соответствуют атрибутам и модификаторам методов.
- Чаще всего используется модификатор `public`, поскольку свойства представляют собой внешнее представление объекта, открывая доступ к закрытым полям.
- «`тип_возврата`» – это тот тип данных, к которому принадлежит то поле, к которому открывает доступ свойство.

Пример

```
class Person  {
    private string name = "";

     public string Name  {
        get    {
            return name;    // возвращаем значение   }
        set    {
            name = value;   // устанавливаем новое   }
     }
}

Person person = new Person();
person.Name = "Иван";
Console.WriteLine person.Name);  /
```

Методы доступа (блоки) `get` и `set`

- Может отсутствовать один из блоков `get` или `set`, но не оба сразу.
- Если отсутствует блок `get`, то свойство доступно только для записи (`write-only`).
- Если отсутствует блок `set`, то свойство доступно только для чтения(`read-only`).
- Когда присутствуют оба блока, то свойство предоставляет доступ и на чтение, и на запись.

Модификаторы блоков `get` и `set`

- Если модификаторы и атрибуты у блоков отсутствуют, то на них распространяется модификаторы свойства.
- Модификатор для блока `set` или `get` можно установить, если свойство имеет оба блока (и `set`, и `get`).
- Только один блок `set` или `get` может иметь модификатор доступа, но не оба сразу.

Модификаторы блоков `get` и `set`

- Модификатор доступа блока `set` или `get` должен быть более ограничивающим, чем модификатор доступа свойства.
  Например, если свойство имеет модификатор `public`, то блок `set/get` может иметь только модификаторы `protected`, `internal`, `protected`, `private`.

Блок `set`

- В блоке `set` свойства используется системная переменная `value`, тип которой совпадает с типом, заявленным в заголовке свойства.
- Переменная `value` используется для задания значения контролируемого свойства.

Пример

```
class Person     {
  private string name;
  public string Name       {
    get    {     return name;     }
    set    {     name = value;    }
  }
}
```

Использование:

```
Person p = new Person();
p.Name = "Иван";
string personName = p.Name;
```

Пример

Свойства позволяют реализовать дополнительную логику, которая может быть необходима, например, при присвоении значения переменной, например, проверку по возрасту:

```
set   {
      if (value < 18)
            Console.WriteLine("Возраст должен быть
> 17");
  else
            age = value;
}
```

Пример 1-1

```
class Rectangle
{
    double height, width, square; //закрытые поля
    public Rectangle()
    {   height = 1; width = 1; square = 1; }
    public Rectangle(double h, double w)
    {   height = h; width = w; }

    public double Height {
    get         { return height;    }
    set         { if (value > 0)    {
      height = value;
      square = height * width;    }
} }
```

Пример 1-2

```
public double Width   {
   get  { return width;       }
   set  { if (value > 0) {
        width = value;
        square = height * width;   }     }
}

public double Square  {
   get  { return height * width;  }
}

}
```

Пример 1-3

```
…

Rectangle r1=new Rectangle();
r1.Height=2;

Console.WriteLine(”Height = {0}, width = {1},
square = {2}”, r1.Height, r1.Width, r1.Square);

r1.Width = 3;
Console.WriteLine(”Height = {0}, width = {1},
square = {2}”, r1.Height, r1.Width, r1.Square);
}
…
```

Пояснение к примеру

- Свойство `Square` предназначено только для чтения поля `square`.
- Значение площади не может устанавливаться независимо от значений высоты и ширины прямоугольника. То есть его необходимо пересчитывать при каждом изменении ширины или высоты.
- ***Важно:*** данное значение является зависимым, следовательно, явное хранение его в объекте нецелесообразно.

Пример 2-1

```
class Rectangle {
  double height; width; // закрытые поля
  public Rectangle()  {
    height = 1; width = 1;}
  public Rectangle(double h, double w)    {
    height = h; width = w; }

  public double Height {
  get  { return height;      }
  set  { if (value > 0)      {
    height = value; }        }
  }
```

Пример 2-2

```
  public double Width {
     get  { return width;     }
     set  { if (value > 0)    {
        width = value  }      }
  }

  public double Square      {
     get  { return height * width;      }
  }
}
```

Пример 2-3

```
…
Rectangle r1=new Rectangle(5, 6);
Console.WriteLine(”Height = {0}, width = {1},
square = {2}”, r1.Height, r1.Width,
r1.Square);
…
```

Поля `square` в классе не существует, но внешнее представление объекта включает в себя свойство только для чтения `Square`, имитируя наличие поля.

Автоматические свойства

Если имеется десяток и более полей, то определять каждое поле и писать для него однотипное свойство очень утомительно.

Фреймворк .NET позволяет использовать автоматические свойства.

Они имеют сокращенное объявление:

Автоматические свойства

```
class Person    {

  public string Name { get; set; }
  public int Age { get; set; }

  public Person(string name, int age) {
    Name = name;   Age = age;
  }

}
```

Автоматические свойства

- В случае автоматических свойств также создаются поля для свойств, только не программистом, а компилятором, который автоматически их генерирует при компиляции.
- Преимущество автосвойств, в том, что при необходимости мы можем развернуть автосвойство в обычное свойство, добавив в него определенную логику.
- Нельзя создать автоматическое свойство только для записи, как в случае со стандартными свойствами.

Автоматические свойства

Автосвойствам можно присвоить значения по умолчанию (инициализация автосвойств):

```
class Person     {
  public string Name { get; set; } = "Иван";
  public int Age { get; set; } = 23;
}
```

В структурах инициализацию автосвойств использовать нельзя.

Модификаторы автоматических свойств

Автосвойства также могут иметь модификаторы доступа:

```
class Person   {
  public string Name { private set; get;}

  public Person(string n)    {
    Name = n;
  }
}
```

Автоматические свойства

- Можно убрать блок `set` и сделать автосвойство доступным только для чтения.
- В этом случае для хранения значения этого свойства для него неявно будет создаваться поле с модификатором `readonly`, поэтому следует учитывать, что подобные `get-свойства` можно установить либо из конструктора класса, как в примере выше, либо при инициализации свойства.

Автоматические свойства

```
class Person
{
    public string Name { get;} = "Tom"
}
```

Сокращенная запись свойств

```
class Person  {
private string name;

//public string Name { get {return name;}}
//эквивалентно

public string Name => name;
}
```