Utilizando as propriedades get / set

Propriedades combinam os aspectos de métodos e campos. Para o utilizador de um objeto, uma propriedade parece ser um campo, acessando a propriedade requer a mesma sintaxe. Para o implementador de uma classe, uma propriedade é um ou dois blocos de código, representando um obter acessador e/ou um Definir acessador. O bloco de código para o **get** o acessador é executado quando a propriedade é leitura; o bloco de código para o **set** o acessador é executado quando a propriedade é atribuída um novo valor. Uma propriedade sem um **set** o acessador é considerado somente leitura. Uma propriedade sem um **get** o acessador é considerado somente para gravação. Uma propriedade que possui os acessadores é leitura-gravação.

Ao contrário dos campos, propriedades não são classificadas como variáveis. Portanto, você não pode passar uma propriedade como um ref (referência de C#) ou check-out (referência de C#) parâmetro.

Propriedades têm muitos usos: eles podem validar dados antes de permitir que uma alteração; modo transparente, eles podem expor dados em uma classe onde esses dados são recuperados, na verdade, a partir de outra fonte, como, por exemplo, um banco de dados; eles podem executar uma ação quando dados são alterados, como disparar um evento ou alterar o valor de outros campos.

Propriedades são declaradas no bloco de classe, especificando o nível de acesso do campo, seguidas do tipo da propriedade, seguidas do nome da propriedade e seguidas de um bloco de código que declara um **get**-assessor e/ou um **set** acessador. Por exemplo:

C#

```
public class Date
{
    private int month = 7;  // Backing store

    public int Month
    {
        get
        {
            return month;
        }
        set
        {
            if ((value > 0) && (value < 13))
            {
                month = value;
            }
        }
    }
}
```

Neste exemplo, `Month` é declarada como uma propriedade assim que o **set** acessador pode Certifique-se de que o `Month` valor é definido entre 1 e 12. O `Month` propriedade utiliza um campo privado para acompanhar o valor real. O local real dos dados de uma propriedade é conhecido como "fazendo o armazenamento" a definição da propriedade. É comum para as propriedades usar campos particulares, como um armazenamento de backup. O campo será marcado como particular para certificar-se de que ele só pode ser alterado chamando a propriedade. Para obter mais informações sobre restrições de acesso público e particular, consulte Modificadores de acesso (guia de programação do C#).

Propriedades de auto-implementado fornecem sintaxe simplificada para declarações de propriedade simples. Para obter mais informações, consulte Auto-implementado propriedades (guia de programação do C#).

O acessador get

O corpo da **get** acessador semelhante ao de um método. Ele deve retornar um valor do tipo de propriedade. A execução da **get** o acessador é equivalente ao ler o valor do campo. Por exemplo, quando você retornar a variável private da **get** acessador e otimizações forem ativadas, a chamada para o **get** método do acessador estiver embutido pelo compilador, portanto não há nenhuma sobrecarga de chamada de método. No entanto, um virtual **get** método do acessador não pode ser embutido porque o compilador não sabe em qual método, na verdade, pode ser chamado em tempo de execução de tempo de compilação. A seguir está uma **get** acessador retorna o valor de um campo particular `name`:

C#

```
class Person
{
    private string name;  // the name field
    public string Name    // the Name property
    {
        get
        {
            return name;
        }
    }
}
```

Quando você faz referência a propriedade, exceto quando o destino de uma atribuição, o **get** o acessador é invocado para ler o valor da propriedade. Por exemplo:

C#

```
Person person = new Person();
//...

System.Console.Write(person.Name);  // the get accessor is invoked here
```

O **get** acessador deve terminar em um retornar ou lança instrução e controle não podem fluir fora do corpo do acessador.

É um estilo de programação ruim para alterar o estado do objeto usando o **get** acessador. Por exemplo, o acessador seguir produz o efeito colateral de alterar o estado do objeto sempre que o `number` campo é acessado.

C#

```
private int number;
public int Number
{
    get
    {
        return number++;   // Don't do this
    }
}
```

O **get** acessador pode ser usado para retornar o valor do campo ou para computá-lo e devolvê-lo. Por exemplo:

C#

```
class Employee
{
    private string name;
    public string Name
    {
        get
        {
            return name != null ? name : "NA";
        }
    }
}
```

No segmento de código anterior, se você não atribuir um valor para o `Name` propriedade, ela retornará o valor NA.

O conjunto de acessador

O **set** o acessador é semelhante a um método cujo tipo de retorno é void. Ele usa um parâmetro implícito chamado *value*, cujo tipo é o tipo da propriedade. No exemplo a seguir, um **set** acessador é adicionado para o `Name` propriedade:

C#

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

Quando você atribui um valor à propriedade, o **set** o acessador é chamado usando-se um argumento que fornece o novo valor. Por exemplo:

C#

```
Person person = new Person();
person.Name = "Joe";  // the set accessor is invoked here

System.Console.Write(person.Name);  // the get accessor is invoked here
```

É um erro para usar o nome do parâmetro implícito, *value*, para uma declaração de variável local em um**set** acessador.

Comentários

Properties can be marked as **public**, **private**, **protected**, **internal**, or **protected internal**. Esses modificadores de acesso definem como os usuários da classe podem acessar a propriedade. O **get** e **set**acessadores para a mesma propriedade podem ter modificadores de acesso diferentes. Por exemplo, o**get** pode ser **public** para permitir o acesso de leitura e de fora o tipo e o **set** pode ser **private** ou**protected**. Para obter mais informações, consulte Modificadores de acesso (guia de programação C#).

Uma propriedade pode ser declarada como uma propriedade estática usando o **static** palavra-chave.Isso torna a propriedade disponível para chamadores a qualquer momento, mesmo que exista nenhuma ocorrência da classe. Para obter mais informações, consulte Classes estáticas e membros de classe estáticos (guia de programação C#).

Uma propriedade pode ser marcada como uma propriedade virtual usando o virtual palavra-chave. Isso permite que as classes derivadas substituir o comportamento de propriedade usando o Substituirpalavra-chave. Para obter mais informações sobre essas opções, consulte Herança (guia de programação do C#).

Uma propriedade de substituição de uma propriedade virtual também pode ser lacrado, especificando que para classes derivadas não é mais virtual. Por fim, uma propriedade pode ser declarada abstrata.Isso significa que não há nenhuma implementação na classe e classes derivadas devem escrever sua própria implementação. Para obter mais informações sobre essas opções, consulte Classes abstratas e seladas e membros de classe (guia de programação do C#).

Observação

É errado usar uma virtual (C# Reference), Resumo (referência de C#), ou Substituir (referência de C#)modificador em um acessador de um estático propriedade.

Exemplo

Este exemplo demonstra as propriedades de instância, estática e somente leitura. Ele aceita o nome do funcionário do teclado, incrementos `NumberOfEmployees` por 1 e exibe o funcionário de nome e número.

C#

```
public class Employee
{
    public static int NumberOfEmployees;
    private static int counter;
    private string name;

    // A read-write instance property:
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    // A read-only static property:
    public static int Counter
    {
        get { return counter; }
    }

    // A Constructor:
    public Employee()
    {
        // Calculate the employee's number:
```

```
        counter = ++counter + NumberOfEmployees;
    }
}

class TestEmployee
{
    static void Main()
    {
        Employee.NumberOfEmployees = 107;
        Employee e1 = new Employee();
        e1.Name = "Claude Vige";

        System.Console.WriteLine("Employee number: {0}", Employee.Counter);
        System.Console.WriteLine("Employee name: {0}", e1.Name);
    }
}
/* Output:
    Employee number: 108
    Employee name: Claude Vige
*/
```

Este exemplo demonstra como acessar uma propriedade na classe base que está ocultos por outra propriedade que tem o mesmo nome em uma classe derivada.

C#

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

public class Manager : Employee
{
    private string name;

    // Notice the use of the new modifier:
    public new string Name
    {
        get { return name; }
        set { name = value + ", Manager"; }
    }
}

class TestHiding
{
    static void Main()
    {
        Manager m1 = new Manager();

        // Derived class property.
        m1.Name = "John";
```

```
        // Base class property.
        ((Employee)m1).Name = "Mary";

        System.Console.WriteLine("Name in the derived class is: {0}", m1.Name);
        System.Console.WriteLine("Name in the base class is: {0}",
((Employee)m1).Name);
    }
}
/* Output:
    Name in the derived class is: John, Manager
    Name in the base class is: Mary
*/
```

Estes são os pontos importantes no exemplo anterior:

- A propriedade `Name` na classe derivada oculta a propriedade `Name` na classe base. Nesse caso, o**new** modificador é usado na declaração da propriedade na classe derivada:

  C#

  ```
  public new string Name
  ```

- A projeção `(Employee)` é usado para acessar a propriedade oculta na classe base:

  C#

  ```
  ((Employee)m1).Name = "Mary";
  ```

  Para obter mais informações sobre como ocultar membros, consulte a novo modificador (referência de C#).

Neste exemplo, duas classes, `Cube` e `Square`, implementar uma classe abstrata, `Shape`e substituir seu resumo `Area` propriedade. Observe o uso da Substituir modificador nas propriedades. O programa aceita o lado como entrada e calcula as áreas para o quadrado e cubo. Ele também aceita a área como entrada e calcula o lado correspondente para o quadrado e cubo.

C#

```
abstract class Shape
{
    public abstract double Area
    {
        get;
        set;
    }
}

class Square : Shape
{
    public double side;

    public Square(double s)  //constructor
    {
        side = s;
```

```
    }

    public override double Area
    {
        get
        {
            return side * side;
        }
        set
        {
            side = System.Math.Sqrt(value);
        }
    }
}

class Cube : Shape
{
    public double side;

    public Cube(double s)
    {
        side = s;
    }

    public override double Area
    {
        get
        {
            return 6 * side * side;
        }
        set
        {
            side = System.Math.Sqrt(value / 6);
        }
    }
}

class TestShapes
{
    static void Main()
    {
        // Input the side:
        System.Console.Write("Enter the side: ");
        double side = double.Parse(System.Console.ReadLine());

        // Compute the areas:
        Square s = new Square(side);
        Cube c = new Cube(side);

        // Display the results:
        System.Console.WriteLine("Area of the square = {0:F2}", s.Area);
        System.Console.WriteLine("Area of the cube = {0:F2}", c.Area);
        System.Console.WriteLine();

        // Input the area:
        System.Console.Write("Enter the area: ");
        double area = double.Parse(System.Console.ReadLine());

        // Compute the sides:
```

```
        s.Area = area;
        c.Area = area;

        // Display the results:
        System.Console.WriteLine("Side of the square = {0:F2}", s.side);
        System.Console.WriteLine("Side of the cube = {0:F2}", c.side);
    }
}
/* Example Output:
    Enter the side: 4
    Area of the square = 16.00
    Area of the cube = 96.00

    Enter the area: 24
    Side of the square = 4.90
    Side of the cube = 2.00
*/
```

C# 3.0 – Propriedades Automáticas

Uma das novidades na versão 3.0 do Visual C#. É o recurso Automatic properties(Propriedades Automáticas).
Nas versões anteriores ao Visual C# 3.0, se precisássemos criar propriedades dentro de uma classe C#, era necessário declarar o CAMPO para armazenar o valor e dois métodos dentro da propriedade, o GET e o SET.
Veja um exemplo usando propriedades codificadas em C# 2.0:
using System;

```
namespace Escola
{
    public class Aluno
    {
        private string nome;
        private int idade;
        private decimal media;

        public string Nome
        {
            get { return nome; }
            set { nome = value; }
        }

        public int Idade
        {
            get { return idade; }
            set { idade = value; }
        }
        public decimal media
        {
            get { return media; }
            set { media = value; }
        }
    }
}
```

Agora veja o mesmo exemplo fazendo uso do novo recurso oferecido pelo Visual C# 3.0.

```
namespace Escola
{
    public class Aluno
    {
        public string Nome { get; set; }
        public int Idade { get; set; }
        public decimal media { get; set; }
    }
}
```

Leia mais: http://sosprogramadores.blogspot.com/2010/10/c-30-propriedades-automaticas.html#ixzz2KvhZOiIW