IFRN

Tecnologia em Análise e Desenvolvimento de Sistemas

Propriedades e Indexadores

Prof. Gilbert Azevedo

Conteúdo

- Propriedades
  - Definição e utilização de propriedades
  - Propriedades somente-leitura e somente-escrita
  - Propriedades em interfaces
  - Implementação explícita de interfaces
- Indexadores
  - Definição e utilização de indexadores
  - Sobrecarga de indexadores
  - Indexadores em interfaces

Propriedades

- As propriedades definem campos lógicos em uma estrutura, classe ou interface
  - Possibilita uma forma mais natural de acesso aos dados da estrutura ou classe, excluindo a necessidade de criação dos tradicionais métodos acessadores *get* e *set*
  - Estabelece um *link* para um campo físico ou um método e possui o funcionamento análogo a um campo público
- [AccessModifier] Type PropertyName {
- get { // Leitura do valor da propriedade }
- set { // Escrita do valor da propriedade }
- }

Exemplo de Propriedades

```
class Triangulo {
    private double b;
    private double h;
    public double Base {
        get { return b; }
        set { if (value > 0) b = value; else b = 0; }
    }
    public double Altura {
        get { return h;}
        set { if (value > 0) h = value; else h = 0;    }
    }
}
```

Acesso às Propriedades

- Instanciando o objeto
  - Triangulo t = new Triangulo();
- Escrevendo um valor na propriedade (set)
  - t.Base = 10;
  - t.Altura = 20;
- Lendo o valor da propriedade (get)
  - Console.WriteLine(t.Base);
  - Console.WriteLine(t.Altura);

Propriedades Somente-Leitura

- As propriedades são somente leitura quando apenas o bloco *get* está definido
  - class ScreenPosition
  - {
  - private int x;
  - public int X { get { return x; } }
  - }

  - ScreenPosition origem = new ScreenPosition();
  - origem.X = 10; // Erro
  - Console.WriteLine(origem.X); // OK

Propriedades Somente-Escrita

- As propriedades são somente escrita quando apenas o bloco *set* está definido
  - class ScreenPosition
  - {
  - private int x;
  - public int X { set
  - { if (value >= 0 and value <= 1280) x=value; } }
  - }
  - ScreenPosition origem = new ScreenPosition();
  - origem.X = 10; // OK
  - Console.WriteLine(origem.X); // Erro

Restrições das Propriedades

- Propriedades não armazenam valores na memória (não são atributos)
- Não podem ser usadas como argumentos *ref* ou *out* *(*não são alocadas na memória)
  - Metodo(*ref* origem.X);
- Contêm no máximo um *get* e um *set*
- Os *gets* e *sets* não podem receber parâmetros
  - Os dados atribuídos ao *set* são acessados pelo identificador *value*

Propriedades em Interfaces

- É possível declarar propriedades em uma interface.
  - Neste caso, *gets* e *sets* ficam vazios.
  - interface IScreenPosition
  - {
  - int X { get; set; }
  - int Y { get; set; }
  - }
- As estruturas e classes que implementam tais interfaces devem obrigatoriamente definir os *gets* e *sets* declarados.

Propriedades em Estruturas

- Implementando as interfaces em uma estrutura

```
struct ScreenPosition : IScreenPosition {
    private int x;
    private int y;
    public int X {
        get { return x; }    set { x = value; }
    }
    public int Y {
        get { return y; }    set { y = value; }
    }
}
```

Propriedades Virtuais

- Numa classe, as propriedades podem ser virtuais, permitindo a redefinição nas implementações

```
class ScreenPosition : IScreenPosition {
    private int x; private int y;
    public virtual int X {
        get { return x; }    set { x = value; }
    }
    public virtual int Y {
        get { return y; }    set { y = value; }
    }
}
```

Implementação Explícita

- Implementação explícita: propriedades e métodos são públicos e não virtuais e acessados por referências da interface
  - class ScreenPosition : IScreenPosition {
  - private int x; private int y;
  - int IScreenPosition.X {
  - get { return x; } set { x = value; }
  - }
  - int IScreenPosition.Y {
  - get { return y; } set { y = value; }
  - }
  - }

Indexadores

- Os indexadores definem propriedades que utilizam a lógica de *array* em uma estrutura, classe ou interface
  - Possibilita uma forma mais natural de acesso aos dados da estrutura ou classe, excluindo a necessidade de criação dos tradicionais métodos *get* e *set* com um parâmetro de indexação
  - Estabelece um *link* para um campo físico, definindo um funcionamento semelhante ao de um *array*
- O indexador usa ***this*** na sua definição e o operador **[]** para recuperar os dados associados a ele

Classe com Indexador

```
class Agenda {
    private string[] nomes = new string[100];
    private string[] fones = new string[100];
    private int qtd;

    public void Adicionar(string nome, string fone)
    {
        nomes[qtd] = nome;
        fones[qtd] = fone;
        qtd++;
    }
```

Definição e Acesso ao Indexador

- Propriedade: retorna o campo qtd
  - public int Qtd {
  - get { return qtd; }
  - }
- Indexador: retorna o elemento *index* do campo nomes
  - public string ***this***[int index] {
  - get { return nomes[index]; }
  - }
- Acesso ao indexador
  - Agenda agenda = new Agenda();
  - for (int i = 0; i < agenda.Qtd; i++)
  - Console.WriteLine(agenda**[**i**]**);

Sobrecarga de Indexador

- Indexador de Leitura e Escrita: retorna o fone do *index* nome
  - public string ***this***[string nome] {
  - get {
  - int index = Array.IndexOf(nomes, nome);
  - if (index != -1) return fones[index];
  - else return "";
  - }
  - set {
  - int index = Array.IndexOf(nomes, nome);
  - if (index != -1) fones[index] = value;
  - }
  - }
- Modificando o fone do *index* Contato
  - agenda["Contato"] = "1234-5678";

Indexadores e Arrays

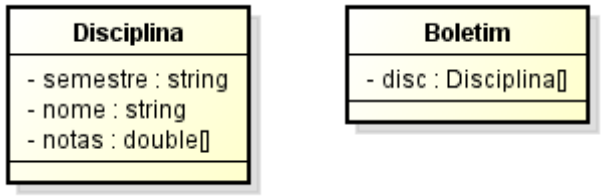
- Os indexadores podem utilizar subscritos não numéricos, enquanto os arrays só podem utilizar subscritos inteiros
  - public int this [string s] { ... }
- Os indexadores podem ser sobrecarregados
  - public string this [int index] { ... }
  - Public string this [string s] { ... }
- Os indexadores não podem ser utilizados como parâmetros *ref* e *out*.

Indexadores em Interfaces

- Para declarar um indexador numa interface, o corpo dos *gets* e *sets* são substituídos por um ponto e vírgula
  - Interface IRawInt {
  - bool this [int index] {get; set;}
  - }
- Toda classe ou estrutura que implementar a interface deve implementar os métodos de acesso
- Os métodos implementados podem ser virtuais
  - class RawInt : IRawInt {
  - public virtual bool this [int index] { ... }
  - }

Exercício

- 1. Implementar uma classe Disciplina para representar a realização de uma disciplina do curso, contendo informações sobre semestre cursado, nome da disciplina e notas (1º bimestre, 2º bimestre e final). Todos os atributos da classe devem ser lidos e escritos com propriedades. Não permitir a entrada da nota da prova final, caso o aluno tem obtido média 6 nas notas dos bimestres. Incluir propriedades para retornar a média final e a situação (aprovado ou reprovado).
- 2. Implementar uma classe Boletim para representar o boletim de um aluno permitindo o registro de até 100 disciplinas. Incluir métodos para inserir e remover disciplinas no boletim, propriedade para retornar o número de disciplinas cadastradas e indexadores para acessar a listagem total de disciplinas cadastradas e a listagem de disciplinas de um semestre específico.



Referencias Bibliográficas

- Introduction to C# Programming with Microsoft .Net
  - Microsoft Official Course 2609A
- Microsoft Visual C# 2005 - Passo a passo
  - John Sharp, Bookman, 2007
- Microsoft Asp.Net – Passo a passo
  - George Sheperd, Bookman, 2007
- Microsoft VS 2005 Express Edition Documentation
- UML – Uma Abordagem Prática
  - Gilleanes T. A. Guedes, Novatec, 2004