Aula 2: Orientação a Objetos com C#

Professor(a): Virgínia Fernandes Mota
virginia@teiacoltec.org

TECNOLOGIAS DE PROGRAMAÇÃO- SETOR DE INFORMÁTICA

Orientação a Objetos com C#

- Ao término desta aula, você será capaz de entender:
  - Encapsulamento e Modificadores de Acesso
  - Construtores
  - Herança
  - Polimorfismo
  - Arrays
  - Tipos implícitos e a palavra VAR
  - Classes Abstratas e Interfaces
  - Métodos e atributos estáticos

Encapsulamento e Modificadores de Acesso

- Modificadores de acesso: public e private!
- Como fazer getters e setters em C#?

```
1  class Conta{
2      public Cliente titular;
3      public int numero;
4      public double saldo; //quero ser capaz de ler o valor mas
           s   a classe pode ter acesso de escrita
5  }
```

- Properties

```
1  class Conta{
2      public Cliente titular;
3      public int numero;
4      public double Saldo {get; private set}; //letra mai scula
           por conven   o
5  }
```

Encapsulamento e Modificadores de Acesso

```
1 Conta c1 = new Conta();
2 c1.Saca(100); //ok
3 c1.Saldo -= 100.0; //n o compila
4
5 MessageBox.Show("O saldo    " + c1.Saldo); //compila
```

Encapsulamento e Modificadores de Acesso

```
1 class Conta{
2     public Cliente Titular {get; set}; //N o est  verdadeiramente
          encapsulado
3     public int Numero {get; set}; //N o est  verdadeiramente
          encapsulado
4     public double Saldo {get; private set};
5 }
```

- Encapsulando atributos de maneiras diferentes.

Encapsulamento e Modificadores de Acesso

- Mas o que está implementado no get e no set? E se eu quiser alterá-los?

```
class Cliente{
    public string Cpf { //Cpf agora    uma propriedade!
        get{
            return cpf;
        }
        set{
            this.cpf = value;
        }
    }
}
```

Encapsulamento e Modificadores de Acesso

- Além de marcar um método, propriedade ou atributo de uma classe como público, podemos marcar a própria classe como pública.
- Quando marcamos a classe como pública, ela se torna visível em todos os pontos da aplicação.

Construtores

```
1 Cliente cliente = new Cliente();
2 cliente.Nome = "Han Solo";
```

- Vamos criar um novo construtor, um que receba nome como parâmetro?

```
1 public class Cliente{
2     (atributos)
3     public Cliente(string nome){
4         this.Nome = nome;
5     }
6 }
```

- Diferente do Java, o construtor vazio (padrão) deixa de existir! Logo apenas instanciações new Cliente("nome") serão compiladas.
- Podem ser criados diversos construtores com parâmetros diferentes.

Herança

- Como funciona a Herança em C#?
- Herança Simples!
- Vamos supor que desejamos criar a classe ContaPoupança que é filha da classe Conta.

```
1  class ContaPoupanca : Conta { // :    equivalente ao extends
       do Java
2    ...
3  }
```

Herança

- Para sobrescrever os métodos

```
1  class ContaPoupanca : Conta { // :    equivalente ao extends
       do Java
2   ...
3      public override void Saca(double valor){ //para indicar
           que    sobrescrita do m todo da classe m  e
4      ...
5      }
6  }
7
8  class Conta{
9  ...
10     public virtual void Saca(double valor){ //para indicar que
           o m todo pode ser sobrescrito
11     ...
12     }
```

- Mas lembrem-se que Saldo era private quando definimos a classe Conta. O que fazer?
- Basta usar o modificador de acesso protected.

Herança

- Resumo:
  - protected para que atributos private sejam acessados pelos filhos
  - : define que a classe estende de outra classe
  - para sobrescrevermos métodos precisamos usar override na sobrescrita e virtual na classe mãe.

Polimorfismo

- O polimorfismo funciona como no Java.
- Se quiséssemos criar uma classe com um método que recebe como parâmetro Conta, ele seria capaz de trabalhar com Conta e todas as duas filhas.
- Mas como isso funciona?
- Exemplo: O C# sabe que ContaPoupanca herda todos os atributos e métodos de Conta, e portanto tem a certeza que existe o atributo Saldo, e que ele poderá invocá-lo sem maiores problemas!
- Para utilizar métodos da superclasse utilizamos base (o equivalente ao super do Java).

Quatro Pilares da Orientação a Objetos

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

Trabalhando com Arrays

- Arrays em C# são muito parecidos com os Arrays em Java.

```
1     //Arrays de tipos b sicos
2     int[] numeros = new int[10];
3     numeros[0] = 42;
4     numeros[4] = 21;
5
6     //Arrays de Objetos
7     Conta[] contas = new Conta[2];  //contas.Length = 2
8     contas[0] = new Conta();
9     contas[1] = new ContaPoupanca();
10
11    //for each
12    foreach(Conta c in contas){
13        Console.WriteLine("O saldo da conta    : " + c.Saldo +
              "\n");
14    }
```

Tipos implícitos e a palavra VAR

- Em C# é possível "adivinhar"o tipo da variável.

```
1       int idade;
2       bool EMaiorDeIdade = (idade >= 18);
3
4       //declara  o impl cita ou tipagem impl cita
5       int idade;
6       var EMaiorDeIdade =  (idade >= 18); //
7
8       //tamb m funciona para objetos;
9       var c1 = new Conta();
```

- Usando var é obrigatório definir um valor inicial a ela logo na declaração para que o compilador consiga inferir qual o tipo que será utilizado.

Classes Abstratas e Interfaces

- Quando trabalhamos com Herança em Java vimos dois outros conceitos: Classes Abstratas e Interfaces.
- Como eles são trabalhados em C#?

Classes Abstratas

- Se no nosso sistema só precisamos instanciar ContaPoupança e ContaCorrente, sabemos que não é necessário uma Conta genérica.
- Para impedir, basta tornar a classe conta Abstrata. Assim ela não poderá ser instanciada.

```
1    abstract class Conta{
2        ...
3    }
```

Métodos Abstratos

- Para criarmos métodos abstratos e as classes filhas devem obrigatoriamente sobrescrever (override) o método.
- Para isso, basta trocar na classe mãe o método virtual por abstract.

```
1     abstract class Conta{
2         ...
3         //era public virtual void Saca (double valor) {...}
4         public abstract void Saca (double valor);
5     }
```

- Método abstratos só podem existir em Classes Abstratas.

Interfaces

- Uma nova regra de negócio para criarmos "contratos"entre classes.
- Vamos supor que temos agora ContaInvestimento e ContaPoupanca que devem calcular um tributo e ContaCorrente não precisa.
- Queremos criar um relatório (Classe TotalizadorDeTributos) que recebe todos os tipos de conta que tem cálculo de tributo.
- Já sabemos que a solução para esse problema e que evita repetição de código é a criação de uma Interface.

Interfaces

- Uma interface em C# é definida por interface e é implementada seguindo a linha da Herança.

```
1       interface ITributavel{
2           public double CalculaTributos();
3       }
4
5       class ContaPoupanca : Conta, ITributavel{ // equivalente a
            extends Conta implements Tributavel
6           ...
7       }
```

- Comportamentos em comum entre classes: Lembrando que Interfaces não tem atributos, nem properties e seus métodos não tem implementação.

Métodos e atributos estáticos

- Métodos e atributos estáticos: Métodos e atributos que são da Classe!
- No nosso exemplo, vamos contar o número de contas do sistema.

```
1     //para contar o n mero de Contas Correntes
2     class ContaCorrente : Conta{
3         public static int TotalDeContas = 0;
4
5         public ContaCorrente(){
6             ContaCorrente.TotalDeContas++;
7         }
8
9         //m todo est tico para dizer o pr ximo n mero de
              conta
10        public static int ProximoNumero(){
11            return ContaCorrente.TotalDeContas + 1;
12        }
13    }
```

Para pesquisarmos

http://msdn.microsoft.com/en-us/library/

Na próxima aula...

Tópicos Avançados