IFRN

Tecnologia em Análise e Desenvolvimento de Sistemas

Herança, Classes Abstratas, Interfaces

Prof. Gilbert Azevedo

Conteúdo

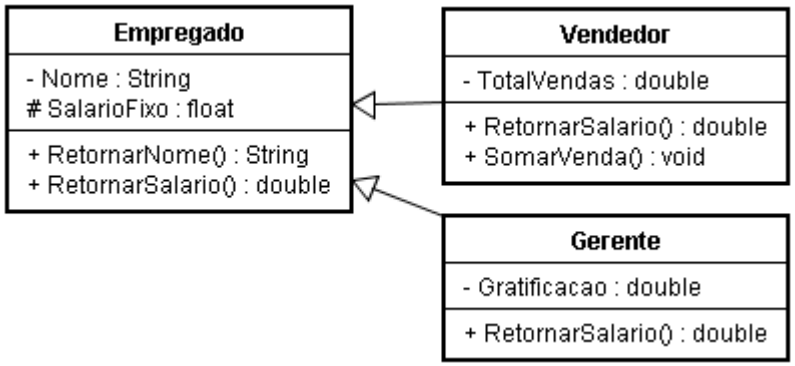
- Conceito de Herança em POO
- Herança e UML
- Encapsulamento e Polimorfismo
- Herança em C#
  - Construtores, Referências, Métodos Virtuais
- Interfaces
- Classes Abstratas

Herança em POO

- Herança é um conceito chave usado na orientada ao objeto para descrever uma relação entre classes
- A herança evita a reescrita de código e especifica um relacionamento de especialização/generalização
- Através da herança uma classe copia ou herda todas as propriedades, atributos e métodos de uma outra classe, podendo estender sua funcionalidade
- A classe que cede os membros para a outra é chamada super-classe, classe base ou classe ancestral
- A classe que herda os membros de uma outra é chamada sub-classe ou classe derivada

Herança e UML

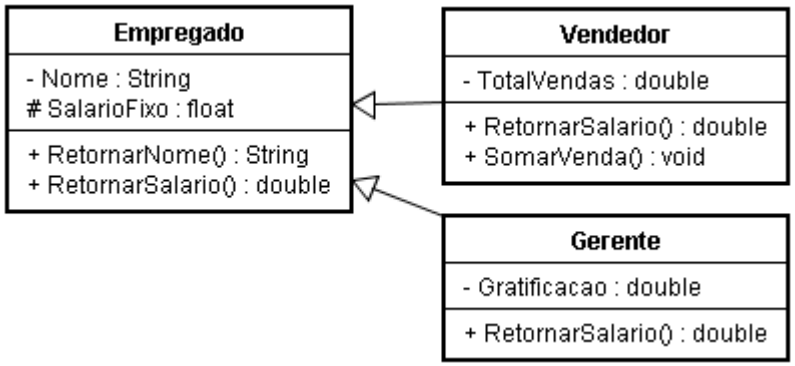
- A UML (Unified Modeling Language) – Linguagem de Modelagem Unificada – é uma linguagem padrão baseada em diagramas utilizada para modelar sistema orientados a objetos



- Em um Diagrama de Classes, a herança é indicada por uma seta da classe derivada para a classe ancestral

Encapsulamento e UML

- O encapsulamento em um diagrama de classes é representado pelos símbolos: - # e +
  - O símbolo – indica os membros privados
  - O símbolo # indica os membros protegidos
  - O símbolo + indica os membros públicos

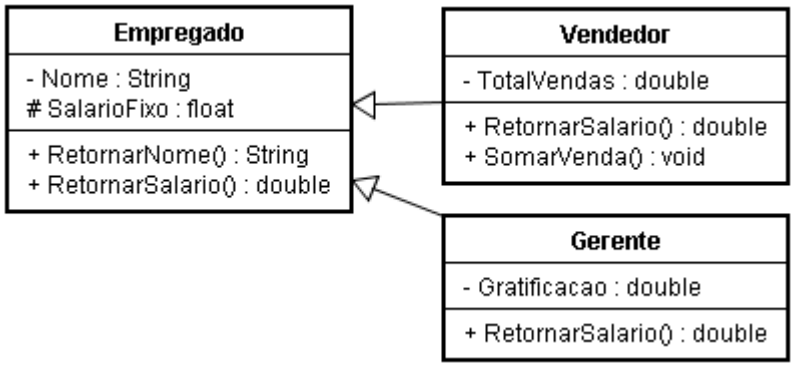


Modificadores de Acesso no C#

- Modificador private
  - O membro da classe (propriedade, atributo ou método) é visível apenas dentro da classe
- Modificador protected
  - O membro é visível na classe e nas classes descendentes
- Modificador public
  - O membro é visível fora da classe
- Modificador internal
  - O membro é público apenas no assembly (projeto)

Polimorfismo

- Polimorfismo é o conceito da POO que ocorre quando objetos de classes distintas em uma hierarquia realizam uma mesma operação (mesmo método) de forma diferente
  - O método RetornarSalario nas classes Empregado, Vendedor e Gerente é um exemplo de polimorfismo



Herança em C#

- A sintaxe para declarar uma classe que herda de outra em C# é:
  - class DerivedClass : BaseClass {
  - ...
  - }
- Sub-classes podem ser normalmente classes base para outras heranças:
  - class DerivedSubClass : DerivedClass {
  - ...
  - }
- Em C#, as classes herdam apenas de uma classe base
- Uma classe *sealed* (selada) não pode ser herdada

System.Object

- A classe System.Object é a classe ancestral de qualquer classe em C#, mesmo que não declarada
  - class Empregado { ...
  - }
  - class Empregado : System.Object { ...
  - }
- Alguns métodos herdados de System.Object
  - Equals – Testa se dois objetos são iguais
  - GetHashCode – Retorna o código de hash para o objeto
  - GetType – Retorna informação sobre a classe do objeto
  - ToString – Retorna o objeto como string

Herança e Construtores

- Os construtores da classe ancestral podem (e devem) ser chamados na classe derivada com a instrução ***base***
  - class Empregado {
  - private string nome;
  - protected double salarioFixo;
  - public Empregado(string nome, double salarioFixo) {
  - this.nome = nome;
  - this.salarioFixo = salarioFixo; }
  - public string RetornarNome() { return nome; }
  - public virtual double RetornarSalario() {
  - return salarioFixo; }
  - }

Herança e Construtores

- Classe Gerente
  - class Gerente : Empregado {
  - private double gratificacao;
  - public Gerente(string nome, double salarioFixo, double
  - gratificacao) : ***base***(nome, salarioFixo) {
  - this.gratificacao = gratificacao; }
  - public override double RetornarSalario() {
  - return salarioFixo + gratificacao;
  - }
  - }

Métodos Virtuais

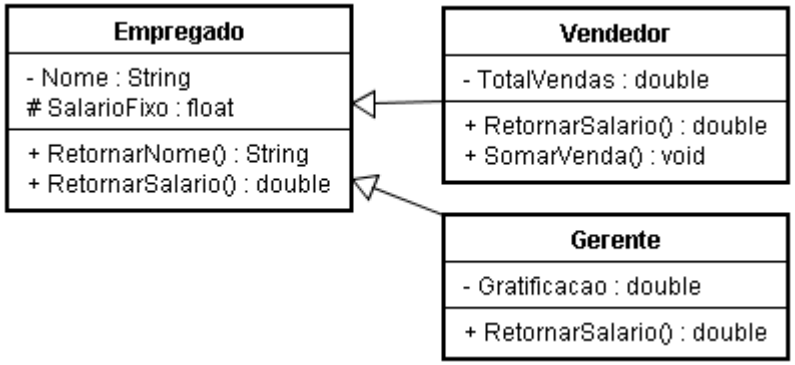
- No C#, métodos Virtuais são usados para implementar o polimorfismo. A classe que define o método usa a palavra *virtual.* As classes derivadas usam *override.*
  - Empregado
    - public *virtual* double GetSalario() {
    - return salarioFixo;}
  - Vendedor
    - public *override* double GetSalario() {
    - return salarioFixo + 0.05 * totalVendas;}
  - Gerente
    - public *override* double GetSalario() {
    - return salarioFixo + gratificacao; }

Herança e Referências

- É possível referenciar objetos em uma hierarquia com referências de classes ancestrais:
  - Object x = new Gerente("Pedro", 1000, 500);
  - Empregado y = new Gerente("Maria", 2000, 800);
  - Gerente z = new Gerente("Paulo", 3000, 900);
  - Console.WriteLine((x as Gerente).RetornarSalario());
  - Console.WriteLine(y.RetornarSalario());
  - Console.WriteLine(z.RetornarSalario());
- Apenas os membros definidos na classe da referência são acessíveis por ela

Operadores is

- O operador *is* é utilizado para verificar se um objeto é de uma determinada classe ou descendente desta classe, retornando verdadeiro ou falso
  - Vendedor v = new Vendedor();
  - if (v is object) ... // Verdadeiro
  - if (v is Empregado) ... // Verdadeiro
  - if (v is Vendedor) ... // Verdadeiro
  - if (v is Gerente) ... // Falso



Operadores as

- O operador as é utilizado para alterar o tipo da referência de um objeto, retornado *null* quando não for possível.
  - Object v = new Vendedor();
  - (v as Vendedor).RetornarSalario(); // Ok
  - (v as Gerente).RetornarSalario(); // Null
- Type-Casting
  - É a operação de conversão do tipo da referência
  - (v as Vendedor).RetornarSalario();
  - ((Vendedor) v).RetornarSalario();

Interfaces

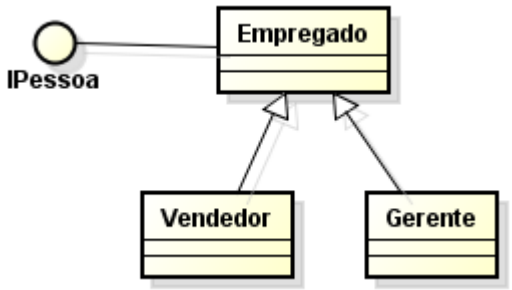
- Interfaces são utilizadas para definir funcionalidades requeridas para uma classe sem se preocupar com sua implementação
  - Estabelece um "contrato" para uma futura classe, ou seja, um comportamento desejado para uma classe
  - É composta por uma lista de métodos que devem ser implementados por uma classe
  - Uma classe pode implementar mais de uma interface
- Limitações
  - Não permitem atributos, nem construtores
  - Todos os métodos da interface são públicos

Interfaces no C#

- Interfaces no C# iniciam com a letra I
  - interface IPessoa {
  - string RetornarNome();
  - }
- Classe com implementação da interface. Os métodos listados na interface devem ser públicos.
  - class Empregado : IPessoa {
  - private string nome;
  - public string RetornarNome() {
  - return nome; }
  - ...
  - }

Referências de Interface

- É possível referenciar um objeto com uma referência da interface
- Apenas os métodos listados na interface podem ser invocados pela referência
  - IPessoa w;
  - w = new Gerente("Josué", 4000, 300);
  - Console.WriteLine(w.RetornarNome());



Classes Abstratas

- Uma classe é abstrata quando pelo menos uma parte de sua implementação não é realizada
- As classes abstratas estão situadas entre uma classe normal (totalmente implementada) e uma interface (totalmente não implementada)
- As classes abstratas são usadas para evitar duplicação de códigos entre classes que possuam uma parte da funcionalidade em comum
- Classes abstratas não podem ser instanciadas

Métodos Abstratos

- Os métodos abstratos em uma classe devem ser identificados com a palavra reservada *abstract* e não tem corpo (implementação)
- A classe abstrata Figura possui um atributo, um construtor, um método implementado e outro abstrato

```
abstract class Figura {
    private string nome;
    public Figura(string nome) { this.nome = nome; }
    public string GetNome() { return nome; }
    public abstract double GetArea();
}
```

Herdando de uma Classe Abstrata

- Uma classe não-abstrata que herde uma abstrata deve implementar todos os métodos abstratos
- Todo método abstrato é também virtual

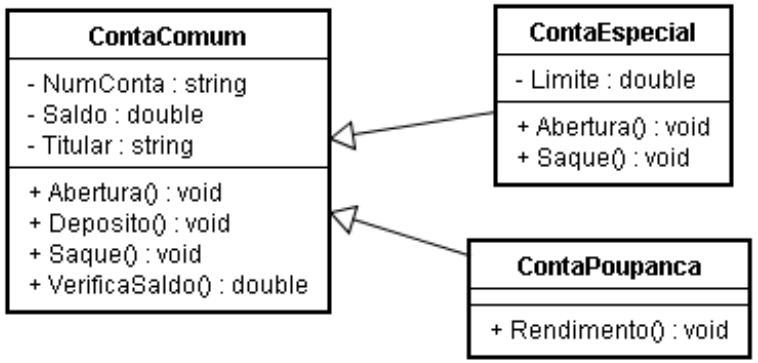
```
class Triangulo : Figura {
    private double b;
    private double h;
    public Triangulo(double aBase, double aAltura) : base("Triângulo") {
        b = aBase; h = aAltura;
    }
    public override double GetArea() {
        return b * h / 2;
    }
}
```

Classes Seladas

- Em uma classe selada, o mecanismo de herança é desativado e nenhuma classe descendente pode ser declarada
  - sealed class UmaClasseSelada {
  - ...
  - }
- Métodos Selados
  - Um método pode ser declarado como *sealed override* indicando a última implementação do método em uma hierarquia de classes

Exercício

- 1. Definir a classe ContaComum com os atributos, Número da Conta, Saldo e Titular e as operações Abertura (recebe o depósito inicial), Depósito, Saque e VerificaSaldo.
- 2. Definir a classe ContaEspecial, descendente de ContaComum, com o atributo de limite. Redefinir as operações de Abertura e Saque.
- 3. Definir a classe ContaPoupanca, descendente de ContaComum. Inserir a operação Rendimento.



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