IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Herança e Polimorfismo

Prof. Gilbert Azevedo

Objetivos

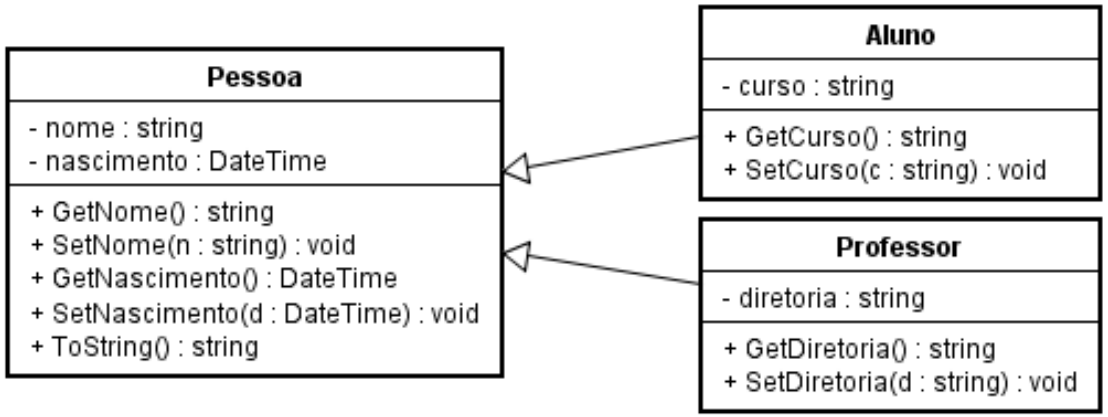
- Utilizar os conceitos de herança e polimorfismo na escrita de programas
- Representar o relacionamento de herança na UML
- Estudar a relação entre herança e encapsulamento
- Escrever classes em C# utilizando herança e polimorfismo
  - Object, Construtores, Polimorfismo e Métodos Virtuais
  - Operadores de tipo: is e as
- Exemplificar o relacionamento de especialização e generalização

Herança em POO

- Encapsulamento, Herança e Polimorfismo são os conceitos chave da Programação Orientação a Objetos.
- A herança permite que uma classe reuse, estenda e/ou modifique uma classe já existente.
- Representa o relacionamento de generalização-especialização entre classes: onde uma classe é especializada em várias outras, podendo as novas classes incorporar novos membros.
- Classe cujos membros são herdados: classe base, super-classe
- Classe que herda os membros: classe derivada ou sub-classe

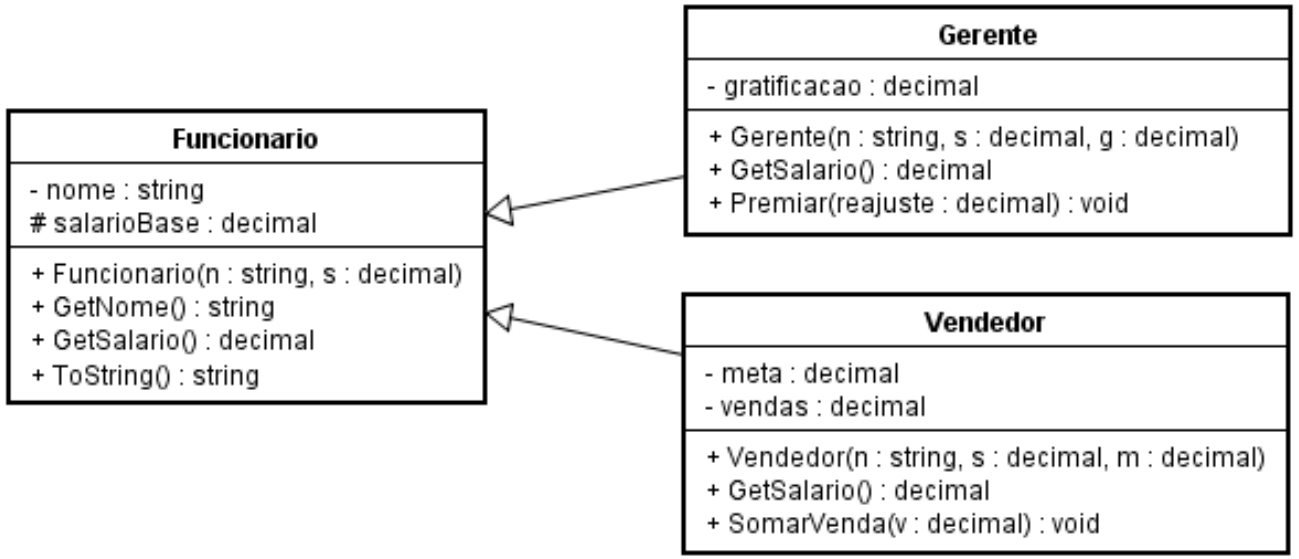
Herança e UML

- Na UML, a herança é indicada por uma seta da sub-classe (derivada) para a super-classe (base).
  - Pessoa é a classe base: tipo menos derivado
  - Aluno e Professor são as classes derivadas: tipo mais derivado



Herança e Encapsulamento

- O encapsulamento protegido informa que um membro da classe é visível para a própria classe e para as classes descendentes
  - – é usado para membros privados (visível na própria classe)
  - # é usado para membros protegidos (visível na classe e descendestes)
  - + é usado para os membros públicos (visível dentro e fora da classe)



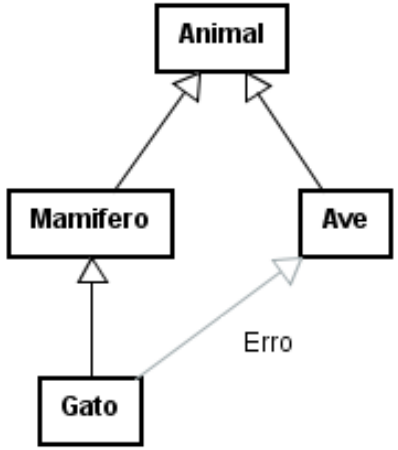
Modificadores de Acesso no C#

- Principais modificadores de acesso do C#, do mais visível para o menos visível
  - public: O membro da classe é visível dentro e fora da classe, inclusive em diferentes projetos (assemblies)
  - internal: O membro é público dentro de um projeto
  - protected: O membro é visível na classe e nas classes descendentes
  - private (padrão): O membro da classe é visível apenas dentro da própria classe

Herança em C#

- Em C#, uma classe herda diretamente de apenas uma classe base – Herança Simples. Mas podem ocorrer vários níveis de herança.
- Sintaxe para herança no C# utiliza o ":"

```
class Animal { }
class Mamifero : Animal { }
class Ave : Animal { }
class Gato : Mamifero { }
class Gato : Mamifero, Ave { } // Erro
```



- A classe Gato herda de Mamífero que herda de Animal.

System.Object

- No C#, Object é a classe base de todas as classes, mesmo que não seja informada explicitamente na definição de uma classe

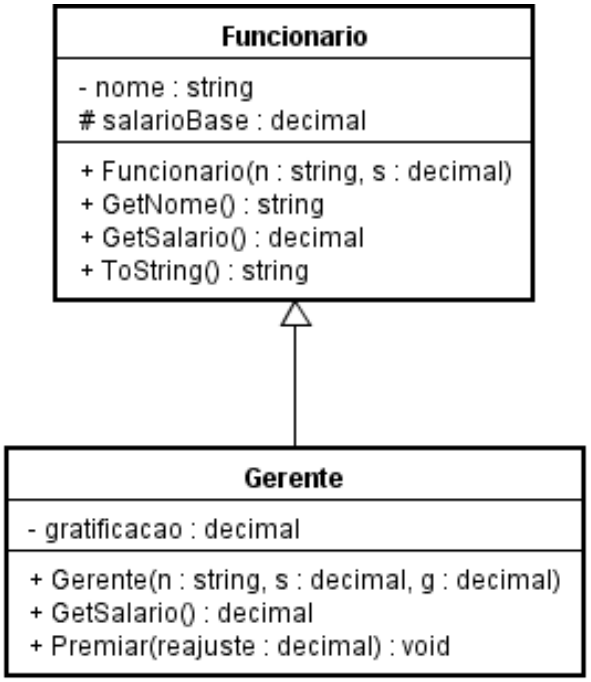
```
class Funcionario { }
class Funcionario : Object { }
```

- Métodos públicos que são herdados de 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 uma string com informações do objeto

Herança e Construtores

- Construtores (e finalizadores) da classe base são os únicos membros não herdados pelas classes derivadas
  - Classe Gerente não herda o construtor da classe Funcionario

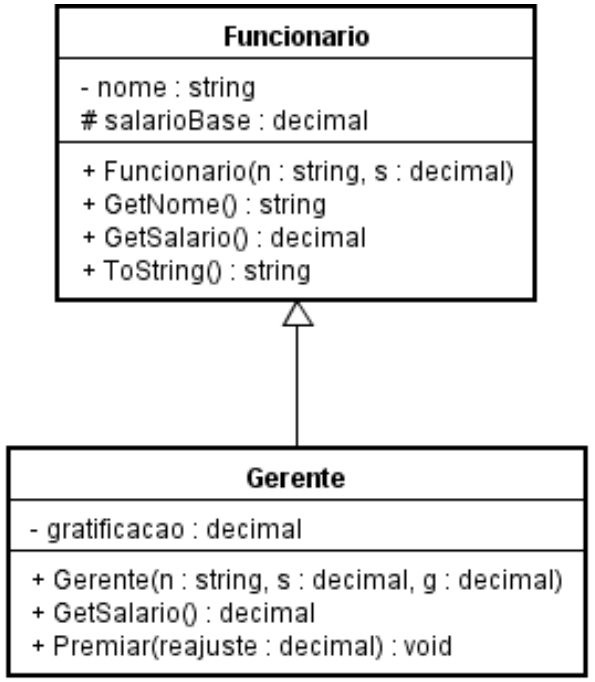
```
class Funcionario {
  private string nome;
  protected decimal salarioBase;
  public Funcionario(string n, decimal s) {
    nome = n;
    salarioBase = s;
  }
}
```



Herança e Construtores

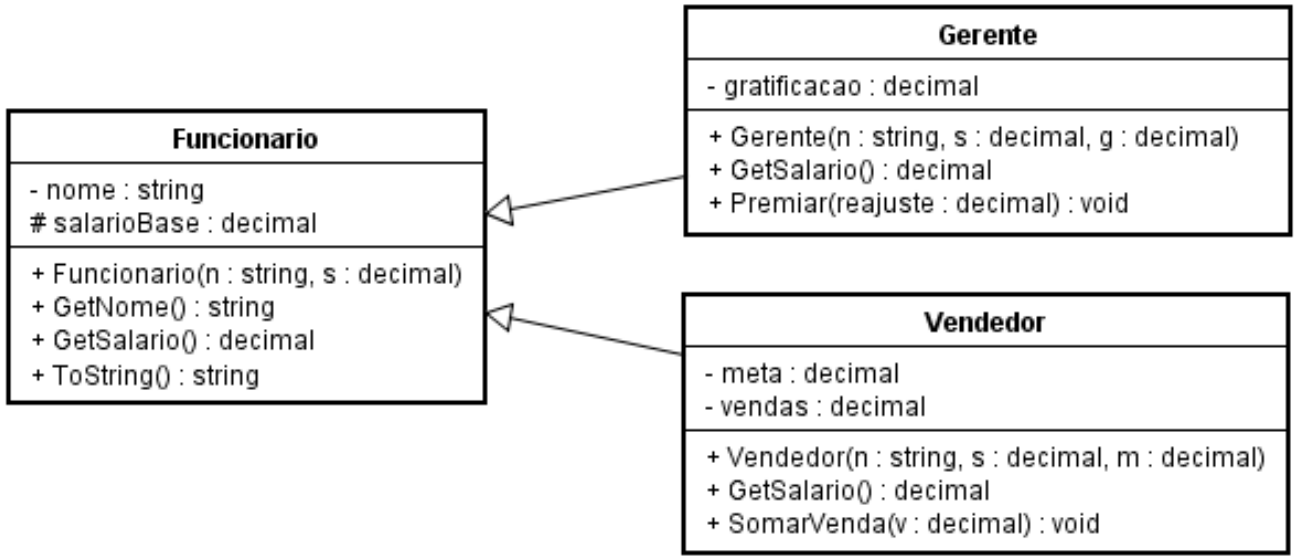
- Os construtores da classe base devem ser chamados pelos construtores da classe derivada com a instrução ***base***

```
class Gerente : Funcionario {
  private decimal gratificacao;
  public Gerente(string n, decimal s, decimal g) :
    base (n, s) {
    gratificacao = g;
  }
}

Gerente g = new Gerente("nome", 1000, 1000);
```



Polimorfismo

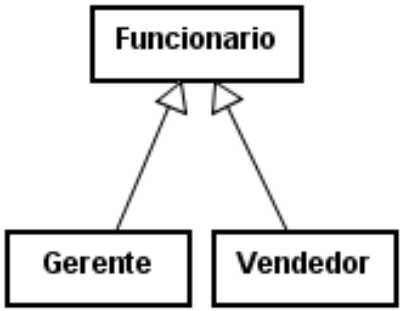
- Polimorfismo é o conceito da POO que ocorre quando objetos de classes distintas em uma hierarquia realizam uma mesma operação de forma diferente. O método herdado é reescrito (*override*).
  - GetSalario nas classes Funcionario, Gerente e Vendedor
  - ToString nas classes Object e Funcionario



Métodos Virtuais

- No C#, os métodos virtuais são usados para realizar o polimorfismo.
- O método é *virtual* na classe base e *override* nas descendentes

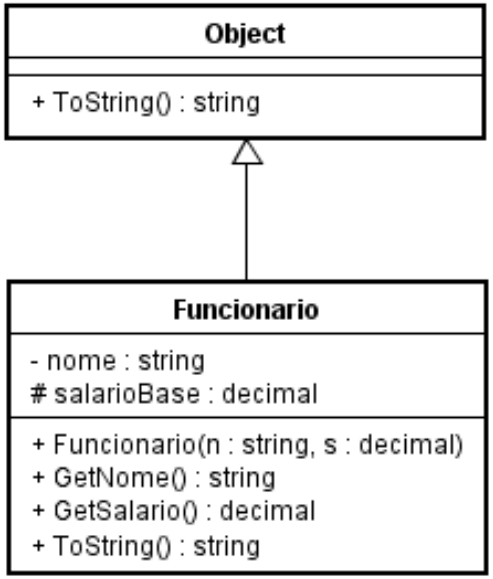
```
// Funcionario
public virtual decimal GetSalario() {
  return salarioBase;
}
// Gerente
public override decimal GetSalario() {
  return salarioBase + gratificacao;
}
// Vendedor
public override decimal GetSalario() {
  return salarioBase + 0.02M * vendas;
}
```



Sobrescrita do ToString

- O método ToString é definido na classe Object e pode ser reescrito em qualquer classe.

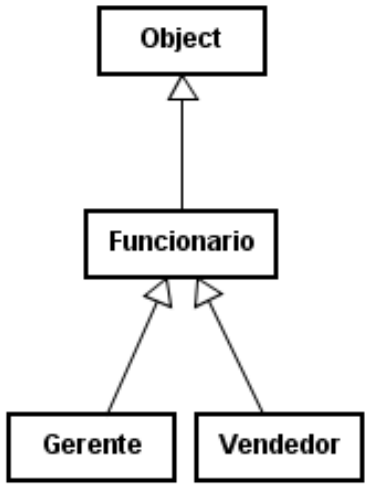
```
class Object {
  public virtual string ToString() { }
}

class Funcionario {
  public override string ToString() {
    return $"{nome} - {GetSalario()}";
  }
}
```



Herança e Referências

- É possível referenciar um objeto de um tipo com uma referência do mesmo tipo ou de um tipo menos derivado

```
Object x      = new Gerente("Pedro", 1000, 500);
Funcionario y = new Gerente("Maria", 2000, 800);
Gerente z     = new Gerente("Paulo", 3000, 900);
Console.WriteLine(x.GetSalario()); // Erro
Console.WriteLine(y.GetSalario()); // dynamic binding
Console.WriteLine(z.GetSalario());
```

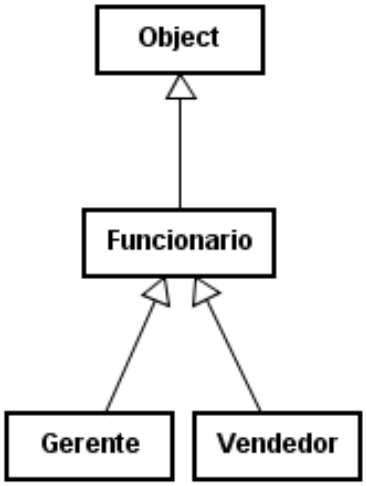


- Apenas os membros da classe da referência são acessíveis embora o objeto seja sempre do tipo do construtor usado.

Operadores is

- A herança expressa um relacionamento do tipo “é um(a)”
- O operador de teste de tipo *is* verifica se o objeto é de uma classe ou de uma classe derivada desta classe, retornando verdadeiro ou falso

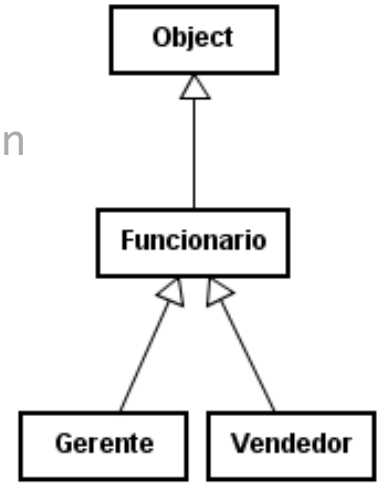
```
Vendedor v = new Vendedor(...);
if (v is object) ...          // Verdadeiro
if (v is Funcionario) ...     // Verdadeiro
if (v is Vendedor) ...        // Verdadeiro
if (v is Gerente) ...         // Falso
```



Operadores as

- O operador de conversão de tipo *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).GetSalario(); // Ok
(v as Gerente).GetSalario();  // NullReferenceException
```



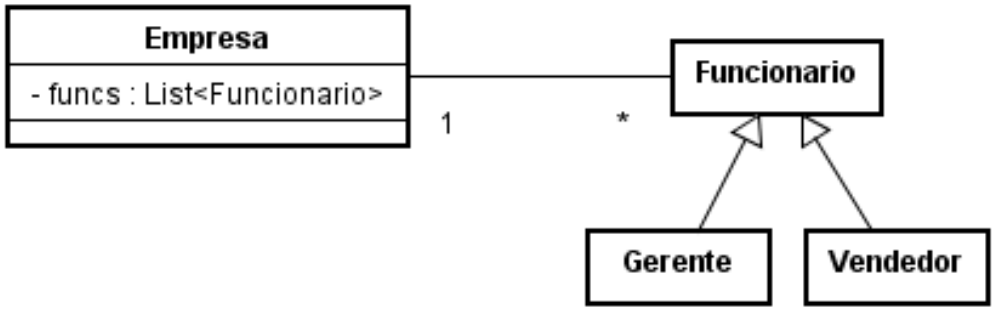
- Conversão de tipo (type-casting) sem verificação

```
((Vendedor) v).GetSalario();  // Ok
((Gerente) v).GetSalario();   // InvalidCastException
```

Especialização/Generalização

- A herança é um relacionamento de especialização-generalização e pode ser usado na representação de entidades
- Em situações específicas, os objetos podem ser referenciados por uma referência da classe base
  - A empresa tem funcionários que podem ser gerentes ou vendedores.

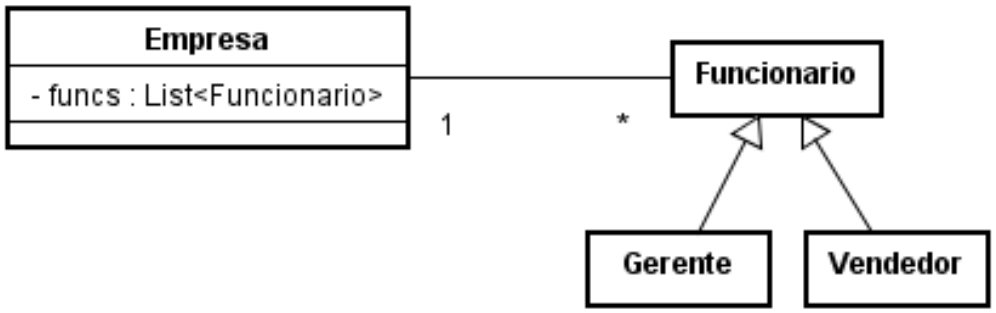
```
List<Funcionario> funcs = new List<Funcionario>();
```



Especialização/Generalização

- O operador de teste de tipo pode verificar se o objeto é de uma classe especializada
  - Por exemplo, se quiser descobrir quem são os gerentes

```
foreach(Funcionario f in funcs)
    if (f is Gerente) Console.WriteLine(f.GetNome());
```



Especialização/Generalização

- O operador de conversão de tipo pode converter a referência para uma classe especializada
  - Por exemplo, adicionar uma venda a um vendedor

```
Vendedor v = funcs[0] as Vendedor;
v.SomarVenda(1000);
```

- Nesso caso, se o funcionário funcs[0] não for um vendedor, uma exceção NullReferenceException é lançada

Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- UML – Uma Abordagem Prática, Gilleanes T. A. Guedes, Novatec, 2004
- Herança (Guia de Programação C#) e Herança em C# e .NET
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/inheritance
  - https://docs.microsoft.com/pt-br/dotnet/csharp/tutorials/inheritance
- Polimorfismo (Guia de Programação C#)
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/polymorphism
- Operadores de teste de tipo e expressão de conversão (referência C#)
  - https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/type-testing-and-cast