Disciplina: POO - Programação Orientada a Objetos

Classe Abstrata
Método Abstrato

Profª. Drª. Giovana Angélica Ros Miola
giovana.miola@fatec.sp.gov.br

Centro Paula Souza

Fatec Presidente Prudente

Classes abstratas

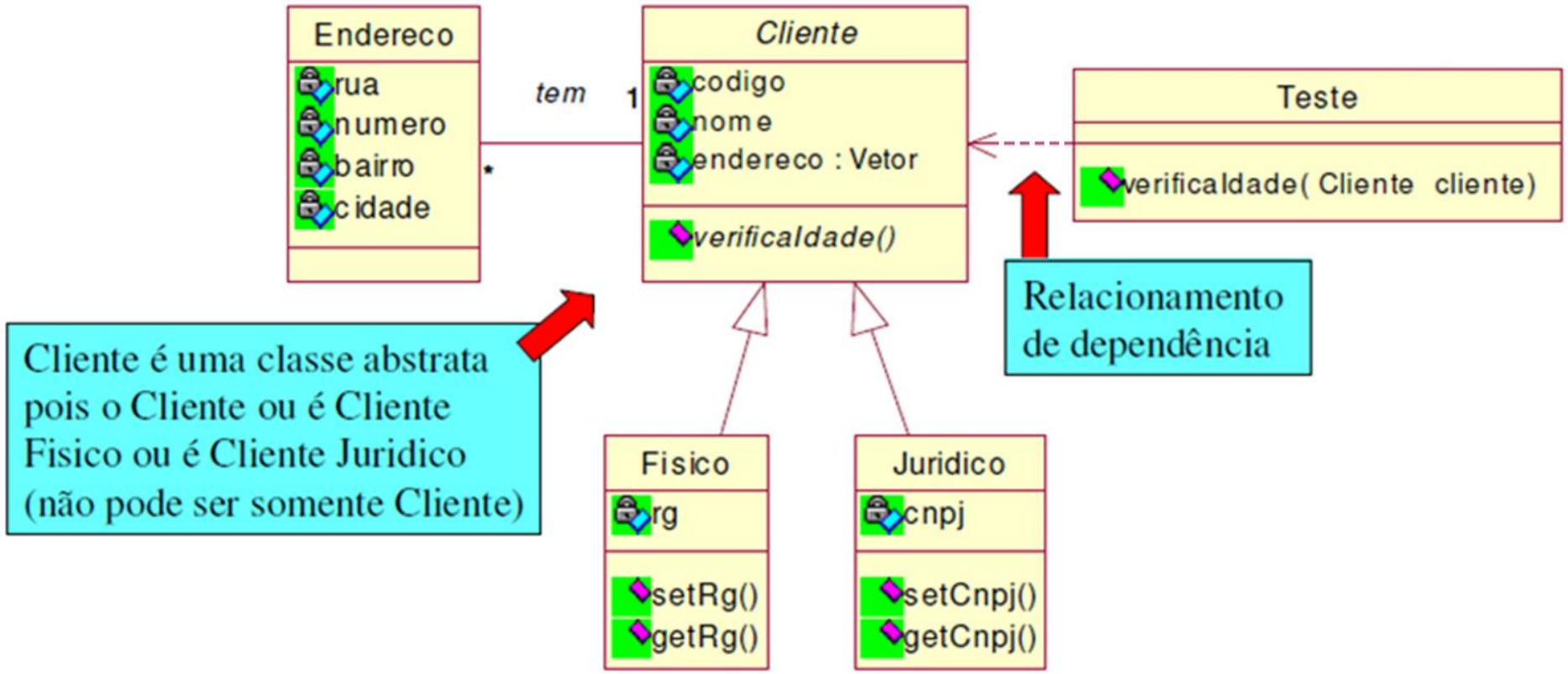
- Classe abstrata é um tipo de classe que somente pode ser herdada e **não instanciada**
- Este tipo de classe é uma classe conceitual que pode definir funcionalidades para que as suas subclasses possam implementá-las
- O propósito da classe abstrata é fornecer uma superclasse apropriada a partir da qual as subclasses possam herdar e compartilhar um **projeto comum**
- Uma classe abstrata (superclasse abstrata) existe apenas para agrupar um comportamento comum de suas subclasses
  - **Não pode ser instanciada**

Classes e Métodos Abstratos

- Para a declaração de classe abstrata deve-se utilizar a palavra-chave **abstract**
- Construtores e métodos **static** não podem ser declarados **abstract**
  - os construtores **static** não são herdados, portanto um construtor **abstract** nunca seria implementado
- Subclasses **não** podem subscrever métodos **static,** portanto **abstract static** nunca seria implementado
- Classes e métodos abstratos
  - Permitem a utilização de polimorfismo
- Um **método abstrato -** não tem corpo (sem lógica)
  - Deve ser obrigatoriamente implementado pelas subclasses

Exemplo – Classes e Métodos abstratos

O método verificaIdade() da classe Teste, recebe por parâmetro um cliente e invoca o método abstrato verificaIdade() para este cliente, que pode ser Fisico ou Juridico. Se for Fisico ele funciona de uma forma, se Juridico de outra.



Classes Abstrata Cliente e Concreta ClienteFisico

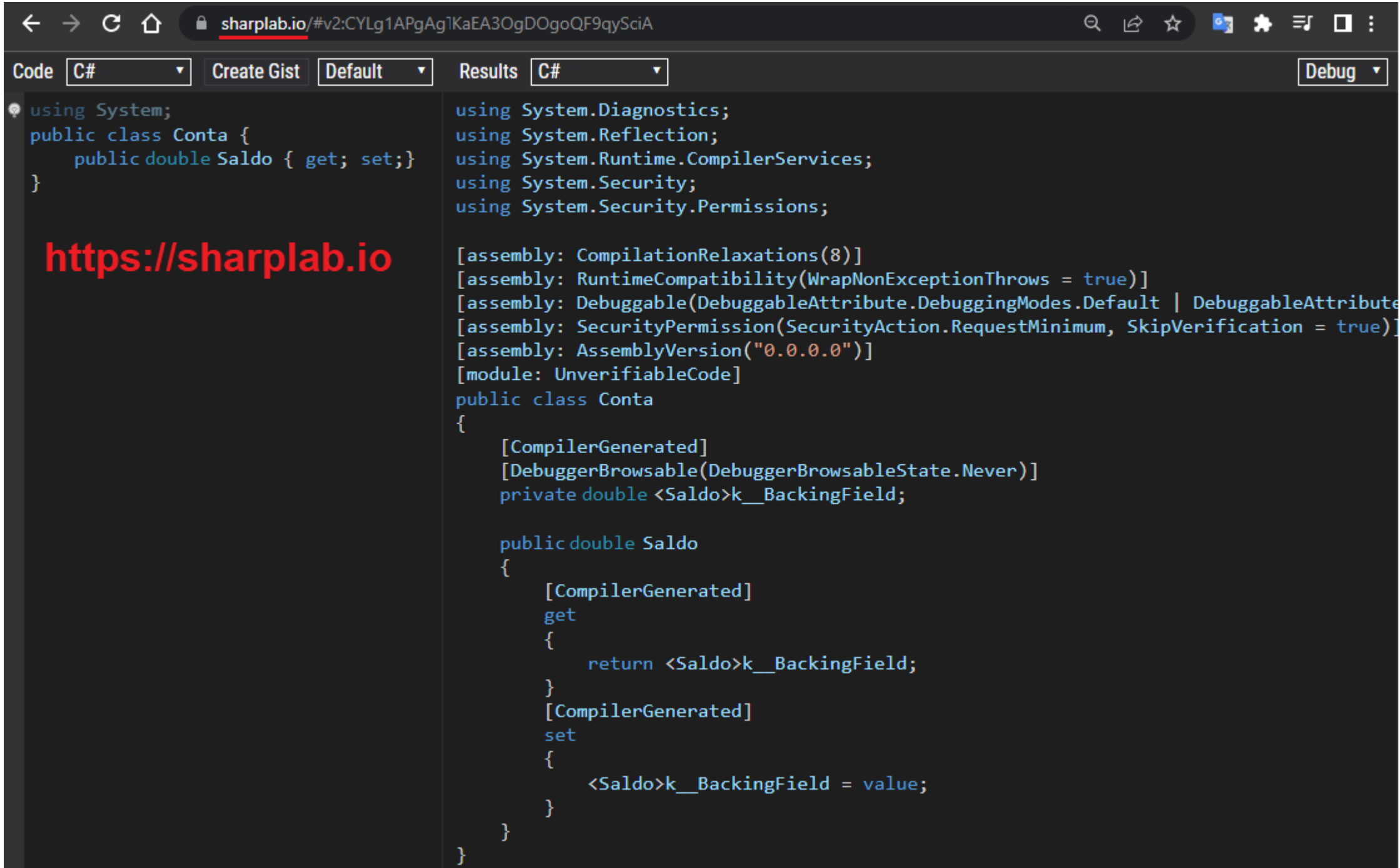
Classe base

```
abstract class Cliente{
        public int Codigo { get; set; }
        public string Nome { get; set; }
        public int Idade { get; set; }

        public void MostraDados(){
            Console.WriteLine("Código: " + Codigo
            +"\tNome: " + Nome + "\tIdade: " +
            Idade);
        }

        public abstract bool VerificaIdade() ;
}
```

Classe derivada

```
class ClienteFisico: Cliente
{
        private int rg;

        public override bool VerificaIdade()
        {
            if (Idade > 18)
                    return true;
            return false;
        }
}
```

Não há desenvolvimento de lógica na classe base. A classe derivada é que DEVERÁ implementar a lógica adequada.


sharplab.io/#v2:CYLg1APgAg1KaEA3OgDOgoQF9qySciA
Code C# Create Gist Default Results C# Debug
using System;
public class Conta {
public double Saldo { get; set;}
}
https://sharplab.io
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Security;
using System.Security.Permissions;
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default | DebuggableAttribute
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerification = true)]
[assembly: AssemblyVersion("0.0.0.0")]
[module: UnverifiableCode]
public class Conta
{
[CompilerGenerated]
[DebuggerBrowsable(DebuggerBrowsableState.Never)]
private double <Saldo>k__BackingField;
public double Saldo
{
[CompilerGenerated]
get
{
return <Saldo>k__BackingField;
}
[CompilerGenerated]
set
{
<Saldo>k__BackingField = value;
}
}
}

Classe Concreta ClienteJuridico e Concreta Teste

Exemplo de relacionamento (dependência) entre classe. Ocorre por meio da passagem de parâmetro com objeto.

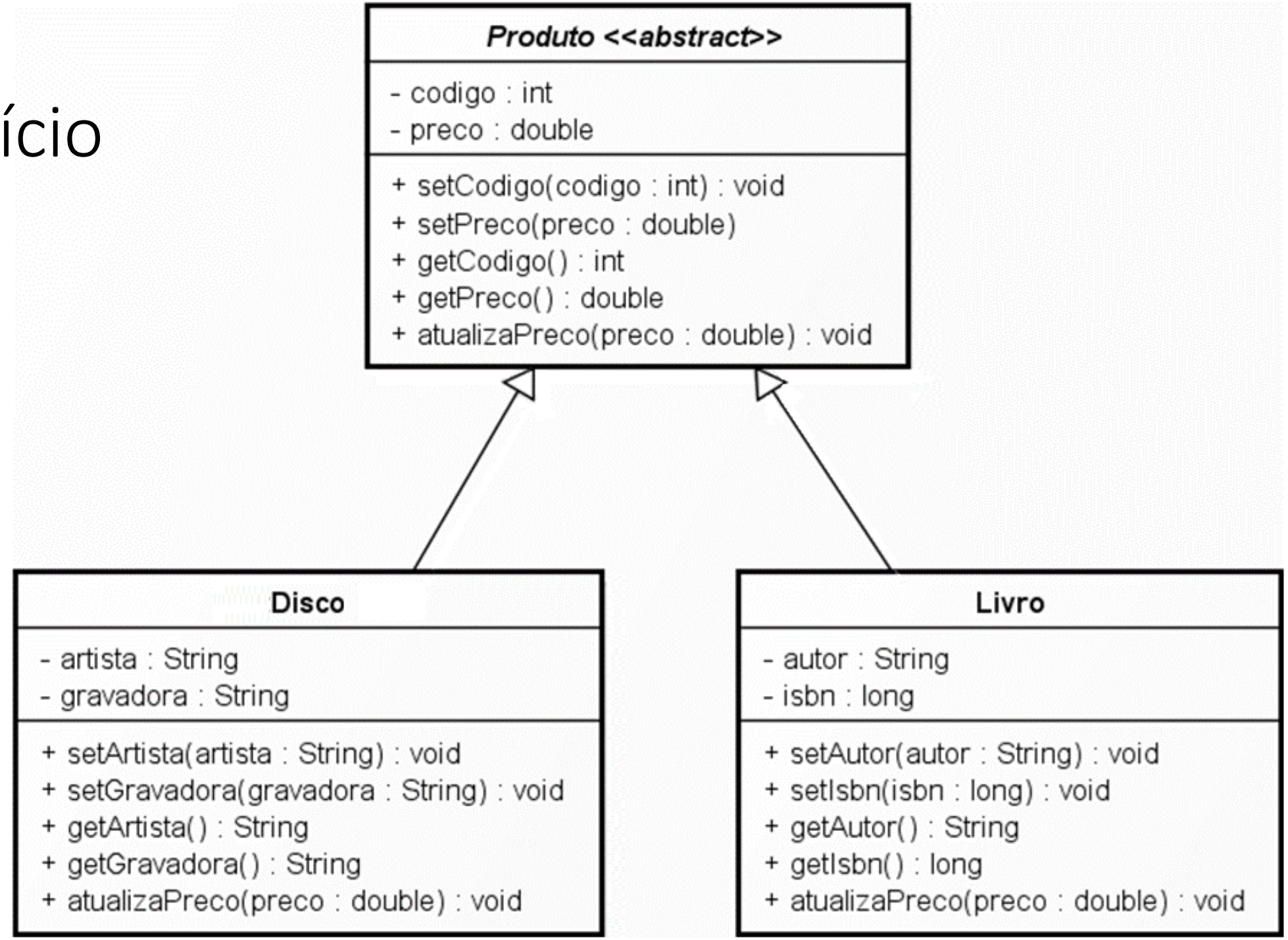
```
class ClienteJuridico : Cliente
{
    public int Cnpj {get; set;};

    public override bool VerificaIdade()
    {
        if (Idade > 50)
            return true;
        return false;
    }
}
```

```
class Teste{
    public bool AvaliaIdade (Cliente objCliente)
    {
        return objCliente.VerificaIdade();
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        //Cliente c = new Cliente();//ERRO
        ClienteFisico cf = new ClienteFisico();
        cf.Idade = 20;
        Teste t= new Teste();
        Console.WriteLine(t.AvaliaIdade(cf));//True
        Console.ReadKey();
    }
}
```

Nesta passagem de parâmetro, o objeto não é nem Físico, nem Jurídico, ele é um Cliente, desta forma por meio da generalização, ocorre a execução na classe correta, pois é verificado qual objeto foi instanciado.

Exercício


Produto <<abstract>>
- codigo : int
- preco : double
+ setCodigo(codigo : int) : void
+ setPreco(preco : double)
+ getCodigo( ) : int
+ getPreco( ) : double
+ atualizaPreco(preco : double) : void
Disco
- artista : String
- gravadora : String
+ setArtista(artista : String) : void
+ setGravadora(gravadora : String) : void
+ getArtista( ) : String
+ getGravadora( ) : String
+ atualizaPreco(preco : double) : void
Livro
- autor : String
- isbn : long
+ setAutor(autor : String) : void
+ setIsbn(isbn : long) : void
+ getAutor( ) : String
+ getIsbn( ) : long
+ atualizaPreco(preco : double) : void

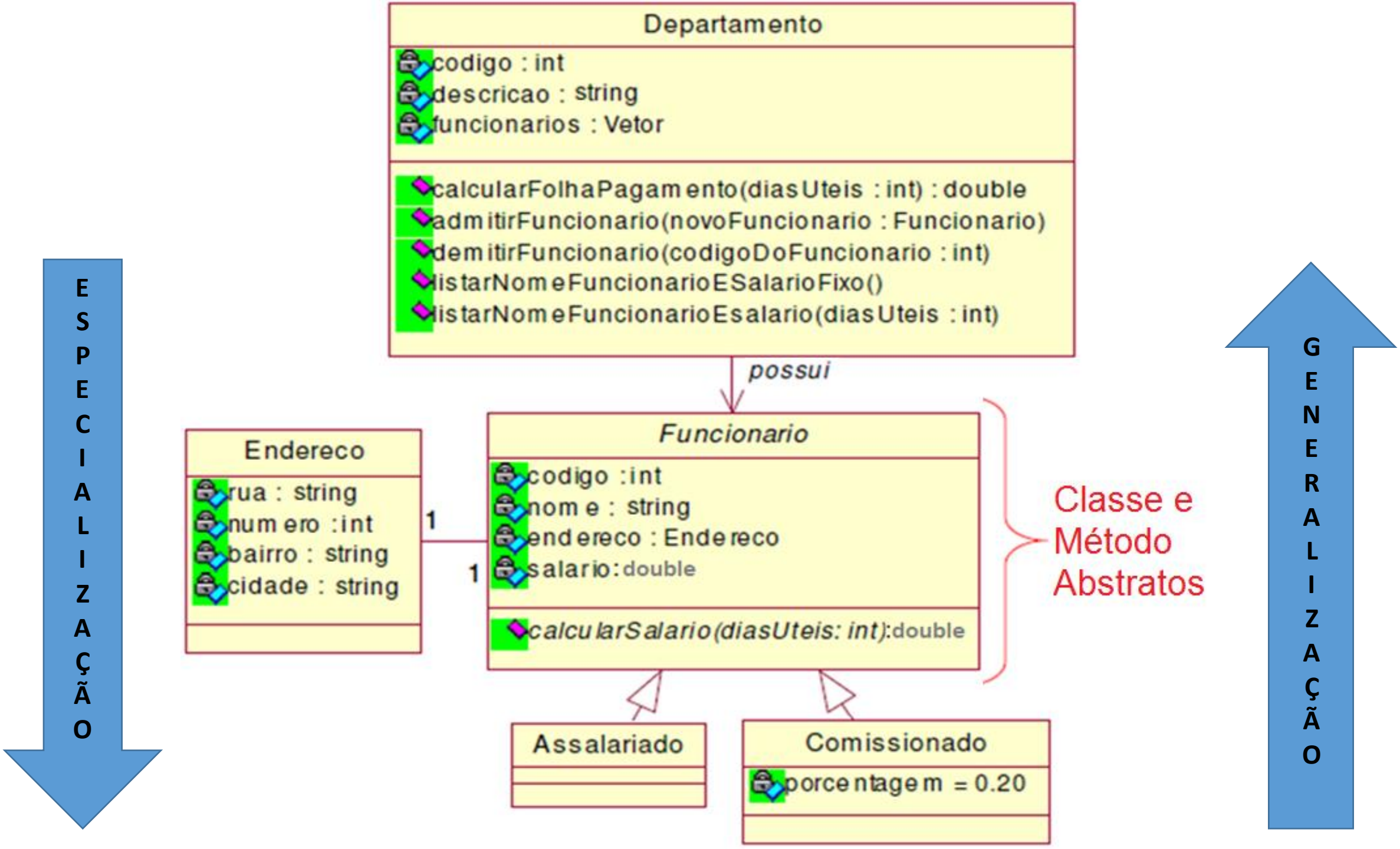
Exercício

- Implementar um sistema de controle dos funcionários de um determinado departamento.
- Existem dois tipos de funcionários
  - Assalariados e comissionados
- Os comissionados possuem uma porcentagem de comissão **de 0.20**%, mas que pode ser alterada mediante o desempenho do funcionário
- O cálculo do salário do assalariado é:
  - (Salario/30) * dias úteis do mês
- O cálculo do salário do comissionado é:
  - Salario/30 * dias úteis * comissão + Salario

Continuação ... Exercício

- Deve ser possível:
  - Admitir um determinado funcionário de um departamento
  - Listar nome e salário dos funcionários
  - Demitir um determinado funcionário pelo seu código
  - Calcular o custo com a folha de pagamento dos departamentos

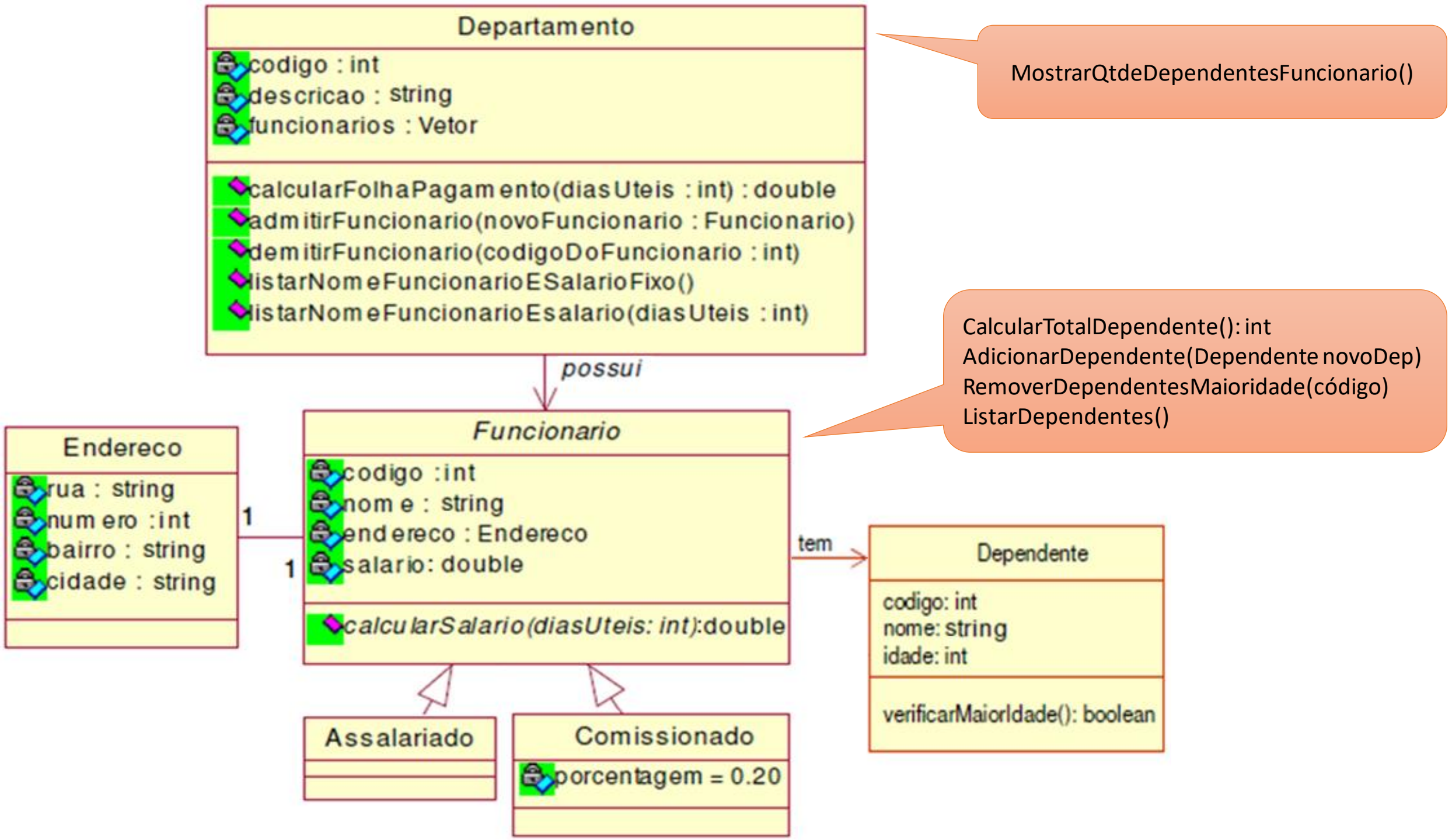
- Implementar um programa principal que:
  - Instancie alguns objetos assalariados e comissionados
  - Realize as tarefas pedidas.


Departamento
codigo : int
descricao : string
funcionarios : Vetor
calcularFolhaPagamento(diasUteis : int) : double
admitirFuncionario(novoFuncionario : Funcionario)
demitirFuncionario(codigoDoFuncionario : int)
listarNomeFuncionarioESalarioFixo()
listarNomeFuncionarioEsalario(diasUteis : int)
possui
Endereco
rua : string
numero : int
bairro : string
cidade : string
1
1
Funcionario
codigo : int
nome : string
endereco : Endereco
salario: double
calcularSalario(diasUteis: int):double
Classe e Método Abstratos
Assalariado
Comissionado
porcentagem = 0.20
ESPECIALIZAÇÃO
GENERALIZAÇÃO

Exercício

- **Aumentando funcionalidades no sistema ...**

- Adicionar a classe Dependente no sistema do exercício anterior
  - Um funcionário pode ter nenhum ou vários dependentes
- O sistema deve:
  - Calcular o total de dependentes
  - Adicionar um determinado dependente de um funcionário
  - Remover um dependente de um determinado funcionário (código do dependente)
  - ListaTr nome e idade dos dependentes
- Complementar a classe principal para:
  - Instanciar alguns objetos dependentes
  - Calcular a quantidade de dependentes de cada funcionário, de cada departamento


Departamento
codigo : int
descricao : string
funcionarios : Vetor
calcularFolhaPagamento(diasUteis : int) : double
admitirFuncionario(novoFuncionario : Funcionario)
demitirFuncionario(codigoDoFuncionario : int)
listarNomeFuncionarioESalarioFixo()
listarNomeFuncionarioEsalario(diasUteis : int)
MostrarQtdeDependentesFuncionario()
possui
Funcionario
codigo :int
nome : string
endereco : Endereco
salario: double
calcularSalario(diasUteis: int):double
CalcularTotalDependente(): int
AdicionarDependente(Dependente novoDep)
RemoverDependentesMaioridade(código)
ListarDependentes()
Endereco
rua : string
numero :int
bairro : string
cidade : string
1
1
tem
Dependente
codigo: int
nome: string
idade: int
verificarMaiorIdade(): boolean
Assalariado
Comissionado
porcentagem = 0.20