CEUB

EDUCAÇÃO SUPERIOR

Programação Orientada a Objetos

ceub.br

CEUB
EDUCAÇÃO SUPERIOR

Programação Orientada a Objetos

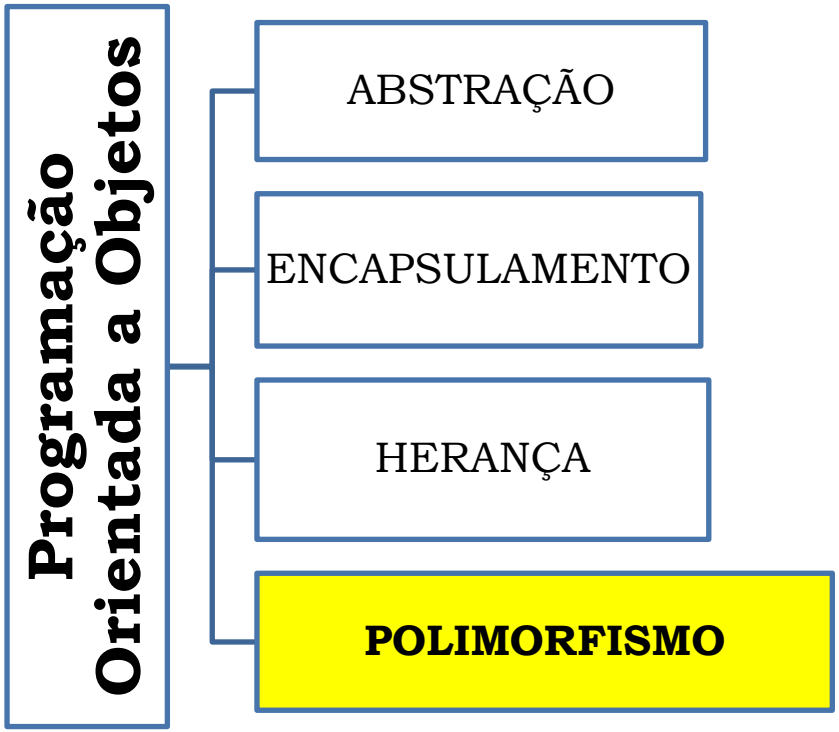
Aula 05 – Polimorfismo, Classes Abstratas e Interfaces

Agenda

Polimorfismo
Sobrecarga
Sobrescrita
Classe Abstratas
Interfaces

Os 4 pilares da Programação Orientada a Objetos

Para que uma linguagem possa ser enquadrada no paradigma de orientação a objetos, ela deve atender a **quatro tópicos** bastante importantes:



Agenda

Polimorfismo
Sobrecarga
Sobrescrita
Classe Abstratas
Interfaces

Polimorfismo

O quarto e último pilar da POO é conhecido como polimorfismo. Trata-se da capacidade de alterar a forma original conforme a necessidade do momento.

A palavra polimorfismo vem do grego e significa muitas formas (poli: muitas, morphos: formas);

Em suma, o polimorfismo consiste na alteração de todo o funcionamento interno de um método herdado de um outro objeto dentro da aplicação.

Polimorfismo

No Polimorfismo temos dois tipos:

- **Polimorfismo Estático ou Sobrecarga**
- **Polimorfismo Dinâmico ou Sobreposição**

O Polimorfismo Estático se dá quando temos a mesma operação implementada várias vezes na mesma classe. A escolha de qual operação será chamada depende da assinatura dos métodos sobrecarregados.

Polimorfismo

O **Polimorfismo Dinâmico** acontece na herança, quando a subclasse sobrepõe o método original. Agora o método escolhido se dá em tempo de execução e não mais em tempo de compilação.

A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

Polimorfismo

MANIPULAÇÃO DE MÉTODOS NAS SUBCLASSES

- **Sobrecarga (*overloading*)**
  - **Ocorre quando uma subclasse define um método com o mesmo nome do método herdado da superclasse, contudo com a sua assinatura diferente.**

- **Sobrescrita (*overriding*)**
  - **Ocorre quando uma subclasse define um método com o mesmo nome e a mesma assinatura do método herdado da superclasse.**
  - **Métodos constantes (final) não podem ser sobrepostos.**

Sobrecarga (*overloading*)

Sobrecarga (overloading)

- **Sobrecarga é utilizada quando você escreve métodos com o mesmo nome mas com assinaturas diferentes.**

- **Uma boa prática é usar a sobrecarga, somente, nos métodos que possuam a mesma funcionalidade.**
- **A sobrecarga pode ser feita igualmente nos métodos construtores.**

EXEMPLO

```
public class ExemploSobrecarga {

   public int somar(int x, int y){
       return (x+y);
   }

   public int somar(int x, int y, int z){
       return (x+y+z);
   }
}
```

Sobrecarga (overloading)

Exemplo

Criar uma classe Calculadora com a seguinte estrutura

```java
public class Calculadora {
    public int Soma(int numero1, int numero2) {
        return numero1 + numero2;
    }

    public double Soma(double numero1, double numero2) {
         return numero1 + numero2;
    }

    public int Soma(int numero1, int numero2, int numero3) {
        return numero1 + numero2 + numero3;
    }

    public float Soma(float numero1, float numero2) {
        return numero1 + numero2;
    }
}
```

Sobrecarga (overloading)

Exemplo

Após criar a classe, criar um aplicativo console e informar os seguintes valores:

```
Calculadora minhaCalculadora = new Calculadora();


int resultado1 = minhaCalculadora.Soma(2, 3);          // chama o método Soma(int a, int b)
double resultado2 = minhaCalculadora.Soma(3.14, 2.71); // chama o método Soma(double a, double b)
int resultado3 = minhaCalculadora.Soma(2, 3, 4);       // chama o método Soma(int a, int b, int c)
float resultado4 = minhaCalculadora.Soma(2.5f, 3.5f);  // chama o método Soma(float a, float b)
```

Sobrecarga (overloading)

DIFERENÇAS this e Bases

- Usados quando for necessário referenciar explicitamente a instância (**this**) ou a superclasse (**Base**).

Em C#, this e base são palavras-chave que permitem acessar membros da classe atual e da classe base (classe pai), respectivamente.

A palavra-chave this é usada para se referir a um membro da classe atual. Por exemplo, se você tiver uma propriedade com o mesmo nome de um parâmetro em um método, pode usar this para se referir à propriedade em vez do parâmetro. Outro exemplo é quando você precisa passar a própria instância da classe como parâmetro para outro método, usando this você pode fazer isso de forma mais fácil.

Já a palavra-chave base é usada para se referir a um membro da classe base. Por exemplo, se a classe atual estender outra classe (classe base) e você quiser chamar um construtor da classe base, você pode usar base para fazer isso.

Sobrecarga (overloading)

DIFERENÇAS this e Bases

Em C#, this e base são palavras-chave que permitem acessar membros da classe atual e da classe base (classe pai), respectivamente.

A palavra-chave this é usada para se referir a um membro da classe atual. Por exemplo, se você tiver uma propriedade com o mesmo nome de um parâmetro em um método, pode usar this para se referir à propriedade em vez do parâmetro.

Outro exemplo é quando você precisa passar a própria instância da classe como parâmetro para outro método, usando this você pode fazer isso de forma mais fácil.

Sobrecarga (overloading)

DIFERENÇAS this e Bases

Já a palavra-chave base é usada para se referir a um membro da classe base.

Por exemplo, se a classe atual estender outra classe (classe base) e você quiser chamar um construtor da classe base, você pode usar base para fazer isso.

Sobrecarga (overloading)

DIFERENÇAS this e Bases

Exemplo

```
public class Pessoa {
    public string Nome { get; set; }

    public Pessoa(string nome) {
        this.Nome = nome;
    }
}

public class Aluno : Pessoa {
    public string Matricula { get; set; }

    public Aluno(string nome, string matricula) : base(nome) {
        this.Matricula = matricula;
    }
}
```

Nesse exemplo, a classe Aluno estende a classe Pessoa.
O construtor da classe Pessoa recebe o nome da pessoa como parâmetro, que é atribuído à propriedade Nome.

Na classe Aluno, o construtor recebe o nome e a matrícula do aluno como parâmetros. Para passar o nome para o construtor da classe base, a palavra-chave base é usada, e para atribuir a matrícula à propriedade Matricula da classe atual, a palavra-chave this é usada.

Sobrescrita (*overriding*)

Sobrescrita (Override)

Sobrescrita ou Sobreposição de métodos (Override)

A Sobrescrita de métodos (override) é um conceito do polimorfismo que nos permite reescrever um método, ou seja, podemos reescrever nas classes filhas métodos criados inicialmente na classe pai.

Sobrescrita (Override)

Com o uso da palavra-chave **virtual** (destinada a métodos, propriedades, indexadores e eventos), determinamos que um membro pode ser sobrescrito em uma classe filha.

Por sua vez, usando a palavra-chave **override** determinamos que na classe derivada, um membro **virtual** da classe base pode ser **sobrescrito**.

Devemos ter em mente que ambas as palavras-chave completam uma à outra.

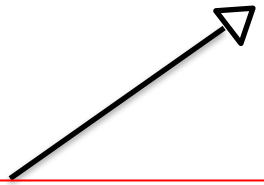
Importante salientar também que a propagação da palavra-chave **virtual** ocorre para descendentes.

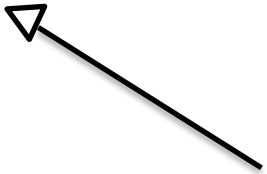
Um método **virtual** pode ser sobrescrito em descendentes e, ainda, em uma classe derivada.

Sobrescrita (Override)

Exemplo Override

```
class Funcionario
{
      private string nome;
      private double salario;
      public string Nome { get => nome; set => nome = value; }
      public double Salario { get => salario; set => salario = value; }
      public virtual double getSalario(int horas)
      {
          salario = horas * 100;
          return salario;
      }
}
```





```
class Estagiario : Funcionario
{
    public override double getSalario(int horas)
    {
       return horas*50;
    }
}
```

```
class Diretor : Funcionario
{
   public override double getSalario(int horas)
   {
       return horas*200;
   }
}
```

Sobrescrita (Override)

Exemplo Override

```
namespace ExemploPolimorfismos
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Diretor diretor = new Diretor();
            Estagiario estagiario = new Estagiario();
            Console.WriteLine("A bonificação do Estagiário será de R$ {0} ", estagiario.getSalario(100));
            Console.WriteLine("A bonificação do Diretor será de R$ {0} ", diretor.getSalario(100));
        }
    }
}
```

Sobrescrita (Override)

Sobrescrita de método

Exemplo2.

- Vamos incluir a classe Funcionário um novo método bonificacao
- Esse método representa uma bonificação que todos os funcionários recebem no fim do ano e é referente a 10% do valor do salário. Porém, o gerente recebe uma bonificação de 15%.

- Como ficaria o código da nossa classe Funcionario?

Sobrescrita (Override)

```
3 references
class Funcionario
{
    private string nome;
    private double salario;
    0 references
    public string Nome { get => nome; set => nome = value; }
    1 reference
    public double Salario { get => salario; set => salario = value; }
    7 references
    public virtual double getSalario(int horas)
    {
        salario = horas * 100;
        return salario;
    }
    5 references
    public virtual double bonificacao()
    {
        double bonificacao = salario* 0.10;
        return bonificacao;
    }
}
```

Sobrescrita (Override)

Se deixarmos a classe Gerente como está, ela vai herdar o método bonificacao da classe Funcionario:

```
2 references
class Gerente : Funcionario
{
    6 references
    public override double bonificacao()
    {
        return base.bonificacao();
    }

    7 references
    public override double getSalario(int horas)
    {
        return base.getSalario(horas);
    }
}
```

Nesse caso, a bonificação do Gerente será 500 ao invés de 750, o que está errrado.

Sobrescrita (Override)

O método bonificacao deve ser reescrito na classe Gerente de modo que ele forneça o valor correto:

```
2 references
class Gerente : Funcionario
{
    5 references
    public override double bonificacao()
    {
        return Salario*0.15;
    }

    7 references
    public override double getSalario(int horas)
    {
        return base.getSalario(horas);
    }
}
```

Sobrescrita (Override)

Agora, se executarmos o mesmo código anterior, ele dará o valor correto!

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Diretor diretor = new Diretor();
        Estagiario estagiario = new Estagiario();
        Gerente gerente = new Gerente();
        Console.WriteLine("O Salário do Estagiário será de R$ {0} ", estagiario.getSalario(100));
        Console.WriteLine("A bonificação do Estagiário será de R$ {0} ", estagiario.bonificacao());

        Console.WriteLine("O Salário do Gerente será de R$ {0} ", gerente.getSalario(100));
        Console.WriteLine("A bonificação do Gerente será de R$ {0} ", gerente.bonificacao());


        Console.WriteLine("A bonificação do Diretor será de R$ {0} ", diretor.getSalario(100));
    }
}
```

Exercícios - POLIMORFISMO

Criar uma superclasse chamada animal e as 3 seguintes subclasses: vaca, gato e carneiro.
Segue as classes com seus respectivos atributos e métodos.

Classe abstrata Animal possui um nome e numeroPatas e um método abstrato emitirSom

Classe Vaca herda de Animal e sobrescreve o método emitirSom. E deverá retornar como saída: Ex
System.out.println(“A vaca que tem 4 patas, faz MUUUU");

Classe Gato herda de Animal e sobrescreve o método emitirSom.

Classe Carneiro herda de Animal e sobrescreve o método emitirSom.

Faça um código para testar as classes herdadas: Classe TesteAnimais

Classes Abstratas

Classes Abstratas

- Ao se criar uma classe para ser estendida, é muito comum não se ter idéia de como codificar os seus métodos, isto é, somente as suas subclasses saberão implementá-los.

- Uma classe deste tipo não pode ser instanciada pois sua funcionalidade está incompleta. Tal classe é dita **abstrata**.

- OBS
- Ao se criar uma classe Abstrata, que tenha métodos abstratos, esses métodos deverão ser criados nas subclasses.

Classes Abstratas

- Podem existir casos em que a classe se comporta como um tipo, logo, supomos que os objetos desse tipo não serão instanciados.
- Nesses casos, chamamos a classe de classe Abstrata.
- O único objetivo dessa classe é servir de superclasse para outras classes.
- As classes que herdam de classes abstratas são conhecidas como classes concretas.
- Temos uma característica exclusiva desse tipo de classe, os métodos abstratos.

Classes Abstratas

- No exemplo em questão, não faz sentido criar objetos do tipo Funcionário, mas sim objetos do tipo Diretor, Professor ou Administrador.

- Nesta caso, especificando Funcionario como classe abstrata, economiza-se código e ganha-se o poliformismo para a criação de métodos genéricos que servirão a diversos objetos.

- C# suporta o conceito de classes abstratas. Pode-se declarar uma classe abstrata usando o modificador abstract.

- Métodos também podem ser declarados abstratos com o modificador abstract. As suas implementações serão feitas nas subclasses.

- As classes abstratas podem ter métodos concretos, campos de dados e construtores. Os objetos das suas subclasses poderão fazer uso deles.

Classes Abstratas

- **Pode acontecer que ao escrever um método para uma classe você não saiba como ele vai ser implementado. Neste caso, a implementação será feita pela classe que herdar o método (a classe filha).**

- **Pode acontecer também que você saiba que um determinado método será sobreposto com certeza na classe filha; então, por que definir sua implementação se ela não será usada ?**

Classes Abstratas

- **Nestes casos você apenas define a assinatura do método e deixa a definição por conta da classe que irá herdar a classe pai.**
- **Estas classes são então chamadas classes abstratas, o método que você não implementou é chamado de método abstrato.**
- **As classes abstratas não podem ser instanciadas através da palavra chave New.**

Classes Abstratas

- Uma classe abstrata é uma classe base genérica
  - Contém métodos abstratos que devem ser implementados nas classes que derivam dela

- Um método abstrato não apresenta implementação na classe base

```
public abstract class Pessoa {
    public abstract void Cadastrar();
    public abstract string Nome { get; set; }
    public abstract int Id { get; }
    public virtual void Viajar() { /* Ação */ }
}
```

- Pode conter membros não-abstratos

Classes Abstratas

- Derivando a classe abstrata e implementando os membros abstratos

```
public class Diretor : Pessoa
{
    public override void Cadastrar()
    { /* Ações */ }
    public override string Nome
    {
        get { /* Implementação get */ }
        set { /* Implementação set */ }
    }
    public override int Id
    {
        get { /* Implementação get */ }
    }
}
```

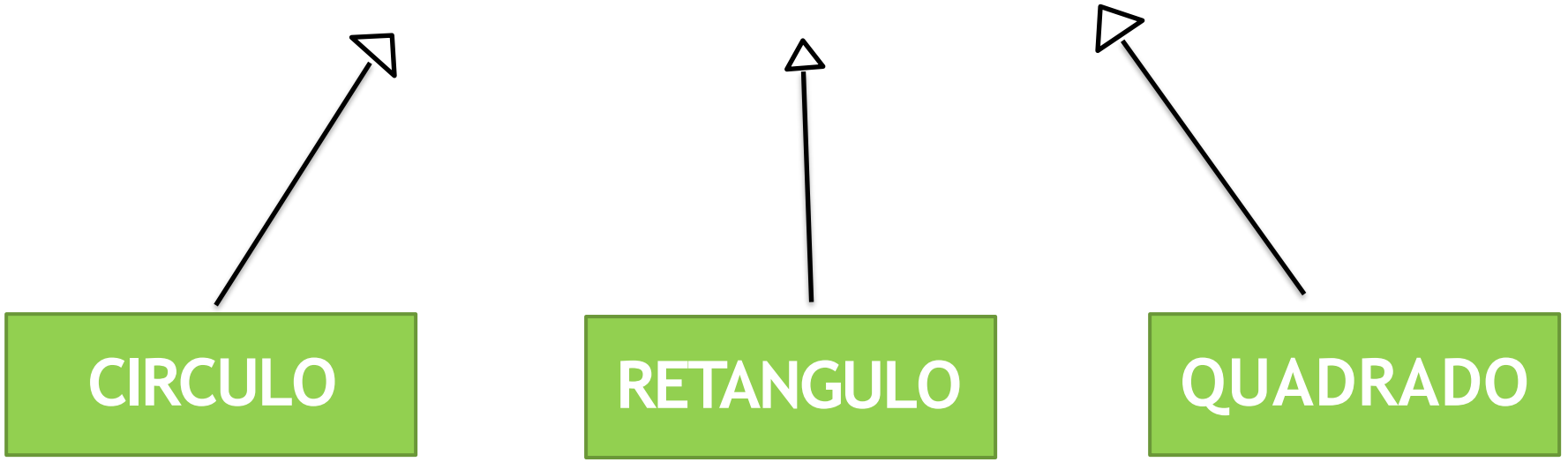
Classes Abstratas
Ex 2

```
public abstract class Forma {

    public abstract void desenhar();

    public abstract void informacoes();

    public void teste(){
        System.out.println("Testando!!");
    }
}
```



Classes Abstratas
Ex 3

Imagine que você está desenvolvendo um jogo que tem diversos personagens. Cada personagem possui um nome, uma vida, um nível e um conjunto de habilidades. Porém, cada personagem tem suas próprias particularidades, como por exemplo, um mago pode lançar feitiços, um guerreiro pode usar armas, e assim por diante.

Sendo assim, crie uma classe abstrata chamada Personagem que define as propriedades comuns a todos os personagens, e um método abstrato UsarHabilidade que será implementado pelas classes derivadas, de acordo com as habilidades de cada personagem

Classes Abstratas
Ex 3

```
public abstract class Personagem
{
    public string Nome { get; set; }
    public int Vida { get; set; }
    public int Nivel { get; set; }

    public abstract void UsarHabilidade();
}
```

```
public class Mago : Personagem
{
    public override void UsarHabilidade()
    {
        Console.WriteLine("O mago lançou um feitiço!");
    }
}
```

```
public class Mago : Personagem
{
    public override void UsarHabilidade()
    {
        Console.WriteLine("O mago lançou um feitiço!");
    }
}
```

Interface

Interfaces

- **Uma interface é parecida com uma classe abstrata, a diferença é que uma classe abstrata pode possuir métodos que não estejam implementados e pode possuir métodos que estejam implementados.**

- **Uma interface possui somente métodos que não estão implementados e que devem ser implementados pela classe que usar a interface.**

Interface

Na programação orientada a objetos, às vezes é útil definir o que uma classe deve fazer, mas não como ela o fará.

Já vimos um exemplo disso: o método abstrato.

Um método abstrato define a assinatura de um método, mas não fornece implementação

Uma subclasse deve fornecer sua própria implementação de cada método abstrato definido por sua superclasse. Portanto, um método abstrato especifica a interface do método, mas não a implementação.

Interface

Embora as classes e métodos abstratos sejam uteis, podemos levar esse conceito um passo adiante.

Em C#, podemos separar totalmente a interface de uma classe de sua implementação usando a palavra-chave interface.

Interface tem objetivo criar um "contrato" onde a Classe que a implementa deve obrigatoriamente obedecer.

Interface

- **É um recurso utilizado para definir os métodos e propriedades de um determinado grupo de classes.**

- **Funcionam como um contrato, e todas as classes que participam do contrato deverão ter aqueles métodos e propriedades.**

- **O funcionamento do método em cada classe pode ser diferente, contanto que o método exista da maneira definida**

Interface

- Dentro das interfaces existem apenas as assinaturas dos métodos e propriedades
- Cabe à classe realizar a implementação das assinaturas, dando comportamento concreto aos métodos
- Mais de uma classe pode implementar a mesma interface
- Uma classe pode implementar mais de uma interface

Interfaces

- **Como o C# não suporta herança múltipla as interfaces permitem que uma classe estenda múltiplas interfaces contornando o problema.**

- **Uma interface no C# não pode conter atributos, somente pode ter métodos, propriedades e eventos. Todos os membros de uma interface são públicos e não podem usar um modificador de acesso.**

Interfaces

- **A classe que implementa a interface deve possuir a definição de todos métodos existentes na interface. Esta definição deve possuir o mesmo nome e a mesma assinatura, retorno e parâmetros, do método na interface.**

- **O nome da classe e o nome da interface são separados por dois pontos(:).**

Interfaces

- **Uma interface define a mesma funcionalidade e comportamento à classes não relacionadas diretamente**
- **Declarando a interface**

```
public interface IProduto
{
    bool EPerecivel { get; }
    Fornecedor RecuperarFornecedor();
    void RegistrarVenda(Cliente cliente);
}
```

Interfaces

- Implementando a interface

```
public class Computador : IProduto
{
	private bool ePerecivel;
	public bool EPerecivel
	{
		get { return ePerecivel; }
	}
	public Fornecedor RecuperarFornecedor()
	{
		return new Fornecedor();
	}
	public void RegistrarVenda(Cliente cliente)
	{
		// Rotina para registrar vendas
	}
}
```

Interfaces

```
if (computador is IProduto)
{
    // ações
}
```

```
IProduto produto = computador as IProduto;

if (produto != null)
{
    Fornecedor fornecedor = produto.RecuperarFornecedor();
}
```

Interfaces

- Pode tornar o comportamento de seus objetos semelhante ao comportamento dos objetos da .NET Framework
- Exemplos:
  - ICollection
  - IComparer
  - IDictionary

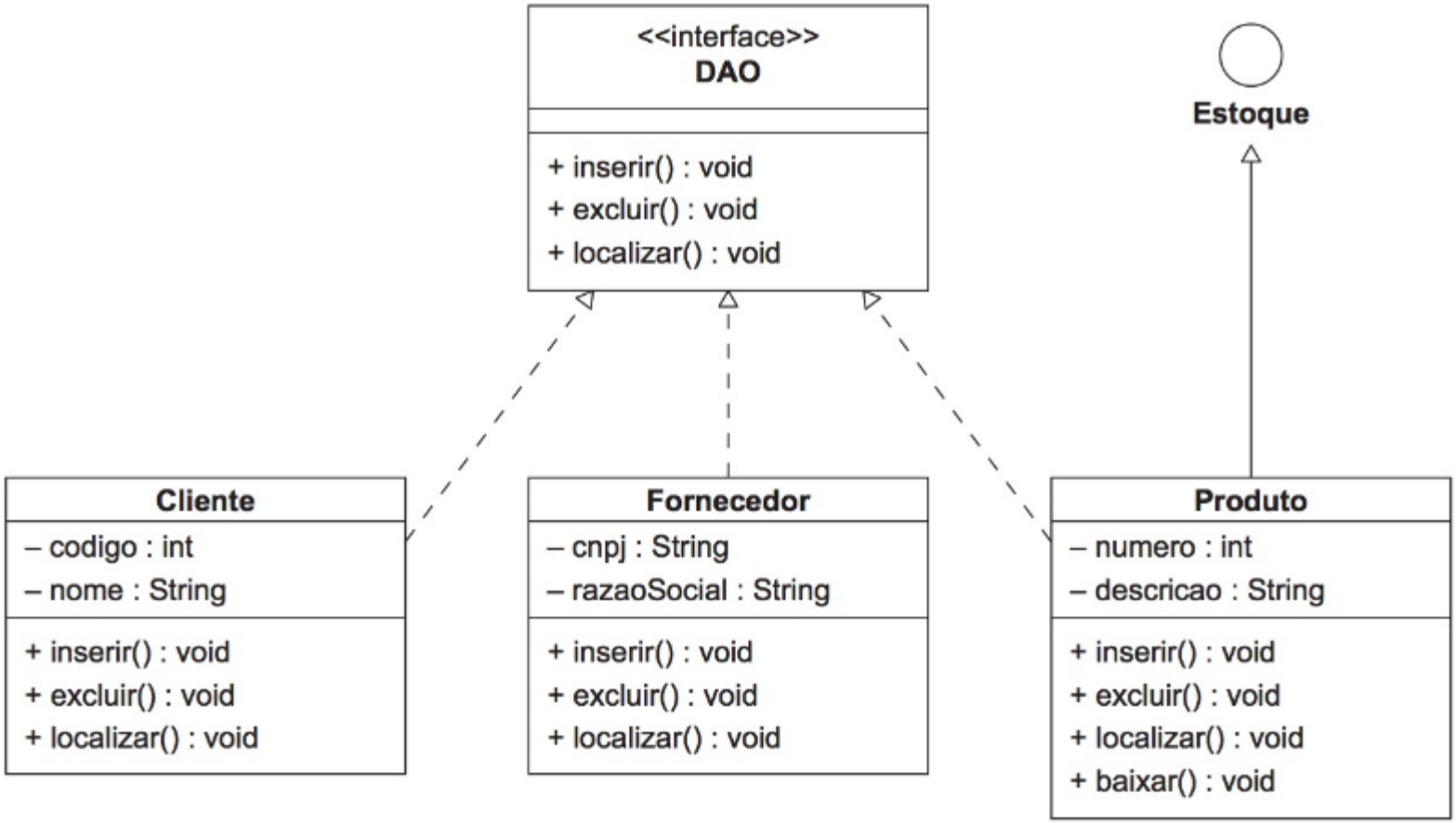
```
public class Cliente : Pessoa, IComparable
{
    ...
}
```

Classes Abstratas
Ex

```
public abstract class Funcionario {
   public abstract double getbonificacao();
}
...
public class Professor: Funcionario{
public double getBonificacao()
   {
      return this.salario * 1,4;
   }
}
```

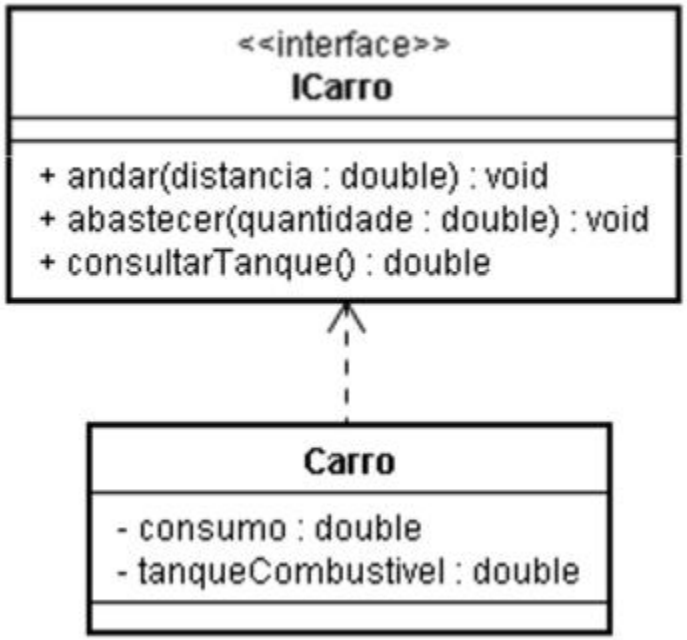
Exercício

- **Implemente o diagrama abaixo**



Exercício

- **Implemente o diagrama abaixo**



REFERÊNCIAS

http://www.hardware.com.br/artigos/programacao-orientada-objetos/

http://www.fontes.pro.br/educacional/materialpag inas/C#/arquivos/jdbc/jdbc.php

http://www.dm.ufscar.br/~waldeck/curso/C#

PORTAL EDUCAÇÃO - Cursos Online : Mais de 900 cursos online com certificado http://www.portaleducacao.com.br/informatica/ar tigos/7852/moderadores-de-acesso#ixzz2AAmxO3JD

http://www.slideshare.net/regispires/C#-08- modificadores-acesso-e-membros-de-classe- presentation

https://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-C#/26366