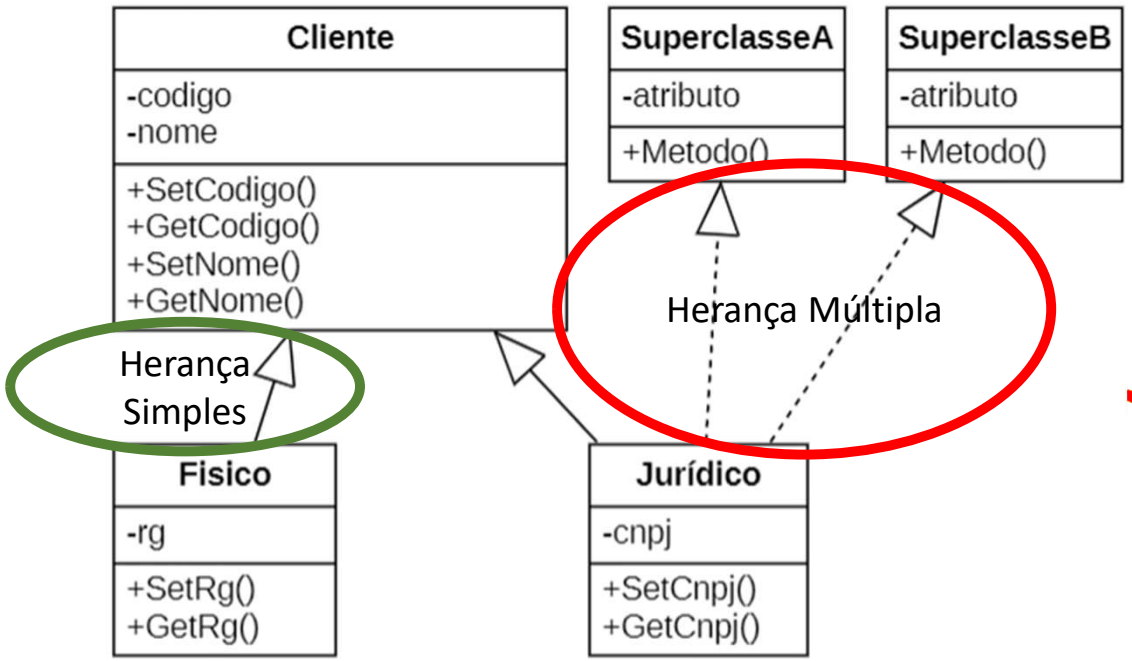
Disciplina: POO - Programação Orientada a Objetos

INTERFACE

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

Herança Simples – C# suporta

Herança Múltipla – C# NÃO suporta, mas a **interface** simula essa situação



C# **não** permite a implementação de herança múltipla

INTERFACE

- É um contrato na forma de uma coleção de declarações de métodos.

- **Quando uma classe implementa uma interface, ela se compromete a implementar todos os métodos declarados nesta.**

- **Interfaces, por natureza, são abstratas**.

Porque utilizar Interfaces?

- Utiliza-se interfaces quando quer classes não relacionadas que implementem métodos similares.

- Através de interfaces, pode-se obter semelhanças (por meio dos métodos) entre classes não relacionadas sem forçar um relacionamento artificial entre elas.

- Outra razão para se utilizar interfaces na programação de objetos, é revelar uma interface de programação de objeto sem revelar essas classes.

Vantagens de implementar interfaces

- Permitem criar sistemas com baixo acoplamento e mais flexível a mudanças

- Torna a vida útil do sistema, porque enquanto as definições da interface não forem alteradas, incluir novos recursos ou re-implementar recursos já existentes decorrentes de mudanças nas regras de negócio não irá 'quebrar' o sistema já existente

- A realização de testes unitários fica mais fácil e o código ganha mais qualidade

INTERFACE

- Pode-se utilizar uma interface como tipo de dados.

- Pode-se utilizar interfaces como mecanismo alternativo para herança múltipla, que permite às classes, ter mais de uma superclasse.

- A herança múltipla não é permitida implementar em C#.

INTERFACE

- As interfaces **não fornecem implementação** para os tipos que elas definem.
- Uma interface define somente **especificações** de métodos (assinaturas).
- **Todos os métodos** são implicitamente **abstratos** e não tem nenhum corpo.
- **Não é possível instanciar** diretamente uma **interface** e criar um membro do tipo da interface

INTERFACE

- **Uma classe deve implementar uma interface** para fornecer os corpos (lógica) de métodos necessários

- As classes só podem herdar membros de uma única superclasse, **mas podem implementar qualquer números de interfaces**

Interface x Classe Abstrata

- A principal diferença entre uma **interface** e uma **classe abstrata** é que a classe abstrata pode ter métodos implementados (reais/concretos) ou não implementados (abstratos).

- Na interface, todos os métodos são **obrigatoriamente abstratos e públicos**.

Interface x Classe

- Uma característica comum entre uma interface e uma classe é que ambas são tipos.

- Isto significa que uma interface pode ser usada no lugar onde uma classe é esperada.

- A declaração de uma interface é praticamente igual a de uma classe, porém utiliza-se a palavra interface ao invés de class .

```
interface Teste
{
   // código da interface
}
```

INTERFACE

- A convenção de nomes do C# para uma interface é seguir a mesma convenção de nomenclatura de classes, porém com um **I** no começo do nome:

```
interface ITeste
{
   double CalculaAlgo(); //assinaturas do métodos
}
```

INTERFACE

- Como a interface ITeste declara o método CalculaAlgo(), toda classe que assina a interface é obrigada a dar uma implementação para essa funcionalidade, **se não implementar o método da interface**, a classe **não compilará**.

Superclasse
Classe Base

Primeiro a superclasse
depois as interfaces

```
// Arquivo Assalariado.cs
public class Assalariado : Funcionario , ITeste
{
    // Implementação dos métodos de Assalariado
    //...
    // método que é obrigado a implementar por essa
    classe, que veio da interface
    public double CalculaAlgo()
    {
        return Salario * 0.02;
    }
```

No método da interface, **não pode** utilizar a palavra override

INTERFACE

- Uma classe pode assinar uma interface sem herdar de outra classe

// Arquivo Produto.cs

```
public class Produto : ITeste
{
    // Implementação dos métodos de Produto
    //...
    // método que é obrigado a implementar, que veio da interface
    public double CalculaAlgo()
    {
        return preco + 50;
    }
}
```

INTERFACE

- Por exemplo, dadas a classe Pessoa e a interface PessoaInterface, as seguintes declarações são válidas:

```
PessoaInterface pi = new Pessoa();
Pessoa pessoa = new Pessoa();
```

- Entretanto, não se pode criar uma instância de uma interface sem implementá-la:

```
PessoaInterface pi = new PessoaInterface(); //ERRO DE COMPILAÇÃO !!!
```

- Outra característica comum é que ambas, interfaces e classes, podem definir métodos, embora uma interface não possa tê-los implementados.

Várias Interfaces

- A classe pode apenas **herdar uma única superclasse**, mas pode implementar **diversas** interfaces.

- Exemplo:

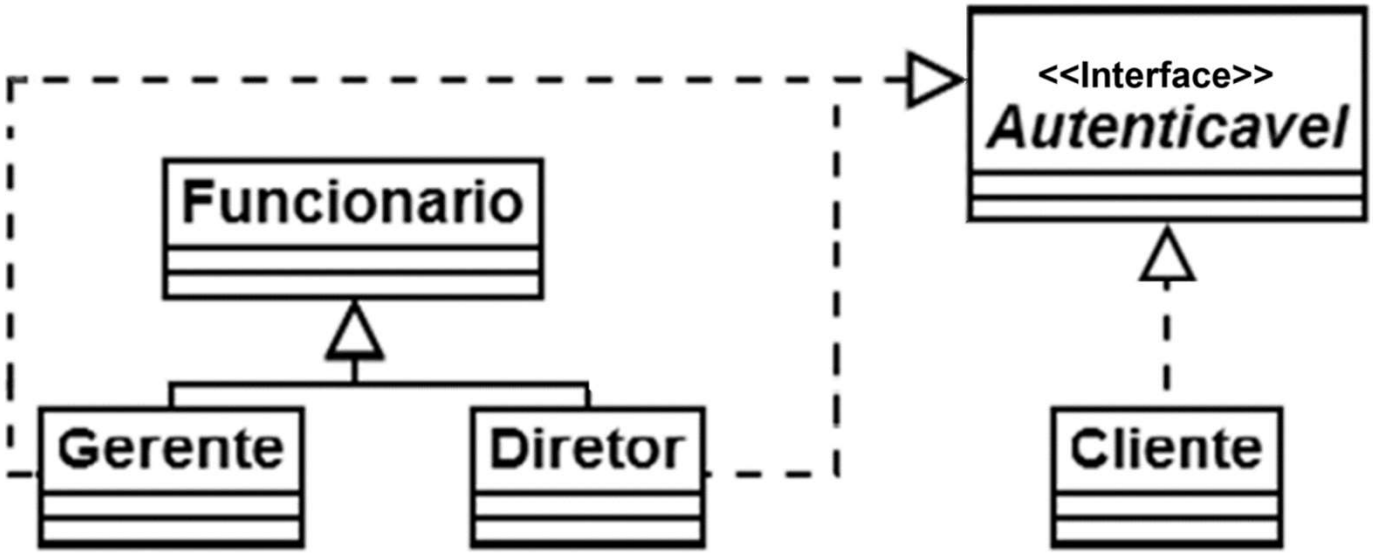
```
public class Pessoa: IPessoaInterface,IEmpresaInterface,IDeptoInterface
{
      //algumas linhas de código
}
```

Exemplo

```
interface IMaquina{
    void Ligue(); //assinatura de método
    void Desligue(); //assinatura de método
}
public class Liquidificador : IMaquina {
   private bool ligado;
   public void Ligue(){
            ligado= true;
   }
   public void Desligue(){
            ligado= false;
   }
}
```

```
public class Microondas : IMaquina{
   private bool ligado;
   public void Ligue(){
            ligado= true;
            acendaLuz();
            inicieMotor();
   }
   public void Desligue(){
            apagueLuz();
            pareMotor();
            ligado= false;
   }
}
```

Exercício




```
interface IAutenticavel
{
        bool Autentica(int senha);
}
```

Superclasse
Classe Base

Interface

```
public class Diretor: Funcionario , IAutenticavel
{
   private int senha;
   // outros atributos e métodos pertinentes a um diretor
   public bool autentica(int senha) {
      if (this.senha != senha)
         return false;
      else {
         Console.WriteLine("É um diretor e tem determinado
         tratamento.");
         return true;
      }
   }
}
```

```
public class Gerente: Funcionario, IAutenticavel
{
   private int senha;

   //outros atributos e métodos pertinentes a um gerente
   public bool autentica(int senha) {
      if (this.senha != senha)
          return false;
      else {
         Console.WriteLine("É um gerente e tem determinado
          tratamento.");
         return true;
      }
   }
   public double Senha
   {
    set  {senha = value;)
   }
}
```

```
public class TestaFuncionario {
 public static void main(String[] args){
     //ler dados dos Gerente

     Autenticavel t = gerente;
     //ler uma senha
     //apresentar o resultado(t.autentica(senha)

  }
}
```

Exercício

- Crie uma interface chamada **I**tributavel, nela terá o método
  double CalculaTributos();
- Este método não recebe nenhum parâmetro e devolve um double que representa o valor do imposto que deve ser pago
- Crie a classe chamada de **ContaCorrente** que implemente a interface **I**tributavel, que pagará 5% de seu saldo como imposto
- Crie outra classe chamada de **SeguroDeVida** e faça com que essa classe implemente a interface **I**tributavel, o método CalculaTributos do SeguroDeVida deve devolver um valor constante de 75 reais
- Faça instâncias de cada classe e teste no método main()

Continuação do exercício

- Crie outra classe chamada TotalizadorDeTributos , que será responsável por acumular os impostos das diferentes classes tributáveis, crie uma instância e teste no método main().

```
public class TotalizadorDeTributos
{
      public double Total { get; private set; }

      public void Adiciona(ITributavel t)
      {
            this.Total += t.CalculaTributos();
      }
}
```