Universidade Federal de Ouro Preto

CSI032 - Programação de Computadores II

Interface

Professor: Dr. Rafael Frederico Alexandre
Contato: rfalexandre@decea.ufop.br
Colaboradores: Eduardo Matias Rodrigues
Contato: eduardo.matias@aluno.ufop.edu.br

Tópicos

Tópicos

Interface
[Caelum, 2019]

1. Toda classe em Java define 2 itens:
   - o que uma classe faz (as assinaturas do métodos);
   - como uma classe faz essas tarefas (o corpo dos métodos e atributos privados);
2. Podemos criar um "contrato"que define o que uma classe deve fazer se quiser ter um determinado status.

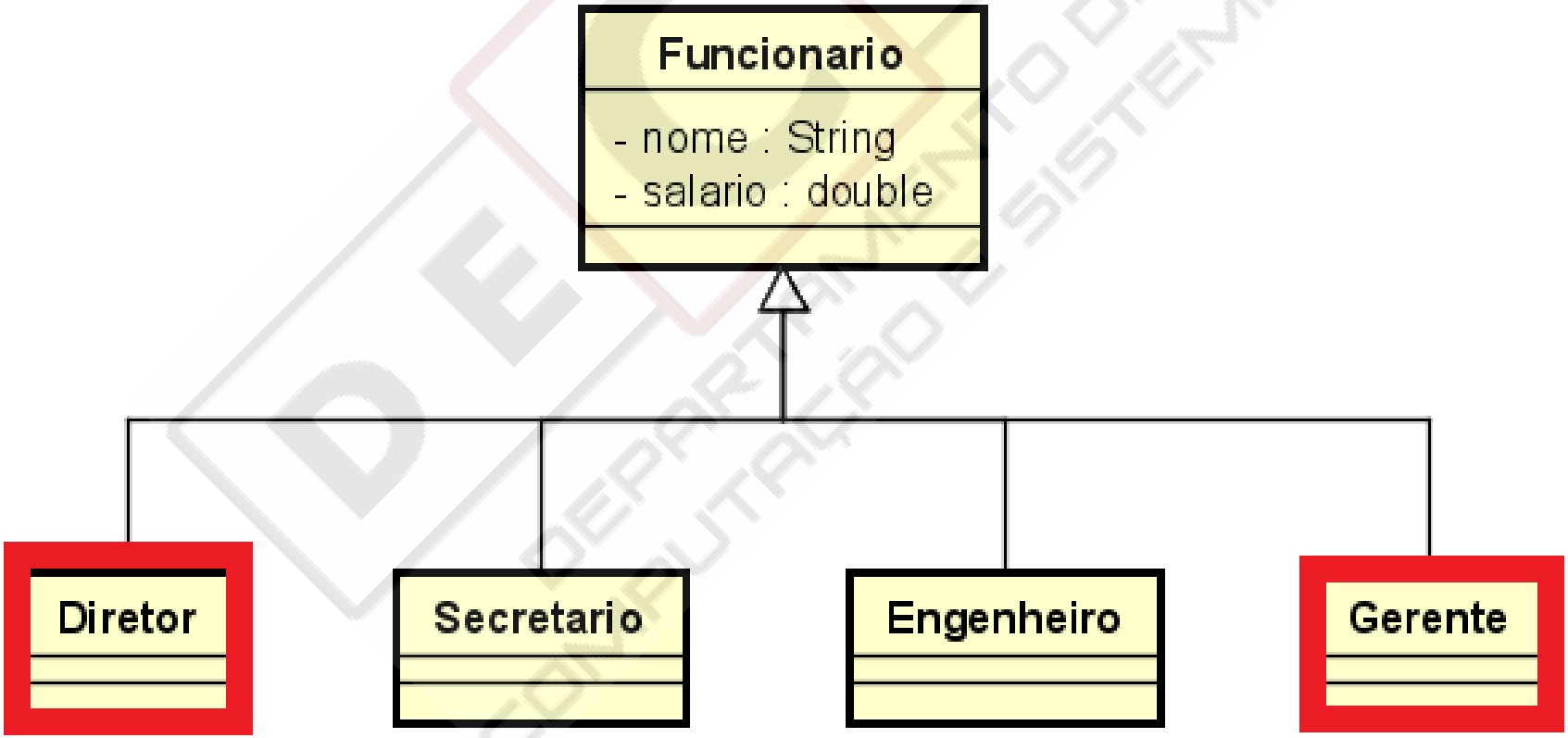
Interface
Introdução

1. As interfaces em Java padronizam as formas em que elementos como sistemas e pessoas podem interagir entre si;
2. Esse conceito é uma abstração do mundo real, ele está presente de várias formas em nosso dia a dia, exemplo:
    1. Os controles do rádio definem a interface entre o usuário e os componentes internos;
        1. Permite que seja realizado um conjunto restritor de operações: ligar/desligar o rádio, aumentar/diminuir o volume, alterar o espectro de frequencia ...
        2. cada rádio implementa esses requisitos (contratos) de forma diferente.

Tópicos

Interface
Problema

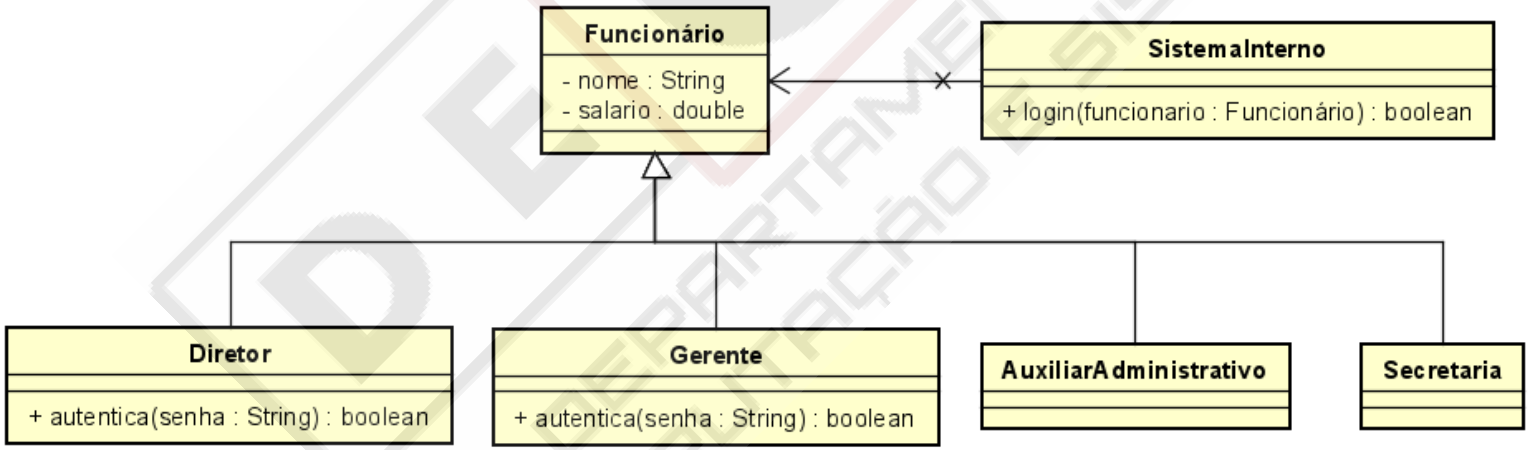
- Suponha um sistema de controle bancário em que apenas Gerentes e Diretores podem acessá-lo.



Interface

Problema, continuação...

1. Tentaremos resolver o problema apenas com o que sabemos até agora;
2. Solução 1: criar uma classe SistemaInterno.



Interface

Solução 1, continuação...

1. As classes que representam funcionários autenticáveis (Diretor e Gerente) possuem um método autentica(String senha), que retorna **true** se a autenticação for bem sucedida e **false** caso contrário;
2. O Sistema interno possui um método login(Funcionario funcionario) que chama o método autentica() do funcionário recebido
3. Obs: cada funcionário autenticável pode ter sua própria maneira de se autenticar, logo, a implementação do método autentica() em Gerente pode ser diferente de uma implementação no Diretor.

Interface

Solução 1: Classe Diretor

```
public class Diretor extends Funcionario {

    private String senha = "Diretor";

    public Diretor(String nome, double salario) {
        super(nome, salario);
    }

    public boolean autentica(String senha) {
        if (this.senha.equals(senha)) {
            return true;
        }
        return false;
    }

}
```

Interface

Solução 1: Classe Gerente

```
public class Gerente extends Funcionario {

    private String senha = "Gerente";

    public Gerente(String nome, double salario) {
        super(nome, salario);
    }

    public boolean autentica(String senha) {
        if (this.senha.equals(senha)) {
            return true;
        }
        return false;
    }
}
```

Interface

Solução 1: Classe Secretária

```
public class Secretaria extends Funcionario {

    public Secretaria(String nome, double salario) {
        super(nome, salario);
    }

    // Não autenticável //
}
```

Interface

Solução 1: Classe SistemaInterno

1. Como nem todo funcionário é autenticável, não podemos chamar o método autentica com uma variável de referência do tipo Funcionário

```
public class SistemaInterno {

    public boolean login(Funcionario funcionario, String senha) {
        // Nem todo funcionário possui o método autentica(senha)
        // Vai dar erro
        return funcionario.autentica(senha);
    }
}
```

Interface
Solução 2

1. Uma outra solução é criar um método de login() para cada funcionário autenticável

```
public class SistemaInterno {

    public boolean login(Diretor diretor, String senha) {
        return diretor.autentica(senha);
    }

    public boolean login(Gerente gerente, String senha) {
        return gerente.autentica(senha);
    }
}
```

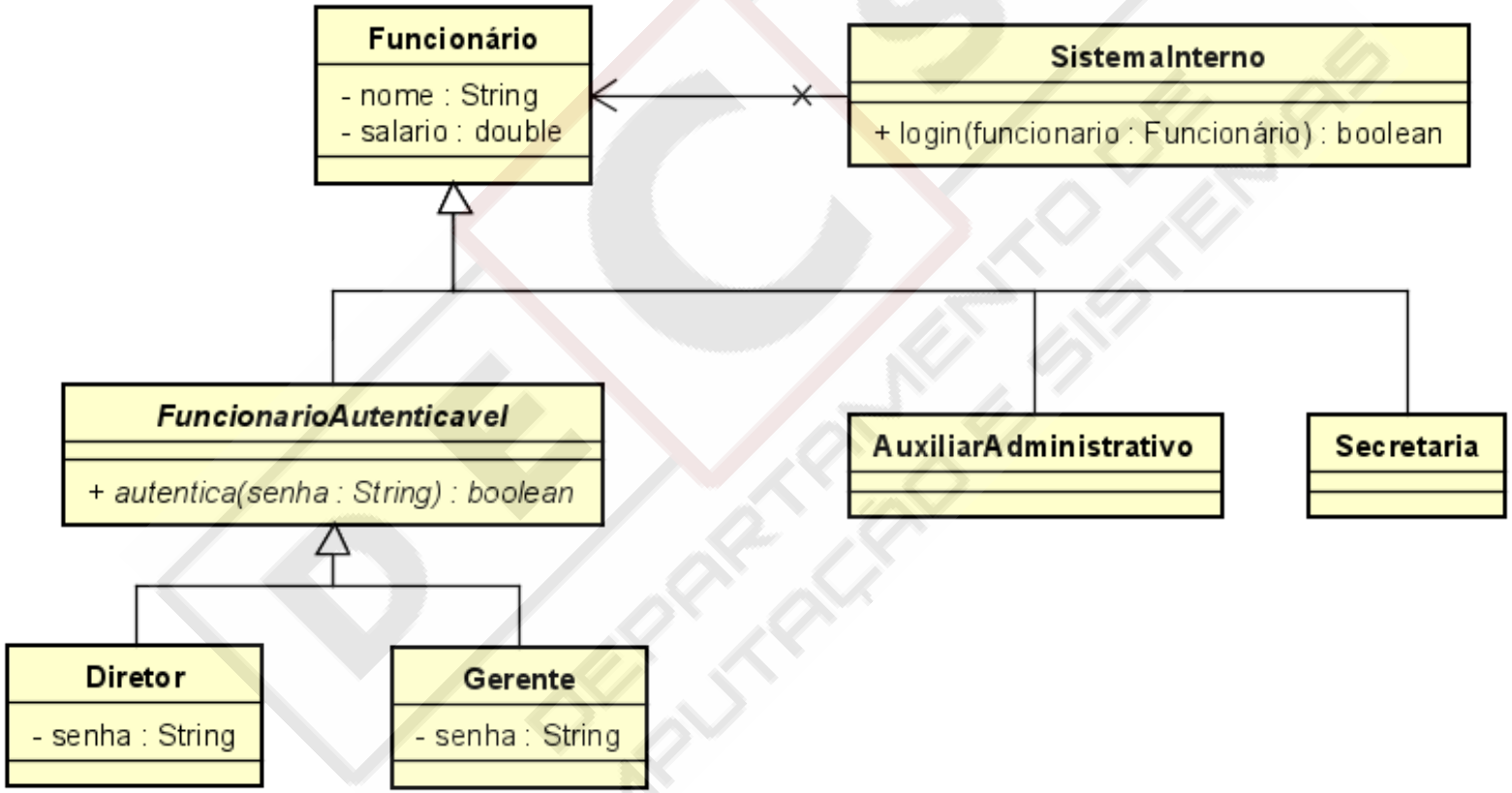
1. Para cada novo tipo de funcionário autenticavel, teríamos que refatorar o código;
    1. Má prática de programação;

Interface
Solução 3

1. Uma maneira mais interessante de se resolver este problema é criar uma classe de funcionários autenticáveis no meio da hierarquia de funcionários;
2. Essa classe deve conter um método abstrato autentica(), assim "suas filhas saberão se autenticar".

Interface

Solução 3, diagrama de classes



Interface

Solução 3, Classe SistemaInterno

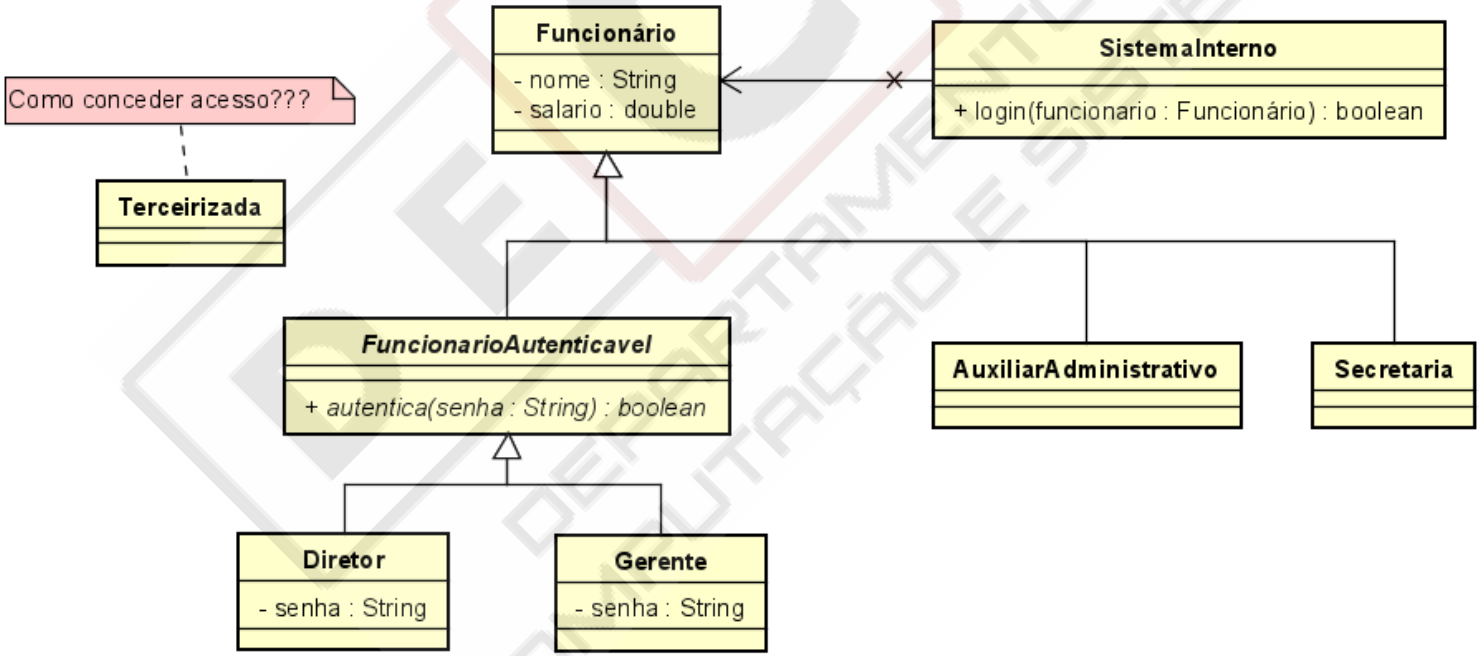
```
public class SistemaInterno {

    public boolean login(FuncionarioAutenticavel funcionario, String
        return funcionario.autentica(senha); // Funciona
    }
}
```

Interface

Solução 3: problema

1. Imagine agora que além de determinados funcionários, as empresas terceirizadas do banco deverão ter acesso ao sistema interno, como resolver o problema?



Interface

Solução 3: problema, continuação ...

1. Poderiamos fazer a classe Terceirizada extender da classe FuncionarioAutenticavel, mas definitivamente uma terceirizada não **é um** funcionário do banco, o que seria uma má prática de programação;
    1. herança deve ser usada apenas se o relacionamento entre classes for **é um**.

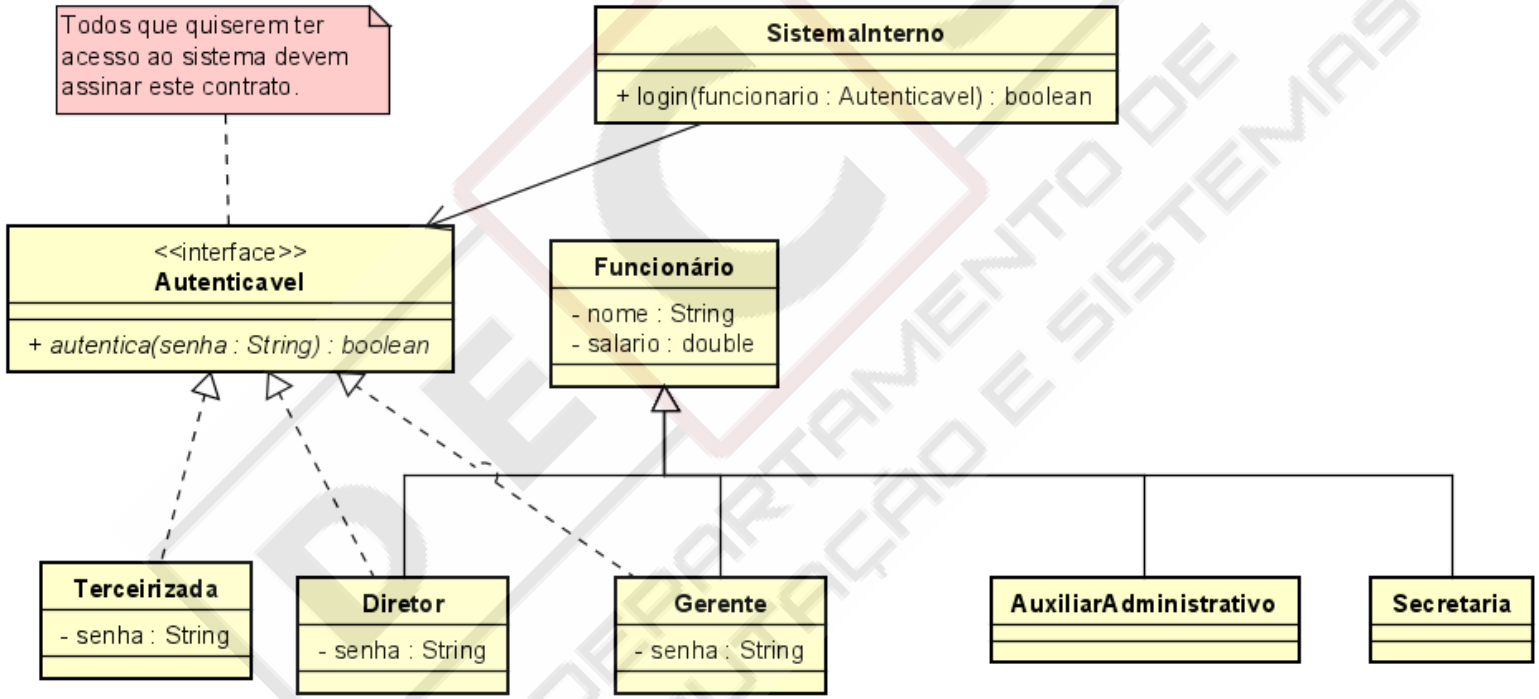
Tópicos

Interface
O contrato

1. Precisamos tratar funcionários, terceirizadas e qualquer outra classe que vier a ter acesso ao sistema da mesma forma na classe SistemaInterno;
    1. Solução: estabelecer um contrato que defina o que uma classe deve fazer se quiser ter determinado status no sistema;
    2. Assim, todas as classes que quiserem ter acesso ao sistema interno deve saber como fazer isso:
        1. Deve ser capaz de autenticar um usuário dada uma determinada senha, devolvendo um booleano indicando o sucesso da operação.

Interface

Interface Autenticavel



Tópicos

Interface
Características

1. **Não** podem ser instânciadas
2. Atributos constantes;
   1. Implicitamente são: **public static final**.
3. Contêm apenas as assinaturas dos métodos;
   1. Implicitamente são: **public abstract**.
4. Só podem ser **implementadas** por classes. Uma classe pode implementar várias Interfaces.
5. Só podem ser herdadas por outra interface

Interface

Criando a interface Autenticavel

```
public interface Autenticavel {

    // Má prática: implicitamente um método já é public abstract
    // public abstract boolean autentica(String senha);

    boolean autentica(String senha);
}
```

Interface

Classe Gerente

```
public class Gerente extends Funcionario implements Autenticavel {

    private String senha = "Gerente";

    public Gerente(String nome, double salario) {
        super(nome, salario);
    }

    @Override
    public boolean autentica(String senha) {
        if (this.senha.equals(senha)) {
            return true;
        }
        return false;
    }
}
```

Interface
Classe Diretor

```
public class Diretor extends Funcionario implements Autenticavel {

    private String senha = "Diretor";

    public Diretor(String nome, double salario) {
        super(nome, salario);
    }

    @Override
    public boolean autentica(String senha) {
        if (this.senha.equals(senha)) {
            return true;
        }
        return false;
    }

}
```

Interface
Classe SistemaInterno

```
public class SistemaInterno {

    public boolean login(Autenticavel funcionario, String senha) {
        return funcionario.autentica(senha);
    }
}
```

1. Podemos referenciar qualquer classe que implemente a Interface Autenticavel com uma variavel do tipo Autenticavel! Assim como faziamos com classes concretas e abstratas.

Interface

Teste

```
public static void main(String[] args) {

    SistemaInterno si = new SistemaInterno();

    Gerente eduardo = new Gerente("Eduardo", 10000);
    Diretor marcela = new Diretor("Marcela", 18000);
    Secretaria fernanda = new Secretaria("Fernanda", 15000);

    Funcionario[] funcionarios = {eduardo, marcela, fernanda};

    for (int i = 0; i < funcionarios.length; i++) {

        Funcionario f = funcionarios[i];
        System.out.println("Nome: " + f.getNome());
        System.out.println("Cargo: " + f.getClass().getSimpleName());

        if (f instanceof Autenticavel) {
            Autenticavel user = (Autenticavel)f;
            if (si.login(user, "Gerente")) {
                System.out.println("Acesso permitido\n");
            } else {
                System.out.println("Acesso negado, senha incorreta\n");
            }
        } else {
            System.out.println("Acesso negado. Funcionário não é autenticável!\n");
        }
    }
}
```

Interface

Saída do programa

```
Nome: Eduardo
Cargo: Gerente
Acesso permitido

Nome: Marcela
Cargo: Diretor
Acesso negado, senha incorreta

Nome: Fernanda
Cargo: Secretaria
Acesso negado. Funcionário não é autenticável!
```

Tópicos

Interface
Herança múltipla

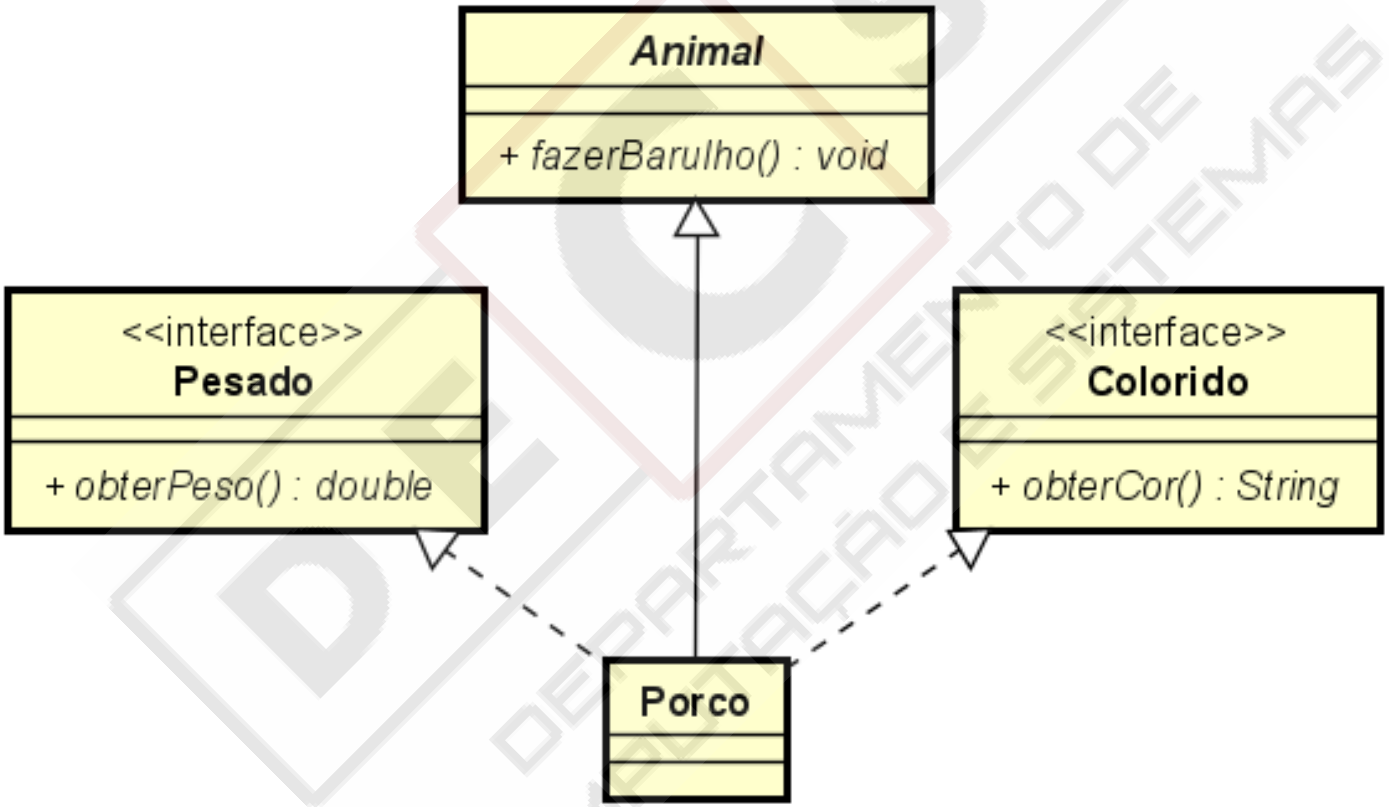
1. Java não oferece suporte a herança múltipla, mas uma classe pode implementar várias Interfaces;
    1. Ainda é necessário implementar os métodos;

Interface

[Malaquias, 2015] Herança múltipla e interface

1. Para nosso exemplo, suponha que possuímos interfaces Pesado e Colorido, uma classe abstrata Animal e queremos implementar uma classe Porco;
   1. Os objetos da classe Porco devem saber seu peso e sua cor, além de serem animais;
   2. Desta forma, podemos herdar diretamente da classe Animal e implementar as duas interfaces.

Cont.
Diagrama de classes



Cont.

```java
public abstract class Animal {
    public abstract void fazerBarulho();
}
```

```java
public interface Colorido {
    String obterCor();
}
```

```java
public interface Pesado {
    double obterPeso();
}
```

1. Podemos nos referir a um porco de 3 formas! Como um animal, colorido ou pesado!

Cont.
Teste

```
public static void main(String[] args) {

    Porco porco = new Porco();

    Animal p2 = new Porco(); // funciona
    Colorido p3 = new Porco(); // funciona
    Pesado p4 = new Porco(); // funciona

    porco.fazerBarulho();
    System.out.println("Peso: " + porco.obterPeso());
    System.out.println("Cor: " + porco.obterCor());
    System.out.println("Enlameado: " + porco.enlameado());
}
```

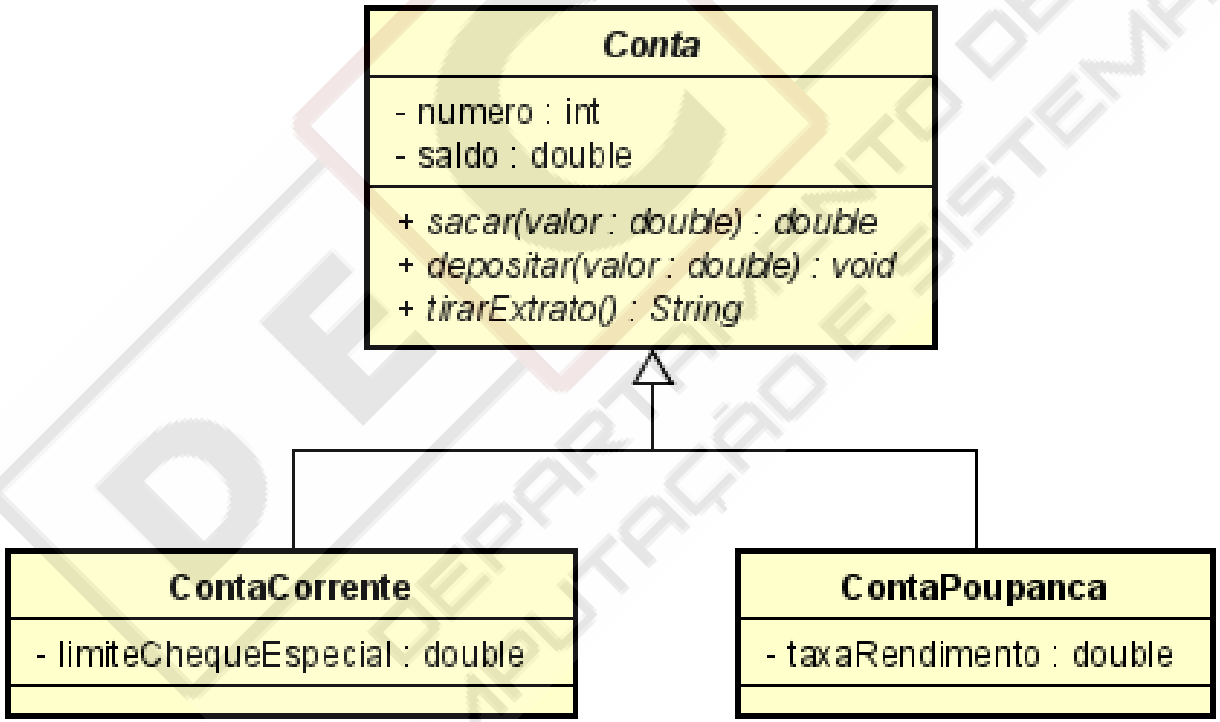
Cont.
Saída do programa

```
oinc oinc oinc oinc
Peso: 45.8
Cor: Rosa
Enlameado: true
```

Tópicos

Exercício
Tributável

1. Implemente o diagrama de classes:



Exercício
Descrição

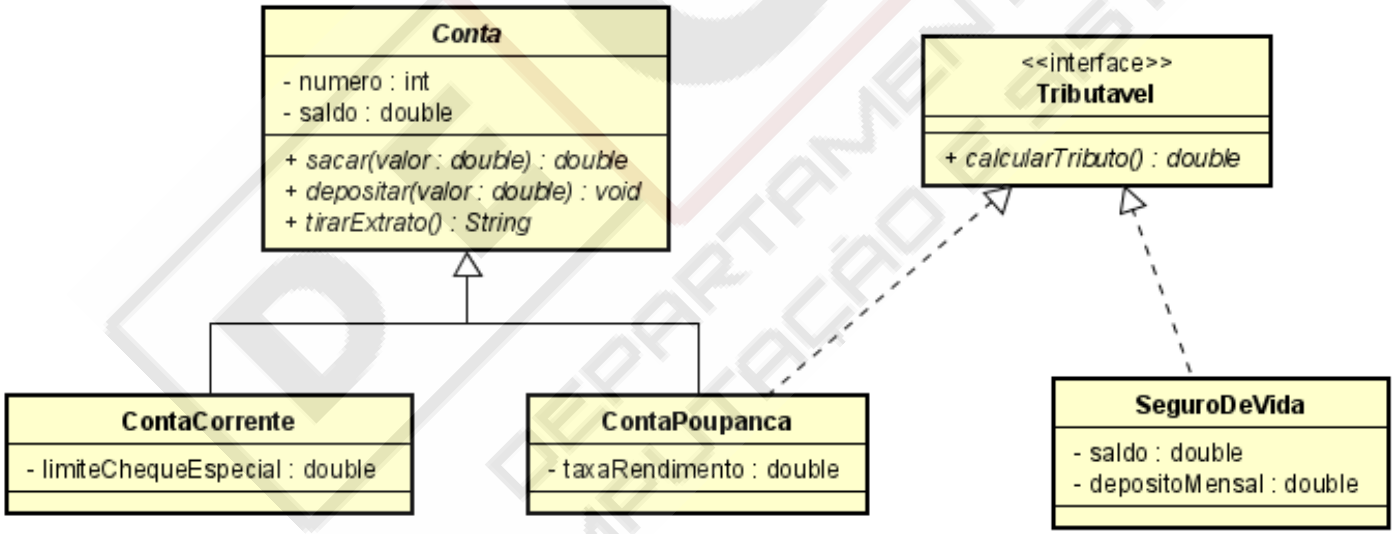
1. Classe Conta
   1. Métodos abstratos sacar(valor), depositar(valor) e tirarExtrato()
2. ContaCorrente
   1. Método sacar deve considerar o limite de cheque especial
   2. método tirarExtrato() deve retornar o saldo em conta acrescido do limite de cheque especial
3. ContaPoupanca
   1. O método tirarExtrato() deve retornar o saldo em conta multiplicado pela taxa de rendimento da conta.

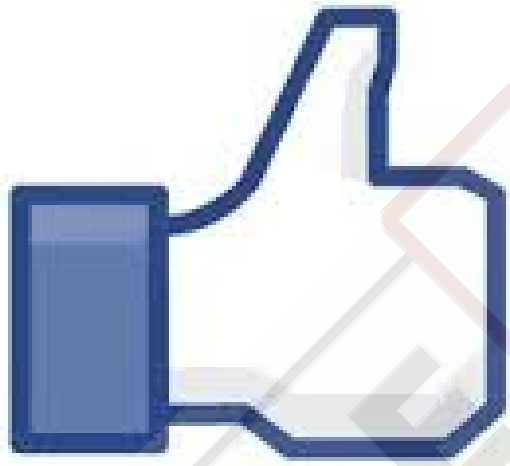
Exercício
Continuação ...

1. Crie uma Interface Tributavel, toda classe tributável deve "saber"calcularTributo();
   1. Contas correntes podem ser tributadas;
   2. O valor do tributo é equivalente a 3% do saldo em conta;
2. Crie uma classe SeguroDeVida, um seguro de vida possui um saldo e uma taxa de depósito mensal. Um seguro de vida também é tributável. O valor do tributo é iqual à 2% do saldo acrescido de 40% do valor do depósito mensal.

Exercício
Tributável

1. Exemplo de diagrama de classes, fique a vontade para fazer as alterações que julgar necessárias. Crie uma classe de teste para o programa.





MUITO OBRIGADO!!!

Referências Bibliográficas I

Caelum (2019).
Java e orientação a objeto.
https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf.

Malaquias, J. R. (2015).
Programação orientada a objetos.
http://www.decom.ufop.br/romildo/2015-2/bcc221/x1-1300012.