PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Milton Palmeira Santana

Anhanguera

Interface

» Imagine que um sistema de controle do banco possa ser acessado pelos diretores do banco, além dos gerentes. Então, teríamos uma classe Diretor:

```
public class Diretor extends Funcionario {

    public boolean autentica(int senha) {
        // Verificar aqui se a senha confere com a recebida como parâmetro.
    }

}
```

Anhanguera

Interface

» Repare que o método de autenticação de cada tipo de Funcionario pode variar muito. Mas vamos aos problemas. Considere o SistemaInterno e seu controle: precisamos receber um Diretor ou Gerente como argumento, verificar se ele se autentica e colocá-lo dentro do sistema.

```
public class SistemaInterno {

    public void login(Funcionario funcionario) {
        // Invocar o método autentica?
        // Não dá! Nem todo Funcionario o tem.
    }
}
```

Anhanguera

Interface

- O SistemaInterno aceita qualquer tipo de Funcionario, tendo ele acesso ao sistema ou não, mas note que nem todo Funcionario tem o método autentica. Isso nos impede de chamar esse método com uma referência apenas a Funcionario (haveria um erro de compilação). O que fazer, então?

```
public class SistemaInterno {

    public void login(Funcionario funcionario) {
        funcionario.autentica(...); // não compila
    }
}
```

Anhanguera

Interface

- Uma possibilidade é criar dois métodos login no SistemaInterno: um para receber Diretor, e outro, Gerente. Já vimos que essa não é uma boa escolha. Por quê?

- Cada vez que criarmos uma nova classe de Funcionario que é autenticável, precisaríamos adicionar um novo método de login no SistemaInterno.

- O que precisamos para resolver nosso problema? Arranjar uma forma de poder referenciar Diretor, Gerente e Cliente de uma mesma maneira, isto é, achar um fator comum.

Anhanguera

Interface

- Se houvesse uma forma na qual essas classes garantissem a existência de um determinado método por meio de um contrato, resolveríamos o problema.

- Toda classe define dois itens:
  - O que uma classe faz (as assinaturas dos métodos);
  - Como uma classe faz essas tarefas (o corpo dos métodos e atributos privados).

Anhanguera

Interface

- Podemos criar um "contrato" o qual define tudo o que uma classe deve fazer se quiser ter um determinado status. Imagine:

- contrato "Autenticavel": Quem quiser ser "Autenticavel" precisa saber: 1. Autenticar uma senha, devolvendo um booleano.

- Quem quiser pode assinar esse contrato, sendo, assim, obrigado a explicar como será feita essa autenticação. A vantagem é que, se um Gerente assinar esse contrato, podemos nos referenciar a um Gerente como um Autenticavel.

Anhanguera

Interface

» Podemos criar esse contrato em Java!

```
public interface Autenticavel {

    boolean autentica(int senha);

}
```

Anhanguera

Interface

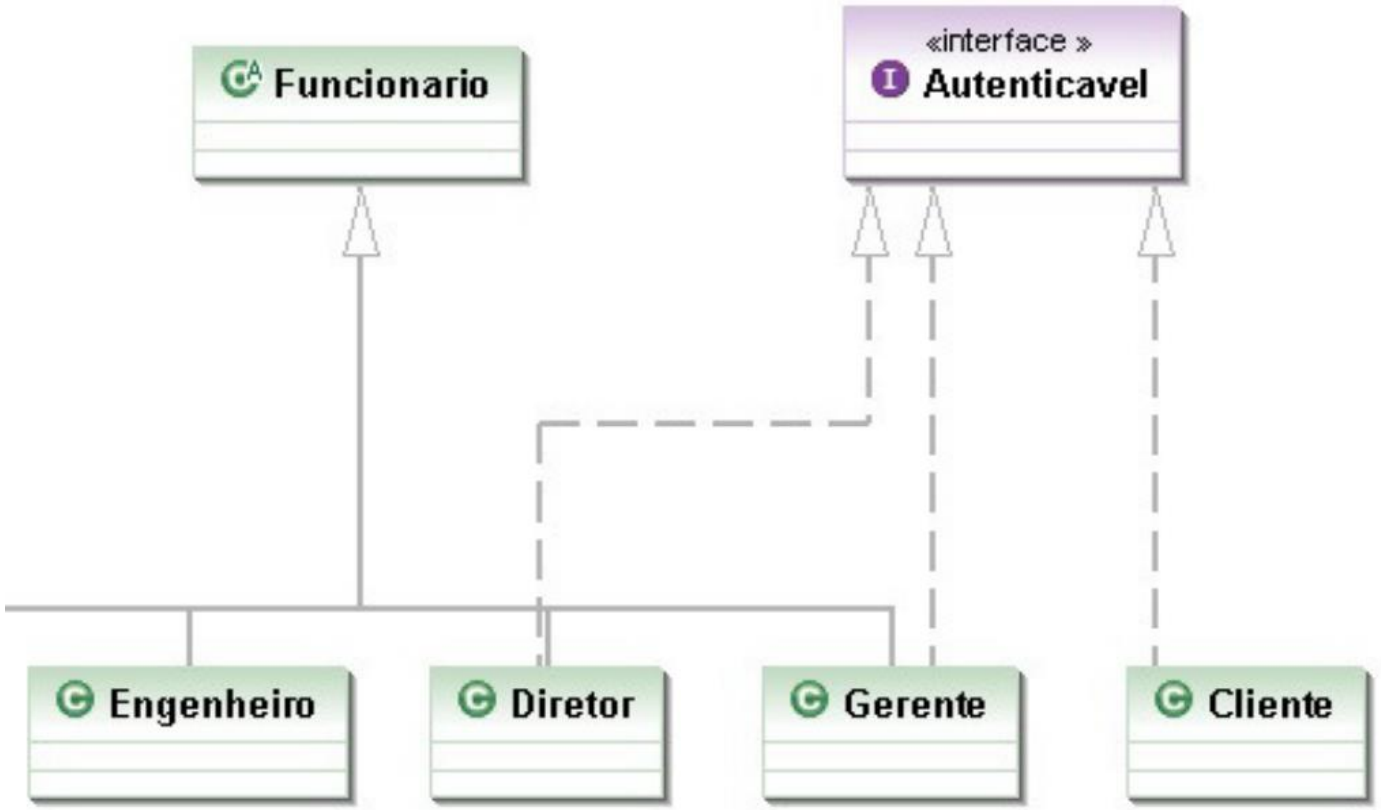
```
public class Gerente extends Funcionario implements Autenticavel {

    private int senha;

    // Outros atributos e métodos.

    public boolean autentica(int senha) {
        if(this.senha != senha) {
            return false;
        }
```

Interface


«interface »
Autenticavel
Funcionario
Engenheiro
Diretor
Gerente
Cliente

Anhanguera

Interface

```
public class SistemaInterno {

  public void login(Autenticavel a) {
     int senha = // Pega senha de um lugar ou de um scanner de polegar.
     boolean ok =    a.autentica(senha);

     // Aqui eu posso chamar o autentica!
     // Não necessariamente é um Funcionario!
     // Além do mais, eu não sei que objeto a
     // referência "a" está apontando exatamente! Flexibilidade.
  }

}
```

Anhanguera

Interface x Herança

- Interface suporta herança múltipla, classes abstratas não.
- Interface permite apenas métodos abstratos.
- Interface não permite atributos.
- Não contém construtor.
- Quando utilizar? O ideal é utilizar a interface quando várias classes diferentes compartilham apenas a assinatura de seus métodos.


Anhanguera