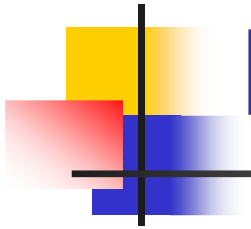
Java Standard Edition (JSE)

Capítulo 08. Interfaces

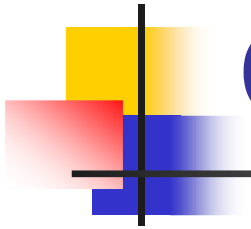
Esp. Márcio Palheta

gtalk: marcio.palheta@gmail.com



Novos recursos a aprender

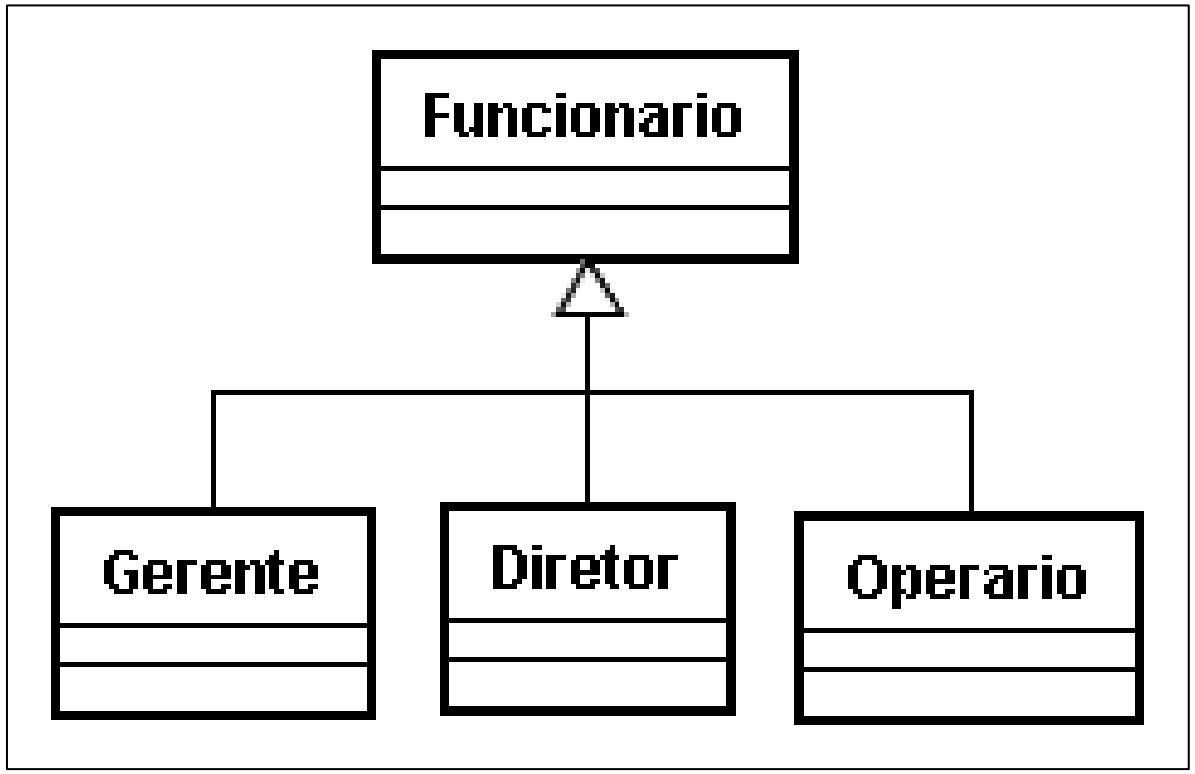
- O que é uma Interface e quando utilizá-la;
- Herança vs Implementação;
- Como criar interfaces em java;
- Como utilizar interfaces criadas;
- Diminuição do acoplamento entre as classes;



Cenário 01: O Diretor

- Imagine que, a partir de agora, o sistema de controle do banco pode ser acessado por Diretores e Gerentes;
- Com isso, deduzimos que Diretor é um tipo de funcionário;
- O método autenticar() de cada funcionário pode variar bastante, a depender de seu cargo;
- Vejamos uma implementação possível:

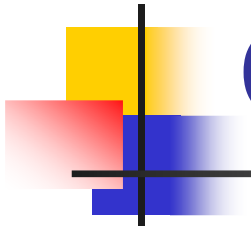
Cenário 01: Representação Gráfica - Herança



Cenário 01: Diretor e Gerente

```
 1 public class Diretor extends Funcionario {
 2     private int senha;
 3
 4⊕    public boolean autenticar(int senha){
 7⊕    public int getSenha() {
10⊕    public void setSenha(int senha) {
13⊕    public double getBonificacao() {
16 }
```

```
 1 public class Gerente extends Funcionario {
 2     private int senha;
 3
 4⊕    public boolean autenticar(int senha){
 7⊕    public int getSenha() {
10⊕    public void setSenha(int senha) {
13⊕    public double getBonificacao() {
16 }
```



Cenário 02: Sessões

- O banco resolveu criar um sistema para controle das sessões de funcionários;
- Para isso, foi criada a classe ControleSessao, responsável por controlar o tempo de conexão(sessão) de cada funcionário;
- A classe possui um atributo que armazena a quantidade de conexões e os métodos login() e logout();

Cenário 02: ControleSessao

- Como implementar o método login, se nem todo funcionário possui o método autenticar()?

```
public class ControleSessao {
    public void login(Funcionario f){
        //Como chamar o método autenticar()???
        //Nem todo Funcionario tem esse método.
    }
}
```

1

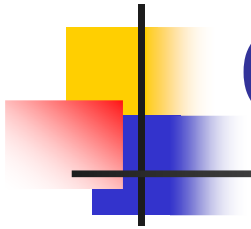
Cenário 02: Alternativas

1

```
void login(Funcionario funcionario) {
    funcionario.autentica();
}
```

2

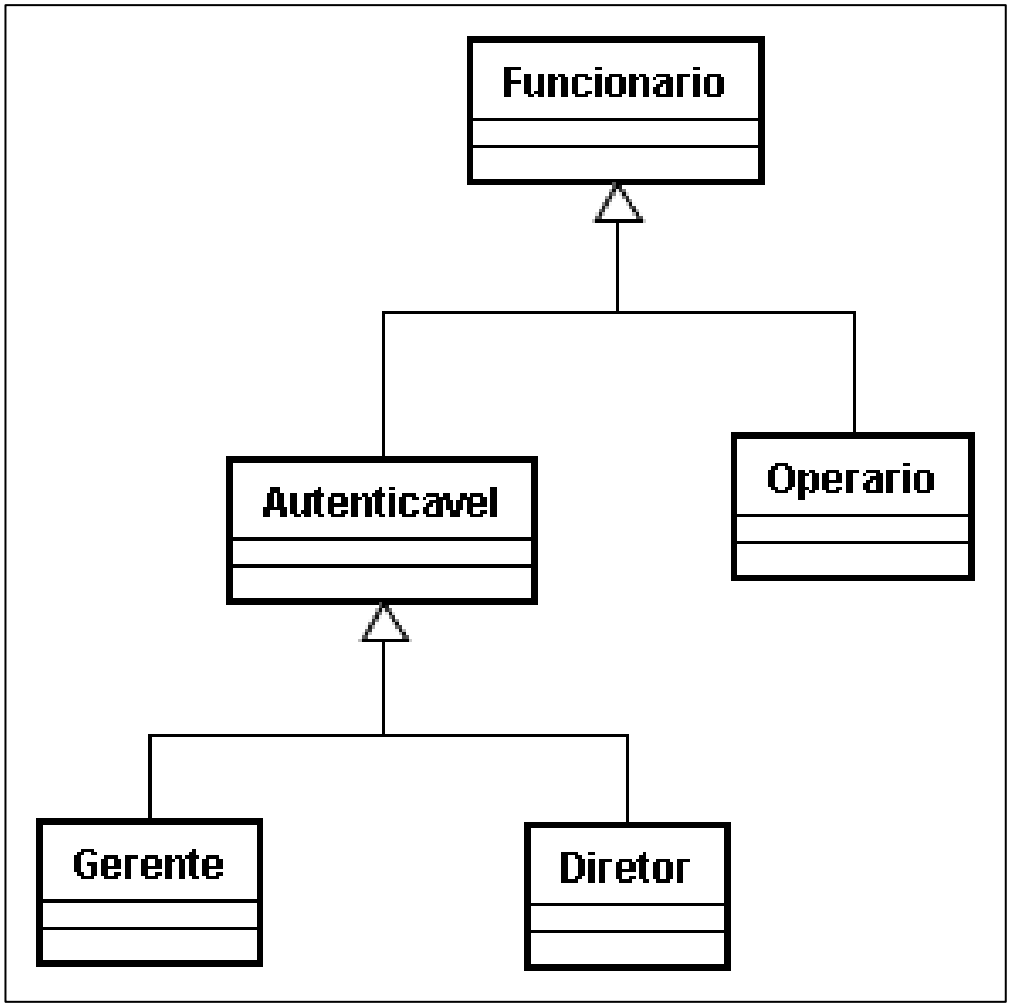
```
void login(Diretor funcionario) {
    funcionario.autenticar();
}
void login(Gerente funcionario) {
    funcionario.autenticar();
}
```

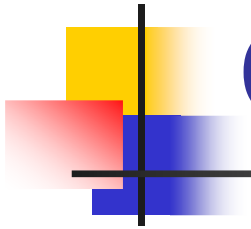


Considerações do modelo:

- A cada nova subclasse de Funcionario que seja "autenticável", precisaremos criar um novo método login();
- Talvez pudéssemos criar uma classe Autenticável, filha de Funcionario e mãe das classes Gerente e Diretor;
- A seguir, é exibido o novo diagrama de classes:

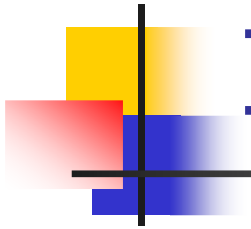
Cenário 02: Outra alternativa





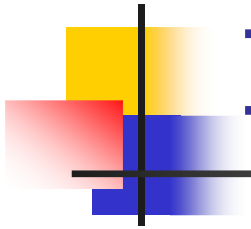
Cenário 02: Um agravante

- No diagrama anterior, a classe Autenticavel herda todas as características de Funcionario;
- E se precisássemos controlar também o login de clientes do banco?
- Cliente deveria possuir o método autenticar();
- Mas seria um Funcionario ?



Interfaces

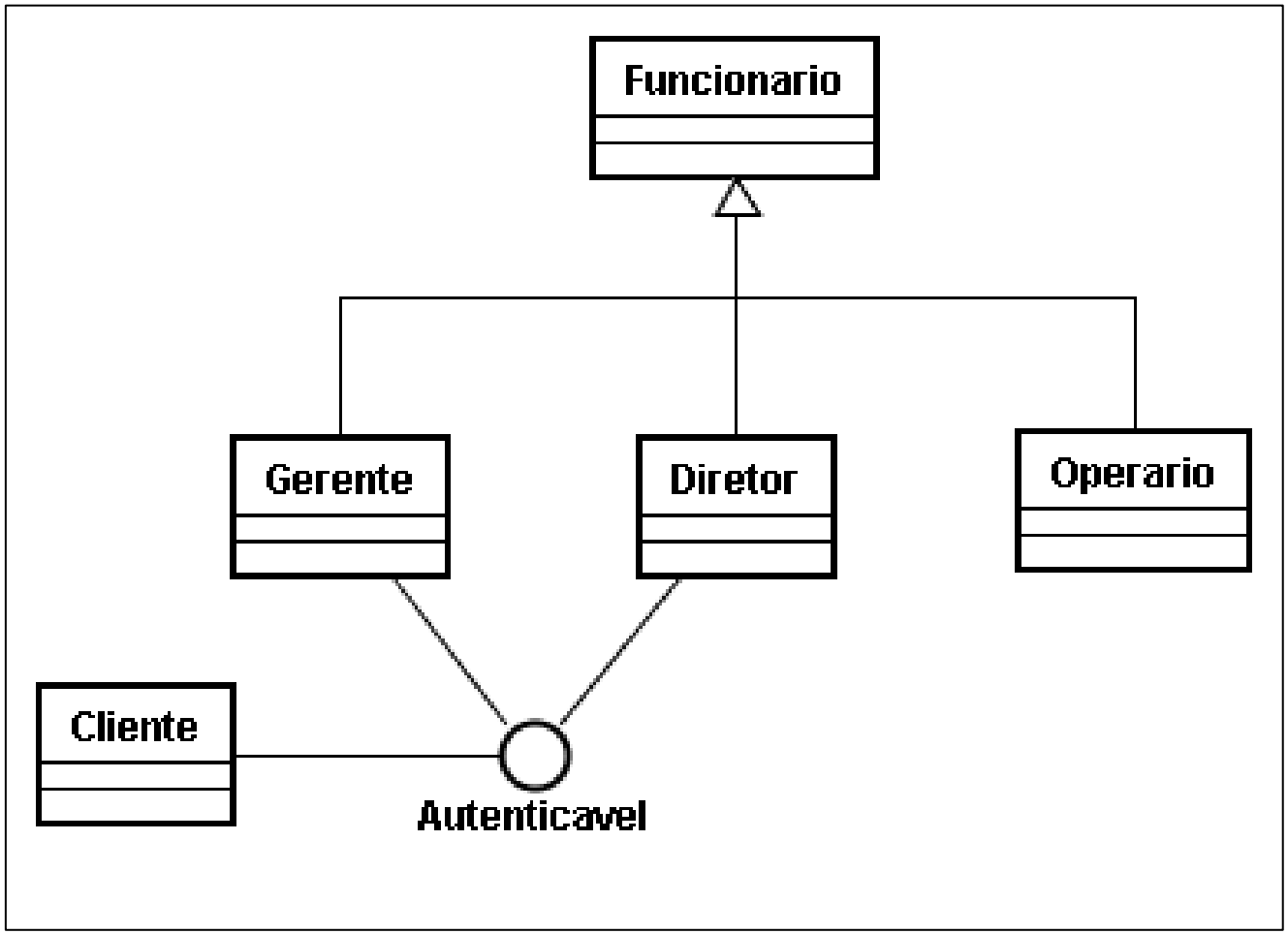
- Precisamos de uma forma de referenciar Diretor, Gerente e Cliente da mesma maneira;
- Podemos criar um "contrato" onde são definidos os termos que devem ser respeitados;
- Uma classe que queira fazer parte do contrato, deve respeitar seus termos;

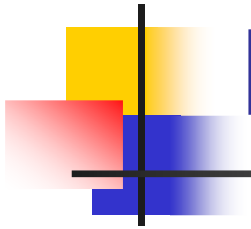


Interfaces – o contrato

- Interfaces definem as regras(métodos) que devem ser seguidos;
- As classes que desejam participar do contrato, devem implementar a interface e seus métodos;
- Como ficaria nosso diagrama de classes, com a inclusão da interface Autenticavel?

Interface - Autenticavel





Declaração da interface

- A interface é um contrato onde, quem assina, se responsabiliza por implementar seus métodos;
- Mostra o que deve ser feito, não como fazer;

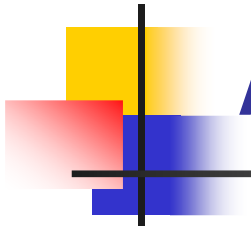
```
1 public interface Autenticavel {
2
3     public boolean autenticar(int senha);
4
5 }
```

Classes que implementam a interface Autenticavel;

```
1  public class Diretor extends Funcionario implements Autenticavel{
2
3      private int senha;
4
5      public boolean autenticar(int senha) {
6          return this.senha == senha;
7      }
8
9      public double getBonificacao() {
10         return getSalario() * 0.20;
11     }
12
13     public int getSenha() {
16     public void setSenha(int senha) {
19 }
```

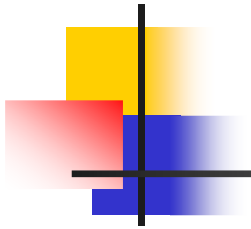
1

2

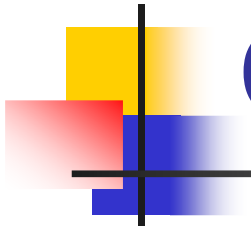


A classe ControleSessao

```
1  public class ControleSessao {
2
3      private static int qtdeSessao;
4
5      public void autenticar(int senha, Autenticavel usuario){
6          if(usuario.autenticar(senha)){
7              qtdeSessao++;
8          }
9      }
10
11     public static int getQtdeSessao() {
12         return qtdeSessao;
13     }
14 }
```

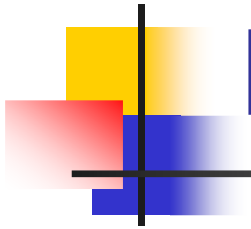


- Herança de interfaces;
- Todos os métodos são abstratos;
  - Implementação de todos os métodos;



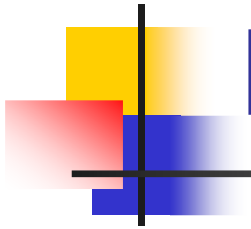
Considerações finais

- O uso de interfaces garantiu o uso do polimorfismo, uma vez que passamos a chamar Diretor e Cliente de Autenticavel;
- Há relacionamento entre classes abstratas e interfaces?
- Quando escolher entre interface ou classe abstrata?



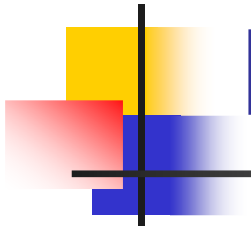
Exercício 01

- Crie o projeto capitulo08;
- Implemente o diagrama de classes do slide 14;
- Implemente a classe ControleSessao;
- Crie uma classe teste, onde são criados N objetos Diretor, Gerente e Cliente;
- Execute login da classe ControleSessao;
- Imprima o total geral de conexões;
- Imprima o total de conexões por tipo;



Bibliografia

- Java - Como programar, de Harvey M. Deitel
- Use a cabeça! - Java, de Bert Bates e Kathy Sierra
- (Avançado) Effective Java Programming Language Guide, de Josh Bloch



Referências WEB

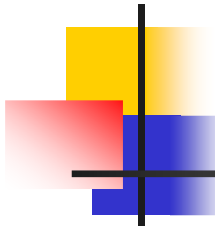
- SUN: www.java.sun.com

Fóruns e listas:

- Javaranch: www.javaranch.com
- GUJ: www.guj.com.br

Apostilas:

- Argonavis: www.argonavis.com.br
- Caelum: www.caelum.com.br



Java 2 Standard Edition (J2SE)

Capítulo 08. Interfaces

Esp. Márcio Palheta

gtalk: marcio.palheta@gmail.com