

Unidade V:

Tipos Abstratos de Dados Básicos


Nativos do Java



PUC Minas

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

- Interface em Java
- Classes *ArrayList*, *Vector* e *LinkedList*
- Classe *Stack*
- Classe *Queue*

- **Interface em Java** 
- Classes *ArrayList*, *Vector* e *LinkedList*
- Classe *Stack*
- Classe *Queue*

- Define e padroniza como coisas distintas interagem entre si
- Permite que classes distintas tenham um conjunto comum e padronizado de métodos (e.g., os cálculos de pagamentos nas classes Funcionario e Mercadoria)

Exemplo de Interface

- A interface Autenticavel tem o método autentica e a classe Geração estende Funcionário e implementa Autenticavel

```
public interface Autenticavel {  
    // ...  
    public abstract boolean autentica(int senha);  
}
```

```
class Gerente extends Funcionario implements Autenticavel {  
    private int senha;  
    // ...  
    public boolean autentica(int senha) {  
        return (this.senha == senha && /* verificar departamento */);  
    }  
}
```

Exemplo de Interface

- A interface Autenticavel tem o método autentica e a classe Geração estende Funcionário e implementa Autenticavel

```
public interface Autenticavel {  
    // ...  
    public abstract boolean autentica(int senha);  
}
```

Inicia com a palavra-chave
interface



```
class Gerente extends Funcionario implements Autenticavel {  
    private int senha;  
    // ...  
    public boolean autentica(int senha) {  
        return (this.senha == senha && /* verificar departamento */);  
    }  
}
```

Exemplo de Interface

- A interface Autenticavel tem o método autentica e a classe Geração estende Funcionário e implementa Autenticavel

```
public interface Autenticavel {  
    // ...  
    public abstract boolean autentica(int senha);  
}
```

Atributos públicos,
estáticos e finais

```
class Gerente extends Funcionario implements Autenticavel {  
    private int senha;  
    // ...  
    public boolean autentica(int senha) {  
        return (this.senha == senha && /* verificar departamento */);  
    }  
}
```

Declaração de Interfaces

-
- Seus
- Seus métodos são públicos e abstratos. Um método abstrato é declarado com a palavra *abstract* e não tem implementação
- Não especifica qualquer detalhe de implementação

Utilizando Interfaces

- Para utilizar uma interface, uma classe concreta deve especificar que ela implementa a interface e declarar cada método na interface com a assinatura especificada na declaração de interface
- Para especificar que uma classe implementa uma interface, adicione a palavra-chave `implements` e o nome da interface ao final da primeira linha da declaração de classe

Implementando Interfaces

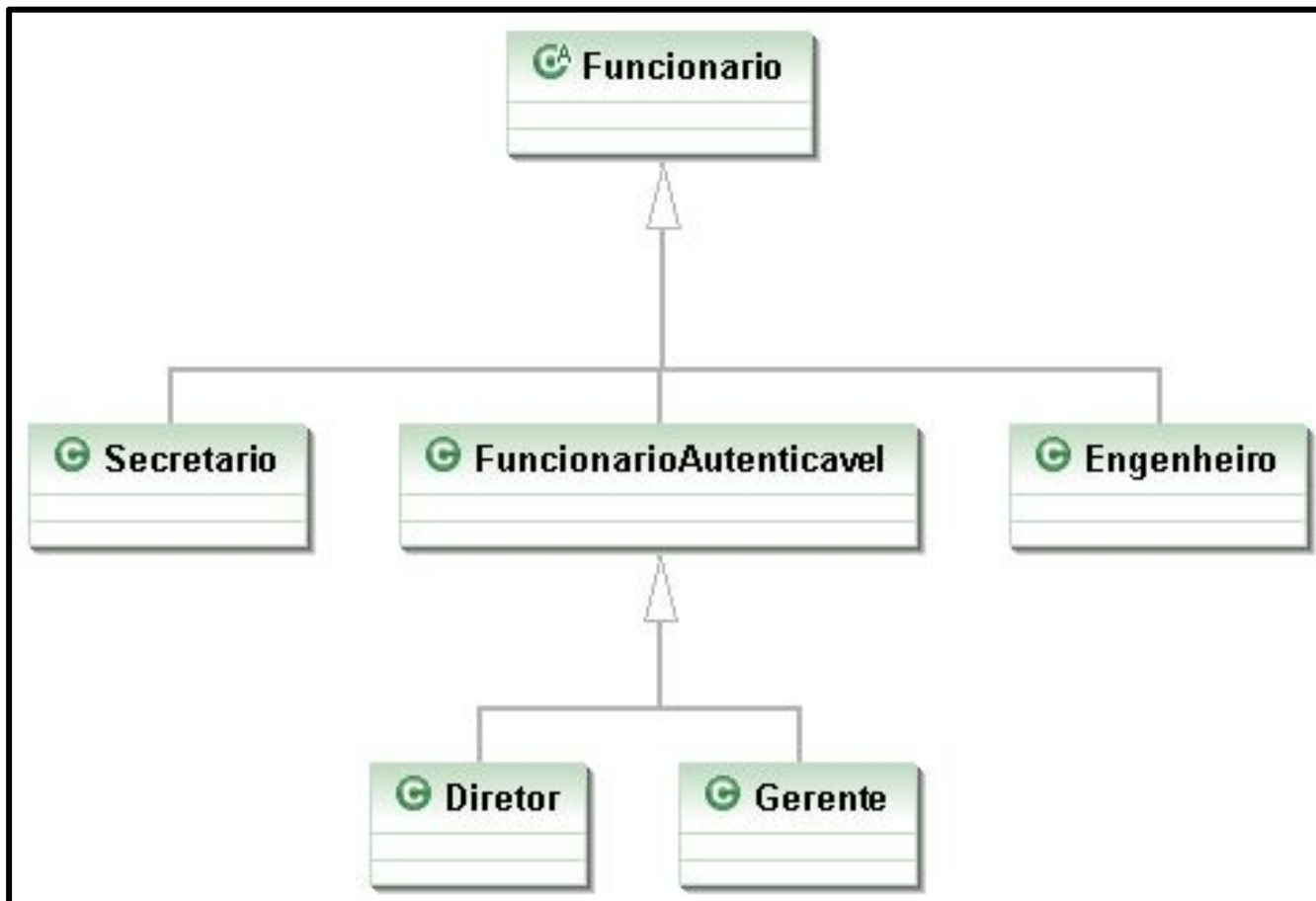
- É como assinar um contrato com o compilador no qual implementamos todos os métodos especificados pela interface ou declaramos nossa classe como *abstract*
- Uma classe que não implementa todos os métodos da interface é uma classe abstrata e deve ser declarada *abstract*

Exemplo de Interface

- Supondo que diretor e gerente acessam a contabilidade de uma empresa; que o acesso do gerente é restrito ao seu setor; e que engenheiro e secretária não tem tal acesso
 - No sistema financeiro, não podemos ter um método boolean `login(Funcionario)` que invoca o método `autentica` dado que nem todo funcionário tem tal método
 - Não é interessante dois métodos `login`, um recebendo `diretor` e outro, `gerente`

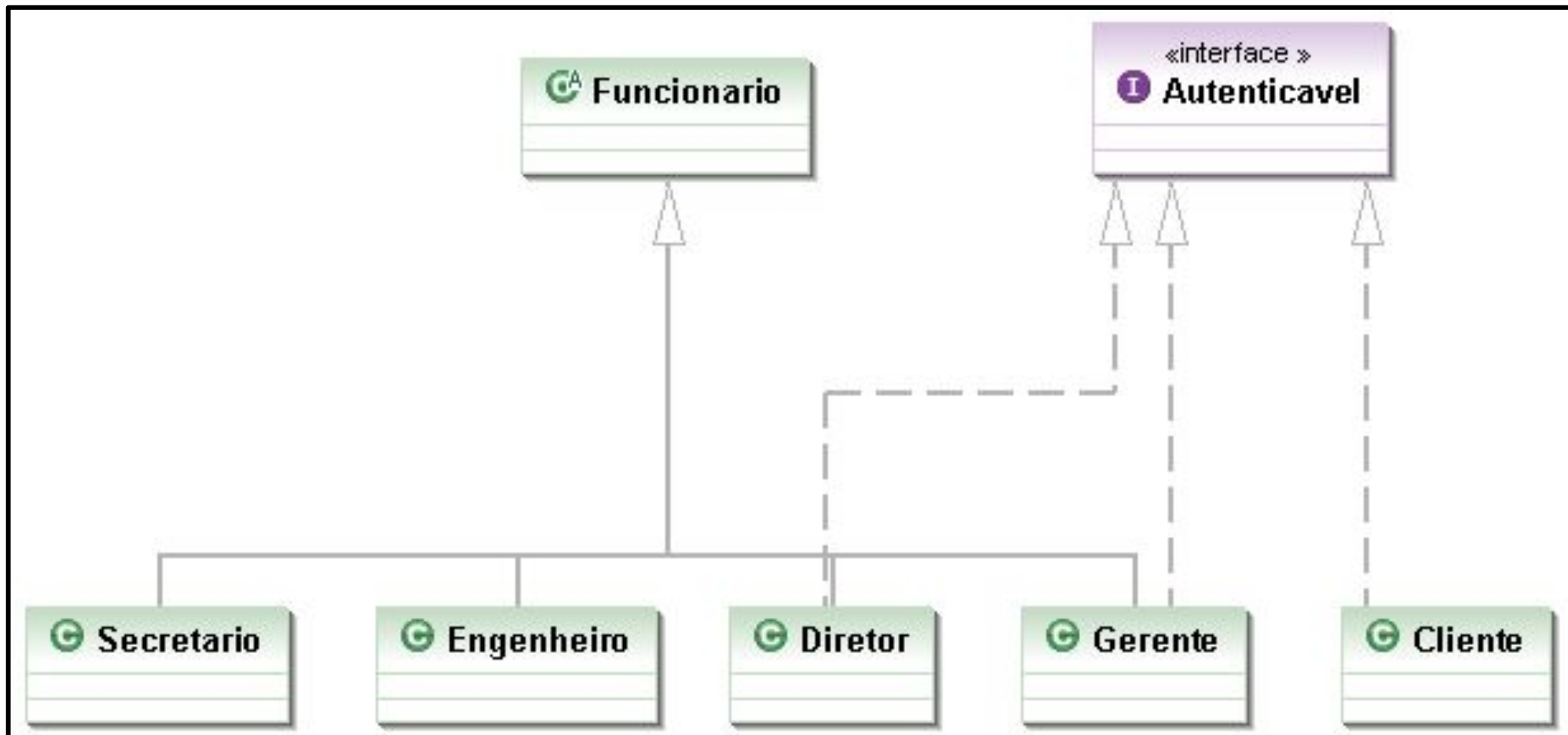
Exemplo de Interface

- A solução abaixo faz sentido, contudo, o cliente (que não é funcionário) também precisará autenticar no sistema



Exemplo de Interface

- Uma solução é criar uma Interface Autenticavel e implementá-la no Diretor, Gerente e Cliente



Exemplo de Interface

- Uma solução é criar uma Interface Autenticavel e implementá-la no Diretor, Gerente e Cliente

```
public interface Autenticavel {  
    // ...  
    public boolean autentica(int senha);  
}
```

```
class Gerente extends Funcionario implements Autenticavel {  
    private int senha;  
    // ...  
    public boolean autentica(int senha) {  
        return (this.senha == senha && /* verificar departamento */);  
    }  
}
```

Exemplo de Interface

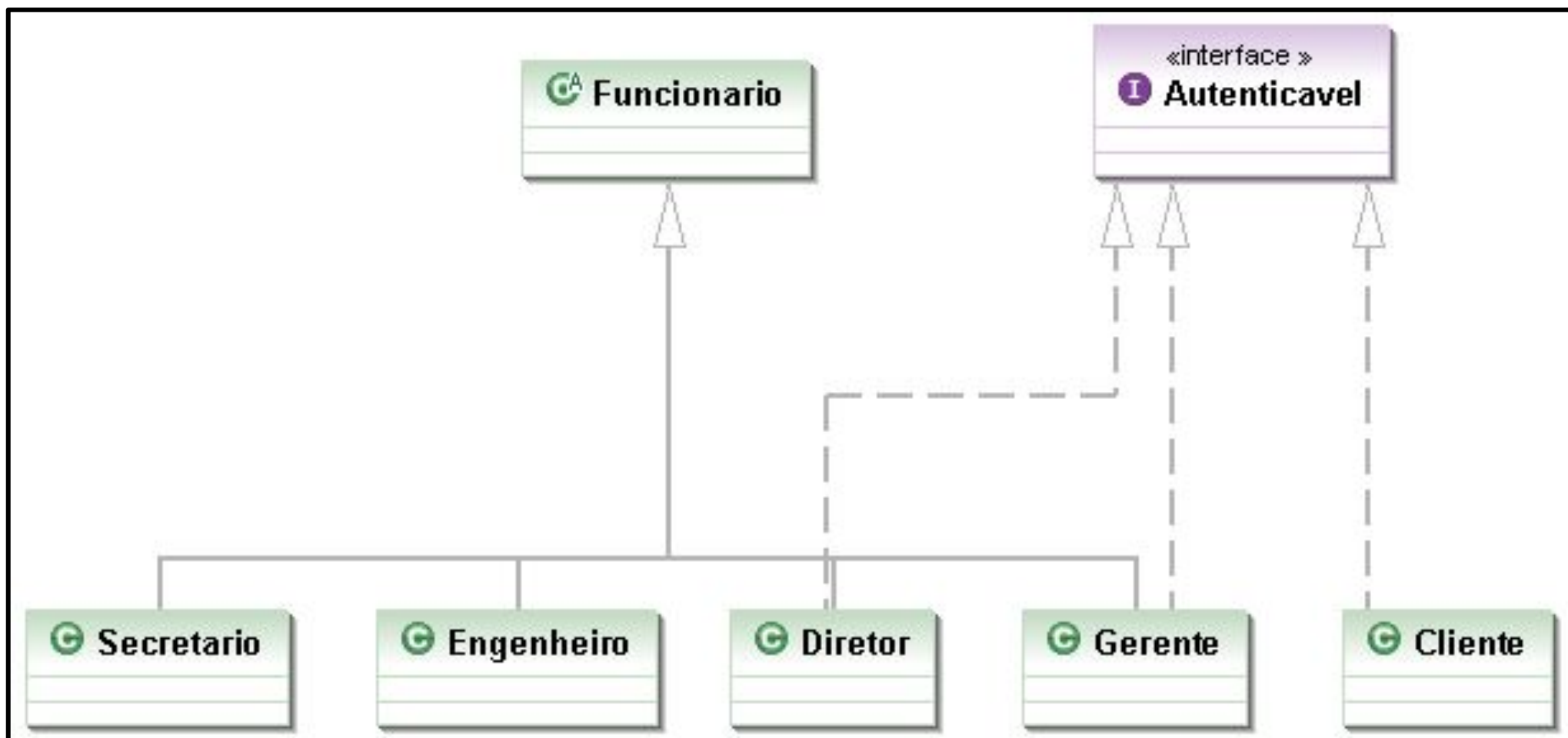
- Uma solução é criar uma Interface Autenticavel e implementá-la no Diretor, Gerente e Cliente

```
class Diretor extends Funcionario implements Autenticavel {  
    // ...  
}
```

```
class SistemaInterno {  
    public void login(Autenticavel a) {  
        int senha = //ê a senha de um banco de dados ou aparelho biométrico  
        boolean ok = a.autentica(senha);  
        // não necessariamente é um Funcionario. Pode ser um Cliente!!!  
        // ...  
    }  
}
```

Exercício: Implemente a Solução

- Uma solução é criar uma Interface Autenticavel e implementá-la no Diretor, Gerente e Cliente



- Interface em Java
- **Classes *ArrayList*, *Vector* e *LinkedList*** 
- Classe *Stack*
- Classe *Queue*

Implementam a Interface *List*

- *ArrayList*, *Vector* e *LinkedList* implementam a interface *List*, logo, elas têm vários métodos similares

```
Vector ve = new Vector();  
ArrayList al = new ArrayList();  
LinkedList ll = new LinkedList();
```

```
ve.add("Atlético-MG");    al.add("Atlético-MG");    ll.add("Atlético-MG");  
ve.add("Cruzeiro");       al.add("Cruzeiro");       ll.add("Cruzeiro");  
ve.add("América");        al.add("América");        ll.add("América");
```

```
System.out.print(ve.size() + " " + al.size() + " " + ll.size());  
System.out.print((String)ve.get(0)+" "+(String)al.get(1)+" "+(String)ll.get(2));
```


```
for (Iterator i = ve.iterator(); i.hasNext();) System.out.println((String)i.next());  
for (Iterator i = al.iterator(); i.hasNext();) System.out.println((String)i.next());  
for (Iterator i = ll.iterator(); i.hasNext();) System.out.println((String)i.next());
```

ArrayList vs Vector vs LinkedList

- *ArrayList/Vector* são *arrays* redimensionáveis e têm comportamento similar
 - Implementação sequencial
 - `get(index)` é eficiente
 - Inserir/remover elementos no meio é ineficiente, pois movimentam elementos
 - Eficiente para caminhar entre elementos
- *LinkedList* são listas duplamente encadeadas
 - Implementação flexível
 - `get(index)` não eficiente
 - Inserir/remover elementos no meio é eficiente
 - Ineficiente para caminhar entre elementos

ArrayList vs Vector

- *Vector* é sincronizada, favorecendo seu uso com *threads*
- Por outro lado, em cenários sem sincronização, *ArrayList* tem um desempenho melhor que o de *Vector*
- *Vector* tem mais métodos por existir desde o Java 1.0

- Interface em Java
- Classes *ArrayList*, *Vector* e *LinkedList*
- **Classe *Stack*** 
- Classe *Queue*

Métodos da Classe *Stack*

- **Elemento pop()**: desempilha o topo da pilha
- **void push(E elemento)**: empilha um elemento
- **boolean empty()**: retorna se a pilha está vazia
- **Elemento peek()**: retorna o topo da pilha, contudo, sem removê-lo
- **int search(Object o)**: retorna a posição de um elemento na pilha

Exemplo com a Classe *Stack*

```
Stack pilha = new Stack();  
  
pilha.push("Atlético-MG");  
pilha.push("Cruzeiro");  
pilha.push("América");  
  
while (pilha.empty() == false){  
    System.out.println("Retirando da pilha: " + pilha.pop());  
}
```

- Interface em Java
- Classes *ArrayList*, *Vector* e *LinkedList*
- Classe *Stack*
- **Classe *Queue*** 