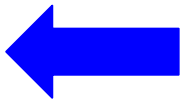
# Unidade VI:
# Tipos Abstratos de Dados Básicos Nativos do Java


Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

# Agenda

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

# Agenda

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

# Interface

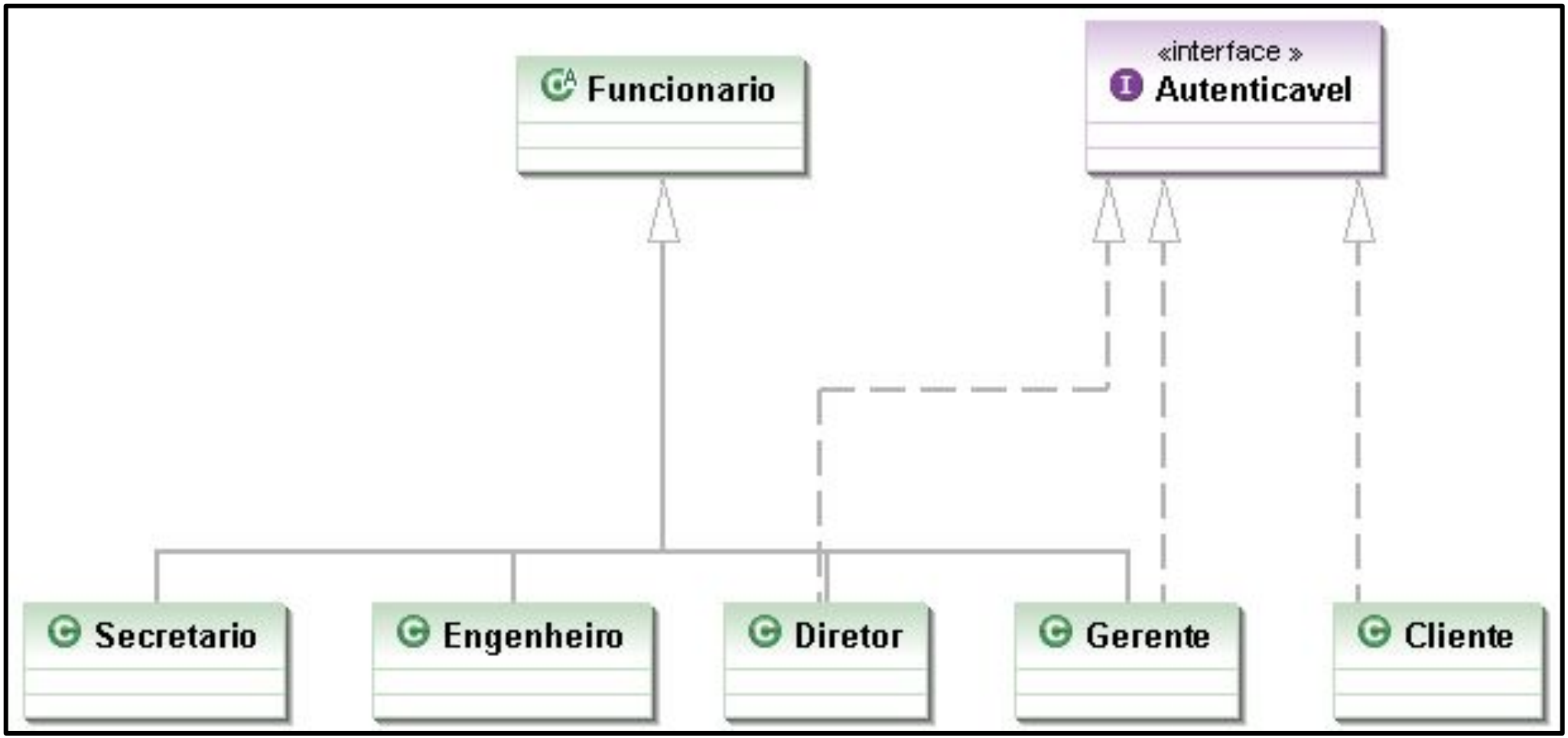
- Define e padroniza como classes distintas interagem entre si. Por exemplo, os cálculos de pagamentos nas classes Funcionario e Mercadoria

- Não especifica qualquer detalhe de implementação

- Ou implementamos todos os métodos especificados pela interface ou declaramos nossa classe como *abstract*

# Exemplo de Interface

- Suponha uma empresa onde apenas o Diretor, Gerente e Cliente acessam o sistema financeiro sendo que o acesso do gerente é restrito ao seu setor



# Exemplo de Interface

- A interface Autenticavel tem o método autentica e a classe Gerente 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 Gerente 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 Gerente 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 */);
    }
}
```

# Exemplo de Interface

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

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

Métodos são públicos e abstratos

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

Métodos abstratos são declarados com a palavra abstract e não têm implementação

# Exemplo de Interface

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

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

Na classe concreta, os métodos da interface seguem a assinatura

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

# Exemplo de Interface

- As classes Diretor e Cliente implementam Autenticavel

```
class Diretor extends Funcionario implements Autenticavel {
        private int senha;
        // ...
        public boolean autentica(int senha) {
            return (this.senha == senha);
        }
}
```

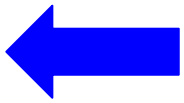
```
class Cliente implements Autenticavel {
        private int senha;
        // ...
        public boolean autentica(int senha) {
            return (this.senha == senha && /* verificar algo do cliente */);
        }
}
```

# Exemplo de Interface

- A classe lê a senha e efetua a autenticação do usuário

```
class SistemaFinanceiro {
	public void login(Autenticavel a) {
		int senha = //lê a senha de um banco de dados ou aparelho biométrico
		if (a.autentica(senha) == true){
			// ...
		}
	}
}
```

# Agenda

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

# Implementam a Interface *List*

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

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

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

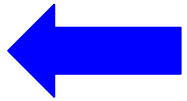
# *ArrayList vs Vector*

- *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

# Agenda

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

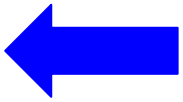
```
import java.util.*;

public class PilhaNativa {
	public static void main(String[] args) throws Exception {
		Stack<String> pilha = new Stack<String>();

		pilha.push("Atlético-MG");
		pilha.push("Cruzeiro");
		pilha.push("América");

		while (pilha.empty() == false){
			System.out.println("Retirando da pilha: " + pilha.pop());
		}
	}
}
```

# Agenda

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

# Métodos da Interface *Queue*

- **boolean add(E e)**: insere o elemento se existir espaço, retornando true em caso de sucesso. Caso contrário, gera uma IllegalStateException
- **boolean offer(E e)**: insere o elemento e retorna true se existir espaço
- **E element()**: recupera a cabeça da fila, sem removê-lo
- **E peek()**: recupera a cabeça, sem removê-lo. Se a fila vazia, retorna null
- **E poll()**: recupera e remove a cabeça. Se a fila vazia, retorna null
- **E remove()**: recupera e remove a cabeça

# Exemplo com a Interface *Queue*

```
import java.util.*;

public class FilaNativa {
	public static void main(String[] args) throws Exception {
		Queue<String> fila = new LinkedList<String>();

		fila.add("Atlético-MG");
		fila.add("Cruzeiro");
		fila.add("América");

		while (fila.isEmpty() == false){
			System.out.println("Retirando da fila: " + fila.remove());
		}
	}
}
```