Interface vs. Implementação

Projeto de Software

Fabio Morais

Baseado no material dos Professores Nazareno Andrade e Hyggo Almeida

Classes vs. Tipos

Existe uma diferença entre uma **classe** e seu **tipo**

- Uma classe define um **tipo** e uma **implementação**
  - **Tipo** define o formato de acesso (**interface**) a objetos da classe
    - O que o objeto é capaz de fazer
  - Um objeto pode ter vários tipos
  - Classes diferentes podem **implementar** o mesmo tipo

O que é feito ≠ como é feito

List<Integer> lista = new ...

- LinkedList<Integer>()
- ArrayList<Integer>()

Qual o tipo de **lista**? Como é a sua **implementação**? Qual a **interface** de acesso a lista?

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description
boolean	**add**(E e) Appends the specified element to the end of this list (optional operation).
void	**add**(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
boolean	**addAll**(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	**addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	**clear**() Removes all of the elements from this list (optional operation).
boolean	**contains**(Object o) Returns true if this list contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns true if this list contains all of the elements of the specified collection.

Tipos e Interfaces de acesso

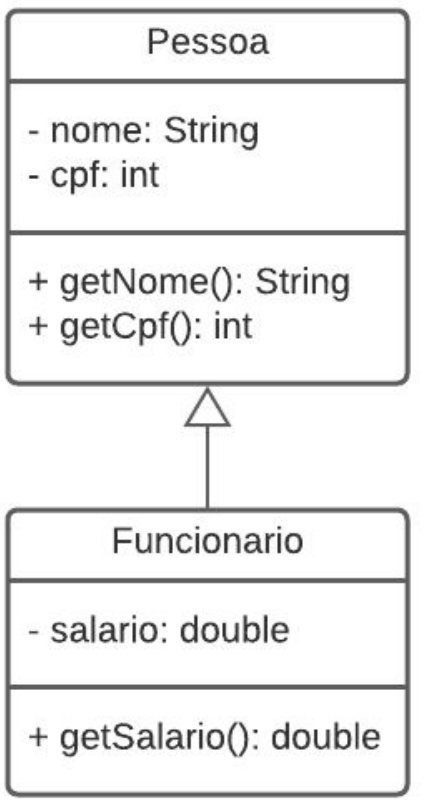
- Objetos são acessados pela sua interface
- Objetos diferentes podem ter interfaces iguais
    - Polimorfismo (objetos com mesmo tipo)
- Clientes mais simples e objetos menos acoplados
    - Flexibilidade: mudanças geram menor impacto

Tipos e Interfaces de acesso

- Existe forte relacionamento entre tipo e classe
- As operações de uma classe definem seu tipo
  - A classe promete oferecer aquelas operações
  - Mas o tipo **não define** a implementação das operações

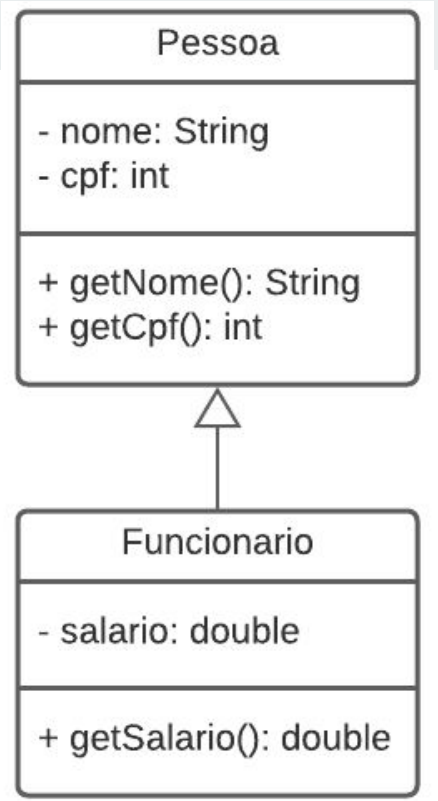
Herança, Tipos e Implementação

- Herança permite criar subtipos (relação **é um**)
  - Funcionario é um **subtipo** de Pessoa
  - ***Pessoa objeto = new Funcionario()***
- Herança também permite reúso de código
  - Funcionario **reusa código** de Pessoa (implementação)



Existem dois tipos de Herança

- Herança de Implementação (ou de classe)
- Herança de Interface (ou de tipo)



Herança de Interface (tipo)

- Define subtipos entre classes
  - Objetos prometem fazer as **mesmas operações**
  - Objetos possuem a mesma **interface** (tipo)

Prometo fazer o mesmo que o outro faz

Herança de Implementação (classe)

- Permite o compartilhamento de código (implementação)
  - Define a **implementação** do tipo de um objeto em função da **implementação** de outro

Implemento da mesma forma que o outro implementa

Em Java

Herança de tipo ou interface (usa *interfaces* do Java)

```
public class Classe implements IPessoa{ . . . }
```

Herança de implementação

```
public class Classe extends Pessoa{ . . . }
```

- Também implica em herança de tipo

Princípio de Orientação a Objetos

Programe para uma **interface** e não para uma **implementação**!

(ou use tipos mais próximos da base da árvore de herança)

Programe para um interface ...

- Contrato entre quem usa a interface e quem implementa
    - Cliente não precisa conhecer o subtipo do objeto que usa
    - Cliente não conhece a classe que implementa o objeto
- A interface é o que há de comum entre os objetos
    - Implementações diferentes para uma mesma interface

Pensando em código

"Preciso de um método que retorne os **últimos 10 itens vendidos**"

Seu código

```
public LinkedList<ItemVenda> getVendidos() { ... }
```

Cliente da sua implementação

```
LinkedList<ItemVenda> lista = xpto.getVendidos();
```

Pensando em código

"... Acho que preciso de uma lista **sem repetições** ..."

Seu código

```
public HashSet<ItemVenda> getVendidos() { ... }
```

Cliente da sua implementação

```
HashSet<ItemVenda> lista = xpto.getVendidos();
```

Pensando em código

"... uma lista sem repetições e **ordenada**"

Seu código

```
public TreeSet<ItemVenda> getVendidos() { ... }
```

Cliente da sua implementação

```
TreeSet<ItemVenda> lista = xpto.getVendidos();
```

Programe para uma interface

- Usar uma lista como Collection deixa a solução genérica

Seu código

```
public Collection<ItemVenda> getVendidos() { ... }
```

Cliente da sua implementação

```
Collection<ItemVenda> lista = xpto.getVendidos();
```

Sobre acoplamento

- Programar para interface reduz o acoplamento

```
List<Integer> lista = xpto.getResultado();
```

```
LinkedList<Integer> lista = xpto.getResultado();
```

Para pensar um pouco

- Em Java, existe alguma diferença entre herdar de uma ***interface*** e de uma ***classe abstrata***?

- E entre um ***interface*** e uma ***classe abstrata*** onde todos os métodos são também abstratos?

Interface vs. Implementação

Projeto de Software