Interfaces

Prof. Pedro Pongelupe

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Engenharia de Software

Sumário

Interface

"**Interface**: parte visível de um módulo a outros módulos. A interface deve oferecer um grupo de métodos coerente. Se uma interface é definida e sempre é mantida, o sistema ganha em extensibilidade e em baixo acoplamento."

Interfaces

- **Interfaces**:
  - “determinado conjunto de métodos que serão implementados em uma classe”.
  - “contrato que define tudo o que uma classe deve fazer se quiser ter um determinado status”.
- Podemos, então, especificar uma interface; e uma ou mais classes “assinariam este contrato”, comprometendo-se a implementar o que foi especificado.

Interfaces

- Interfaces em Java possuem prioritariamente declarações de métodos (sem definição) e atributos "**public static final**".
- A implementação fica a cargo de cada especialização desta interface.
- Interfaces são usadas para definir um protocolo de comportamento que pode ser implementado por qualquer classe na hierarquia de classes.
- Interfaces podem ser declaradas, mas não podem ser instanciadas, assim como classes abstratas.
- É uma saída elegante ao problema da herança múltipla.

Definindo uma interface

- Definição de interfaces:
  - Declaração da interface: declara os atributos tais como nome da interface e se ela herda de outra interface.
  - Corpo da interface: contém as definições de constantes e as declarações dos métodos da interface.

```
interface nomeInterface [extends OutraInterface] {
  corpo da Interface;
}
```

- Para se usar uma interface usa-se a palavra-chave **implements.**

Exemplo: Scanner e classes fecháveis

- A classe Scanner é um leitor de texto simples para converter tipos primitivos utilizando expressões regulares.
- A classe Scanner **implementa** a interface Closeable, portanto é um objeto fechável.
- Um objeto fechável deve implementar o método close:

```
public interface Closeable extends AutoCloseable {
    public void close() throws IOException;
}
```

Exemplo: Scanner e classes fecháveis

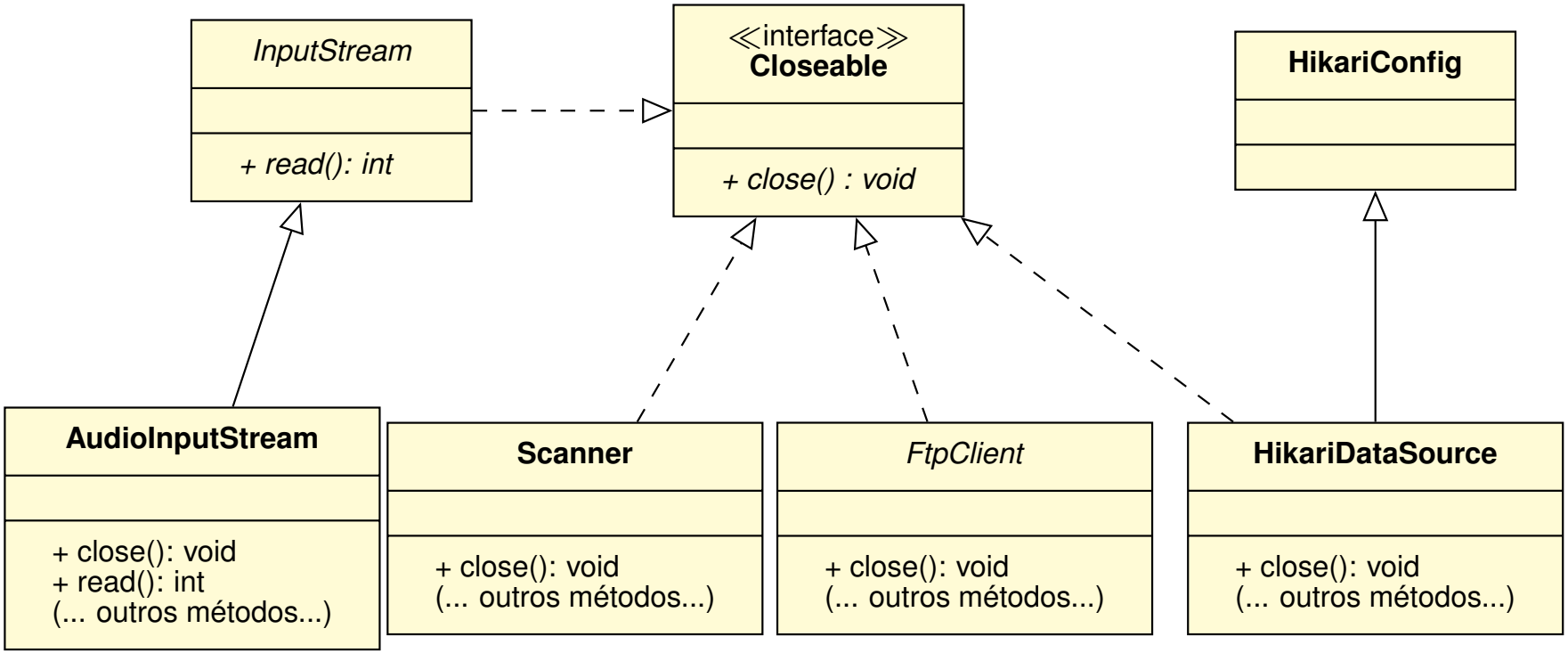
Implementação do método close na classe Scanner

```
@Override
public void close() {
    if (closed)
        return;
    if (source instanceof Closeable) {
        try {
            ((Closeable)source).close();
        } catch (IOException ioe) {
            lastException = ioe;
        }
    }
    sourceClosed = true;
    source = null;
    closed = true;
}
```

Exemplo: Scanner e classes fecháveis

- A interface Closeable declara o método close() mas não a implementa.
- As classes que implementarem esta interface devem implementar o método close().
- Todo objeto será também um fechável.

Exemplo: Scanner e classes fecháveis - UML



Porque utilizar interfaces ao invés de classes abstratas?

- Seria a classe abstrata Closeable abaixo equivalente à interface?

```
abstract class Closeable {
    public abstract void close() throws IOException;
}
```

- Resposta: Não.

Interfaces provêm Herança Múltipla?

- Podem ser encarados como um paliativo, mas são coisas diferentes diferentes:
  - Uma classe herda apenas constantes de uma interface.
  - Uma classe não pode herdar implementações de uma interface.
  - A hierarquia de interfaces é independente da hierarquia de classes. Classes que implementam a mesma interface podem ou não estar relacionadas na hierarquia.
- Java permite herança múltipla de interfaces.

Para que usar Interfaces?

- Use interfaces para definir protocolos de comportamento que possam ser implementados em qualquer lugar na hierarquia de classes.
- Interfaces são úteis para:
  - Capturar similaridades entre classes não relacionadas.
  - Declarar métodos que uma ou mais classes devem inevitavelmente implementar.
  - Revelar interfaces sem revelar os objetos que a implementam (útil na venda de pacotes de componentes).

Métodos default

- Até o Java 7, interface não podia prover nenhuma implementação.
- No Java 8, um método *default* permite definir um método de interface com implementação.
- Permite expandir a interface sem violar o código existente.
- Permite implementar métodos que são opcionais, dependendo da forma como a interface é usada.
- Pode produzir erro de herança múltipla de método.

Métodos default

```
public interface UserProfile {
  // decl. método normal
  int getId();

  // decl. método default
  default int getAdminId() {
   return -1;
  }
}
```

```
class MyUserProfile implements
    UserProfile {
  public int getId() {
    return 101;
  }
}
```

```
class Demo {
  public static void main(String args[]) {
    UserProfile obj = new MyUserProfile();

    System.out.println("ID: " + obj.getId());
    System.out.println("Admin ID: " + obj.getAdminId());
  }
}
```

[1]

[1] Adaptado de: Herbert Schildt. *Java Para Iniciantes*. Bookman 2018.

Métodos default

```
public interface Ordenavel {

    boolean menorQue(Ordenavel o);

    boolean igual(Ordenavel o);

    default boolean diferente(Ordenavel o) {
        return !igual(o);
    }

    default boolean maiorQue(Ordenavel o) {
        return !menorQue(o) && !igual(o);
    }

}
```

Expressões Lambda e interfaces funcionais

- Espressões lambda e interfaces funcionais são elementos da programação funcional incorporados ao Java.
- Programação funcional, com sua ênfase em funções "puras", tratadas como valores de 1a classe (que retornam o mesmo resultado dadas as mesmas entradas, sem a produção de efeitos colaterais) e a imutabilidade simplificam a programação paralela.

Interfaces funcionais (*functional interface*)

São interfaces com um único método abstrato. Sua implementação pode ser feita por uma classe regular, classe interna, classe anônima ou expressão lambda. A boa prática é anotar essas classes com *@FunctionalInterface*.

Expressões Lambda e interfaces funcionais

1. Considere a interface Runnable, disponível no Java.
2. Ela é uma *functional interface* baseada no método run, e pode ser implementada por uma expressão lambda ou referência de método.

```
@FunctionalInterface
public interface Runnable {
    public void run();

}
```

Interfaces funcionais com classes internas

```
public class ImprimeNomeThreadRunnable implements Runnable {

    @Override
    public void run() {
        System.out.println("O nome dessa thread é "
        + Thread.currentThread().getName());
    }
}

public class Driver {

  public static void main(String[] args) {
    System.out.println("O nome dessa thread é " + Thread.currentThread().getName());
    new Thread(new ImprimeNomeThreadRunnable()).start();
  }

}
```

Interfaces funcionais com classes anônimas

1. Solução possível: classe interna anônima que implementa a interface Runnable.

```
public class DriverClasseImplementaInterface {

        public static void main(String[] args) {
                System.out.println("O nome dessa thread é " +
            Thread.currentThread().getName());

                new Thread(new Runnable() {

                        @Override
                        public void run() {
                                System.out.println("O nome dessa thread é " +
                                                Thread.currentThread().getName());
                        }
                }).start();
        }
}
```

Interfaces funcionais com expressões lambda

1. Solução possível: expressão lambda que implementa a interface Runnable.

```
public class DriverLambda {

        public static void main(String[] args) {
                System.out.println("O nome dessa thread é " +
                                Thread.currentThread().getName());

                new Thread(() -> System.out.println("O nome dessa thread é " +
                                Thread.currentThread().getName()))
                        .start();
        }
}
```

Obrigado!!

Muito obrigado pela atenção! Alguma dúvida? Bora praticar!!!

"*Ser radical é atacar o problema em suas raízes*"