Ficha Prática 10

António Nestor Ribeiro, Paulo Azevedo, Mário Martins
{anr,pja,fmm}@di.uminho.pt
PPIV (LESI)

2005/06

Objectivos

1. Interfaces: declaração e utilização.
2. Interfaces como tipos de dados
3. Exercícios.

Interfaces

A declaração de uma interface em Java é muito simples dado que a interface apenas poderá conter um identificador, um conjunto opcional de constantes (ou seja static e final) e um conjunto de assinaturas de métodos que são implicitamentes abstractos.

O código necessário para criar uma interface toma a seguinte forma:

```
public interface MinhaInterface  {

public abstract metodoA();
public abstract metodoB();
....
}
```

Uma classe pode declarar-se do tipo de dados de uma dada interface ao sinalizar tal através da seguinte construção:

```
public class MinhaClasse implements  MinhaInterface  {

....
....

public metodoA() {.....}
public metodoB(){.....}
....
}
```

A classe para correctamente implementar a interface, ou interfaces, declarada(s), tem de fornecer uma concretização para os métodos que estão declarados na interface.

Objectos instância dessa classe podem ser vistos como sendo do tipo de dados da interface, sendo que nesse caso o seu comportamento resume-se aos métodos declarados na interface.

Hierarquia das Interfaces

À semelhança das classes, as interfaces estão organizadas numa hierarquia. No entanto, a herança existente na hierarquia das interfaces é múltipla, o que implica que uma interface pode ser sub-interface de mais do que uma interface.

Podemos ter assim declarações como:

```
public interface MinhaInterface extends Interface1, Interface2 {

public abstract metodoA();
public abstract metodo B();
....
}
```

Quando os métodos herdados das possíveis super-interfaces forem idênticos é necessário tornar não ambígua a utilização do método recorrendo à sintaxe `NomeInterface.metodo()`.

Uma classe pode implementar mais do que uma interface, sendo que para tal deve explicitamente elencar a lista das interfaces.

```
public class MinhaClasse implements MinhaInterface1, MinhaInterface2  {

....
....
```

```
....
}
```

Interfaces e Classes Abstractas

Por vezes o mesmo conceito pode ser expresso através da utilização de classes abstractas ou de interfaces. Tal resulta do facto de que uma classe totalmente abstracta e uma interface terem o mesmo poder expressivo. No entanto existem algumas diferenças entre ambos os conceitos:

- uma classe abstracta pode não ser 100% abstracta, enquanto que uma interface é-o garantidamente;
- uma classe abstracta não impõe às suas subclasses a implementação dos métodos abstractos - uma interface obriga a quem a implementa a tornar os métodos concretos;
- uma classe abstracta pode ser utilizada para criar os mecanismos para a construção de software genérico, extensível e parametrizável. Uma interface não tem genericamente essas preocupações;
- classes e interfaces pertencem a duas hierarquias distintas, podendo no entanto o programador conjugar ambas por forma a aumentar a capacidade expressiva dos seus programas.

A classe Class

A classe `Class` permite que em tempo de execução se possam saber algumas propriedades importantes dos objectos que populam uma aplicação.

Instâncias da classe `Class` representam classes e interfaces num ambiente de execução Java. É possível por exemplo a uma instância desta classe enviar metódos como:

- public static Class forName(String classname)
- public Object newInstance()
- public boolean isInstance(Object obj) // equivalente dinâmico ao operador instanceOf
- public boolean isInterface()
- public Class getSuperClass()

- public Class[ ] getInterfaces()
- etc.

É por exemplo possível determinar quais as interfaces a que determinados objectos (pertencentes por exemplo a uma colecção) respondem.

O método `getInterfaces` pode ser invocado para determinar quais as interfaces com que uma determinada classe está comprometida. Este método devolve um array de objectos do tipo `Class`.

Vejam-se por exemplo as seguintes declarações, em que em tempo de execução se interroga quais as interfaces implementadas.

```
import java.lang.reflect.*;
import java.io.*;

class SampleInterface {

   public static void main(String[] args) {
      try {
          RandomAccessFile r = new RandomAccessFile("myfile", "r");
          printInterfaceNames(r);
      } catch (IOException e) {
          System.out.println(e);
      }
   }

   static void printInterfaceNames(Object o) {
      Class c = o.getClass();
      Class[] theInterfaces = c.getInterfaces();
      for (int i = 0; i < theInterfaces.length; i++) {
         String interfaceName = theInterfaces[i].getName();
         System.out.println(interfaceName);
      }
   }
}
```

Exercícios

1. Crie a interface Audio que deverá ser implementada por forma a classificar os suportes que tenham audio. Essa interface deve poder ter o comportamento que permita saber:

- quantas faixas de audio existem
- qual é a codificação em termos de bitrate (ex: 192kbps, 128kbps, etc.)

2. Crie uma interface Video que permite saber:
   - qual é a sinopse do filme (guardada numa String)
3. Altere as classes existentes por forma a que a classe CD implemente Audio e DVD implemente Video.
4. Crie uma classe DVD-Audio que representa um DVD musical, isto é, que para além do filme tenha faixas de música.
5. Refaça o método que determina o número total de faixas de música, tendo em conta a nova estrutura de classes.
6. Crie as classes de excepção necessárias e codifique as situações em que deve lançar excepções. Sugere-se que valide as pré-condições recorrendo a esta nova técnica de programação, por forma a tornar a classe BibMultimedia robusta.