

Compatibilidade entre Interfaces e Classes

Seja a interface Ordem:

```
public interface Ordem{  
    boolean igual (Ordem elemento);  
    boolean maior (Ordem elemento);  
    boolean menor (Ordem elemento);  
}
```

Identificadores de interfaces podem ser usados na declaração de variáveis:

```
Ordem ord1, ord2;
```

- os referências ord1 e ord2 podem ser associadas a quaisquer instâncias que implementem a interface Ordem.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Supondo a classe ClasseA:

```
public class ClasseA implements Ordem {  
  
    // implementação dos métodos da interface Ordem (igual, maior e menor)  
  
    // outros métodos  
  
    public void m1 () { ... }  
    public void m2 () { ... }  
  
    ...  
  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

e a declaração:

```
ClasseA A1 = new ClasseA();
```

```
ord1 = A1;           // correto
```

```
ord1.m1 ();         //errado – m1() não é método da interface
```

```
ord1.m2 ();         //errado – m2() não é método da interface
```

```
ord1. igual (ord2)  // correto
```

Herança de interfaces

Exemplos retirados de [Martins] “Programação Orientada aos Objectos em Java 2”, F. Mário Martins, FCA, Setembro de 2000.

O que acontece quando há sobreposição de constantes e métodos?

1) Herança simples de constantes

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public interface I1 {
```

```
    // constantes
```

```
        int x = 1;
```

```
        int y = 2;
```

```
        int z = 3;
```

```
    // métodos
```

```
        int soma();
```

```
}
```

```
public interface I2 extends I1 {
```

```
    // constantes
```

```
        int x = 10;
```

```
        int y = 20;
```

```
}
```

```
    // as constantes podem ser
```

```
    // redefinidas
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class Classe1  
    implements I1 {
```

```
    public int soma(){  
        return ( x + y + z );  
    }
```

```
}
```

```
public class Classe2  
    implements I2 {
```

```
    public int soma(){  
        return ( x + y + z );  
    }
```

```
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class Teste {  
  
    public static void main (String args [] ){  
  
        Classe1 C1 = new Classe1();  
        Classe2 C2 = new Classe2();  
  
        System.out.println ( C1.soma() );  
        System.out.println ( C2.soma() );  
    }  
}
```

Qual o output deste programa?

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Resposta:

6

33

Como aceder ao x e ao y da super-interface?

```
public class Classe3 implements I2 {  
  
    public int soma(){  
        return ( I1.x + y + z );  
    }  
}
```

- prefixar a constante com o nome da interface

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

2) Herança múltipla de constantes

```
public interface I3 extends I1 , I2 {  
    // redefine as constantes  
    int x = 100;  
    int y = 200;  
    int z = 300;  
  
    // métodos  
    int soma();    // sobrepõe o método  
}
```

```
public interface I4 extends I1, I2 {  
    //válido  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class Classe3 implements I3 {  
    public int soma() {  
        return (x + y + z );    //válido  
    }  
}
```

```
public class Classe4 implements I4 {  
    public int soma() {  
        return (x + y + z );    // errado - ambíguo  
    }  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class Classe4 implements I4 {  
  
    public int soma() {  
  
        return ( I1.x + I2.y + z );    // correto  
  
    }  
  
}
```

3) Herança simples de métodos

```
public interface I1 {  
  
    // constantes  
    int a = 1;  
    int b = 2;  
  
    // métodos  
    int soma();  
    int prod();  
}
```

```
public interface I2 extends I1 {  
  
    // constantes  
    int b = 20; //redefinição  
  
    // métodos  
    int soma() //sobreposição *  
    int soma (int x);  
    int prod (int x);  
}
```

** tal como na herança de classes, a sobreposição de métodos obriga a igual tipo de resultado*

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class C1 implements I1 {  
    public int soma() { return a + b;}  
    public int prod() { return a * b;}  
}
```

```
public class C2 implements I2 {  
    // métodos de I2  
    public int soma() { return a + b;}  
    public int soma (int x) { return x + a + b; }  
    public int prod (int x) { return a * x * b;}  
  
    //métodos de I1  
    public int prod() {return a * b; }  
}
```

Qual o output do seguinte programa?

```
public class TesteInterface{
    public static void main (String args [] ) {
        C1 c1 = new C1();
        C2 c2 = new C2();
        System.out.println ( c1.soma() );
        System.out.println ( c1.prod() );
        System.out.println ( c2.soma() );
        System.out.println ( c2.soma(10) );
        System.out.println ( c2.prod() );
        System.out.println ( c2.prod(5) );
    }
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

3

2

21

31

20 !!

100

4) Herança múltipla de métodos

```
public interface I11 {  
    // constantes  
    int a = 1;  
    int b = 2;  
  
    // métodos  
    int soma();  
  
    int prod (); **  
  
    float div (int x, int z);  
}
```

```
public interface I22 {  
    // constantes  
    int b = 20;  
  
    //métodos  
    int soma();  
    int soma (int x);  
  
    float prod (); **  
  
    boolean prod (int x, int y);  
}
```


Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public interface I33 extends I11, I22 {
```

```
    int soma (int x); //sobreposição
```

```
    float prod (int x , int y);
```

```
    // errado, sobrepõe o método com um diferente tipo de resultado
```

```
    boolean prod (int k); // sobrecarga
```

```
}
```

*** ao implementarmos I33 ocorrerá um erro de compilação: sobreposição com diferentes tipos de resultados.*

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Observações:

1) Se uma interface declara um método, esse método sobrepõe todos os métodos com a mesma assinatura nas suas super-interfaces;

Se um método sobrepõe outro, terá que ter o mesmo tipo de resultado que o primeiro.

2) Se um método sobrepõe outro então a sua cláusula *throws* não pode entrar em conflito com a dos métodos que sobrepõe.

3) Os métodos são sobrepostos com “base” na sua assinatura;
Se uma interface declara dois métodos com o mesmo nome, e uma interface sobrepõe um deles, uma implementação satisfaz ambas as interfaces.

4) Uma interface herda das suas super-interfaces todos os métodos que não sejam sobrepostos por declaração na interface.

5) É possível uma interface herdar mais do que um método com a mesma assinatura, desde que com o mesmo tipo de resultado.

!! Se o tipo de resultado for diferente, na classe que implementa essa interface ocorrerá um erro.

5) É possível uma interface herdar mais do que um método com a mesma assinatura, desde que com o mesmo tipo de resultado.

!! Na classe que implementa essa interface ocorrerá um erro.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Supondo a classe Aluno, construída nas aulas anteriores,

– Construa uma pequena aplicação que permita as seguintes opções:

- 1 - Criar aluno
- 2 – Consultar aluno, dado o seu número
- 3 – Consultar aluno, dado o seu nome
- 4 – Listar todos os alunos
- 5 – Apagar um aluno
- 6 – Corrigir o nome de um aluno
- 7 – Guardar alunos num ficheiro

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

- Os alunos criados deverão ser armazenados num objecto do tipo `java.util.Vector<Aluno>`.
- A opção 7 deverá escrever num ficheiro (`fichAlunos`) o vector com os alunos existentes no vector de alunos.
- Quando o programa é executado deverá começar por ler o conteúdo desse ficheiro.

Duas versões:

- *um só programa e uma função para cada opção.*
- *as várias funções numa classe separada.*