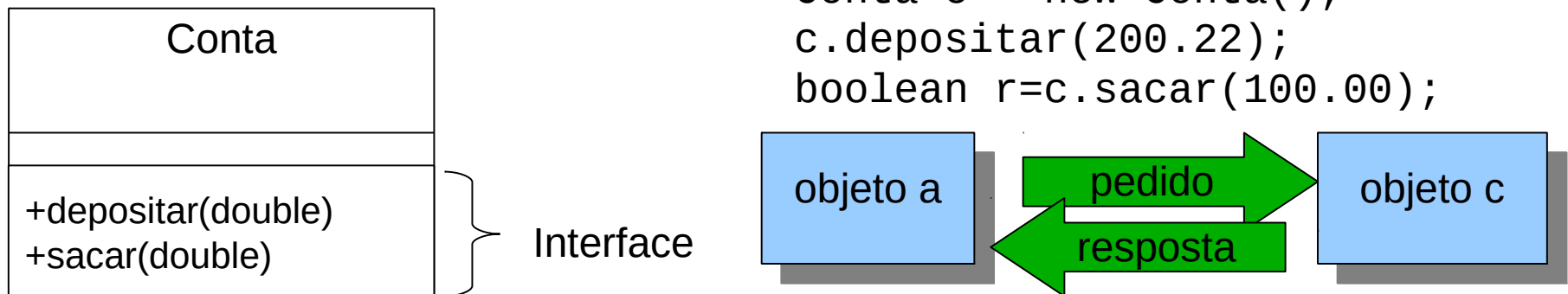


04-Polimorfismo

Conteúdo: Interface em Java. Polimorfismo: implementação de métodos abstratos. redefinição de métodos. upcasting e downcasting.

Interface (1)

- Interface de um objeto: conjunto de pedidos que podem ser feitos a ele, ou seja, os métodos que definem o comportamento dele
- Enviando mensagem a um objeto
 - `<nome_objeto>.<nome_método>(<args>*)`;



Interface (2)

- Em Java: interface é como uma classe 100% abstrata
- Interface Java é equivalente a tipo (conforme GoF)
- Tipo pode ser entendido como um subconjunto de mensagens que um objeto pode receber
- É um contrato entre o código cliente (que usa a classe) e a classe que implementa a interface Java
- É uma declaração formal do contrato por meio de assinatura de métodos sem implementação
- Várias classes, mesmo que não relacionadas, podem implementar o mesmo tipo (interface Java)

Interface (3)

- Posso definir uma interface Java personalizada, ou usar uma pré-definida na biblioteca Java
- Definindo um classe como implementando um tipo interface Java

```
<modificador>* class <nome> implements <nome_interface>* { }
```

- Todos os métodos definidos na interface Java devem ser implementados pela classe concreta
- Uma classe pode implementar várias interfaces Java

```
public class I1 extends Applet implements Runnable,  
    Estable, Sortable, Observable { ... }
```

Teste Rápido

- Seja uma classe Cliente que herda de Pessoa e implementa a interface Sortable. Quais tipos podem referenciar objetos da classe Cliente?

A. Pessoa

B. Cliente

C. Sortable

D. Object

Resposta do Teste Rápido

- Todos. Todas as classes em Java herdam, implicitamente, de Object.

Interface (4)

- Sintaxe de Declaração

```
<modificador>* interface <nome> {  
    <protótipo_método>*  
}
```

- Exemplo

```
public interface Voador {  
    public void decolar();  
    public void aterrizar();  
    public void voar();  
}
```

```
public class Aviao  
    implements Voador {  
    public void decolar() {  
        /* implementação */  
    }  
    public void aterrizar(){  
        /* implementação */  
    }  
    public void voar() {  
        /* implementação */  
    }  
}
```

Interface (5)

- Implementando várias interfaces

```
interface IServico {  
    public boolean iniciar();  
    public boolean parar();  
}  
  
interface IMonitor {  
    public int nivel();  
    public float temperatura();  
}
```

```
class Reator implements  
    IServico, IMonitor {  
    public boolean iniciar() {  
        /* implementação */  
    }  
    public boolean parar() {  
        /* implementação */  
    }  
    public int nivel() {  
        /* implementação */  
    }  
    public float temperatura(){  
        /* implementação */  
    }  
}
```


Interface (6)

- Herança em interfaces. Exemplo:

```
interface I1 {  
    public void m1();  
}  
interface I2 extends I1 {  
    public void m2();  
}  
class C1 implements I2 {  
    public void m1() { /* implementação m1 */ }  
    public void m2() { /* implementação m2 */ }  
}
```

A interface I2 engloba os métodos m1 (herdado de I1) e m2

Os métodos m1 e m2 precisam ser implementados.

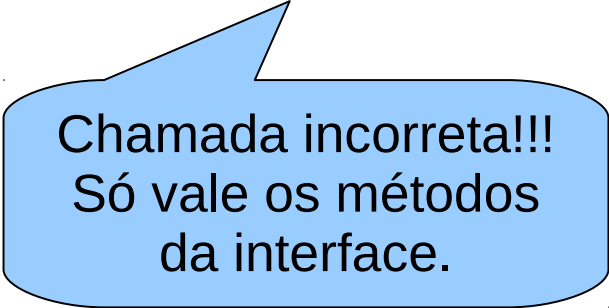
Interface (7)

- Usando referência do tipo interface

- Enxergar um objeto pela “máscara” da interface

```
interface INavegador {  
    public void anterior();  
    public void proximo();  
}  
public class DVD  
    implements INavegador {  
    /* implementações */  
    public void irMenu() { }  
}
```

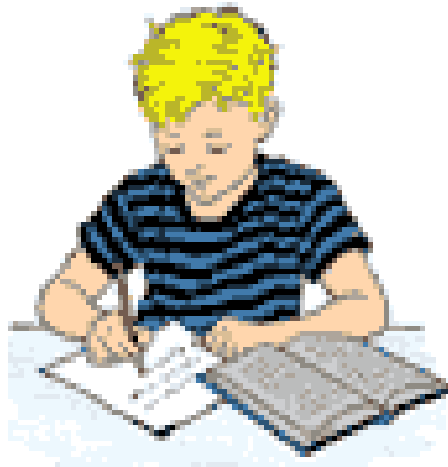
```
public class Testa {  
    public static void  
        main(String args[]) {  
        INavegador controle;  
        controle=new DVD();  
        // Exemplos:  
        controle.anterior();  
        controle.proximo();  
        controle.irMenu();  
    }  
}
```



Chamada incorreta!!!
Só vale os métodos
da interface.

Exercícios de Fixação

- Faça os exercícios 04-polimorfismo-parte-01



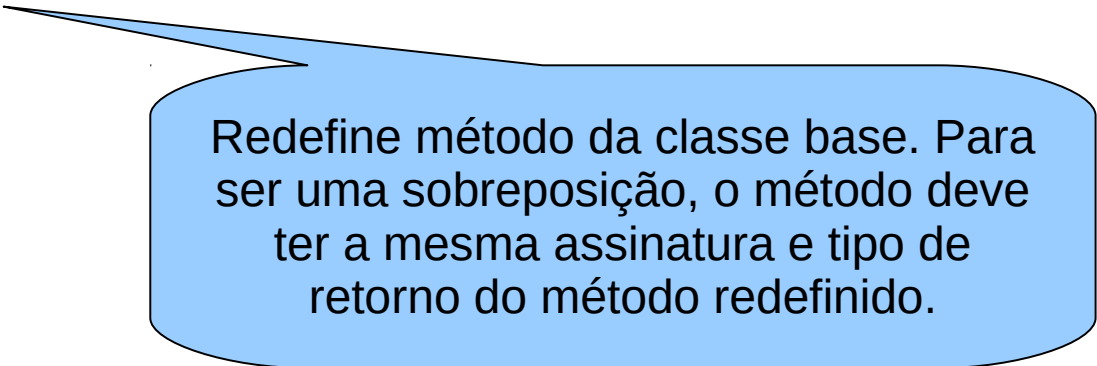
Polimorfismo (1)

- Significado: diferentes formas
- Em POO: dois objetos, pertencentes a mesma classe base, podem se comportar de maneira diferente recebendo a mesma mensagem
- Maneiras de implementar polimorfismo
 - Por implementação de método abstrato
 - Por redefinição (override ou sobreposição) de método herdado

Polimorfismo (2)

- Redefinição de método (*override*)
 - Classe redefine método definido na superclasse

```
public class A {  
    public void mostrar() { System.out.println("A"); }  
}  
  
public class B extends A {  
    public void mostrar() {System.out.println("B"); }  
}
```



Redefine método da classe base. Para ser uma sobreposição, o método deve ter a mesma assinatura e tipo de retorno do método redefinido.

Polimorfismo (3)

- Instrução qualificadora **super**

```
public class A {  
    public void mostrar() { System.out.println("A"); }  
}  
  
public class B extends A {  
    public void mostrar() {  
        super.mostrar();  
        System.out.println("B");  
    }  
}
```



O que a instrução abaixo
irá resultar?
new B().mostrar();

Teste Rápido

- Um programador percebe que, com frequência, vários métodos herdados da super classe não se aplicam a nova classe que está criando. Quais afirmações estão corretas?
 - A. A super classe está errada
 - B. O foco deve ser nos métodos que se aplicam à subclasse, ignorando os outros.
 - C. O uso da herança não está adequado
 - D. A linguagem adotada não deveria ser O.O.

Resposta do Teste Rápido

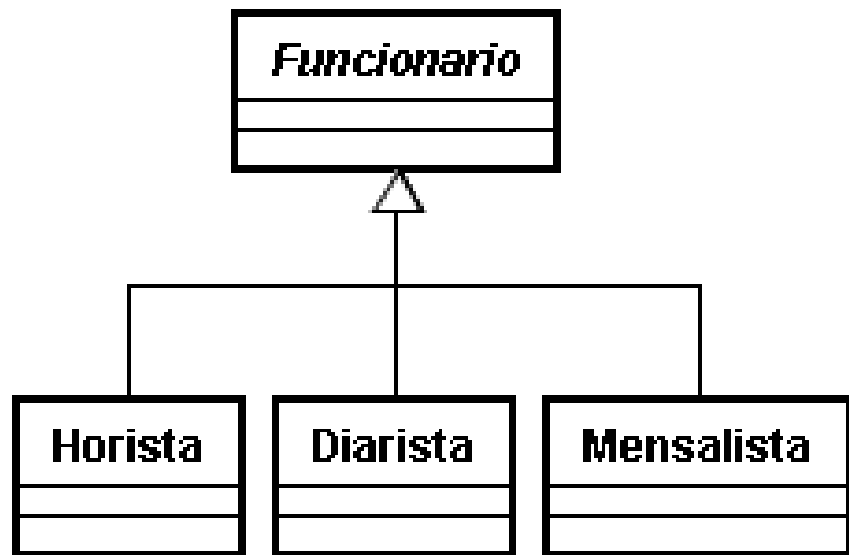
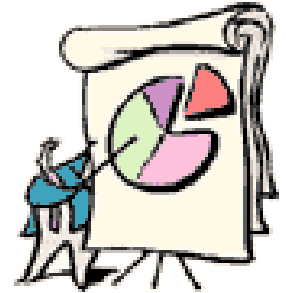
- Somente a C. Geralmente, no afã de se reusar um comportamento criado numa classe, acha-se que a melhor maneira é criar uma subclasse dela.

Polimorfismo (4)

- Classe Abstrata
 - Define uma classe de objetos cuja implementação não conhecemos integralmente
 - Diferente das classes concretas, não pode haver instanciamento direto desse tipo de classe
 - Serve como um molde para que sejam definidas classes concretas, que devem completar a implementação ausente na abstrata
 - De forma análoga, um método abstrato serve apenas para definir uma mensagem na interface, mas não sabemos a implementação

Polimorfismo (5)

- Estudo de caso BlueJ: uso de classe abstrata
 - Código TiposFuncionario



- Analisar código da 4 classes;
- Tentar instanciar Funcionario pelo *Code Pad*;
- Instanciar 2 objetos Horista (f1 e f2);
- Tentar acessar método privado;
- Instanciar outros objetos e gerar pagamento deles;
- Tentar compilar uma das sub-classes sem implementar o método abstrato

Polimorfismo (6)

- Referência polimórfica: consiste em utilizar o tipo de uma das superclasses na hierarquia para referenciar um objeto de uma subclasse
 - Funcionario f1=new Horista("Ana",8.33);
- A interface acessível através da referência polimórfica é a mesma da classe utilizada, não sendo possível usar de forma direta a interface das subclASSES.
- Para acessar a interface da subclASSES, temos que fazer um *casting (mais especificamente downcast)*.

Polimorfismo (7)

```
Funcionario f1=new Horista("Ana",8.33);  
f1.acumularHoras(4);  
//Error: cannot find symbol - method acumularHoras(int)  
Horista h1 = (Horista) f1; // downcast  
h1.acumularHoras(4);  
Funcionario f2 = new Mensalista("João");  
Horista h2 = (Horista) f2; // tenta downcast  
                        // Erro ClassCastException  
System.out.println( f1.calcularPagamento() );  
Funcionario f3=new Mensalista("Ze",4500.22);  
System.out.println(f3.calcularPagamento());
```

Exercícios Práticos

- Faça os exercícios 04-polimorfismo-parte-02
- T3: início do prazo
- Exercício Orientado Banco CobraTudo



Humor



Papo de Bêbado