

Polimorfismo

Prof. Pedro Pongelupe



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

- 1 Polimorfismo
 - Tipos de Polimorfismo: Class. de Carndelli
 - Polimorfismo ad-hoc
 - Polimorfismo universal de inclusão
- 2 Sobrepondo métodos e ocultando campos
 - Palavra-chave super
 - A classe Object
- 3 Exemplo: Estoque polimórfico



Polimorfismo

- Polimorfismo → muitas formas.
- Algumas funções se aplicam a objetos de diferentes classes, alcançando o mesmo resultado semântico.
- É o princípio que permite que classes filhas tenham métodos iguais, mas comportamentos diferentes.
 - Métodos iguais → mesma assinatura.
 - Comportamentos diferentes → ações diferentes.



Polimorfismo: Classificação de Cardelli

- Polimorfismo *ad-hoc*
 - Sobrecarga
 - Coerção
- Polimorfismo universal
 - Polimorfismo universal paramétrico
 - Polimorfismo universal de inclusão



Polimorfismo *ad-hoc*

- Sobrecarga (*overloading*) é uma forma limitada de polimorfismo
 - permite definir funções com o mesmo nome em um mesmo escopo (assinaturas diferentes).
 - o compilador, pelos parâmetros, determina a função a ser chamada: a / b (real), a / b (inteiro)
- Coerção ocorre quando objeto diferente do esperado o compilador força a conversão automática
 - $5.0/2 = ? \mapsto \text{float} / \text{float} = \text{float}$
 - $\text{int} \mapsto \text{float}$ (inteiro convertido para real)

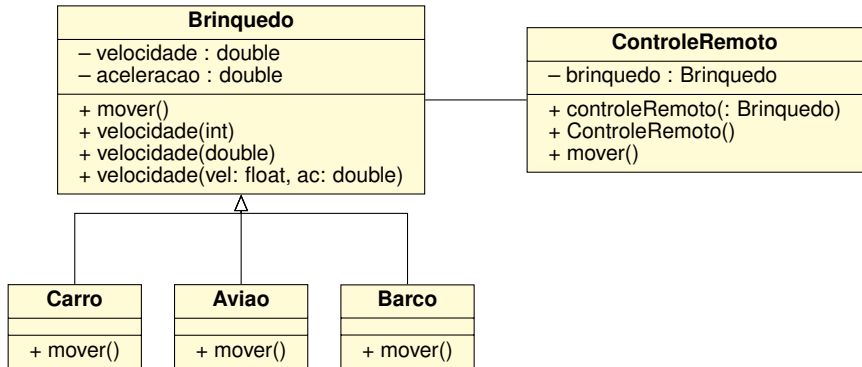


Polimorfismo universal de inclusão

- Funções com mesmo nome e mesma assinatura, mas em escopo diferente.
 - O programador não precisa determinar qual implementação será efetivamente executada.
 - Amarração tardia (*late binding*)
- Pode ocorrer perda de desempenho (por causa da *late binding*)
- Requisitos (deve haver):
 - Herança.
 - Sobreposição de métodos.
 - Referência de uma superclasse apontando para um objeto de uma subclasse (referências polimórficas).



Exemplo de herança simples

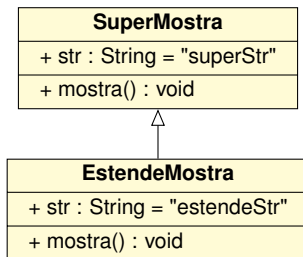


Adaptado de: Bezerra, Gomes, Kulesza, *Programação Orientada a Objetos*, Aula 12.



Sobrepondo métodos e ocultando campos

- Sobreposição (*overriding*): métodos com mesmo nome e assinatura idênticas.
 - Métodos devem ser **não estáticos**.





Sobrepondo métodos e ocultando campos

```
public class SuperMostra {  
    public String str = "superStr";  
  
    public void mostra() {  
        System.out.println("SuperMostra: " + str);  
    }  
}  
  
public class EstendeMostra extends SuperMostra {  
  
    public String str = "estendeStr";  
  
    public void mostra() {  
        System.out.println("EstendeMostra : " + str);  
    }  
}
```



Sobrepondo métodos e ocultando campos

```
public class Aplicacao {  
    public static void main(String[] args) {  
        EstendeMostra est = new EstendeMostra();  
        SuperMostra sup = est;  
  
        System.out.println("est.str = " + est.str);  
        System.out.println("sup.str = " + sup.str);  
  
        est.mostra();  
        sup.mostra();  
    }  
}
```



A palavra-chave `super`

- Disponível para todos os métodos não estáticos de uma classe estendida.
- Pode corresponder ao método equivalente da superclasse.
- Pode corresponder a uma referência explícita à superclasse.
 - `super.método` sempre utiliza a implementação da superclasse do método.



A palavra-chave `final`

- Métodos `final`:
 - Significa que nenhuma classe estendida poderá sobrepor o método.
- Classes `final`
 - Não pode ser sub-classificada (herdada) por outra classe, e todos os métodos são `final`.
 - Segurança: Comportamento não mudará
 - Otimização: Define previamente qual método será chamado (evita *late binding*).

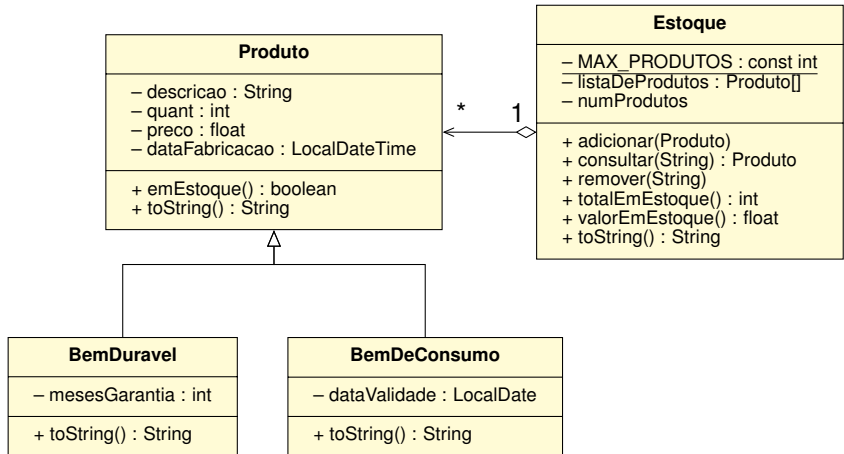


A classe Object

- Todas as classes estendem a classe Object, direta ou indiretamente.
 - Métodos de utilidade geral
 - Métodos de linha (fluxo) de execução (threads)
- Alguns métodos de utilidade geral:
 - **public boolean** equals(Object obj): compara estado de dois objetos.
 - **public int** hashCode(): retorna código hash para uso em tabelas HashTable.
 - **protected** Object clone() **throws** ClonenotSupportedException: retorna um clone.
 - **public final** Class getClass() : retorna o objeto do tipo class que representa a classe deste objeto.



Exemplo: Estoque polimórfico





Exemplo: Produto

```
public class Produto {  
    ...  
  
    /**  
     * Método sobreposto da classe Object.  
     */  
    @Override  
    public String toString() {  
        return "Produto: " + id + " - " + descricao  
            + "    Preço: R$" + preco + "    Quant.: " + quant  
            + "    Fabricação: " + dataFabricacao;  
    }  
  
    @Override  
    protected void finalize() throws Throwable {  
        super.finalize();  
        System.out.println("Finalizando um produto ....");  
        instancias --;  
    }  
}
```



Exemplo: Bem de Consumo e Bem Durável

```
public class BemDeConsumo extends Produto {  
    ...  
  
    @Override  
    public String toString() {  
        return super.toString() + " Data de Validade: " + dataValidade;  
    }  
}  
  
public class BemDuravel extends Produto {  
    ...  
  
    @Override  
    public String toString() {  
        return super.toString() + " Garantia: " + mesesGarantia;  
    }  
}
```




Exemplo: Estoque Polimórfico

```
public class Estoque {  
    private static final int MAX_PRODUTOS = 100;  
    private Produto[] listaDeProdutos;  
    private int numProdutos;  
  
    public void adicionar(Produto p) {  
        if (numProdutos < MAX_PRODUTOS) {  
            listaDeProdutos[numProdutos++] = p;  
        }  
    }  
  
    ...  
  
    @Override  
    public String toString() {  
        StringBuilder valor = new StringBuilder();  
        for (int i = 0; i < numProdutos; i++) {  
            valor.append(listaDeProdutos[i] + "\n");  
        }  
        return valor.toString();  
    }  
  
    public Estoque() {  
        listaDeProdutos = new Produto[MAX_PRODUTOS];  
        numProdutos = 0;  
    }  
}
```



Obrigado!!

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

"Devemos ousar inventar o futuro"