

Programação Orientada a Objetos

Prof. Evandro César Freiburger

Herança de Implementação

- Uma das características mais fortes o modelo de desenvolvimento orientado a objetos é a capacidade de gerar uma classe a partir da herança de outra;
- Em algumas linguagens (C++ por exemplo) é possível gerar uma classe a partir de mais de uma classe – isso é chamado de herança múltipla;
- Em Java apenas uma classe pode ser usada como base para outra – herança simples;
- A palavra reservada para implementar herança em Java é extends;

Herança de Implementação

- Em Java não é possível definir classes a partir do "zero"

```
public class Produto{  
}
```

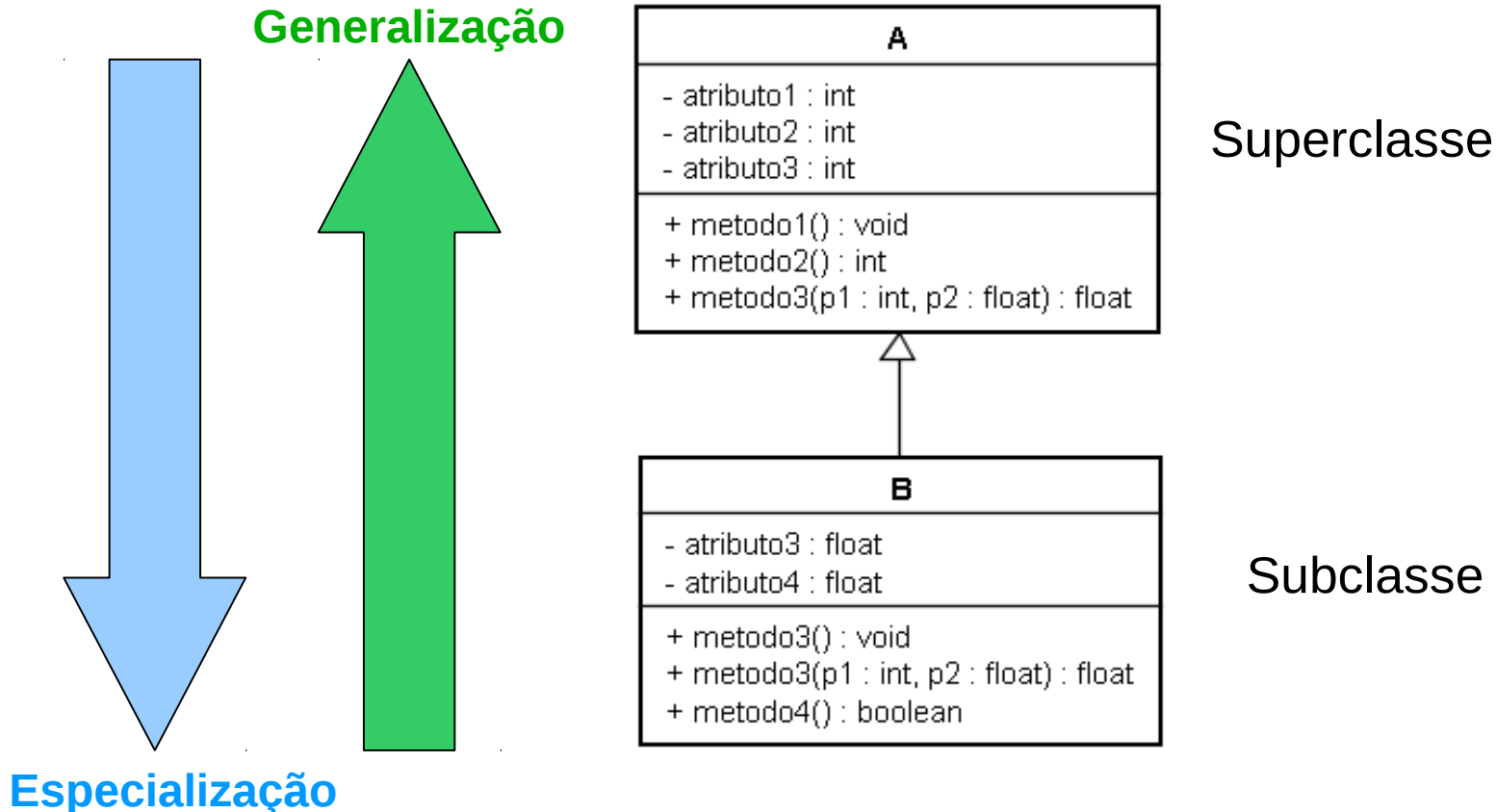
É equivalente a:

```
public class Produto extends Object{  
}
```

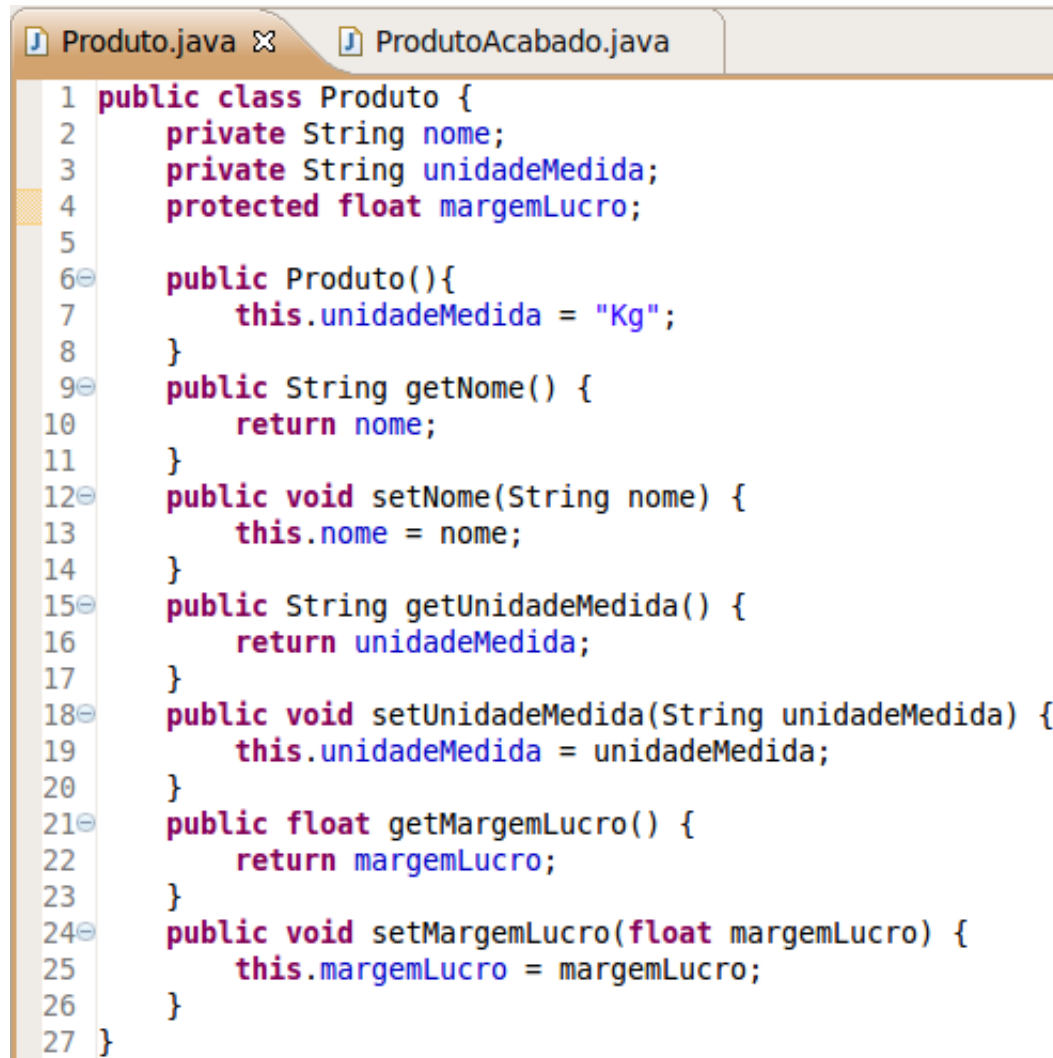
Object é a classe base de todas as classes Java

Herança de Implementação

- Quando não é inserido uma classe base, o compilador considera a herança da classe Object, que é a classe base de todas as classes do java;



Herança de Implementação

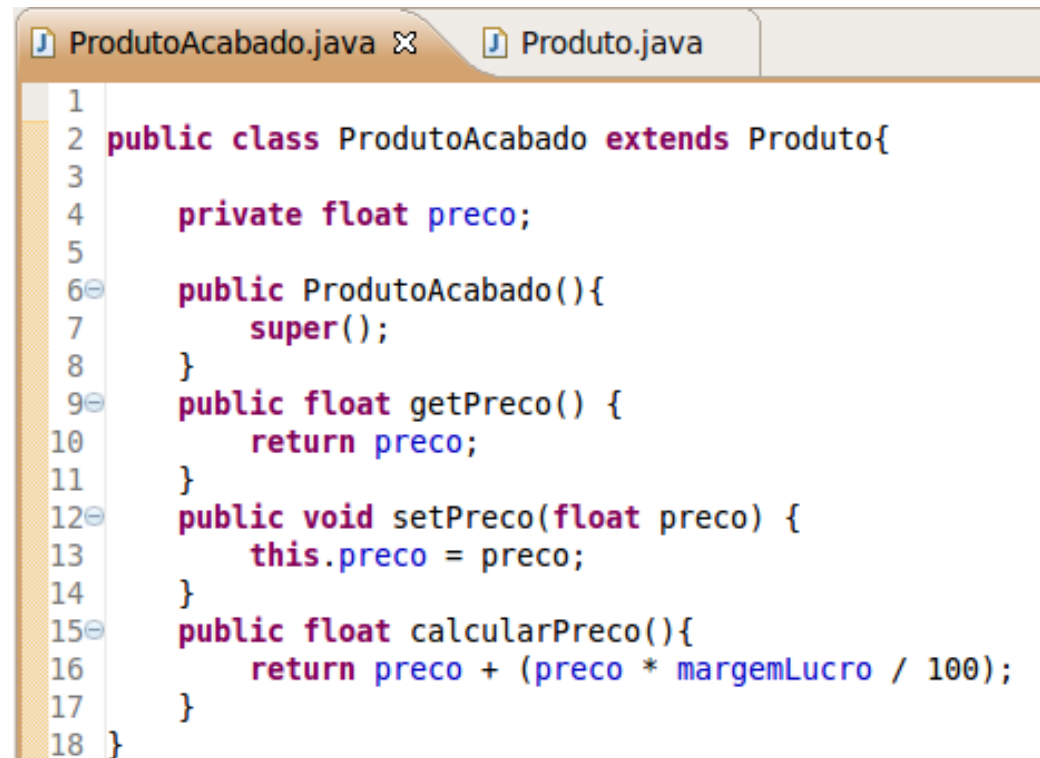


The image shows a screenshot of a Java IDE with two tabs: 'Produto.java' and 'ProdutoAcabado.java'. The 'Produto.java' tab is active, displaying the source code for the 'Produto' class. The code is as follows:

```
1 public class Produto {
2     private String nome;
3     private String unidadeMedida;
4     protected float margemLucro;
5
6     public Produto(){
7         this.unidadeMedida = "Kg";
8     }
9     public String getNome() {
10         return nome;
11     }
12     public void setNome(String nome) {
13         this.nome = nome;
14     }
15     public String getUnidadeMedida() {
16         return unidadeMedida;
17     }
18     public void setUnidadeMedida(String unidadeMedida) {
19         this.unidadeMedida = unidadeMedida;
20     }
21     public float getMargemLucro() {
22         return margemLucro;
23     }
24     public void setMargemLucro(float margemLucro) {
25         this.margemLucro = margemLucro;
26     }
27 }
```

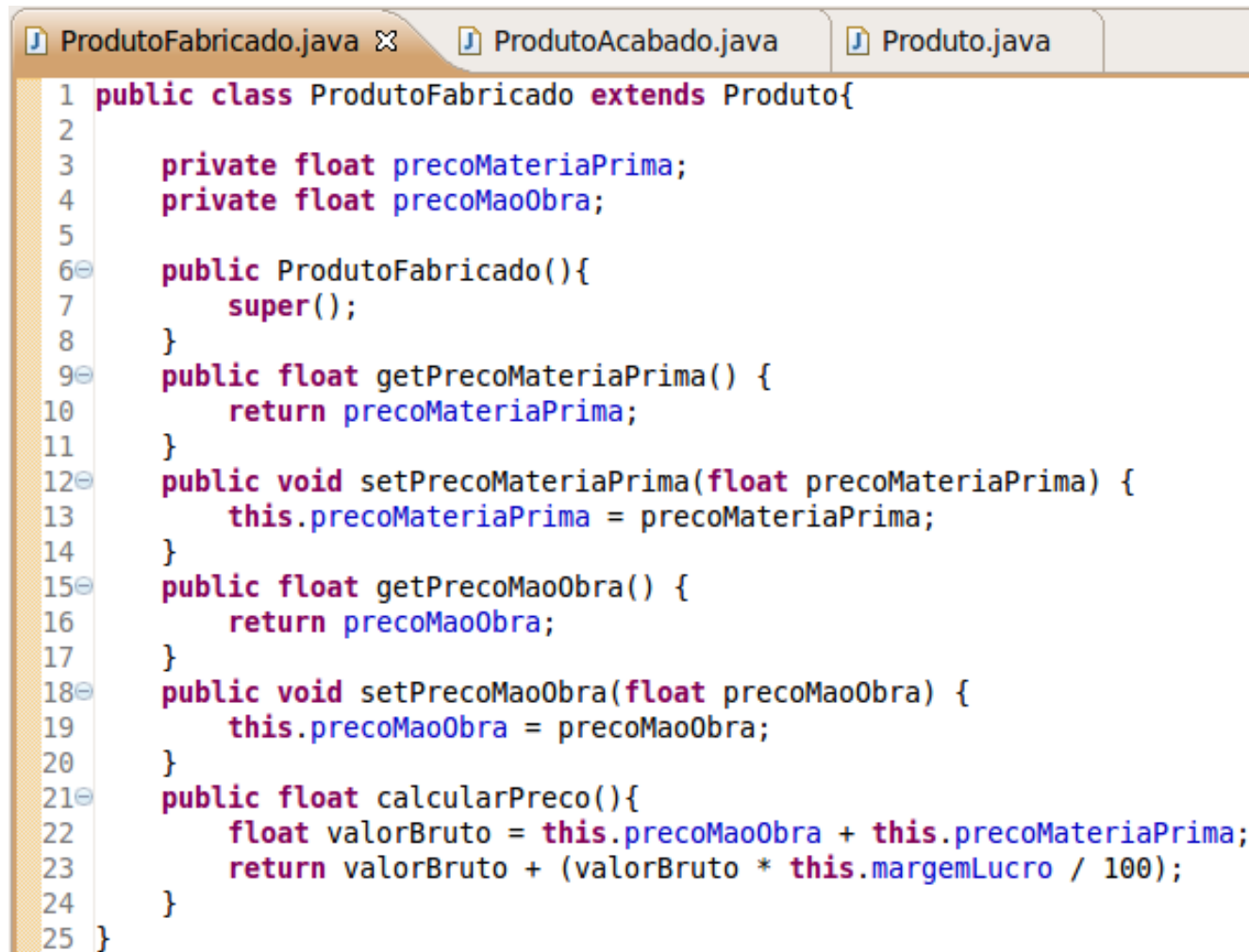
Herança de Implementação

- ProdutoAcabado especializa a classe Produto
- Adiciona características particulares
- Herda a implementação da classe Produto



```
1
2 public class ProdutoAcabado extends Produto{
3
4     private float preco;
5
6     public ProdutoAcabado(){
7         super();
8     }
9     public float getPreco() {
10         return preco;
11     }
12     public void setPreco(float preco) {
13         this.preco = preco;
14     }
15     public float calcularPreco(){
16         return preco + (preco * margemLucro / 100);
17     }
18 }
```

Herança de Implementação



The image shows a screenshot of a Java IDE with three tabs: 'ProdutoFabricado.java', 'ProdutoAcabado.java', and 'Produto.java'. The 'ProdutoFabricado.java' tab is active, displaying the following code:

```
1 public class ProdutoFabricado extends Produto{
2
3     private float precoMateriaPrima;
4     private float precoMaoObra;
5
6     public ProdutoFabricado(){
7         super();
8     }
9     public float getPrecoMateriaPrima() {
10         return precoMateriaPrima;
11     }
12     public void setPrecoMateriaPrima(float precoMateriaPrima) {
13         this.precoMateriaPrima = precoMateriaPrima;
14     }
15     public float getPrecoMaoObra() {
16         return precoMaoObra;
17     }
18     public void setPrecoMaoObra(float precoMaoObra) {
19         this.precoMaoObra = precoMaoObra;
20     }
21     public float calcularPreco(){
22         float valorBruto = this.precoMaoObra + this.precoMateriaPrima;
23         return valorBruto + (valorBruto * this.margemLucro / 100);
24     }
25 }
```

Herança de Implementação

AplicacaoProdutoAcabado.java

```
1 public class AplicacaoProdutoAcabado {  
2     public static void main(String[] args) {  
3  
4         ProdutoAcabado pro = new ProdutoAcabado();  
5         pro.setNome("Refrigerente 2L");  
6         pro.setUnidadeMedida("Un");  
7         pro.setPreco(2.0f);  
8         pro.setMargemLucro(20.0f);  
9  
10        System.out.println("-----");  
11        System.out.println("Nome: "+pro.getNome());  
12        System.out.println("PrecoVenda: "+pro.calcularPreco());  
13    }  
14 }
```


Herança de Implementação

AplicacaoProdutoFabricado.java ✕

```
1 public class AplicacaoProdutoFabricado {
2     public static void main(String[] args) {
3
4         ProdutoFabricado pro = new ProdutoFabricado();
5         pro.setNome("Pão Francês");
6         pro.setUnidadeMedida("Kg");
7         pro.setPrecoMateriaPrima(2.0f);
8         pro.setPrecoMaoObra(1.0f);
9         pro.setMargemLucro(30.0f);
10
11         System.out.println("-----");
12         System.out.println("Nome: "+pro.getNome());
13         System.out.println("PrecoVenda: "+pro.calcularPreco());
14     }
15 }
```

Classes Abstratas

- Em algumas situações é necessário tratar uma hierarquia de herança como se fossem do mesmo tipo;
- Por exemplo:
 - Se fosse necessário criar um vetor de objetos ProdutoAcabado e ProdutoFabricado, com qualquer quantidade e qualquer ordem desses objetos;
 - A partir do vetor obter o valor de venda de cada produto;
- Em OO uma superclasse é um tipo genérico para suas subclasses;

Classes Abstratas

```
AplicacaoProdutos.java x AplicacaoProdutoAcabado.java
1 public class AplicacaoProdutos {
2     public static void main(String[] args) {
3
4         Produto produtos[] = new Produto[2];
5
6         ProdutoFabricado pro1 = new ProdutoFabricado();
7         pro1.setNome("Pão Francês");
8         pro1.setUnidadeMedida("Kg");
9         pro1.setPrecoMateriaPrima(2.0f);
10        pro1.setPrecoMaoObra(1.0f);
11        pro1.setMargemLucro(30.0f);
12
13        ProdutoAcabado pro2 = new ProdutoAcabado();
14        pro2.setNome("Refrigerente 2L");
15        pro2.setUnidadeMedida("Un");
16        pro2.setPreco(2.0f);
17        pro2.setMargemLucro(20.0f);
18
19        produtos[0] = pro1;
20        produtos[1] = pro2;
21
22        for(Produto pro : produtos){
23            System.out.println("-----");
24            System.out.println("Nome: "+pro.getNome());
25            System.out.println("PrecoVenda: "+pro.calcularPreco());
26        }
27    }
28 }
```

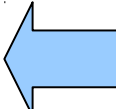
Gera erro, pois o
tipo genérico
Produto não
possui o método
calcularPreco()

Classes Abstratas

- Implementar o método concreto calcularPreco() na classe Produto, não tem sentido, pois nela não existem informações necessárias para o cálculo;
- A saída é usar o recurso de classes Abstratas;
- Na verdade usar o recurso de método abstrato;
- Método abstrato não possui corpo, apenas assinatura;
- Quando uma classe possui pelo menos um método abstrato, ela se torna abstrata;

Classes Abstratas

```
ProdutoAbstrato.java x AplicacaoProdutos.java AplicacaoP
1 public abstract class ProdutoAbstrato {
2     private String nome;
3     private String unidadeMedida;
4     protected float margemLucro;
5     public ProdutoAbstrato(){
6         this.unidadeMedida = "Kg";
7     }
8     public String getNome() {
9         return nome;
10    }
11    public void setNome(String nome) {
12        this.nome = nome;
13    }
14    public String getUnidadeMedida() {
15        return unidadeMedida;
16    }
17    public void setUnidadeMedida(String unidadeMedida) {
18        this.unidadeMedida = unidadeMedida;
19    }
20    public float getMargemLucro() {
21        return margemLucro;
22    }
23    public void setMargemLucro(float margemLucro) {
24        this.margemLucro = margemLucro;
25    }
26
27    public abstract float calcularPreco();
28 }
```



Classes Abstratas

- Mudanças em ProdutoAcabado e ProdutoFabricado;

```
public class ProdutoAcabado extends ProdutoAbstrato{
```

```
public class ProdutoFabricado extends ProdutoAbstrato{
```

Classes Abstratas

```
AplicacaoProdutos.java x AplicacaoProdutoAcab ProdutoAcabado.java
1 public class AplicacaoProdutos {
2     public static void main(String[] args) {
3
4         ProdutoAbstrato produtos[] = new ProdutoAbstrato[2];
5
6         ProdutoFabricado pro1 = new ProdutoFabricado();
7         pro1.setNome("Pão Francês");
8         pro1.setUnidadeMedida("Kg");
9         pro1.setPrecoMateriaPrima(2.0f);
10        pro1.setPrecoMaoObra(1.0f);
11        pro1.setMargemLucro(30.0f);
12
13        ProdutoAcabado pro2 = new ProdutoAcabado();
14        pro2.setNome("Refrigerente 2L");
15        pro2.setUnidadeMedida("Un");
16        pro2.setPreco(2.0f);
17        pro2.setMargemLucro(20.0f);
18
19        produtos[0] = pro1;
20        produtos[1] = pro2;
21
22        for(ProdutoAbstrato pro : produtos){
23            System.out.println("-----");
24            System.out.println("Nome: "+pro.getNome());
25            System.out.println("PrecoVenda: "+pro.calcularPreco());
26        }
27    }
28 }
```

Polimorfismo dinâmico,
decidido em tempo de
execução;