

Programação Orientada a Objetos

Herança, subtipos e
redefinição de métodos

Sérgio Soares
scbs@cin.ufpe.br

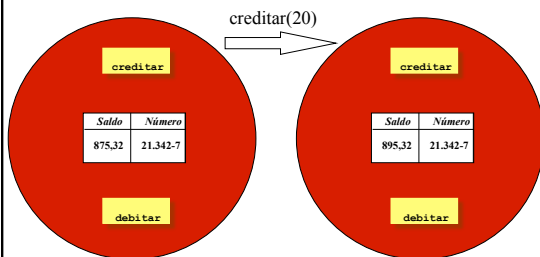
Introdução

- Imagine que temos uma aplicação bancária com
 - Uma classe `Conta`, que possui número, saldo, e os métodos `creditar` e `debitar`
 - Uma classe `Banco` que possui um array de `Conta` (**lembra do cadastro de contas?**), que armazena as contas do banco

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

2

Estados do Objeto Conta



Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

3

Motivação

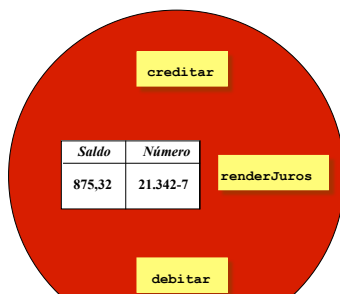
- Imagine agora que surge um novo requisito
 - O banco precisa trabalhar com poupanças que rendem juros uma vez por mês

O QUE FAZER?

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

4

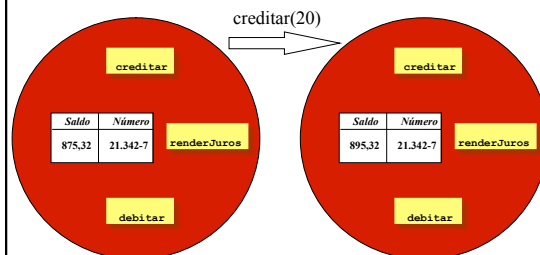
Objeto Poupança



Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

5

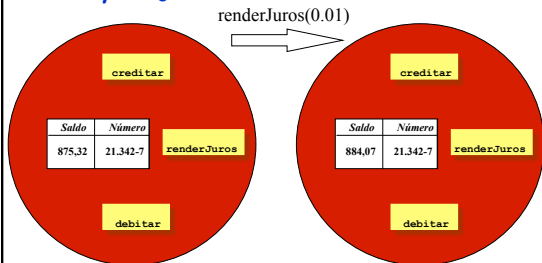
Estados do Objeto Poupança



Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

6

Estados do Objeto Poupança



Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

7

Classe de Poupanças: Assinatura

```
public class PoupancaD {
    public PoupancaD (String n) {}
    public void creditar(double valor) {}
    public void debitar(double valor) {}
    public String getNumero() {}
    public double getSaldo() {}
    public void renderJuros(double taxa) {}
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

8

Classe de Poupanças: Descrição

```
public class PoupancaD {
    private String numero;
    private double saldo;
    public void creditar (double valor) {
        saldo = saldo + valor;
    } // ...
    public void renderJuros(double taxa) {
        this.creditar(saldo * taxa);
    }
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

9

E a aplicação bancária?

- Precisamos alterar a classe Banco (que tem um array de Conta) para trabalhar com objetos Poupanca

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

10

Classe de Bancos: Assinatura

```
public class BancoD {
    public BancoD() {}
    public void cadastrarConta(Conta c) {}
    public void creditarConta(String numero,
                             double valor) {}
    public void cadastrarPoupanca(PoupancaD p) {}
    public void creditarPoupanca(String numero,
                                 double valor) {}
    // ...
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

11

Classe de Bancos: Descrição (1)

```
public class BancoD {
    private Conta[] contas;
    private int indiceC;
    private PoupancaD[] poupancas;
    private int indiceP;
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

12

Classe de Bancos: Descrição (2)

```
public void cadastrarConta(Conta c) {
    contas[indiceC] = c;
    indiceC = indiceC + 1;
}

public void cadastrarPoupanca(PoupancaD p) {
    poupancas[indiceP] = p;
    indiceP = indiceP + 1;
}
```

qual a diferença?

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

13

Classe de Bancos: Descrição (3)

```
private Conta procurarConta(String numero) {
    int i = 0;
    boolean achou = false;
    Conta resposta = null;
    while ((! achou) && (i < indiceC)) {
        if (contas[i].getNumero().equals(numero)) {
            achou = true;
            resposta = contas[i];
        } else {
            i = i + 1;
        }
    }
    if (!achou)
        throw new RuntimeException("Não achou");
    return resposta;
}
```

Atenção: Por enquanto vamos indicar erros assim, mas isso vai mudar

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

Classe de Bancos: Descrição (4)

```
public void debitarConta(String numero,
                        double valor) {
    Conta c;
    c = this.procurarConta(numero);
    c.debitar(valor);
}

public void creditarConta(String numero,
                        double valor) {
    Conta c;
    c = this.procurarConta(numero);
    c.creditar(valor);
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

15

Problemas

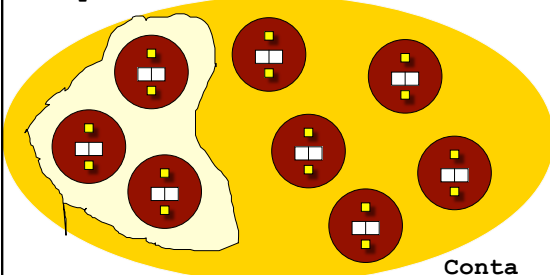
- Duplicação desnecessária de código:
 - a definição de PoupancaD é uma simples extensão da definição de Conta
 - clientes de Conta que precisam trabalhar também com PoupancaD terão que ter código especial para manipular poupanças
- Falta refletir relação entre tipos do "mundo real"

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

16

Subtipos e Subclasses

Poupanca



Conta

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

17

Herança

- Necessidade de estender classes
 - alterar classes já existentes e adicionar propriedades ou comportamentos para representar outra classe de objetos
 - criar uma hierarquia de classes que "herdam" propriedades e comportamentos de outra classe e definem novas propriedades e comportamentos

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

18

Subclasses

- **Comportamento**
objetos da subclasse comportam-se como os objetos da superclasse
- **Substituição**
objetos da subclasse podem ser usados no lugar de objetos da superclasse

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

19

Herança

- **Reuso de Código**
a descrição da superclasse pode ser usada para definir a subclasse
- **Extensibilidade**
algumas operações da superclasse podem ser redefinidas na subclasse

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

20

Classe de Poupanças: Com herança

```
public class Poupanca extends Conta {  
    public Poupanca (String numero) {  
        super(numero);  
    }  
    public void renderJuros(double taxa) {  
        double juros = this.getSaldo() * taxa;  
        this.creditar(juros);  
    }  
}
```

Construtores da superclasse não são herdados, mas **devem** ser utilizados (via **super**)

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

21

Extends

- **subclasse extends superclasse**
- Mecanismo para definição de herança e subtipos
- Herança simples: só pode-se herdar uma classe por vez

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

22

Extends: Restrições (1)

- Atributos e métodos privados são herdados, mas não podem ser acessados diretamente

```
public class Poupanca extends Conta {  
    public Poupanca (String numero) {  
        super(numero);  
    }  
    public void renderJuros(double taxa) {  
        double juros = this.saldo * taxa;  
        this.saldo = this.saldo + juros;  
    }  
}
```

Erro

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

23

Extends: Restrições (2)

- Qualificador **protected**: visibilidade restrita ao pacote e as subclasses em outros pacotes

```
package br.ufpe.cin.banco;  
public class Conta {  
    protected double saldo;  
    //...
```

Funciona, mas não use!

```
package br.ufpe.cin.bancoItau;  
import br.ufpe.cin.banco.Conta;  
public class Poupanca extends Conta {  
    //...  
    public void renderJuros(double taxa) {  
        double juros = this.saldo * taxa;  
        this.saldo = this.saldo + juros;  
    }  
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

24

Extends: Restrições (3)

- Construtor default só é disponível se também for disponível na superclasse

```
public class Poupanca extends Conta {  
    public Poupanca () {  
        super();  
    }  
    // ...  
}
```

Só funciona se Conta também tiver um construtor default

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

25

Usando Poupanças

(testando a classe em um método *main*)

```
...  
Poupanca poupanca;  
poupanca = new Poupanca("21.342-7");  
poupanca.creditar(500.87);  
poupanca.debitar(45.00);  
System.out.println(poupanca.getSaldo());  
...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

26

Subtipos: Substituição

```
...  
Conta conta;  
conta = new Poupanca("21.342-7");  
conta.creditar(500.87);  
conta.debitar(45.00);  
System.out.println(conta.getSaldo());  
...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

27

Herança (parte 2)

Redefinição de métodos

Na aula passada...

- Definimos a classe *Poupanca*, uma subclasse de *Conta*
- Vimos que o *Banco* pode trabalhar tanto com *Conta* quanto com *Poupanca* (sem a necessidade de código especial)
- Mas o *Banco* precisa ter código específico para lidar com *Poupanca*

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

29

Finalizando a implementação de Banco

- Como fazer o *Banco* render juros de uma *Poupanca* que esta no array de *Conta*?

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

30

Funciona????

```
...
Conta conta;
conta = new Poupanca("21.342-7");
...
conta.renderJuros(0.01);
System.out.println(conta.getSaldo());
...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

31

Subtipos: Verificação Dinâmica com Casts

```
...
Conta conta;
conta = new Poupanca("21.342-7");
...
((Poupanca) conta).renderJuros(0.01);
System.out.println(conta.getSaldo());
...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

32

Substituição e Casts (1)

- Nos contextos onde contas são usadas pode-se usar poupanças
- Nos contextos onde poupanças são usadas pode-se usar contas com o uso explícito de *casts*

```
Conta conta;
conta = new Poupanca("21.342-7");
((Poupanca) conta).renderJuros(0.01);
System.out.println(conta.getSaldo());
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

33

Substituição e Casts (2)

- *Casts* correspondem a verificação dinâmica de tipos e podem gerar exceções (Cuidado!)
- *Casts* não fazem conversão de tipos

```
Conta conta;
conta = new Conta("21.342-7");
((Poupanca) conta).renderJuros(0.01);
System.out.println(conta.getSaldo());
```

Erro em tempo de execução

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

34

Classe Banco: Assinatura

```
public class Banco {
    public Banco () {}
    public void cadastrar(Conta conta) {}
    public void creditar(String numero,
                        double valor) {}
    public void debitar(String numero,
                       double valor) {}
    public double getSaldo(String numero) {}
    public void transferir(String contaOrigem,
                          String contaDestino,
                          double valor) {}
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

35

Subtipos: Substituição

```
...
Banco banco = new Banco();
banco.cadastrar(new Conta("123-4"));
banco.cadastrar(new Poupanca("567-8"));
banco.creditar("123-4", 129.34);
banco.transferir("123-4", "567-8", 9.34);
System.out.print(banco.getSaldo("567-8"));
...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

36

Subtipos: Verificação Dinâmica com instanceof

```
...
Conta c = this.procurar("567-8");
if (c instanceof Poupanca)
    ((Poupanca) c).renderJuros(0.01);
else
    throw new RuntimeException("Não é poupança");
...
```

Por enquanto vamos usar esse comando sempre que quisermos dar uma mensagem de erro

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

37

Verificação Dinâmica de Tipos (1)

- **Casts e instanceof:**
 - ((Tipo) variável)
 - variável instanceof Tipo
 - O tipo de variável deve ser supertipo de Tipo
 - deve haver uma relação de hierarquia

```
Conta conta;
conta = ...
((String) conta).length();
```

Código sem sentido: erro de compilação

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

38

Verificação Dinâmica de Tipos (2)

- **Casts e instanceof:**
 - Casts geram exceções quando instanceof retorna false

```
Conta c = this.procurar("567-8");
if (c instanceof Poupanca)
    ((Poupanca) c).renderJuros(0.01);
else
    ((Poupanca) c).renderJuros(0.01);
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

39

Verificação Dinâmica de Tipos (3)

- **Casts e instanceof:**
 - Casts são essenciais para verificação estática de tipos (compilação)

Não compila

```
Conta c = this.procurar("567-8");
if (c instanceof Poupanca)
    c.renderJuros(0.01);
else
    // ...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

40

Mas software sempre muda ...

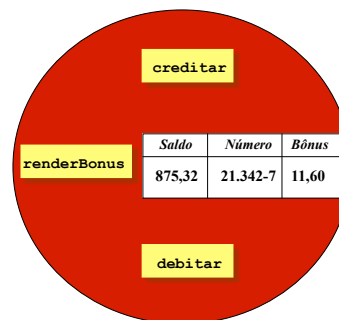
- Surge um novo requisito:
 - Precisamos criar uma conta especial que deve armazenar um bônus a cada crédito que a mesma receber
 - Além disso, a conta deve permitir render este bônus, somando seu valor ao saldo da conta

Atenção: O método creditar dessa nova conta funciona ligeiramente diferente que o de conta e poupança

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

41

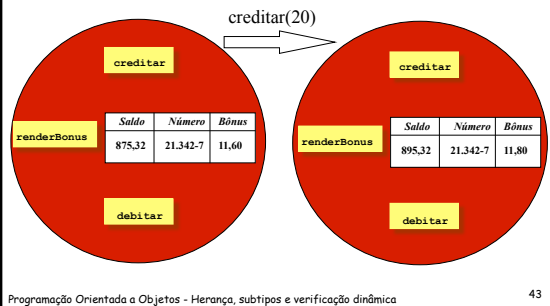
Objeto Conta Especial



Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

42

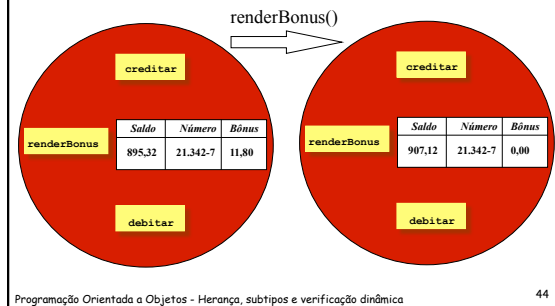
Estados de uma Conta Especial



Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

43

Estados de uma Conta Especial



Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

44

Contas Especiais: Assinatura

```
public class ContaEspecial extends Conta {
    public ContaEspecial(String numero) {}
    public void renderBonus() {}
    public double getBonus() {}
    public void creditar(double valor) {}
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

45

Contas Especiais: Descrição (1)

```
public class ContaEspecial extends Conta {
    private double bonus;
    public ContaEspecial(String numero) {
        super(numero);
        bonus = 0.0;
    }
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

46

Contas Especiais: Descrição (2)

```
public void creditar(double valor) {
    super.creditar(valor);
    bonus = bonus + (valor * 0.01);
}
public void renderBonus() {
    super.creditar(this.bonus);
    bonus = 0;
}
public double getBonus() {
    return this.bonus;
}
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

47

Redefinição de Métodos

- Invariância: tipos dos argumentos e resultados da *redefinição* tem que ser iguais aos tipos da *definição*
- Semântica e Visibilidade dos métodos redefinidos deve ser preservada
 - *creditar* de *ContaEspecial* deve fazer o mesmo que o *creditar* de *Conta* para os atributos herdados
- Só é possível acessar a definição dos métodos da superclasse imediata (via *super*)

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

48

Usando ContaEspecial (testando a classe em um método main)

```
...
ContaEspecial conta;
conta = new ContaEspecial("21.342-7");
conta.creditar(200.00);
conta.debitar(100.00);
conta.renderBonus();
System.out.print(conta.getSaldo());
...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

49

Ligações Dinâmicas

```
...
Conta conta;
conta = new ContaEspecial("21.342-7");
((Conta) conta).creditar(200.00);
conta.debitar(100.00);
if (conta instanceof ContaEspecial) {
    ((ContaEspecial) conta).renderBonus();
}
System.out.print(conta.getSaldo());
...
```

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

50

Ligações Dinâmicas

- Dois métodos com o mesmo nome e tipo:
 - definição e redefinição, qual será **executado?**
- O código é escolhido **dinamicamente** (em tempo de execução), não estaticamente (em tempo de compilação)
- Escolha é feita com base na classe do **objeto** associado à variável destino do método

Programação Orientada a Objetos - Herança, subtipos e verificação dinâmica

51