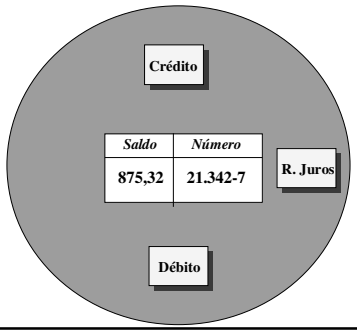
Introdução à Programação
Orientada a Objetos com Java

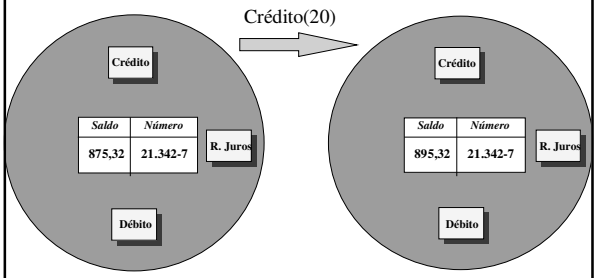
Herança, subtipos, polimorfismo
e dynamic binding

Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco

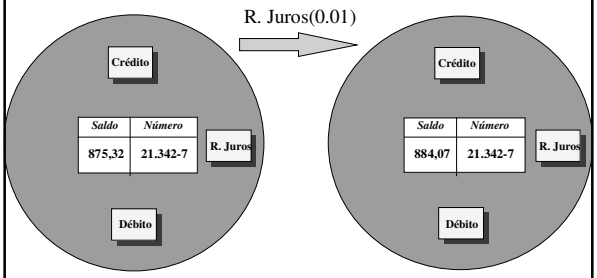
Objeto Poupança



Estados do Objeto Poupança



Estados do Objeto Poupança



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) { }
}
```

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);
    }
}
```

Classe de Bancos: Assinatura

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

Classe de Bancos: Descrição

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

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

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

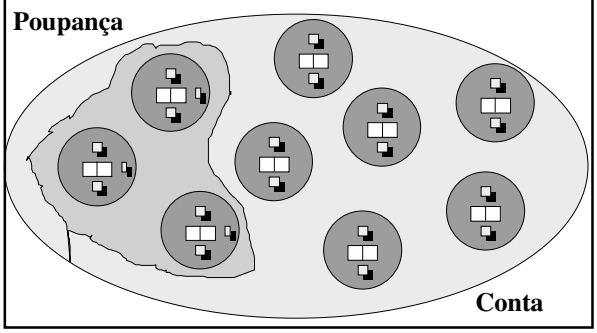
```
private Conta procurarConta(String numero) {
   int        i = 0;
   boolean achou = false;
   Conta resposta = null;
   while ((! achou) && (i < indiceC)) {
      if (contas[i].getNumero().equals(numero))
         achou = true;
      else
         i = i + 1;
   }
   if (achou) resposta = contas[i];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

```
   public void debitarConta(String numero,
                            double valor) {
      Conta c;
      c = this.procurarConta(numero);
      if (c != null)
         c.debitar(valor);
      else
         System.out.println("Conta inexistente!");
   }
```

Problemas

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

Subtipos e Subclasses



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

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

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

Classe de Poupanças: Assinatura

```
public class Poupanca extends Conta {
    public Poupanca (String numero) { }
    public void renderJuros(double taxa) { }
}
```

Classe de Poupanças: Descrição

```
public class Poupanca extends Conta {

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

Extends

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

Extends: Restrições

- Atributos e métodos privados são herdados, mas não podem ser acessados diretamente
- Qualificador protected: visibilidade restrita ao pacote e as subclasses de outros pacotes
- Construtores não são herdados
- Construtor *default* só é disponível se também for disponível na superclasse

Usando Poupanças

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

Subtipos: Substituição

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

Herança

- Polimorfismo
  - Uma conta pode ser
    - —uma poupança
    - —uma conta especial
  - Um transporte pode ser
    - —um carro
    - —um avião
    - —um barco

Subtipos: Verificação Dinâmica com Casts

```
...
Conta  conta;
conta = new Poupanca("21.342-7");
...
((Poupanca) conta).renderJuros(0.01);
conta.imprimirSaldo();
...
```

Substituição e Casts

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

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) {}
}
```

Subtipos: Substituição

```
...
Banco        banco = new Banco();
banco.cadastrar(new Conta("21.345-7"));
banco.cadastrar(new Poupanca("1.21.345-9"));
banco.creditar("21.345-7",129.34);
banco.transferir("21.345-7","1.21.345-9",9.34);
System.out.print(banco.getSaldo("1.21.345-9"));
...
```

Exercício

- Modifique a classe Banco para que seja possível render juros de uma poupança. Isto é, adicione um novo método que rende os juros da poupança cujo número é parâmetro deste método; a taxa de juros corrente deve ser um atributo de Banco.

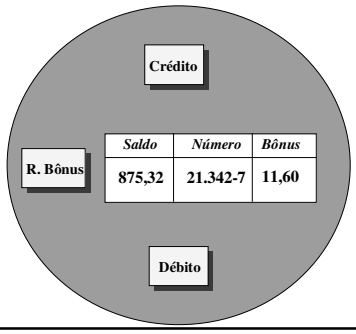
Subtipos: Verificação Dinâmica com instanceof

```
...
Conta c = this.procurar("123.45-8");
if (c instanceof Poupanca)
   ((Poupanca) c).renderJuros(0.01);
else
   System.out.print("Poupança inexistente!")
...
```

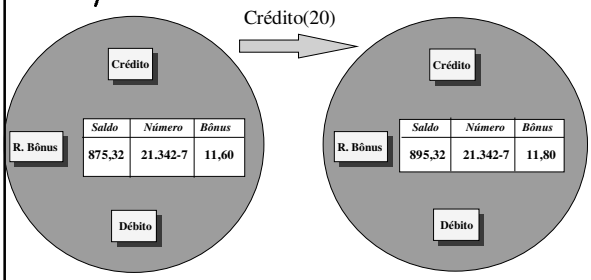
Verificação Dinâmica de Tipos

- *Casts* e instanceof:
  - ((Tipo) *variável*)
  - *variável* instanceof Tipo
  - O tipo de *variável* deve ser supertipo de Tipo
  - *Casts* geram exceções quando instanceof retorna *false*
  - *Casts* são essenciais para verificação estática de tipos (compilação)

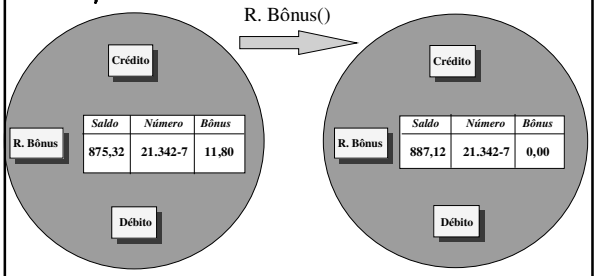
Objeto Conta Especial



Estados de uma Conta Especial



Estados de uma Conta Especial



Contas Especiais: Assinatura

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

Contas Especiais: Descrição

```
public class ContaEspecial extends Conta {

   private double bonus;

   public ContaEspecial (String numero) {
      super (numero);
      bonus = 0.0;
   }
```

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

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
- Só é possível acessar a definição dos métodos da superclasse imediata (via super)

Usando Contas Especiais

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

Ligações Dinâmicas

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

Ligações Dinâmicas

- Dois métodos com o mesmo nome e tipo: definição e redefinição, qual usar?
- 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