Orientação a Objetos e Java

Sérgio Soares
sergio@dsc.upe.br

Classes Abstratas

Objetivo

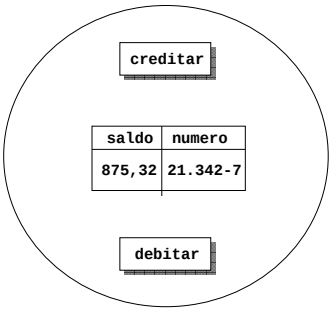
Depois desta aula você será capaz de desenvolver sistemas mais reusáveis e extensíveis, através da utilização de classes abstratas que permitem relacionar classes que compartilham parte dos seus códigos mas que se comportam de forma radicalmente diferente em alguns casos.

Classes Abstratas

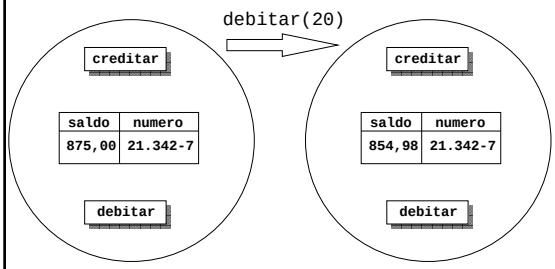
Leitura prévia essencial

- Seções 7.13 a 7.20 do livro *Java: how to program* (de Harvey e Paul Deitel)

Objeto Conta Imposto



Estados do Objeto Conta Imposto



Conta Imposto: Assinatura

```
public class ContaImposto {
  public ContaImposto (String numero) {}
  public void creditar(double valor) {}
  public void debitar(double valor) {}
  public String getNumero() {}
  public double getSaldo() {}
}
```

Conta Imposto: Assinatura

```
public class ContaImpostoM extends Conta {

  public ContaImpostoM(String numero) {}
  public void debitar(double valor) {}

}
```

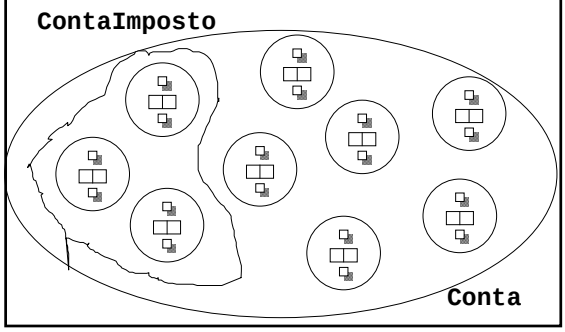
Conta Imposto: Descrição

```
public class ContaImpostoM extends Conta {

  private static final double TAXA = 0.0038;

  public ContaImpostoM (String numero) {
    super (numero);
  }
  public void debitar(double valor) {
    double imposto = (valor * taxa);
    super.debitar(valor + imposto);
  }
}
```

Subtipos e Subclasses



Subclasses e Comportamento

- Objetos da subclasse *comportam-se* como os objetos da superclasse
- Redefinições de métodos devem preservar o comportamento (semântica) do método original
- Grande impacto sobre manutenção/evolução de software...

Revisão/Otimização de Código

```
...
double m(Conta c) {
  c.creditar(x);
  c.debitar(x);
  return
    c.getSaldo();
}
...
```

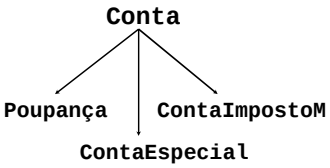
```
...
double m(Conta c) {
   return
     c.getSaldo();
}
...
```

Modificação é correta? Em que contextos?

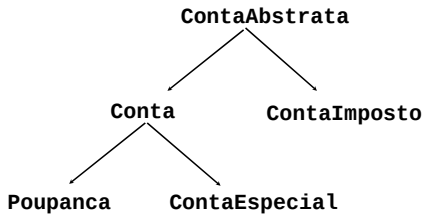
Subclasses e Evolução de Software

- Deveria ser possível raciocinar sobre o código usando-se apenas a definição dos tipos das variáveis envolvidas (**`Conta`**)
- O comportamento do código deveria ser independente do tipo do objeto (**`Conta`**, **`ContaEspecial`**, **`ContaImposto`**) associado a uma dada variável em tempo de execução

Reuso sem Subtipos



Reuso preservando Subtipos



Definindo Classes Abstratas

```
public abstract class ContaAbstrata {
  private String numero;
  private double saldo;
  public ContaAbstrata (String numero) {
    this.numero = numero;
    saldo = 0.0;
  }
  public void creditar(double valor) {
    saldo = saldo + valor;
  }
```

Definindo Classes Abstratas

```
  public double getSaldo() {
    return saldo;
  }
  public String getNumeto() {
    return numero;
  }
  public abstract void debitar(double valor);
  protected void setSaldo(double saldo) {
    this.saldo = saldo;
  }
}
```

Revisão/Otimização de Código

```
...
double m(ContaA c) {
  c.creditar(x);
  c.debitar(x);
  return
    c.getSaldo();
}
...
```

⇨

```
...
double m(ContaA c) {
   return
     c.getSaldo();
}
...
```

Modificação é correta? Em que contextos?

Classes Abstratas

- Possibilita herança de código preservando comportamento (semântica)
- Métodos abstratos:
  - geralmente existe pelo menos um
  - são implementados nas subclasses
- Não cria-se objetos:
  - mas podem (devem) ter construtores, para reuso
  - métodos qualificados como **`protected`** para serem acessados nas subclasses

Contas: Descrição Modificada

```
public class Conta extends ContaAbstrata {
  public Conta(String numero) {
    super (numero);
  }
  public void debitar(double valor) {
    this.setSaldo(getSaldo() - valor);
  }
}
```

Poupanças: Descrição Original

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

Conta Especial: Descrição Original

```
public class ContaEspecial extends Conta {
  public static final double TAXA = 0.01;
  private double bonus;
  public ContaEspecial (String numero) {
    super (numero);
  }
  public void creditar(double valor) {
    bonus = bonus + (valor * TAXA);
    super.creditar(valor);
  }
  ...
}
```

Conta Imposto: Descrição

```
public class ContaImposto
      extends ContaAbstrata {
  public static final double TAXA = 0.0038;
  public ContaImposto (String numero) {
    super (numero);
  }
  public void debitar(double valor) {
    double imposto = valor * TAXA;
    double total = valor + imposto;
    this.setSaldo(getSaldo() – total);
  }
}
```

Substituição e Ligações Dinâmicas

```
...
ContaAbstrata  ca, ca’;
ca = new ContaEspecial(¨21.342-7¨);
ca’ = new ContaImposto(¨21.987-8¨);
ca.debitar(500);
ca’.debitar(500);
System.out.println(ca.getSaldo());
System.out.println(ca’.getSaldo());
...
```

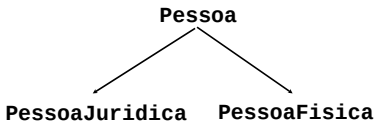
Classes Abstratas: Utilização

- Herdar código sem quebrar noção de subtipos, preservando o comportamento do supertipo
- Generalizar código, através da abstração de detalhes não relevantes
- Projetar sistemas, definindo as suas arquiteturas e servindo de base para a implementação progressiva dos mesmos

Contas: Projeto OO

```
public abstract class ContaProjeto {
  private String numero;
  private double saldo;
  public abstract void creditar(double valor);
  public abstract void debitar(double valor);
  public String getNumero() {
    return numero;
  protected setSaldo(double saldo) {
    this.saldo = saldo;
  }
  ...
}
```

Pessoa: Reuso e Subtipos



Pessoa: Projeto OO

```
public abstract class Pessoa {
  private String nome;
  ...
  public abstract String getCodigo();
}
```

Pessoa Física: Projeto OO

```
public class PessoaFisica
                        extends Pessoa {
  private String cpf;
  ...
  public String getCodigo() {
    return cpf;
  }
}
```

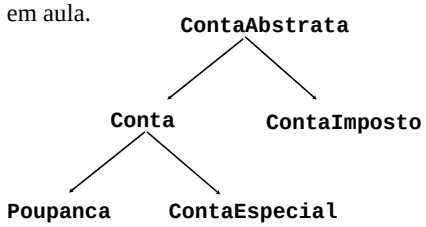
Pessoa Jurídica: Projeto OO

```
public class PessoaJuridica
                      extends Pessoa {
  private String cnpj;
  ...
  public String getCodigo() {
    return cnpj;
  }
}
```

```
public class RepositorioPessoasArray {
  private Pessoa[] pessoas;
  ...
  public Pessoa procurar(String codigo) {
    Pessoa p = null;
    boolean achou = false;
    for (int i=0; i<indice && !achou; i++) {
      p = pessoas[i];
      if (p.getCodigo().equals(codigo))
        achou = true;
      else
        p = null;
    }
    return p;
  }
}
```

Exercícios

- Modifique a classe Banco para que seja possível armazenar todos os tipos de contas vistos em aula.



Classes Abstratas

Resumo

- Importância de redefinir métodos preservando a semântica dos métodos originais
- Cláusula `abstract` para classes
- Cláusula `abstract` para métodos
- Classes abstratas e projeto e estruturação de sistemas

Classes Abstratas

Leitura adicional

- Capítulo 7 do livro *Thinking in Java* (de Bruce Eckel)
- Seções 4.7 e 4.10 do livro *A Programmer's Guide to Java Certification* (de Khalid Mughal e Rolf Rasmussen)