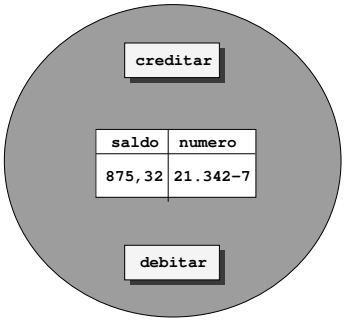
Introdução à Programação
Orientada a Objetos com Java

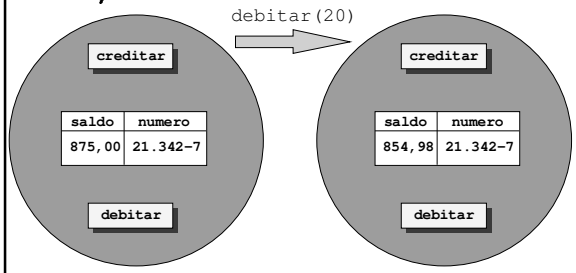
Classes Abstratas

Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco

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) {}
}
```

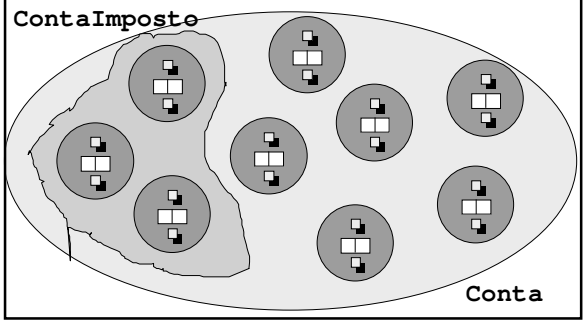
Conta Imposto: Descrição

```
public class ContaImpostoM extends Conta {

    private static final double taxa = 0.001;

    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();
}
...
```

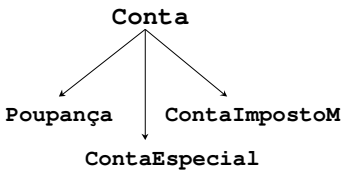
```
...
double m(Conta c) {
   return
     c.getSaldo();
}
...
```

Modificação é correta? Em que contextos?

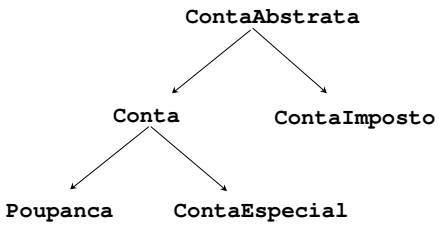
Subclasses e Evolução de Software

- 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 abstract void debitar(double valor);
    protected void setSaldo(double saldo) {
        this.saldo = saldo;
    }
    public double getSaldo() {
        return saldo;
    }
    /* ... */
}
```

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 (n);
    }
    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 = 0.001;

    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

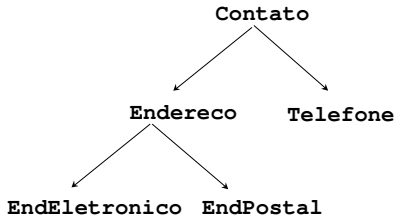
Contas: Projeto OO

```
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;
    }
    /* ... */
}
```

Cliente: Projeto OO

```
public abstract class Cliente {
    private String nome;
    private ConjuntoContato contatos;
    /* ... */
    public void incluirContato(Contato contato) {
        contatos.incluir(contato);
    }
    public abstract Endereco getEndereco();
    public abstract Contato getContato(String tipo);
    /* ... */
}
```

Contato: Reuso e Subtipos



Contato: Projeto OO

```
public abstract class Contato {
    private String tipo;
    public Contato (String tipo) {
        this.tipo = tipo;
    }
    public abstract String getInfoRotulo();
}
```

Endereço: Projeto OO

```
public abstract class Endereco extends Contato {
    public Endereco (String tipo) {
        super (tipo);
    }
}
```

Endereço Eletrônico: Projeto OO

```
public class EnderecoEletronico extends Endereco {
    private String email;
    public EnderecoEletronico(String email) {
        super ("EnderecoEletronico");
        this.email = email;
    }
    public String getInfoRotulo() {
        return ("Email: " + email);
    }
}
```

Endereço Residencial: Projeto

```
public class EnderecoPostal extends Endereco {
    private String rua;
    private String cidade;
    // ...
    public EnderecoPostal(String cidade,
                          String rua,/*...*/) {
        super ("EnderecoPostal");
        this.cidade = cidade;
        this.rua = rua;
        // ...
    }
    public String getInfoRotulo() {
        return ("Rua: " + rua /*...*/);
    }
}
```

Telefone: Projeto

```
public class Telefone extends Contato {
    private String ddd;
    private String numero;
    public Telefone(String ddd, String numero) {
        super ("Telefone");
        this.numero = numero;
        this.ddd = ddd;
    }
    public String getInfoRotulo() {
        return ("DDD: " + ddd +
                "Numero: " + numero);
    }
}
```