

Introdução à Programação



Classes Abstratas

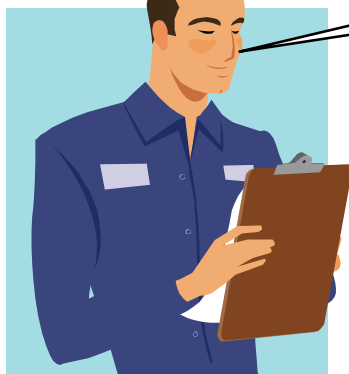
Classes Abstratas



Como representar um conceito tão genérico?

Quero que você escreva uma classe que represente Veiculos.

Sem problema, chefe.



**Classes
Abstratas!**



Tópicos da Aula

- ◆ Hoje, aprenderemos a trabalhar com classes abstratas
 - Conceito
 - Importância
 - Utilização
 - Estrutura de uma classe abstrata em Java
 - Usando classe abstrata com herança
- ◆ Veremos, ainda, duas técnicas de refactoring
 - Extract class
 - Padrão template method
 - Uso com classes abstratas

Classe Abstrata

- ◆ Um classe abstrata é uma classe que representa um conceito genérico que está parcialmente descrita
 - Pode conter métodos implementados
 - Pode conter **métodos abstratos**, ou seja, métodos sem nenhuma implementação
- ◆ Representa um conceito, mas sem se preocupar com alguns detalhes de implementação
- ◆ Comportamento genérico é geralmente implementado

Importância de Classes Abstratas

- ◆ Possibilita herança de código preservando comportamento (semântica)
 - O que é genérico é herdado
 - O que é específico é implementado nas subclasses
- ◆ Adia especificidades de implementação para fases posteriores de desenvolvimento
 - Quem se preocupa é desenvolvedor de subclasse
- ◆ Grande impacto sobre manutenção/evolução de software...
 - Pode aumentar reuso e extensibilidade
 - Torna o software mais flexível

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

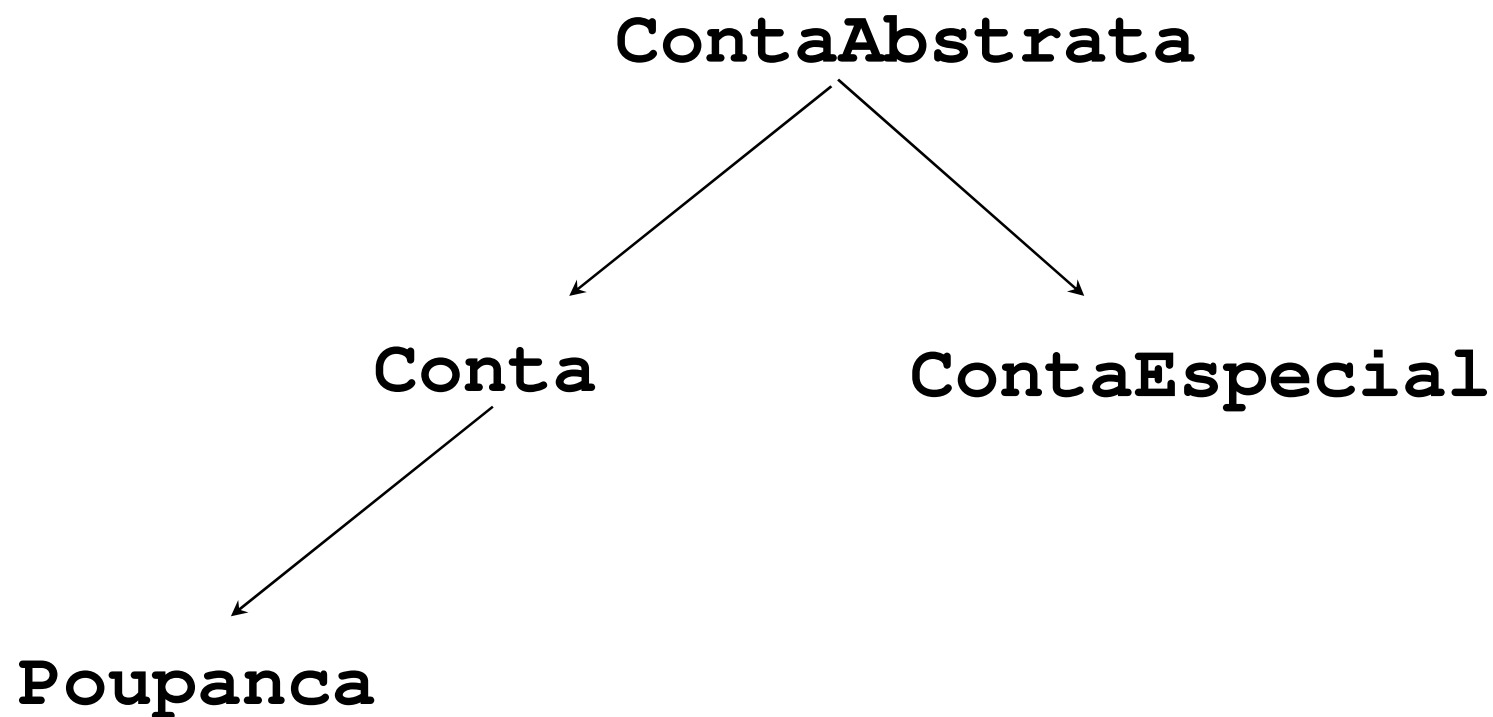
Mais sobre Classes Abstratas

- ◆ Métodos abstratos:
 - Geralmente existe pelo menos um
 - São implementados nas subclasses
- ◆ Não se criam objetos a partir de classes abstratas:
 - Mas, podem (devem) ter construtores, para reuso
 - Criam-se objetos a partir das subclasses, desde que estas não sejam também abstratas
- ◆ Subclasses devem redefinir métodos abstratos para não serem também classes abstratas

Classes Abstratas e Herança

- ◆ Uma classe abstrata representa uma entidade abstrata que é insuficientemente definida para ser útil sozinha
- ◆ Como ela não pode ser instanciada, não há sentido em falar de classe abstrata, se não houver herança
 - Subclasses serão instanciadas
- ◆ Podemos declarar uma variável do tipo da classe abstrata e associá-la a diferentes objetos das subclasses durante o fluxo do programa
 - Respeita sistema de tipos

Exemplo de Uso: Conta



Definindo uma Classe Abstrata em Java

```
public abstract class ContaAbstrata {  
    private String numero;  
    private double saldo;  
  
    public abstract void debitar(double valor)  
        throws SaldoInsuficienteException;  
  
    public ContaAbstrata (String numero) {  
        this.numero = numero;  
        saldo = 0.0;  
    }  
  
    public String getNumero() {  
        return numero;  
    }  
  
    public double getSaldo() {...}  
    public void setSaldo {...}  
}
```

Palavra reservada que indica que classe ou método é abstrato

Método abstrato

Classe Conta herdando Comportamento

```
public class Conta extends ContaAbstrata{  
  
    public void debitar(double valor)  
        throws SaldoInsuficienteException{  
        if (valor <= getSaldo()){  
            setSaldo(getSaldo() - valor);  
        } else {  
            throw new  
                SaldoInsuficienteException(getNumero());  
        }  
    }  
  
    public Conta(String numero) {  
        super(numero);  
    }  
}
```

Implementando o
método abstrato
do pai

Construtor utiliza
construtor do pai

Classe ContaEspecial Herdando Comportamento

```
public class ContaEspecial extends ContaAbstrata{
```

```
    private double limite;
```

Define um novo
atributo

```
    public void debitar(double valor) throws ...{
```

```
        if (valor <= (getSaldo()+limite)){
```

```
            setSaldo(getSaldo() - valor);
```

```
        } else {
```

```
            throw ...//exceção
```

```
        }
```

```
    }
```

Implementando o
método abstrato
do pai

```
    public ContaEspecial(String numero, double limite) {
```

```
        this.limite = limite;
```

```
        super(numero);
```

```
    }
```

```
}
```

Construtor utiliza
construtor do pai

Classe Poupanca Herda de Conta

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

Definição de novo
método

Usando uma Variável do Tipo ContaAbstrata

```
...  
MiniJavaSystem sys = new MiniJavaSystem();  
ContaAbstrata ca;  
ca = new ContaEspecial("21.342-7", 1000);  
ca.creditar(500);  
ca.debitar(600);  
sys.println(ca.getSaldo());  
ca = new Conta("21.111-7");  
ca.creditar(500);  
ca.debitar(600);  
sys.println(ca.getSaldo());  
...
```

Qual das duas
linhas lançará
uma exceção?

Em tempo de execução é decidida qual
versão de debitar deve ser utilizada

Refactoring: *Extract Class*

- ◆ Consiste em mover alguns métodos e atributos de uma classe para uma nova classe
- ◆ Quando usar?
 - Uma classe está grande demais, dificultando assim o seu entendimento
 - Notar que um subgrupo de métodos e atributos pode formar uma nova entidade lógica
 - Que pode ser reutilizado por outra classe

Deve-se ter cuidado para não tirar métodos e atributos que são inerentes ao conceito que a classe representa!

Exemplo: Classe Cliente

```
public class Cliente {  
    private String nome;  
    private String cpf;  
    private String rg;  
    private String numTelefone;  
    private String codigoAreaTelefone;  
    public String getNome() {  
        return nome;  
    }  
    public String getCPF() {  
        return cpf;  
    }  
    public String getTelefoneCompleto() {  
        return (codigoAreaTelefone+numTelefone);  
    }  
}
```

Código pode ser
movido para uma nova
classe

Extract Class: Criando Classe Telefone

```
public class Telefone {  
    private String numTelefone;  
    private String codigoAreaTelefone;  
    public String getTelefoneCompleto() {  
        return (codigoAreaTelefone+numTelefone);  
    }  
}
```

Código movido para a
classe Telefone

Extract Class: Adaptando Classe Cliente

```
public class Cliente {  
    private String nome;  
    private String cpf;  
    private String rg;  
    private Telefone telefone;  
    public String getNome() {  
        return nome;  
    }  
    public String getCPF() {  
        return cpf;  
    }  
    public String getTelefoneCompleto() {  
        return (telefone.getTelefoneCompleto());  
    }  
}
```

Classe Cliente usa
agora classe
Telefone

Refactoring: *Template Method*

- ◆ Consiste em generalizar em um método abstrato (***template method***) de uma superclasse, métodos definidos em subclasses diferentes, que possuem uma seqüência similar de operações
 - A parte invariável dos métodos fica no método abstrato da superclasse
 - As partes variáveis ficam nos métodos das subclasses

Método debitar de Conta e ContaEspecial

```
public class Conta extends ContaAbstrata{  
    public void debitar(double valor)  
        throws SaldoInsuficienteException{  
        if (valor <= getSaldo()){  
            setSaldo(getSaldo() - valor);  
        } else {  
            throw new  
                SaldoInsuficienteException(getNumero());  
        }  
    }  
}
```

Parte invariável
do algoritmo

```
public class ContaEspecial extends ContaAbstrata{  
    public void debitar(double valor)  
        throws SaldoInsuficienteException{  
        if (valor <= (getSaldo()+ limite)){  
            setSaldo(getSaldo() - valor);  
        } else {  
            throw new  
                SaldoInsuficienteException(getNumero());  
        }  
    }  
}
```

Parte variável do
algoritmo

Aplicando o Template Method

```
public abstract class ContaAbstrata {  
    ...  
  
    public void debitar(double valor)  
        throws SaldoInsuficienteException {  
        if (ehPossivelDebito(valor)) {  
            setSaldo(getSaldo() - valor);  
        } else {  
            throw new  
                SaldoInsuficienteException(getNumero());  
        }  
    }  
  
    public abstract boolean ehPossivelDebito(double valor);  
}
```

Template
method

Método abstrato que
será implementado nas
subclasses

Re-estruturando Conta e ContaEspecial

```
public class Conta extends ContaAbstrata{  
    ...  
    public boolean ehPossivelDebito(double valor){  
        return (valor <= getSaldo());  
    }  
}
```

Estas subclasses
devem
implementar
este método

```
public class ContaEspecial extends ContaAbstrata{  
  
    public boolean ehPossivelDebito(double valor){  
        return (valor <= (getSaldo()+ limite);  
    }  
}
```

Definição do método debitar desaparece das classes. Agora está na superclasse abstrata

Mais sobre *Template Method*

- ◆ Um *Template Method* define um algoritmo utilizando métodos abstratos
- ◆ Subclasses redefinem os métodos abstratos para prover um comportamento concreto
- ◆ Quando usar?
 - Para implementar partes invariantes de um algoritmo uma única vez e deixar subclasses implementarem o comportamento variável
 - Para evitar duplicação de comportamento comum entre subclasses

Resumindo ...

◆ Classes abstratas

- Importância
- Utilização
- Declaração uma classe abstrata em Java
- Relação de classe abstrata com herança
- Abstração do conceito de Conta

◆ Refactoring

- Extract class
- Padrão template method