Programação Orientada a Objetos

Aula 04: Exceções

Ivo Calado
ivo.calado@ifal.edu.br

Instituto Federal de Educação, Ciência e Tecnologia de Alagoas

23 de Novembro de 2016

Roteiro

1 Introdução

Motivação

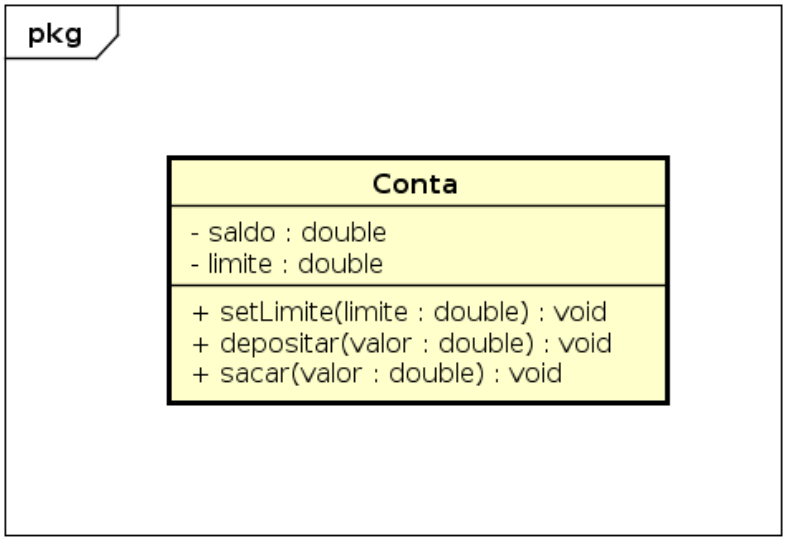
Cenário

Você necessita desenvolver um sistema de auto-atendimento para um banco. Como seria modelada a entidade Conta em termos de atributos e ações?

Motivação

Cenário

Você necessita desenvolver um sistema de auto-atendimento para um banco. Como seria modelada a entidade Conta em termos de atributos e ações?



Utilizando esta classe

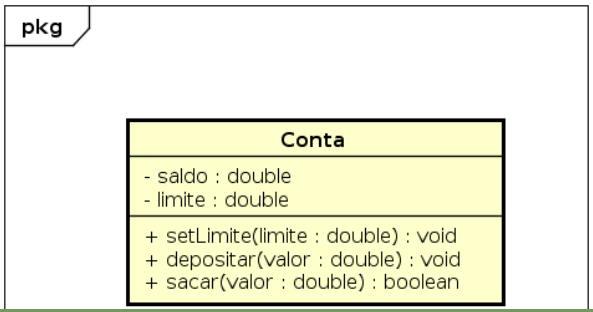
```
CaixaElet caixa = new CaixaElet();
Conta minhaConta = new Conta();
minhaConta.depositar(100);
minhaConta.setLimite(100);
minhaConta.sacar(200);
caixa.emitir(200);
```

- E se o saque for maior que o saldo+limite, como indicar que o saque não foi possível e impedir a emissão do dinheiro?

Utilizando esta classe

```
CaixaElet caixa = new CaixaElet();
Conta minhaConta = new Conta();
minhaConta.depositar(100);
minhaConta.setLimite(100);
minhaConta.sacar(200);
caixa.emitir(200);
```

- E se o saque for maior que o saldo+limite, como indicar que o saque não foi possível e impedir a emissão do dinheiro?



Utilizando a nova classe I

Definição:

```
boolean sacar(double valor) {
     // posso sacar até saldo+limite
     if (valor > this.saldo + this.limite) {
       return false;
     } else {
       this.saldo = this.saldo - valor;
     return true;
   }
}
```

Utilização:

Utilizando a nova classe II

```
CaixaElet caixa = new CaixaElet();
Conta minhaConta = new Conta();
minhaConta.depositar(100);
minhaConta.setLimite(100);
if (minhaConta.sacar(1000)) {
    caixa.emitir(1000);
    caixa.exibirMensagem("Saque realizado com
        sucesso!");
} else {
    caixa.exibirMensagem("Saldo insuficiente"
       );
}
```

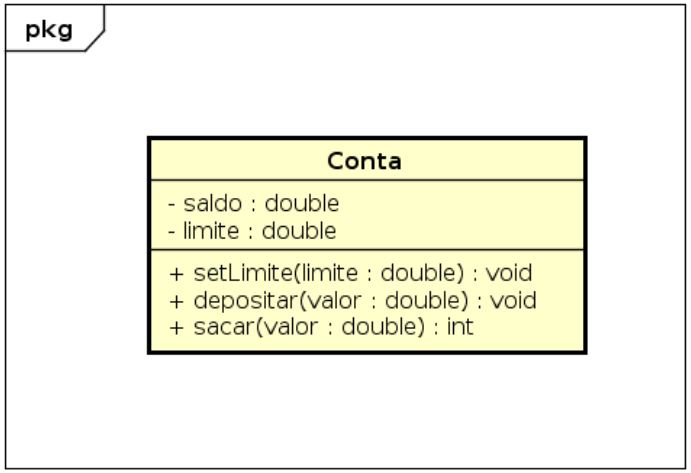
Utilizando a nova classe III

Magic numbers I

- E se precisássemos sinalizar mais do que um tipo de insucesso? Por exemplo, se o usuário informou um valor negativo para saldo
- Poderíamos fazer uso da estratégia conhecida como *magic numbers* que consiste em definir diferentes códigos de erro para casa situação. Nossa classe ficaria assim:

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Magic numbers II



Definição:

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Magic numbers III

```
int sacar(double valor) {
    // posso sacar até saldo+limite
    if(valor <= 0) {
      return -2;
    } else if (valor > this.saldo + this.
       limite) {
      return -1;
    } else {
      this.saldo = this.saldo - valor;
    return 0;
  }
}
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Magic numbers IV

Utilização:

```
CaixaElet caixa = new CaixaElet();
Conta minhaConta = new Conta();
minhaConta.depositar(100);
minhaConta.setLimite(100);
int resultado = minhaConta.sacar(1000)
if (resultado == 0) {
    caixa.emitir(1000);
    caixa.exibirMensagem("Saque realizado com
        sucesso!");
} else if(resultado == -1){
```

Magic numbers V

```
    caixa.exibirMensagem("Saldo insuficiente!
        ");
} else {
    caixa.exibirMensagem("Valor negativo
        solicitado!");
}
```

Magic numbers

Qual o principal problema da estratégia *magic numbers*

Essa estratégia recebe o nome de *magic numbers* justamente porque os valores retornados não possuem uma semântica clara definida e dependem extensamente da documentação associada.

Mais problemas (2)?

- E se o programador ao utilizar a classe **Conta** esquecesse de checar o retorno do método sacar?

```
CaixaElet caixa = new CaixaElet();
Conta minhaConta = new Conta();
minhaConta.depositar(100);
minhaConta.setLimite(100);
caixa.emitir(1000);
caixa.exibirMensagem("Saque realizado com
   sucesso!");
```

Mais problemas (2)?

- E se o programador ao utilizar a classe **Conta** esquecesse de checar o retorno do método sacar?

```
CaixaElet caixa = new CaixaElet();
Conta minhaConta = new Conta();
minhaConta.depositar(100);
minhaConta.setLimite(100);
caixa.emitir(1000);
caixa.exibirMensagem("Saque realizado com
    sucesso!");
```

Consequência: o sistema ficará num estado inconsistente

Solução: Exceções

Execeções

Uma **exceção** representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema. Trata-se de um valioso recurso disponível em **algumas** linguagens de programação como Java

O processo de chamada de métodos em Java I

Antes de vermos a solução do nosso problema, é necessário conhecer como Java gerencia a chamada de vários métodos. Para tal, criar a seguinte aplicação:

1. Criar uma classe C1, com os seguintes métodos:
   1. metodo1, metodo2, metodo3 e metodo4
   2. **metodo1** imprime "Inicio metodo 1", invoca **metodo2** e imprime "Fim metodo 1"
   3. **metodo2** imprime "Inicio metodo 2", invoca **metodo3** e imprime "Fim metodo 2"
   4. **metodo3** imprime "Inicio metodo 3", invoca **metodo4** e imprime "Fim metodo 3"
   5. **metodo4** imprime "Inicio metodo 4", realiza a divisão 10/2 e imprime "Fim metodo 4"

O processo de chamada de métodos em Java II

2. Criar uma classe C2 com método main que deve realizar os seguintes passos:
   1. Instanciar um objeto da classe C1
   2. Imprimir uma mensagem de boas vindas
   3. invocar o método **metodo1** da classe
   4. Imprimir uma mensagem indicando o término do método

Código:

O processo de chamada de métodos em Java III

```
public class C1 {
    public void metodo1() {
        System.out.println("=>inicio método 1"
            );
        metodo2();
        System.out.println("=>fim método 1");
    }

    private void metodo2() {
        System.out.println("==>inicio método 2
            ");
        metodo3();
```

O processo de chamada de métodos em Java IV

```
        System.out.println("==>fim método 2");
    }

    private void metodo3() {
        System.out.println("===>inicio método
            3");
        metodo4();
        System.out.println("===>fim método 3")
            ;
    }

    private void metodo4() {
```

O processo de chamada de métodos em Java V

```
        System.out.println("====>inicio método
            4");
        int i = 10/2;
        System.out.println("====>fim método 4"
           );
    }
}
```

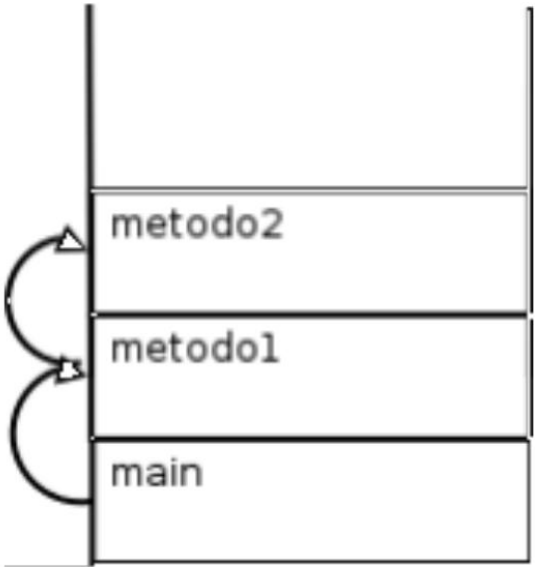
Saída:

O processo de chamada de métodos em Java VI

```
>inicio método main
=>inicio método 1
==>inicio método 2
===>inicio método 3
====>inicio método 4
====>fim método 4
===>fim método 3
==>fim método 2
=>fim método 1
>fim método main
```

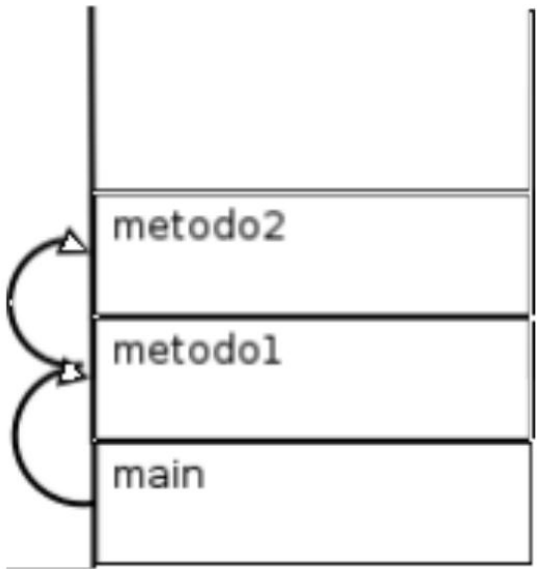
A pilha de chamada (*Call Stack*) em Java

As chamadas são empilhadas e após o final do método são desempilhados



A pilha de chamada (*Call Stack*) em Java

As chamadas são empilhadas e após o final do método são desempilhados



- Agora vamos modificar o método **metodo4** para que seja realizada uma divisão por zero e verificar a saída

Stack trace

- A divisão por zero irá disparar uma **exceção** que deve ser tratada

- Quando uma exceção é lançada ela começa a interromper todos os métodos da pilha de chamadas até ser **capturada** ou então até a aplicação ser **encerrada**

Capturando uma exceção I

Para capturarmos uma exceção utilizamos a cláusula **try-catch**

```
public class C2 {
    public static void main(String[] args) {
        C1 c1 = new C1();
        System.out.println(">inicio método
            main");
        try {
            c1.metodo1();
        } catch (ArithmeticException e) {
            System.out.println("!!!Erro de
                divisão por zero!!!");
        }
```

Capturando uma exceção II

```
        System.out.println(">fim método main")
            ;
    }
}
```

Saída:

```
>inicio método main
=>inicio método 1
==>inicio método 2
===>inicio método 3
====>inicio método 4
!!!Erro de divisão por zero!!!
>fim método main
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Capturando uma exceção III

- Vamos agora testar o processo a saída se movermos o bloco try-catch para algum outro método da pilha de chamada. Exemplo:

```
public void metodo1() {
    System.out.println("=>inicio método 1");
    try {
        metodo2();
    } catch (ArithmeticException e) {
        System.out.println("!!!Erro de divisão
            por zero!!!");
    }
    System.out.println("=>fim método 1");
```

Capturando uma exceção IV

```
}
```

Saída:

```
>inicio método main
=>inicio método 1
==>inicio método 2
===>inicio método 3
====>inicio método 4
!!!Erro de divisão por zero!!!
=>fim método 1
>fim método main
```