Programação Orientada a Objetos

Carlos Henrique Bughi, MSc

UNIVALI
Escola Politécnica

Onde estamos?
(e para onde vamos)

Manipulando coleções de objetos

Tratamento de exceções

Padrões de Projeto

Aula 14

Tratamento de exceções

UNIVALI

Escola Politécnica

Problema: como tratar erros

- Suponha que um erro ocorra enquanto um programa esteja rodando;
- Erros podem ocorrer por diversos motivos
  - Falha de rede,
  - Falta de espaço em disco,
  - Término inesperado da conexão com um banco;
  - Índice inválido para um vetor;
  - Manipulação de variáveis nulas;
  - Por fim, qualquer coisa que tenha um comportamento não esperado pelo programa;

Problema: como tratar erros

- Os usuários finais esperam que o programa saiba lidar com erros que possam ocorrer;
- Quando ocorrer um erro, o programa pode agir de duas formas:
  - Retornar a um estado seguro e permitir ao usuário a execução de uma nova ação;
  - Permitir que o usuário salve todas as informações e finalize o programa corretamente.

Problema: como tratar erros

- Imagine um programa que utiliza um arquivo da rede;
- O que acontece se o servidor no qual o programa está tentando conectar está desligado? Duas coisas podem ocorrer:
  - O programa trava e termina, e o usuário terá uma péssima experiência;
  - Ou você pode fazer um tratamento de exceção;

O que é uma exceção

- Exceções são condições excepcionais que podem ocorrer em um programa e que podem ser **descobertas** e **tratadas**.
- É importante diferenciar o descobrimento do erro e o tratamento do erro
  - É muito frequente descobrir algo errado em um lugar mas querer tratar o erro em outro lugar;
  - Por exemplo, na classe Calculadora, tratar o erro de divisão por zero é ruim porque é um método de "baixo nível", que não sabe sequer que tipo de interface está sendo usada (gráfica, console, etc).

Exceções

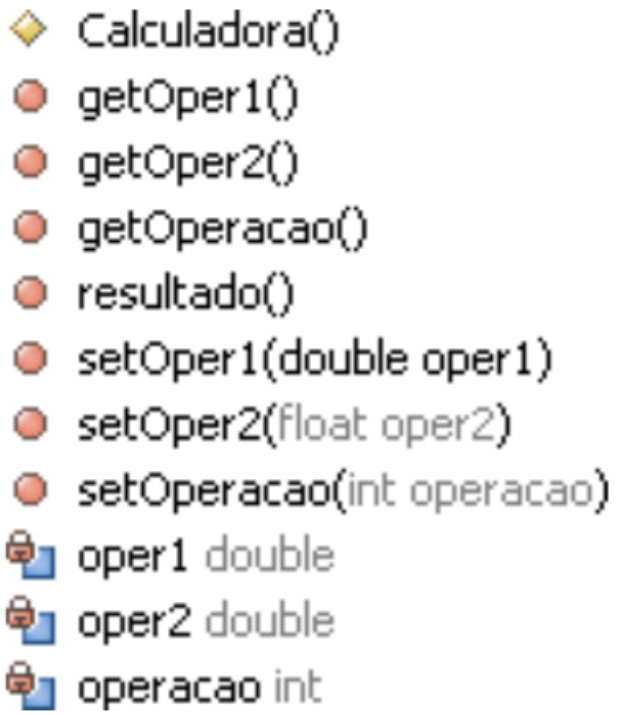
- Vamos usar um mecanismo novo para **retornar erros**;
- Sabemos que o retorno normal de valores por um método usa “return”
- O retorno anormal (indicando um erro) usa outra palavra para retornar do método
  - A palavra é **throw**
- Da mesma forma que “return”, “throw” retorna imediatamente do método;
- Diferentemente de “return, “throw” só retorna objetos especiais, chamados exceções;
  - A exceção pode conter uma mensagem indicando qual o erro ocorrido;

Exceções

- “throw” faz com que todos os métodos chamados retornem, até o ponto em que algum método **capture a exceção** para tratar o erro.
  - Esta captura é feita com um bloco “**try-catch**”;

Exceções

- Voltando ao exemplo da calculadora, como podemos tratar o problema da divisão por zero?
- Considere a seguinte estrutura da classe Calculadora:



Exceções

- Atualmente, o método responsável por realizar a operação está implementado da seguinte maneira:

```
public double resultado(){
    switch (this.operacao){
        case 1:  return this.oper1/this.oper2;
        case 2:  return this.oper1*this.oper2;
        case 3:  return this.oper1+this.oper2;
        case 4:  return this.oper1-this.oper2;
        default: return 0.0;
    }
}
```

Exceções

- Neste método iremos detectar o erro e lançar uma exceção

```
public double resultado() throws CalculadoraException{
    switch (this.operacao){
        case 1:
            if (this.oper2 == 0)
                throw new CalculadoraException("Divisão por zero");
            else
                return this.oper1/this.oper2;
        case 2:  return this.oper1*this.oper2;
        case 3:  return this.oper1+this.oper2;
        case 4:  return this.oper1-this.oper2;
        default: return 0.0;
    }
}
```

Exceções

- Observe a cláusula “throws” que diz que tipo de exceção o método pode lançar
  - É importante dizer tanto o que um método retorna normalmente, quanto o que ele retorna quando há erro

```
public double resultado() throws CalculadoraException{
```

Retorno normal

Retorno anormal

Exceções

- Java tem muitas exceções para indicar várias condições de erro
  - Neste exemplo, estamos criando nossa própria exceção para representar o erro (exemplo: ArithmeticException se dividir por zero);
  - Perceba que a exceção é dada quando utilizamos a cláusula “throw”
    - throw new CalculadoraException("Divisão por zero");
    - Neste momento, o método é interrompido e a exceção é retornada;

Exceções

- Código da classe de exceção para calculadora (CalculadoraException);
  - Lembre-se que toda classe de exceção deve herdar Exception;

```
public class CalculadoraException extends Exception{

    public CalculadoraException(String motivo) {
        super(motivo);
    }

}
```

Exceções

- Vamos ver agora como se dá o tratamento da exceção gerada:

```
public Main() {
    Calculadora calc = new Calculadora();
    calc.setOper1(5);
    calc.setOper2(0);
    calc.setOperacao(1);
    try {
        System.out.println(calc.resultado());
    } catch (CalculadoraException ex) {
        System.out.println(ex.getMessage());
    }
}
```

Exceções

- No código anterior, vemos como capturar uma exceção com try-catch e como obter a mensagem que está dentro da exceção;
- A cláusula try pode ser seguida de quantos catch forem necessários (um para cada tipo de exceção), ou ainda um único catch tratando a exceção genérica Exception
- Outra possibilidade é não tratar a exceção neste momento e sim deixá-la passar para um nível;
  - Para isso, basta adicionar a cláusula throws na assinatura do método;

Exceções

- Voltando ao método resultado() da classe Calculadora, vamos ver como adicionar mais de uma exceção de retorno;
- Para isso, suponha que será tratado também o erro de não definição da operação desejada:

```
public double resultado() throws CalculadoraException, OperacaoException{
    switch (this.operacao){
        case 1:
            if (this.oper2 == 0)
                throw new CalculadoraException("Divisão por zero");
            else
                return this.oper1/this.oper2;
        case 2:  return this.oper1*this.oper2;
        case 3:  return this.oper1+this.oper2;
        case 4:  return this.oper1-this.oper2;
        default: throw new OperacaoException();
    }
}
```

Duas exceções separadas por vírgula

Exceções

- A classe OperacaoException não possui um construtor com argumento pois ela sempre terá a mesma mensagem de erro:

```
public class OperacaoException extends Exception{

    public OperacaoException() {
        super("Operação não foi definida");
    }

}
```

Exceções

- Tratando agora as duas exceções temos: (opção 1)

```
public Main() {
    Calculadora calc = new Calculadora();
    calc.setOper1(5);
    calc.setOper2(0);
    calc.setOperacao(1);
    try {
        System.out.println(calc.resultado());
    } catch (OperacaoException ex) {
        System.out.println(ex.getMessage());
        ex.printStackTrace();
    } catch (CalculadoraException ex) {
        System.out.println(ex.getMessage());
        ex.printStackTrace();
    }

}
```

1º.

2º.

Imprime o rastro do erro

Regra básica, ao utilizar vários catch, inicie pelas exceções mais especializadas deixando por último a mais genérica (Exception)

UNIVALI

Escola Politécnica

Exceções

- Tratando agora as duas exceções temos: (opção 2)

```
public Main() {
    Calculadora calc = new Calculadora();
    calc.setOper1(5);
    calc.setOper2(0);
    calc.setOperacao(1);
    try {
        System.out.println(calc.resultado());
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
        ex.printStackTrace();
    }
}
```

Exceção padrão

Exceções

- Tratando agora as duas exceções temos: (opção 3)

```
public Main() throws CalculadoraException, OperacaoException {
    Calculadora calc = new Calculadora();
    calc.setOper1(5);
    calc.setOper2(0);
    calc.setOperacao(1);
    System.out.println(calc.resultado());
}

public static void main(String[] args) {
    try {
        new Main();
    } catch (OperacaoException ex) {
        ex.printStackTrace();
    } catch (CalculadoraException ex) {
        ex.printStackTrace();
    }
}
```

O método passa as exceções adiante

Exceções

- Caso tenha um código que deva ser executado mesmo se um erro acontecer, é possível utilizar a cláusula “finally”

```
public Main(){
    Calculadora calc = new Calculadora();
    calc.setOper1(5);
    calc.setOper2(2);
    calc.setOperacao(1);
    String resultado = "";
    try{
        resultado = String.valueOf(calc.resultado());
    } catch(Exception e){
        resultado = e.getMessage();
    } finally{
        System.out.println("Resultado da operação: "+resultado);
    }
}
```