Módulo 03

Introdução ao Tratamento de Exceções

- Conceito de Exceção
- Bloco try/catch
- Estratégias para Criação de Classes de Dados

Créditos

Autor

Prof. Alessandro Cerqueira

(alessandro.cerqueira@hotmail.com)

Classes de Dados (model)

Métodos de Validação

- No módulo anterior foi apresentada uma primeira estratégia para construção dos métodos de validação que consistia dos seguintes passos:

1. Para cada atributo, criar um método *`public boolean validar`*`[`*`NomeAtributo`*`]` e que receberá um único parâmetro do mesmo tipo do atributo.

```
Ex: public boolean validarIdade(int idade) { ... }
    public boolean validarNome(String nome) { ... }
```

2. Codificar nesses métodos as regras que venham a invalidar o valor recebido; nesse caso retornar `false`. Se ao final, o valor passou por todos os testes, retornar `true`.
3. Nos métodos `set`, só permitir a atribuição após a validação.

- Essa estratégia não é boa, por duas razões:
  - O método de validação informa que o valor é **inválido**, mas não informa **por que é inválido**.
  - Para executar o método, é necessário que já exista um objeto da classe para a validação.

Uso e Tratamento de Exceções

Métodos de Validação

- Vamos introduzir o conceito de **Exceção** presente em Java e em muitas linguagens de programação.
- Uma **Exceção** é um **objeto** utilizado para sinalizar que ocorreu um problema durante a execução e que informa qual é a razão do problema ter ocorrido. Esse objeto é instância da classe **`Exception`** ou de uma de suas especializações.
- Como o Tratamento de Exceções envolve vários aspectos a serem aprendidos em Java (`throw`, `throws`, `try`, `catch`, `finally`), vamos apresentar os passos de como vamos escrever e usar os métodos de validação. Assim, vamos conhecer cada aspecto dessa técnica.

Uso e Tratamento de Exceções

(Passo 1) Criação da Classe de Exceção

- Ao trabalharmos com exceções adotamos a prática de **criar uma nova classe** para descrever uma classe de problemas que podem ocorrer.
- Como queremos sinalizar problemas na validação de dados, vamos criar a classe **ModelException** (pacote model) com o seguinte código:

```
1 package model;
2
3 public class ModelException extends Exception {
4
5     public ModelException(String msgDeErro) {
6         super(msgDeErro);
7     }
8
9 }
```

Uso e Tratamento de Exceções

(Passo 2) Métodos de Validação

- Os métodos de validação deixarão de retornar `boolean` e passarão a serem **`public static void`**; ou seja, não retornarão valor (void) e serão métodos estáticos (tópico futuro).
- Para cada situação em que o valor passado não é válido para o atributo, deveremos lançar a sinalização de exceção (**`throw`**) e instanciar um objeto **`ModelException`** descrevendo qual é a razão pela qual o valor é inválido.

```
public static void validarIdade(int idade) throws ModelException {
    if(idade < 0 || idade > VALOR_MAX_IDADE)
        throw new ModelException("Idade Inválida: " + idade);
}
```

Uso e Tratamento de Exceções

(Passo 2) Métodos de Validação

- O **`throw`** indica que uma exceção ocorreu e que precisa ser tratada imediatamente pelo programa. Assim, a sequência de instruções é abortada e o fluxo de execução do programa será automaticamente redirecionado para onde a exceção será tratada (Bloco **`try/catch`**).
- Como **`ModelException`** é uma *Checked Exception* (tópico futuro), então é obrigatório que coloquemos no cabeçalho do método a indicação **`throws`** (com 's') com a indicação da classe do objeto de exceção que pode ser lançado.

```
public static void validarNome(String nome) throws ModelException {
    if(nome == null || nome.length() == 0)
        throw new ModelException("O nome não pode ser nulo!");
    for(int i = 0; i < nome.length(); i++) {
        char c = nome.charAt(i);
        if( !Character.isAlphabetic(c) && !Character.isSpaceChar(c) )
            throw new ModelException("No nome, há um caracterer inválido '" + c + "' na posição " + i);
    }
}
```

Uso e Tratamento de Exceções

(Passo 3) Validação nos Métodos *Set*

- Na primeira abordagem de validação, tínhamos um comando `if` com a chamada e verificação do valor passado por parâmetro.
- Na abordagem com uso de **Exceções**, não é necessária a presença do `if`, pois a atribuição só será realizada se **não** for lançada a exceção.
- Como durante a execução do método `set` pode ser lançada uma exceção, é obrigatório colocarmos **`throws`** na especificação do método por dois motivos:
  1) O método não fará o tratamento da exceção (Bloco `try`/`catch`), pois isso é feito nos métodos que utilizam os objetos de dados.
  2) Por usar um método que faz esse lançamento.
- Assim, todos os métodos set terão uma forma semelhante ao código abaixo:

```
public void setNome(String novoNome) throws ModelException {
    Pessoa.validarNome(novoNome);
    this.nome = novoNome;
}
```

Uso e Tratamento de Exceções

(Passo 4) Chamada dos Métodos *Set* no Construtor

- Como o método construtor faz a chamada aos métodos set, então também será obrigatória a colocação do **throws** na especificação do método pelos mesmos motivos foram apresentados no slide anterior:

  1) O método não fará o tratamento da exceção (Bloco try/catch), pois isso é feito nos métodos que utilizam os objetos de dados.

  2) Por usar um método que faz esse lançamento.

- Com esses quatro primeiros passos, fechamos os procedimentos necessários para o uso de exceções nos métodos de validação.

```
public Pessoa(String c, String n, int i) throws ModelException {
    super(); // Chamada ao construtor da superclasse (Object) - tópico futuro
    this.setCpf(c);
    this.setNome(n);
    this.setIdade(i);
}
```

Uso e Tratamento de Exceções

(Passo 5) Uso do Bloco `try/catch`

- Nos trechos de código onde tivermos a instanciação de objetos de dados ou a chamada de métodos `set`, deveremos fazer o tratamento da exceção com a colocação de um bloco **`try/catch`**.
- O bloco **`try`** deverá conter as instruções onde pode ocorrer o lançamento de uma exceção.

```
try {
    // Instanciando um objeto Pessoa
    Pessoa p1 = new Pessoa(cpf, nome, idade);
    // Informo ao usuário que o objeto foi guardado
    JOptionPane.showMessageDialog(btOk, "Objeto Pessoa Criado: " + p1);
} catch (ModelException e1) {
    // Exibe uma 'Dialog' posicionada próximo ao botão
    // btOk com a mensagem de erro da exceção
    JOptionPane.showMessageDialog(btOk, e1.getMessage());
    // Imprime na console o traçado de execução da Stack
    e1.printStackTrace();
}
```

Uso e Tratamento de Exceções

(Passo 5) Uso do Bloco `try/catch`

- Caso ocorra uma exceção, o fluxo do programa seguirá imediatamente para o bloco **`catch`** associado à exceção.
  - No nosso caso, se durante a instanciação de um objeto de dados, um método de validação lançar a exceção, o fluxo cairá no bloco `catch` associado a `ModelException`.

```
try {
    // Instanciando um objeto Pessoa
    Pessoa p1 = new Pessoa(cpf, nome, idade);
    // Informo ao usuário que o objeto foi guardado
    JOptionPane.showMessageDialog(btOk, "Objeto Pessoa Criado: " + p1);
} catch (ModelException e1) {
    // Exibe uma 'Dialog' posicionada próximo ao botão
    // btOk com a mensagem de erro da exceção
    JOptionPane.showMessageDialog(btOk, e1.getMessage());
    // Imprime na console o traçado de execução da Stack
    e1.printStackTrace();
}
```

Uso e Tratamento de Exceções

(Passo 5) Uso do Bloco `try/catch`

- A declaração do bloco **`catch`** se assemelha à declaração de parâmetros em um método, sendo que esse parâmetro é um ponteiro para o objeto de exceção que ocorreu.
  - No nosso caso, o parâmetro será um ponteiro para `ModelException`.
- No bloco **`catch`** colocamos o código para o tratamento da exceção (ex. comunicação do problema ao usuário, redefinição dos passos de execução do programa, suspenção do programa, etc.)

```
try {
    // Instanciando um objeto Pessoa
    Pessoa p1 = new Pessoa(cpf, nome, idade);
    // Informo ao usuário que o objeto foi guardado
    JOptionPane.showMessageDialog(btOk, "Objeto Pessoa Criado: " + p1);
} catch (ModelException e1) {
    // Exibe uma 'Dialog' posicionada próximo ao botão
    // btOk com a mensagem de erro da exceção
    JOptionPane.showMessageDialog(btOk, e1.getMessage());
    // Imprime na console o traçado de execução da Stack
    e1.printStackTrace();
}
```

Uso e Tratamento de Exceções

(Passo 5) Uso do Bloco `try/catch`

- Há dois métodos úteis nos objetos de exceção:
  - **`getMessage()`** – Nos dá o texto que descreve o problema da exceção.
  - **`printStackTrace()`** – Escreve na console o traçado de exceção do programa até o ponto onde ocorreu a exceção

```
try {
    // Instanciando um objeto Pessoa
    Pessoa p1 = new Pessoa(cpf, nome, idade);
    // Informo ao usuário que o objeto foi guardado
    JOptionPane.showMessageDialog(btOk, "Objeto Pessoa Criado: " + p1);
} catch (ModelException e1) {
    // Exibe uma 'Dialog' posicionada próximo ao botão
    // btOk com a mensagem de erro da exceção
    JOptionPane.showMessageDialog(btOk, e1.getMessage());
    // Imprime na console o traçado de execução da Stack
    e1.printStackTrace();
}
```

Uso e Tratamento de Exceções

Passos Fututos

- Há mais aspectos relacionados às exceções, mas eles serão abordados em tópicos futuros.
- Por enquanto, os conhecimentos aqui apresentados são suficientes para codificarmos as classes de dados e realizarmos o tratamento de exceções nos trechos de código onde manipulamentos objetos de dados.