Programação Orientada a Objetos

Aula 12 - Lançamento de Exceções

Apresentação

Na aula passada, aprendemos como tratar situações anormais de execução dos programas, mais especificamente como tratar exceções que podem acontecer em situações não comuns. Nesta aula, iremos aprender como gerar nossas próprias exceções ao se perceber uma situação inadequada para continuação da execução do nosso programa. Boa aula!

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Criar suas próprias exceções customizadas.
- Lançar novas exceções.
- Usar a cláusula finally do bloco try/catch.
- Repassar exceções de que não queremos tratar.

Contornando a Situação!

No mundo Java, quando uma coisa ruim acontece, temos sempre a chance de contornar a situação, seja ela qual for. É de nossa responsabilidade decidir se queremos tratar do problema que nos foi passado, ou simplesmente repassá-lo para a próxima vítima. É sobre esse contexto que estudaremos nesta aula.

Relembrando!

Vimos na aula anterior por que o tratamento de exceções é importante para que possamos criar programas robustos e tolerante a falhas. Aprendemos que as exceções são todas subclasses de Throwable. Vimos também como um método informa ao compilador que ele poderá causar risco (cláusula throws), e a sintaxe básica para capturar e tratar uma exceção (bloco try/catch). Agora, vamos um pouco mais a fundo no tratamento de exceções... chegou a hora de virar gente grande! Vamos lá então... mãos à obra!

Cláusula Finally: para as coisas que você deseja fazer não importa como.

Lembra que no capítulo anterior vimos a sintaxe básica do bloco **try/catch**? Pois então, ele ainda quarda algumas cartas na manga para combater as exceções do mundo Java. E para que serve essa danada dessa cláusula finally? Vou contar uma breve historinha para que possamos entender melhor o seu sentido.

Imagine que você esta tentando cozinhar alguma coisa, você tem que primeiramente ligar o fogão, certo? Ok, supondo que o alimento preparado tenha queimado! Você terá que **desligar o fogão**! Poxa, sou um péssimo cozinheiro! É verdade! Mas agora suponha que você tem visto muito Ana Maria Braga e está craque na cozinha e o que você fez ficou uma maravilha, como fica na televisão, você ainda tem que **desligar o fogão** no final de tudo!

Perceba que não importa qual for o resultado, FALHA ou SUCESSO, você sempre terá que realizar a mesma ação ao **final** do processo. É no bloco **finally** que colocaremos o código que deverá rodar, *não importa o que aconteça*.

Então, vamos dar uma olhada na sintaxe:

```
try {
    ligarFogao();
    fogao.assar();
} catch ( FogaoExcecao ex) {
    ex.printStackTrace();
} finally {              // Executará
    desligarFogao(); //NÃO importa o que aconteça
}
```

Qual a vantagem de termos o bloco **finally** nesta situação? Alguém se arrisca? Sem o **finally** temos que colocar o método desligarFogao() em dois locais, primeiro dentro do bloco **try**, para o caso em que tudo ocorra como o esperado, e outra vez dentro do bloco **catch**, caso alguma exceção seja lançada. Em resumo, o bloco finally permite que coloquemos todos os nossos códigos de limpeza em um único local, ao invés de duplicarmos como no exemplo a seguir:

```
try {
    ligarFogao();
    fogao.assar();
    desligarFogao();
} catch ( FogaoExcecao ex) {
    ex.printStackTrace(); //Código DUPLICADO!
    desligarFogao();  //Sem o finally as coisas ficam mais difíceis!
}
```

Desafio!

Se dentro de um bloco TRY ou CATCH tivesse um ‘return’ em seu corpo, o que aconteceria? O bloco finally ainda sim executaria, ou seria ignorado? Pense... ao final deste assunto falaremos a resposta!

Agora, vamos dar uma pausa para colocarmos em prática o que aprendemos até agora, então, vamos lá, mãos a obra! E continue pensando no nosso Desafio!

Atividade 01

1. Crie uma classe que contenha um método que lance uma exceção padrão da API a sua escolha.
2. Crie um programa Java que utilize o método criado acima e trate a exceção dentro de um bloco try/catch/finally.
3. Faça com que o programa criado nos passos acima execute de duais maneiras: Sucesso e Falha. Dentro de cada bloco *try/catch/finally* imprima uma mensagem informando que o código foi processado. Faça isso para os dois casos.

Exercitando a Mente

Para melhor fixar o nosso entendimento, vamos analisar o código abaixo e responder as seguintes perguntas:

- Qual você acha que seria a saída do programa abaixo?
- Qual seria a saída do programa se fizéssemos a seguinte alteração no código: String teste = “yes”; ? Assuma que ExcecaoAssustadora estende de Exception.

```
1  public class TesteExcecao {
2     public static void main(String args []) {
3        String teste = “no”;
4        try {
5           System.out.println(“Início do try”);
6           facaCoisaPerigosa(teste);
7           System.out.println(“Final do try”);
8        } catch ( ExcecaoAssutadora ex ) {
9           System.out.println(“Excecao assustadora”);
10       } finally {
11          System.out.println(“finally”);
12       }
13       System.out.println(“Final do main”);
14    }
15
16    static void facaCoisaPerigosa(String teste)  throws ExcecaoAssustadora {
17       System.out.println(“Início coisa perigosa”);
18       if ( “yes”.equals(teste) ) {
19          throw new ExcecaoAssustadora();
20       }
21       System.out.println(“Final da coisa perigosa”);
22       return;
23    }
24 }
```

Agora que aprendemos mais um pouco sobe a sintaxe para tratamento de exceções, voltamos a nossa pergunta **desafio**. Você já sabe a resposta? Espero que sim! Então, vamos lá... Lembra que foi dito no início da aula que o código do bloco **finally** executaria não importando o que acontecesse? Pois é puramente verdade! Mesmo que tenhamos um retorno dentro do bloco **try** ou do **catch**, ainda sim o nosso **finally** será executado! Lembre-se, não importa o que aconteça, o **finally** sempre será executado!

Vídeo 02 - Finally

Capturando Múltiplas Exceções

Por essa você não esperava! Você sabia que um método pode lançar várias exceções? Pois é, um método poderá lançar quantas exceções sejam necessárias. Mas, na declaração do método, será obrigatório informar cada uma delas, contudo, se duas ou mais exceções possuem uma superclasse em comum, então, nesse caso o método pode declarar apenas as superclasses na cláusula **throws**.

```
1 public class Lavanderia {
2     public void lavar() throws ExcecaoCalca, ExcecaoCamisa {
3         // Código que poderá lança qualquer uma das Exceções.
4     }
5 }
```

No código acima, temos um exemplo de um método lançando mais de uma exceção, considerando que ExcecaoCalca e ExcecaoCamisa fossem filhas de ExcecaoRoupa, o mesmo código poderia ser escrito da seguinte forma.

```
1 public class Lavanderia {
2     public void lavar() throws ExcecaoRoupa {
3         // Código que poderá lança qualquer uma das Exceções.
4     }
5 }
```

Como consequência desse fato, quem utilizar-se desse método deverá capturar cada uma das exceções. Mais um detalhe, a ordem em que você captura as múltiplas exceções dentro do bloco catch é de extrema importância, mas veremos isso com mais detalhes adiante. Vamos dar uma olhada na sintaxe para capturar várias exceções em um único bloco **try/catch**.

```
public class Besteirol {
    public void fazBesteira(){
        Lavanderia lavanderia = new Lavanderia();
        try {
            lavanderia.lavar();
        } catch (ExcecaoCalca) {
            // código de recuperação
        } catch (ExcecaoCamisa) {
            // código de recuperação
        }
    }
}
```

Veja que no código acima colocamos duas cláusulas **catch**, isso porque o método **lavanderia.lavar()** poderá lançar tanto **ExcecaoCalca** quanto **ExcecaoCamisa**. Observe que neste caso não importa a ordem em que as capturamos no bloco catch, pois ambas possuem o mesmo nível de hierarquia, ou seja, são subclasses de **ExcecaoRoupa**. Sendo assim, esse mesmo código também poderia ser escrito da seguinte maneira.

```
public class Besteirol {
    public void fazBesteira(){
        Lavanderia lavanderia = new Lavanderia();
        try {
            lavanderia.lavar();
        } catch (ExcecaoRoupa) {
            // código de recuperação
        }
    }
}
```

Note que agora temos apenas um bloco **catch** tratando a ExcecaoRoupa. Como essa última é superclasse de ambas as exceções (ExcecaoCalca e ExcecaoCamisa), eu posso simplificar meu código capturando apenas uma exceção que seja superclasse das demais.

Esse tipo de abstração também tem as suas desvantagens. Perceba que dessa forma daremos o mesmo tratamento para ambos os tipos de exceções. Sendo assim, se quisermos especificar um tratamento diferenciado de acordo com a exceção levantada, temos que de fato utilizar *múltiplas cláusulas catch*.

Criando Exceções

Estamos quase um especialista no tratamento de exceções, já aprendemos várias coisas, como criar métodos que lançam exceção, tratá-las dentro do bloco **try/catch**, aprendemos a utilizar o bloco **finally**, até mesmo a lançar várias exceções e capturá-las uma a uma. Porém, ainda não sabemos como criar as nossas próprias exceções. Então, chegou a hora de resolver esse problema, sem mais demoras, vamos direto ao assunto.

```
1 public class ExcecaoRoupa extends Exception {
2     public ExcecaoRoupa (String mensagem) {
3         super(mensagem);
4     }
5 }
```

Pronto, muito simples, não acha? É isso mesmo, a API Java é muito fácil de se trabalhar e poderosa também, para criarmos nossa própria exceção basta criar uma classe que estenda de Exception e sobrescrever o método getMessagem() ou qualquer outro que você desejar. A mesma classe poderia ser escrita da seguinte forma.

```
1 public class ExcecaoRoupa extends Exception {
2     @Override
3     Public String getMessage() {
4         return “Voce não está utilizando a roupa de forma adequada!”;
5     }
6 }
```

Mas, fazendo dessa forma, fica bem menos flexível do que a primeira, notem que assim não podemos mais modificar a mensagem a ser exibida quando lançarmos uma exceção do tipo **ExcecaoRoupa**. Esse exemplo só foi para demonstrar que podemos sobrescrever qualquer método da classe Exception para customizar nossas próprias exceções da forma que melhor nos atenda.

Lançando Exceções

Vídeo 03 - Throw

Chegamos ao momento especial em relação ao tratamento de exceções. Até agora estávamos preocupado em tratar as exceções que poderiam acontecer, tentando contornar a situação utilizando o bloco **try/catch**. Mas, agora é a nossa vez de lançarmos nossas próprias exceções, agora que aprendemos como criá-las, vamos ver como lançá-las! Um dia da caça outro do caçador!

Lembra deste código? Vamos incluir agora o seu conteúdo para lançar uma exceção do tipo **ExcecaoRoupa**.

```
1 public class Lavanderia {
2    public void lavar() throws ExcecaoRoupa {
3       if ( material == sapato )
4          throw new ExcecaoRoupa(“Apenas roupas! Lave o sapato na mão!”);
5       }
6    }
7 }
```

No código acima, verificamos se o material a ser levado é um sapato, caso verdadeiro, lançamos uma exceção informando que apenas Roupas poderão ser lavadas na máquina de lavar roupas, então, lançamos uma exceção.

Em Java, utilizamos a cláusula **throw** para lançar uma exceção, e como uma exceção também é um objeto, utilizamos o **new** para criá-la. Fácil não é? Como dito no capítulo anterior, toda a API Java para tratamento e criação de exceções foi desenvolvida de uma forma simples, limpa e intuitiva, nos fornecendo recursos bastante poderosos para que possamos tornar nossas aplicações seguras e confiáveis.

Vídeo 04 - Criando Exceções

Agora que já estamos grandinhos e sabemos criar as nossas próprias exceções, vamos colocar em prática toda essa habilidade para que possamos fixar melhor o que foi aprendido até o momento. Então, vamos lá, chega de moleza!

Atividade 02

1. Crie uma classe de exceção que estenda de Exception chamada de **ExcecaoAlfabeto**.
2. Crie um programa Java com um método que receba uma String de parâmetro chamado **validarLetrasAlfabeto**. Esse método deve verificar se cada um dos caracteres da String é uma letra do alfabeto, se for encontrado algum caractere especial, devemos lançar uma exceção do tipo criado no exercício anterior.
3. Crie duas classes de exceção que estenda de **ExcecaoAlfabeto** criada no primeiro exercício, cada uma com o seguinte nome (**ExcecaoLetras** e **ExcecaoNumeros**).
4. Ajuste o programa criado no passo dois criando mais dois métodos, um que valida se a String do parâmetro contém apenas caracteres que são letras, e se não for lance uma exceção do tipo **ExcecaoLetra** e no segundo valide se a String contém apenas números, em caso negativo lance a exceção **ExcecaoNumero**.

Repassando Exceções

Até agora aprendemos como tratar as exceções que um determinado método que utilizamos possa lançar. Mas, quando não queremos tratá-las, o que fazer? Pode não ser nossa responsabilidade tratar um determinado tipo de exceção. Nesse tipo

de situação devemos então repassá-las para o próximo método da pilha. E como fazemos isso? Vamos reutilizar a nossa classe Besteirol criada anteriormente.

```
1 public class Besteirol {
2     public void fazBesteira() throws ExcecaoRoupa {
3         Lavanderia lavanderia = new Lavanderia();
4         lavanderia.lavar();
5     }
6 }
```

O que foi que aconteceu com a cláusula **try/catch** que tinha aqui antes? Bom, agora não precisamos mais dela, uma vez que repassamos o problema, não precisamos mais tratar dele. Para isso, utilizamos a cláusula **throws** na declaração do método contendo a mesma exceção declarada no método **lavanderia.lavar()**. A regra básica para satisfazer as vontades do compilador é a seguinte: ou você trata a exceção dentro de um bloco **try/catch** ou você repassa o problema na declaração do seu método utilizando a cláusula **throws**. É simples assim, se você não quer resolver o problema, então, repasse para quem quiser tratá-lo, o que importa é fazer o compilador feliz!

Resumo

Bom, chegamos ao final de mais uma aula bastante emocionante. Vimos mais detalhadamente a sintaxe do bloco **try/catch** e do nosso novo amigo **finally**. Aprendemos também que podemos lançar vários tipos de exceções, de acordo com a nossa necessidade, e assim como podemos lançar, podemos também capturar várias exceções em um único bloco **catch**. Aprendemos a criar nossas próprias exceções e também como lançá-las. Por fim, vimos como repassar as exceções que não queremos ou não devemos tratar.

Autoavaliação

1. Suponha que temos os seguintes tipos de exceções: ExcecaoRoupa, ExcecaoRoupadeBaixo, ExcecaoCalcinha. Seguindo a ordem, ExcecaoRoupa é superclasse de ExcecaoRoupadeBaixo que, por sua vez, é superclasse de ExcecaoCalcinha. Qual a ordem que devemos capturar essas exceções em um mesmo bloco try/catch? Explique por que, nesse caso, a ordem é relevante!

Referências

KATHY SIERRA, Bert Bates. **Head First Java**. 2nd ed. 2005.

KATHY SIERRA, Bert Bates. **SCJP Sun Certified Programmer for Java 6 Study Guide**. 2008.