Disciplina de Programação Orientada a Objetos - POOS3

Curso Superior de ADS - 3º Semestre

(Professor Dênis Leonardo Zaniro)

Manipulação de exceções

Roteiro

- Tratamento de exceções: estrutura *try...catch()*
- O bloco *finally*
- Lançamento de exceções: a cláusula *throws*
- Implementação de classes de exceção e a instrução *throw*

Objetivos

1- Especificar código necessário para tratar exceções.

2- Criar classes que representem exceções e especificar código para lançar exceções.

Tratamento de exceções: estrutura *try...catch()*

Nenhum programa desenvolvido até esse momento considerou tratar exceções (*exceptions*). Exceções são indicações de erros que podem ocorrer durante a execução de um programa, interrompendo seu funcionamento normal. Tratar uma exceção significa definir um código especial (comportamento alternativo) que somente será executado caso uma exceção seja lançada.

Embora o tratamento de exceções ainda não tenha sido considerado pelos programas, em diversas situações, foram definidas instruções "de risco", isto é, instruções que poderiam lançar alguma exceção. O trecho de código dado a seguir exemplifica uma instrução de risco:

```
import javax.swing.JOptionPane;
public class Principal {
  public static void main(String args[]) {
    int n;
    n = Integer.parseInt(JOptionPane.showInputDialog("N:"));
    //instrução que poderia lançar uma exceção
  }
}
```

De acordo com o exemplo anterior, percebe-se que o método estático *parseInt()* da classe Integer tenta efetuar uma conversão de uma *string* lida do usuário para um número inteiro. O "risco" dessa conversão é que, caso o usuário digite um valor não

numérico, a conversão falhará ocasionando uma exceção. Nesse caso, uma exceção chamada *NumberFormatException* será lançada.

Outra situação que poderia lançar uma exceção diz respeito à manipulação de *arrays*. Caso haja a tentativa de acessar uma posição inexistente de um *array*, uma exceção chamada *ArrayIndexOutOfBoundsException* será lançada. Sem a existência de algum código de tratamento para esse tipo de exceção, a execução do programa será abruptamente interrompida.

Deve-se pensar em uma exceção como um objeto (na verdade, exceções são classes em Java, como será visto adiante), assim, para tratar uma exceção, é necessário: 1) definir um bloco de código que notifique ao compilador que dentro desse bloco há uma ou várias instruções de risco; e 2) para o bloco definido, estabelecer um ou mais blocos de código que serão executados caso a exceção ocorra.

Para desempenhar essas duas tarefas, deve-se usar a estrutura ***try...catch()***. A sintaxe básica é:

```
try {
  ...
}
catch(TipoExcecao var) {
  ...
}
```

Dentro do bloco ***try***, deve-se inserir as instruções que podem lançar alguma exceção, ou seja, instruções que devem ser controladas. Caso uma exceção realmente seja lançada por alguma instrução definida no bloco *try*, a execução desse bloco será interrompida, e imediatamente todo o código definido no bloco ***catch()*** será executado. É importante observar que o código definido em um bloco *catch()* somente será executado se ocorrer alguma exceção cujo tipo seja compatível ao tipo declarado no cabeçalho do bloco *catch()*.

Portanto, ao declarar o cabeçalho de um bloco *catch()*, é necessário definir um parâmetro do tipo da exceção que poderá ocorrer no bloco *try* correspondente. Considerando-se o exemplo de conversão para um número inteiro citado anteriormente, o código de tratamento de exceção seria:

```
import javax.swing.JOptionPane;
public class Principal {
  public static void main(String args[]) {
    int n;
    try {
      n = Integer.parseInt(JOptionPane.showInputDialog("N:"));
      //instrução que poderia gerar uma exceção
    }
```

```
    catch(NumberFormatException erro) {
      JOptionPane.showMessageDialog(null,”Erro na conversão”);
    }
  }
}
```

Pelo exemplo apresentado, percebe-se que a conversão somente será efetuada se a *string* lida do usuário realmente for um valor numérico. Caso contrário, uma exceção será lançada interrompendo a conversão. A partir daí, a máquina virtual buscará no programa o código de tratamento específico para essa exceção. Como existe um bloco *catch()*, cujo parâmetro está de acordo com o tipo de exceção ocorrida, todo o código definido nesse bloco será executado. É importante observar que, se o tipo do parâmetro declarado no cabeçalho do bloco *catch()* não for compatível com o tipo da exceção lançada, então o tratamento não será realizado, e a execução do programa será ainda interrompida.

Como citado anteriormente, uma exceção deve ser considerada como um objeto. A ocorrência de uma exceção refere-se, portanto, à criação de um objeto que guarde dados sobre a exceção ocorrida. Esse objeto será capturado pelo parâmetro declarado no cabeçalho do bloco *catch()*. Isso significa que esse parâmetro passará a guardar uma referência para o objeto, portanto, por meio dele, será possível ter acesso a métodos relacionados à exceção ocorrida.

Todo objeto de exceção possui um método chamado *printStackTrace()* que permite mostrar na tela informações sobre a exceção (Mensagem em vermelho). O trecho de código abaixo mostra o uso desse método para o exemplo anterior:

```
...
  try {
    n = Integer.parseInt(JOptionPane.showInputDialog(“N:”));
  }
  catch(NumberFormatException erro) {
    erro.printStackTrace();
  }
...
```

É importante observar que, para um mesmo bloco *try*, é possível haver dois ou mais blocos *catch()*, desde que os tipos de exceção dos seus parâmetros sejam diferentes. É particularmente útil especificar mais de um bloco *catch()* quando o bloco *try* envolver instruções que possam lançar exceções diferentes. Cada tipo de exceção deverá ser tratada por um determinado bloco *catch()*. O trecho de código dado a seguir mostra a sintaxe básica para definir mais de um bloco *catch()*.

```
try {
  ...
}
```

```
catch(TipoExcecao1 var) {
  ...
}
catch(TipoExcecao2 var) {
  ...
} ...
```

O bloco *finally*

Em muitas situações, ao especificar a estrutura *try...catch( )*, é necessário que, havendo ou não uma exceção, alguma tarefa seja obrigatoriamente executada ao final de uma operação. Por exemplo, durante a conexão com um banco de dados, será necessário, após efetuar uma determinada operação, que o programa encerre essa conexão, independentemente de ter havido alguma exceção ou não.

Para "forçar" a execução de uma ou várias instruções, independentemente de alguma exceção ter sido lançada, é necessário usar o bloco ***finally***. Esse bloco é opcional e, se for usado, deverá ser inserido após o último bloco *catch( )* em uma estrutura *try...catch()*. A sintaxe básica para a estrutura *try...catch()...finally* é dada a seguir:

```
try {
  ...
}
catch(TipoExcecao var) {
  ...
}
//Poderia haver outros blocos catch() aqui
finally {
  ...
}
```

O trecho de código dado a seguir mostra um método que recebe, como parâmetros, uma posição e um valor, e armazena em um *array a* o valor recebido na posição informada. Imagine que o *array* já tenha sido declarado e instanciado adequadamente, e o método *atribuiValor()* tenha acesso direto a esse *array*.

```
public boolean atribuiValor(int pos, int valor) {
  try {
    a[pos] = valor;
    return true;
  }
  catch(ArrayIndexOutOfBoundsException erro) {
    return false;
```

```
  }
  finally {
    ... //código que será executado ao final do método
  }
}
```

De acordo com esse exemplo, todo o código definido no bloco *finally* será executado, apesar de haver instruções *return* especificadas nos blocos *try* e *catch()*. Antes de a instrução *return* ser executada, o bloco *finally* será executado. Nesse ponto, é importante observar que, se não houvesse o bloco *finally*, e houvesse instruções após o bloco *catch()*, haveria um erro de compilação (*Unreachable code*).

Para o caso dessa exceção específica, é importante ressaltar que a melhor alternativa seria evitar que a exceção fosse lançada, usando-se, por exemplo, a estrutura IF/ELSE. O exemplo anterior apenas foi usado para demonstrar que o bloco *finally*, sempre que existir em uma estrutura *try..catch()*, será obrigatoriamente executado.

Encaminhamento de exceções: a cláusula *throws*

Em muitos casos, um determinado método ou ainda uma classe não possui responsabilidade de tratar exceções que podem ocorrer no código. Entretanto, ainda existe a necessidade de indicar a outros métodos ou classes que alguma exceção pode ser lançada durante a sua execução. Para indicar que um método apenas repassará alguma exceção, deve-se usar a cláusula ***throws*** na sua declaração.

A seguir, é apresentado o código de uma classe que possui como responsabilidade efetuar operações em um banco de dados. Cada método, nessa classe, é responsável por uma operação específica e todos eles, como devem se comunicar com o banco de dados, precisam tratar uma exceção chamada SQLException (localizada no pacote *java.sql*). Entretanto, esses métodos não serão os responsáveis em definir um tratamento específico para essa exceção, caso ela ocorra. Essa responsabilidade será delegada a outras classes que requisitarem os serviços da classe relacionada com o banco de dados.

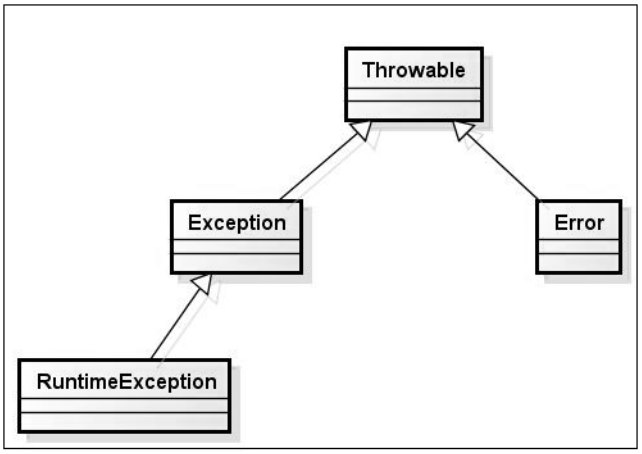
```
import java.sql.SQLException;
public class EmpregadoBD {
  public void salvaEmpregado(Empregado e) throws SQLException {
    ... //código que pode gerar a exceção SQLException
  }
  public void alteraEmpregado(Empregado e) throws SQLException {
    ... //código que pode gerar a exceção SQLException
  }
  ...
}
```

A exceção SQLException é uma exceção obrigatória em Java, isto é, se, na declaração de um método em uma classe, existir a cláusula *throws SQLException*, a classe

responsável por invocar esse método será obrigada a tratar essa exceção - usando a estrutura *try...catch()* - ou então delegar essa responsabilidade a outra classe. Encare essa situação como uma lei: ou trate ou declare a exceção. É importante observar que nem todas as exceções são obrigatórias, o que sugere que exista em Java uma classificação de exceções, como será estudado adiante.

Implementação de classes de exceção e a instrução *throw*

Como já foi explicado, o lançamento de uma exceção corresponde à criação de um objeto em Java. Dessa forma, existe uma hierarquia de classes na API Java que permite representar tipos de exceções diferentes, conforme mostra o diagrama de classes a seguir.



De acordo com o diagrama de classes anterior, a classe mãe de todas as classes que representam exceções em Java é a classe Throwable. Dessa classe, herdam as classes Exception e Error. Qualquer classe que estenda a classe Exception, direta ou indiretamente (Desde que não seja da classe RuntimeException), representará uma exceção obrigatória (*checked exception*). A classe SQLException, já mencionada, é um exemplo de uma classe que estende a classe Exception, representando, portanto, uma exceção que deverá ser obrigatoriamente tratada (ou declarada).

Já a classe Error representa exceções que estão, muitas vezes, fora de controle do programa como, por exemplo, problemas no hardware, falta de espaço em disco, falta de memória, etc. Um exemplo de uma classe que estende indiretamente a classe Error é a classe StackOverflowError.

Embora a classe RuntimeException estenda a classe Exception, ela representa exceções que não são obrigatórias (*unchecked exceptions*). As classes de exceção já observadas nos programas como, por exemplo, as classes NumberFormatException, ArrayIndexOutOfBoundsException e NullPointerException estendem direta ou indiretamente a classe RuntimeException.

Dessa forma, para se criar uma exceção, deve-se implementar uma classe que estenda qualquer uma das classes Exception, RuntimeException ou Error. Se a classe criada estender a classe Exception, então, automaticamente, a exceção representada por essa

classe será obrigatória. Caso a classe estenda a classe RuntimeException ou a classe Error, então a exceção representada não será obrigatória.

Uma vez que a classe que representa a exceção tenha sido implementada, deve-se definir em qual trecho de código e em quais circunstâncias a exceção será criada (encare a exceção como um objeto do tipo da classe de exceção implementada). Nesse ponto, é importante ressaltar que os exemplos anteriores apenas trataram ou então repassaram exceções lançadas por outros códigos, mas nenhuma exceção foi criada e lançada pelo nosso próprio código.

Para exemplificar o lançamento de exceções de classes implementadas em nosso código, vamos implementar uma classe bastante simples chamada Pessoa. Essa classe possuirá apenas um único atributo chamado *idade*, e o seu valor não poderá ser negativo. Para garantir que essa restrição seja cumprida, será implementada uma classe chamada IdadeNegativaException que representará uma exceção a ser lançada, caso haja a tentativa de se informar um valor de idade negativo para uma pessoa qualquer. A implementação dessa classe é apresentada abaixo:

```
public class IdadeNegativaException extends Exception {
  public String toString() {
    return "Erro: idade negativa.";
  }
}
```

Como pode ser observado no código, a classe IdadeNegativaException é uma classe comum, exceto pelo fato de estender a classe Exception. Como uma classe comum, essa classe poderia possuir quaisquer atributos e métodos, além de construtores. Além disso, essa classe representa uma exceção que deverá ser obrigatoriamente tratada em algum ponto do código.

A classe Pessoa será codificada sabendo-se que qualquer tentativa de definição de uma idade negativa causará o lançamento da exceção do tipo IdadeNegativaException. Além disso, a classe Pessoa não será a responsável em tratar a exceção, mas apenas lançá-la (criando um objeto do tipo IdadeNegativaException). Para lançar uma exceção, deve-se usar a instrução ***throw*** (e não *throws*), conforme é mostrado pelo código da classe Pessoa dado abaixo:

```
public class Pessoa {
  private int idade;
  public void setIdade(int i) throws IdadeNegativaException {
    if(i < 0)
      throw new IdadeNegativaException();
    else
      this.idade = i;
  }
  ...
```

```
}
```

De acordo com o código acima, o método *setIdade()* lançará uma exceção do tipo IdadeNegativaException, caso a idade informada como argumento seja negativa. Nesse ponto, a máquina virtual interromperá a execução do método e buscará o código de tratamento dessa exceção.

Como o método *setIdade()* não possui o código de tratamento dessa exceção, mas apenas delega essa responsabilidade a outra classe, a máquina virtual buscará esse código no método responsável pela invocação do método *setIdade()*. Caso esse método também delegue essa responsabilidade a outro lugar no código, a máquina virtual continuará realizando a busca até encontrar o código de tratamento para a exceção lançada.

Nesse exemplo, o método *setIdade()* para um objeto do tipo Pessoa será chamado a partir do método *main()* da classe Principal. Portanto, no método *main()*, deve-se especificar o código de tratamento dessa exceção, conforme é mostrado a seguir:

```
public class Principal {
  public static void main(String args[]) {
    Pessoa p = new Pessoa();
    try {
      p.setIdade(20); //a idade da pessoa valerá 20
      p.setIdade(-10); //a exceção será lançada e idade será 20
    }
    catch(IdadeNegativaException erro) {
      System.out.println(erro.toString());
    }
  }
}
```

Ainda há três questões importantes sobre esse exemplo que deverão ficar como exercícios. Primeiramente, caso o método *main()* não tratasse a exceção lançada pelo método *setIdade()*, haveria erro de compilação? E se o método *main()* apenas declarasse essa exceção (usando a cláusula *throws*)? Como a máquina virtual Java se comportaria diante dessa situação?

Em segundo lugar, vamos supor que fosse necessário adicionar um atributo *nome* à classe Pessoa e o valor desse atributo não pudesse ser igual a *null*. Como poderíamos criar uma exceção que fosse lançada caso houvesse a tentativa de atribuir o valor *null* ao atributo *nome*? Haveria duas exceções em uma mesma classe?

Como terceira questão, caso fosse necessário que a exceção lançada no método *setIdade()* guardasse o valor negativo informado como argumento, como deveríamos fazer? Vamos supor que, no código de tratamento de erros (classe Principal), fosse necessário imprimir o valor negativo informado como argumento que tivesse originado a exceção.

Referências bibliográficas

Deitel, H. M.; Deitel, P. J. Java - Como Programar. 4. ed., Bookman, 2002.

Oracle. *Learning the Java Language*. Disponível em: http://download.oracle.com/javase/tutorial/java/. Data de acesso: 02/01/2016.

Santos, R. Introdução à Programação Orientada a Objetos usando JAVA. Elsevier, 2003.

Sierra, K; Bates, B. Use a Cabeça! Java. 2. ed., O Reilly, 2005.