Tratamento de exceções no Java

Exceções em Java

A linguagem Java possui um mecanismo especial para o tratamento de erros que possam ocorrer em tempo de execução do programa. Diferentemente de outras linguagens, o surgimento de um erro ocasiona a interrupção imediata do programa, porém em Java podemos tratar esta situação de erro de uma forma adequada e evitando, assim, a interrupção do programa.

Uma exceção, basicamente é uma classe de Java representada na sua forma mais genérica pela classe `java.lang.Exception`, logo todas as exceções que ocorram ao longo da execução do seu programa podem ser tratadas como objetos do tipo `Exception`.

Uma característica importante sobre exceções é que, pelo fato delas poderem ocorrer a qualquer momento, estas são literalmente "lançadas" de volta para a cadeia de execução e chamada das classes.

Bloco try / catch

Como a exceção é lançada por toda a cadeia de classes do sistema, a qualquer momento é possível se "pegar" essa exceção e dar a ela o tratamento adequado.

Para se fazer este tratamento, é necessário pontuar que um determinado trecho de código que será observado e que uma possível exceção será tratada de uma determinada maneira, segue um exemplo deste novo bloco:

```
package material.excecao;

/**
 * Classe utilizada para demonstrar o bloco try / catch.
 */
public class ExemploExcecao {
  public static void main(String[] args) {
    try {
      /* Trecho de código no qual uma
       * exceção pode acontecer.
       */
    } catch (Exception ex) {
      /* Trecho de código no qual uma
       * exceção do tipo "Exception" será tratada.
       */
    }
  }
}
```

No caso acima, o bloco `try`, é o trecho de código em que uma exceção é esperada e o bloco `catch`, em correspondência ao bloco `try`, prepara-se para "pegar" a exceção ocorrida e dar a ela o tratamento necessário. Uma vez declarado um bloco `try`, a declaração do bloco catch torna-se obrigatória.

Na linha 12 o bloco `catch` declara receber um objeto do tipo `Exception`, lembrando do conceito da herança, todas as exceções do Java são classes filhas de `Exception`.

Algo importante de ser comentado, é que quando uma exceção ocorre, as demais linhas de código deixam de ser executadas até encontrar o bloco catch que irá tratar a exceção.

Exemplo:

```java
package material.excecao;

import java.util.Scanner;

/**
 * Classe que demonstra o uso do try / catch.
 */
public class ExemploTryCatch {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    try {
      System.out.print("Digite um valor inteiro..:");
      int numero1 = s.nextInt();
      System.out.print("Digite um valor inteiro..:");
      int numero2 = s.nextInt();

      System.out.println(numero1+ " + " + numero2 + " = " +
(numero1+numero2));
    } catch (Exception ex) {
      System.out.println("ERRO - Valor digitado nao e um numero
inteiro!");
    }
  }
}
```

Observemos o exemplo acima. Neste código, caso aconteça um erro na linha 13, as linhas de 14 a 17 não seriam executadas, retornando então o código a ser executado apenas a partir da linha 19 (trecho correspondente ao bloco `catch`).

Uma vez que uma exceção foi tratada por um bloco `catch`, a execução do programa segue normalmente.

Caso não ocorra erro durante a execução teremos a seguinte saída no console:

```
C:\>javac material\excecao\ExemploTryCatch.java
C:\>java material.excecao.ExemploTryCatch
Digite um valor inteiro..:5
Digite um valor inteiro..:3
5 + 3 = 8
```

Se no lugar de um número inteiro for digitado outra informação teremos a seguinte saída no console:

```
C:\>javac material\excecao\ExemploTryCatch.java
C:\>java material.excecao.ExemploTryCatch
Digite um valor inteiro..:10
Digite um valor inteiro..:abc
ERRO - Valor digitado nao e um numero inteiro!
```

Palavra-chave throw

Também é possível que você próprio envie uma exceção em alguma situação especifica, como em uma situação de login em que o usuário digita incorretamente sua senha. Para realizarmos tal tarefa é necessária a utilização da palavra-chave `throw` da seguinte maneira:

```
throw new << Exceção desejada >>();
```

Vamos ver um exemplo de lançamento de uma exceção do tipo `Exception`:

```java
package material.excecao;
```

```java
import java.util.Scanner;

/**
 * Classe utilizada para demonstrar o uso da palavra chave throw,
 * utilizada quando queremos lançar uma exceção.
 */
public class ExemploThrow {
  public static final String SENHASECRETA = "123456";

  public static void main(String[] args) {
    try {
      Scanner s = new Scanner(System.in);
      System.out.print("Digite a senha: ");
      String senha = s.nextLine();
      if(!senha.equals(SENHASECRETA)) {
        throw new Exception("Senha invalida!!!");
      }
      System.out.println("Senha correta!!!\nBem vindo(a)!!!");
    } catch (Exception ex) {
      System.out.println(ex.getMessage());
    }
  }
}
```

Note que, assim que a palavra-chave `throw` for utilizada, podemos tentar tratar a exceção no bloco `try` / `catch` ou lançar essa exceção nesse método, caso a exceção em questão seja do tipo checked. Observe também que a palavra reservada `new` foi utilizada, visto que a exceção é um novo objeto que deve ser criado na memória. Isso se faz necessário para que a exceção possa ser lançada por toda a pilha de execução até que seja devidamente tratada ou acarrete no término da aplicação.

Quando executamos este código, temos a seguinte saída no console, note que no primeiro teste entramos com uma senha invalida `abc`, portanto foi lançado a exceção, no segundo teste entramos com a senha valida `123456` e executamos a aplicação por completo.

```
C:\>javac material\excecao\ExemploThrow.java
C:\>java material.excecao.ExemploThrow
Digite a senha: abc
Senha invalida!!!

C:\>javac material\excecao\ExemploThrow.java
C:\>java material.excecao.ExemploThrow
Digite a senha: 123456
Senha correta!!!
Bem vindo(a)!!!
```

Alguns métodos importantes da classe `Exception`:

`printStackTrace()` - Imprime em tela a pilha de execução. Muito comum para auxiliar no diagnóstico de erros.

`getMessage()` - Retorna uma String com a mensagem contida na exceção.

`getClass()` - Retorna uma String com o nome complete da classe da exceção.

Bloco finnaly

A palavra-chave `finally` representa um trecho de código que será sempre executado, independentemente se uma exceção ocorrer. Por exemplo:

```java
package material.excecao;

import java.util.Scanner;

/**
 * Classe que demonstra o uso do bloco finally.
 */
public class ExemploFinally {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    try {
      int dividendo, divisor;
      System.out.print("Digite o valor do dividendo: ");
      dividendo = s.nextInt();
      System.out.print("Digite o valor do divisor: ");
      divisor = s.nextInt();

      if(divisor == 0) {
        throw new Exception("Nao eh permitido fazer uma divisao por
zero!");
      }

      System.out.println(dividendo+" / "+divisor+" = "+(dividendo /
divisor));
    } catch (Exception ex) {
      System.out.println("Erro: " + ex.getMessage());
    } finally {
      System.out.println("Bloco Finally.");
    }
  }
}
```

No exemplo anterior, a mensagem “Bloco Final!”, sempre será exibida, ocorrendo ou não um `Exception`.

Caso não ocorra erro durante a execução teremos a seguinte saída no console:

```
C:\>javac material\excecao\ExemploFinally.java
C:\>java material.excecao.ExemploFinally
Digite o valor do dividendo: 7
Digite o valor do divisor: 2
7 / 2 = 3
Bloco Finally
```

Se digitarmos zero no valor do divisor será lançado uma exceção, note que o bloco `finally` será executado mesmo que ocorra alguma exceção:

```
C:\>javac material\excecao\ExemploFinally.java
C:\>java material.excecao.ExemploFinally
Digite o valor do dividendo: 6
Digite o valor do divisor: 0
ERRO: Nao eh permitido fazer uma divisao por zero!
Bloco Finally
```

O bloco `finally` é muito utilizado quando queremos liberar algum recurso, como: uma conexão com o banco de dados, um arquivo de dados, etc. Veremos mais adiante alguns códigos que fazem uso do `finally` para este propósito.

Palavra-chave throws

Caso em algum método precise lançar uma exceção, mas você não deseja tratá-la, quer retorna-la para o objeto que fez a chamada ao método que lançou a exceção, basta utilizar a palavra chave `throws` no final da assinatura do método.

Quando utilizamos o `throws` precisamos também informar qual ou quais exceções podem ser lançadas.

Exemplo:

```
package material.excecao;

import java.util.Scanner;

/**
 * Classe utilizada para demonstrar o uso da palavra chave throws,
 * deixando para quem chamar o método tratar alguma possivel exceção.
 */
public class ExemploThrows {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    try {
      ExemploThrows et = new ExemploThrows();

      System.out.print("Digite o valor do dividendo: ");
      double dividendo = s.nextDouble();

      System.out.print("Digite o valor do divisor: ");
      double divisor = s.nextDouble();

      double resultado = et.dividir(dividendo, divisor);

      System.out.println("O resultado da divisao eh: " + resultado);
    } catch (Exception ex) {
      System.out.println(ex.getMessage());
    }
  }

  public double dividir(double dividendo, double divisor) throws
Exception {
    if(divisor == 0) {
      throw new Exception("Nao e permitido fazer uma divisao por
zero!");
    }

    return dividendo / divisor;
  }
}
```

Note que na linha 29 declaramos na assinatura do método que ele pode ou não lançar uma exceção do tipo `java.lang.Exception`.

E o método `public static void main(String[] args)` será responsável por tratar alguma exceção que o método `dividir(double dividendo, double divisor)` possa lançar.

Caso não ocorra erro durante a execução teremos a seguinte saída no console:

```
C:\>javac material\excecao\ExemploThrows.java
C:\>java material.excecao.ExemploThrows
Digite o valor do dividendo: 7
Digite o valor do divisor: 2
O resultado da divisao eh: 3.5
```

Se digitarmos zero no valor do divisor será lançado uma exceção pelo método `dividir()`, esta exceção será tratada pelo método `main()`:

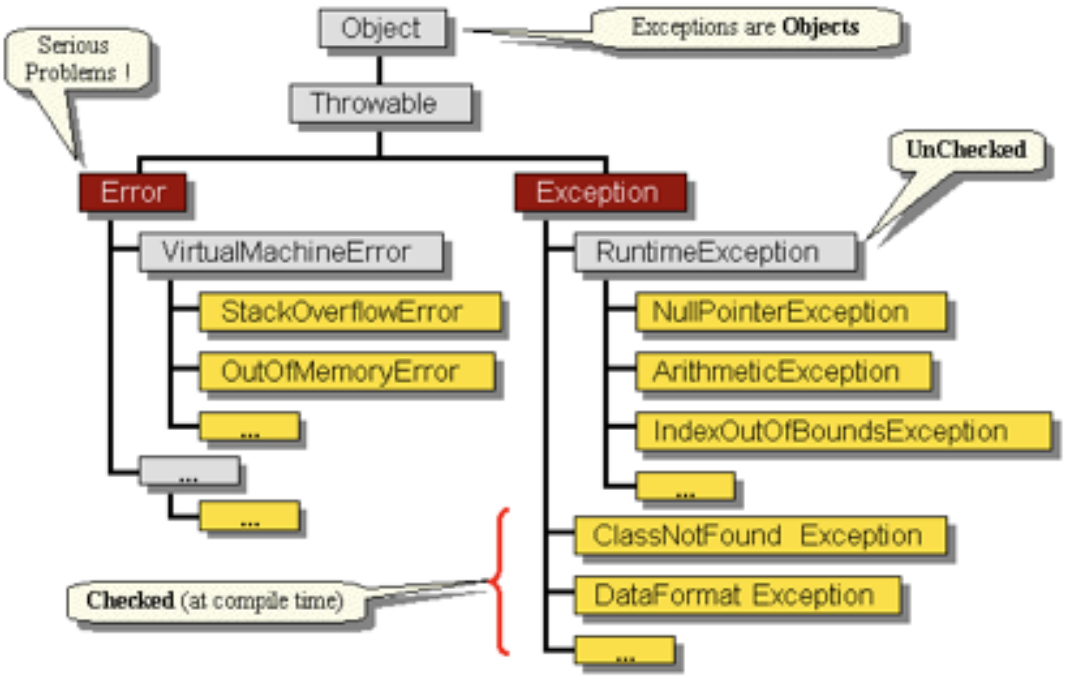
```
C:\>javac material\excecao\ExemploThrows.java
C:\>java material.excecao.ExemploThrows
Digite o valor do dividendo: 5
Digite o valor do divisor: 0
Nao e permitido fazer uma divisao por zero!
```

Hierarquia das Exceptions

A princípio, Java possui diversos tipos específicos de exceção, cada um deles diretamente relacionado a uma operação específica, por este motivo para que uma classe seja considerada uma exceção é imprescindível que ela de alguma forma seja filha de `java.lang.Exception`.

As exceções que são subclasses de `java.lang.Exception` são normalmente divididas em duas parte as Checked e Unchecked, que explicaremos mais adiante.

No quadro abaixo, temos um exemplo da hierarquia de algumas subclasses de Exception. Uma das subclasses mais comuns é a `java.lang.RuntimeException`, sobretudo para os usuários finais, pois como podem acontecer em qualquer parte do código durante sua execução, normalmente, quando não tratadas ocasionam na interrupção do programa.



Checked Exceptions

As checked exceptions, são exceções já previstas pelo compilador. Usualmente são geradas pela palavra chave `throw` (que é discutido mais adiante). Como ela é prevista já em tempo de compilação, se faz necessária a utilização do bloco `try` / `catch` ou da palavra chave `throws`.

A princípio, todas as classes filhas de Exception são do tipo checked, exceto pelas subclasses de `java.lang.RuntimeException` (exceção em tempo de execução).

Unchecked Exceptions

Um dos fatores que tornam a `RuntimeException` e suas classes filhas tão específicas em relação as demais subclasses de `Exception` é que elas são exceções não diretamente previstas por acontecer em tempo de execução, ou seja, são unchecked exceptions. Por exemplo:

```
package material.excecao;

import java.util.Scanner;

/**
 * Classe utilizada para demonstrar exceções filhas de
RuntimeException.
 */
public class ExemploRuntimeException {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print("Digite um numero inteiro..: ");
    int numero = s.nextInt();

    System.out.println("Numero lido: " + numero);
  }
}
```

Observe que no trecho de código acima, temos uma variável do tipo `int` recebendo uma entrada do usuário também do tipo `int`. Porém, vamos supor que nosso usuário não digite um inteiro, mas sim um caractere.

```
C:\>javac material\excecao\ExemploRuntimeException.java
C:\>java material.excecao.ExemploRuntimeException
Digite um numero inteiro..: abc
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at material.excecao.ExemploRuntimeException.main(ExemploRuntime
Exception.java:15)
```

Este tipo de exceção, que acontece somente em tempo de execução, a princípio não era tratado e uma exceção do tipo `java.util.InputMismatchException` será gerada e nosso programa é encerrado. Agora iremos prever este erro, utilizando o bloco `try` / `catch`:

```
package material.excecao;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
```

```
 * Classe utilizada para demonstrar exceções filhas de
RuntimeException.
 */
public class ExemploRuntimeException2 {
  public static void main(String[] args) {
    int numero = 0;
    boolean validado = false;
    while(!validado) {
      try {
        Scanner s = new Scanner(System.in);
        System.out.print("Digite um numero inteiro..: ");
        numero = s.nextInt();
        validado = true;
      } catch (InputMismatchException ex) {
        System.out.println("O valor inserido nao e um numero
inteiro!");
      }
    }
    System.out.println("O valor lido foi: " + numero);
  }
}
```

Desta forma, a digitação incorreta do usuário será tratada e uma mensagem de erro será exibida caso uma exceção aconteça, a variável validado não receberia o valor true.

```
zxcvr
C:\>javac material\excecao\ExemploRuntimeException2.java
C:\>java material.excecao.ExemploRuntimeException2
Digite um numero inteiro..: abc
O valor inserido nao e um numero inteiro!
Digite um numero inteiro..: x
O valor inserido nao e um numero inteiro!
Digite um numero inteiro..: 7
O valor lido foi: 7
```

Errors

Errors são um tipo especial de Exception que representam erros da JVM, tais como estouro de memória, entre outros. Para este tipo de erro normalmente não é feito tratamento, pois sempre quando um `java.lang.Error` ocorre a execução do programa é interrompida.

Tratando múltiplas Exceções

É possível se tratar múltiplos tipos de exceção dentro do mesmo bloco `try` / `catch`, para tal, basta declara-los um abaixo do outro, assim como segue:

```
package material.excecao;

import java.util.InputMismatchException;

/**
 * Classe utilizada para demonstrar o uso de varios catchs.
 */
```

```java
public class ExemploMultiplasExcecoes {
  public static void main(String[] args) {
    try {
      /* Trecho de código no qual uma exceção pode acontecer. */
    } catch (InputMismatchException ex) {
      /* Trecho de código no qual uma exceção
         do tipo "InputMismatchException" será tratada. */
    } catch (RuntimeException ex) {
      /* Trecho de código no qual uma exceção
         do tipo "RuntimeException" será tratada. */
    } catch (Exception ex) {
      /* Trecho de código no qual uma exceção
         do tipo "Exception" será tratada. */
    }
  }
}
```

As exceções tratadas pelos catchs devem seguir a ordem da mais especifica para a menos especifica.

Criando sua exceção

Na linguagem Java podemos também criar nossas próprias exceções, normalmente fazemos isso para garantir que nossos métodos funcionem corretamente, dessa forma podemos lançar exceções com mensagens de fácil entendimento pelo usuários, ou que possa facilitar o entendimento do problema para quem estiver tentando chamar seu método possa tratar o erro.

No exemplo abaixo vamos criar uma exceção chamada **ErroDivisao** que será lançada quando ocorrer uma divisão incorreta, para isso precisamos criar esta classe filha de Exception.

```java
package material.excecao;

/**
 * Exceção que deve ser lançada quando uma divisão é invalida.
 */
public class ErroDivisao extends Exception {
  public ErroDivisao() {
    super("Divisao invalida!!!");
  }
}
```

Agora vamos criar a classe TesteErroDivisao, que iremos utilizar para testar o lançamento da nossa exceção ErroDivisao, crie o método restoDaDivisao que irá calcular o resto da divisão de dois números e lançara a exceção ErroDivisao caso tenha o valor do divisor maior que o valor do dividendo.

```java
package material.excecao;

import java.util.Scanner;

/**
 * Classe utilizada para demonstrar o uso da exceção ErroDivisao.
 */
public class TesteErroDivisao {
  public static void main(String[] args) {
    try {
```

```
      Scanner s = new Scanner(System.in);
      System.out.print("Digite o valor do dividendo: ");
      int numero1 = s.nextInt();

      System.out.print("Digite o valor do divisor: ");
      int numero2 = s.nextInt();

      TesteErroDivisao teste = new TesteErroDivisao();

      System.out.println("Resto: " + teste.restoDaDivisao(numero1,
numero2));
    } catch (ErroDivisao ex) {
      System.out.println(ex.getMessage());
    }
  }

  public int restoDaDivisao(int dividendo, int divisor) throws
ErroDivisao {
    if(divisor > dividendo) {
      throw new ErroDivisao();
    }

    return dividendo % divisor;
  }
}
```

Quando executamos a classe TesteErroDivisao caso não ocorra nenhum problema será apresentado o resto da divisão, se ocorrer algum erro será apresentado a mensagem “Divisão Invalida!!!”.

```
C:\>javac material\excecao\TesteErroDivisao.java
C:\>java material.excecao.TesteErroDivisao
Digite o valor do dividendo: 5
Digite o valor do divisor: 2
Resto: 1

C:\>javac material\excecao\TesteErroDivisao.java
C:\>java material.excecao.TesteErroDivisao
Digite o valor do dividendo: 3
Digite o valor do divisor: 7
Divisao invalida!!!
```