

Disciplina: PROGRAMAÇÃO I

Aula 9: Tratamento de exceções

I Apresentação

Trabalharemos com o tratamento de exceções em Java, analisando algumas situações inesperadas, tais como: divisão por zero, erro de conversão de tipos de dados, erro na abertura de um arquivo, entre outras.

Todas essas situações em Java são chamadas de exceções, e existe um mecanismo específico para tratá-las. Abordaremos também o processo de captura de uma exceção com o conjunto try/catch, além do conceito de lançamento de exceção com o uso de throw e proteção de métodos com o uso de throws.

Objetivo

- Examinar o conceito de exceção;
- Identificar e aplicar o conceito de captura de exceção com try/catch;
- Definir e aplicar o conceito de lançar exceção com throw e throws.

Primeiras palavras

Uma coisa muito comum no uso de sistemas computacionais são os problemas oriundos de codificação errada ou do uso incorreto da aplicação pelo usuário. Problemas de execução são normais, mas devemos tomar o máximo de cuidado para minimizá-los, uma vez que dificilmente iremos abranger todos os prováveis problemas que poderão ocorrer durante a execução de programa.

Antigamente, as linguagens de programação não ofereciam suporte ao tratamento de erros e os programas simplesmente encerravam a sua execução. Isso é um grande problema atualmente, pois, se antigamente, os sistemas executavam em ambientes locais e controlados, hoje em dia os sistemas executam de forma distribuída e remota.

Ter necessidade de reiniciar um sistema nos dias de hoje pode ser um grande transtorno e, principalmente, pode gerar enormes prejuízos. Para evitar situações como essas, as linguagens de programação evoluíram, dando suporte ao tratamento de erros durante a execução da aplicação.

A esse tratamento de erros durante a execução chamamos Tratamento de Exceções.

Um tratamento de exceções, quando utilizado em sistemas computacionais, diminui a probabilidade do sistema encerrar abruptamente¹, prejudicando um ou mais usuários, além de poder impactar em outros sistemas que, por ventura, possam estar relacionados a ele.



Fonte: shutterstock

| Tratamento de exceções

O **tratamento de exceções** é comum em várias linguagens de programação atualmente, sendo a **linguagem Java um exemplo delas**.

O tratamento de exceções pode ser usado para controlar condições anormais de processamento (execução) ou de compilação.

O tratamento de exceções para possíveis problemas de compilação não será tratado aqui, ficando o Tratamento de Exceções em tempo de execução (processamento) como alvo do nosso estudo.

O tratamento de exceções em tempo de execução permite que problemas de código ou ações equivocadas dos usuários encerrem inadvertidamente nossa aplicação. Podemos tratar exceções em parte do código, proteger métodos de forma que os mesmos só possam ser usados em conjunto com o tratamento de exceções e ainda podemos fazer com que o sistema gere exceções.

As palavras reservadas para o Tratamento de Exceções na linguagem são:

1

Try

3

Finally

2

Catch

4

Throw

Throws

Pode ocorrer uma ou mais exceções em um mesmo trecho de código, por isso, **devemos identificar as possíveis exceções que podem ser geradas por cada trecho de código durante o desenvolvimento.**

As exceções obedecem a uma hierarquia, razão pela qual devemos tratar primeiramente as exceções mais especializadas, deixando as exceções mais genéricas por último.

Exceções mais comuns


- 1 Entrada de dados: o usuário insere dados incompatíveis, tais como digitar um texto ao invés de um valor, ou digitar um valor real onde era esperado um inteiro;
- 2 Tentar abrir um arquivo inexistente;
- 3 Tentar acessar um elemento inexistente de um vetor;
- 4 Tentar realizar uma consulta em um servidor de banco de dados que gere algum erro, como a inserção de um registro com chave primária duplicada (chaves primárias são únicas e não há dois registros com o mesmo valor em um campo que é uma chave primária);
- 5 Conexões de redes perdidas.

Características

- 1 Exceções são condições anormais que podem representar erros ocorridos durante a execução. Irão interromper o fluxo de execução normal, desviando o processamento para a realização do tratamento de exceções;
- 2 O processo de geração de exceções realizado pela linguagem é chamado disparo ou lançamento;
- 3 O processo de tratamento de exceções é chamado de captura;
- 4 Exceções que não forem capturadas pela ausência ou falta de tratamento adequado devem ser repassadas. O repasse de exceções será objeto de um tópico específico a ser apresentado em breve;
- 5 Exceções em Java são objetos e, por isso, toda geração de exceção gera um objeto;

7 O processo de tratamento de exceções é conhecido pelo paradigma de Tentar-e-Capturar (try-and-catch), no qual, se um comando pode gerar uma exceção, ele deve ser tentado, e, se durante sua execução uma exceção for gerada, a mesma deverá ser capturada.

Tratamento de Exceções com o uso de:

 Clique nos botões para ver as informações.

Try



O try delimita o bloco com o conjunto de instruções que podem gerar exceções. Nele devem ser contidas todas as instruções que poderão gerar exceções, ou um conjunto de instruções, entre as quais uma ou mais poderão gerar uma exceção. É necessário ao menos um bloco catch ou finally.

Catch



Cada cláusula catch é responsável por realizar um tratamento de exceção, mas de forma hierárquica. Temos exceções específicas (ou especialistas) e exceções mais genéricas (generalistas); portanto, podemos realizar o tratamento específico ou genérico para cada tipo de exceção gerada. Podemos utilizar uma ou mais cláusulas catch quanto forem necessárias. Após a exceção ser lançada, o fluxo passa para o conjunto de cláusulas catch e é esperado que uma delas capture a exceção. Dessa forma, as exceções especialistas devem ser incluídas na frente das cláusulas de tratamento de exceções generalistas. Veremos mais adiante a hierarquia entre as exceções.

Finally



Essa cláusula é opcional e, quando existe, será sempre executada. Independentemente de se ocorreu ou não a exceção, a cláusula finally será sempre executada. Entre as suas funções, pode-se destacar a liberação de recursos. Muitas exceções são geradas em função de alocações de recursos, tais como: acesso a arquivos, acesso a servidores de banco de dados, conexões de rede, dentre outros. Todos esses recursos devem ser liberados após a realização da tarefa, ou da existência da exceção, se for o caso. Dessa forma, esses recursos devem ser liberados para não sobrecarregar o sistema.

Sintaxe para o uso do try / catch / finally

```
try {  
    Comandos do Try  
}  
  
catch ( TipoDaExceção1 objetoDaExceção1 ) {  
    Tratamento da Exceção1  
}  
  
catch ( TipoDaExceção2 objetoDaExceção2 ) {
```



```
Tratamento da Exceção2  
  
}  
// outras exceções se necessário  
[finally] { // opcional  
    Comandos do Finally  
}
```

I Tipo e Objeto da Exceção

O Tipo da Exceção é a declaração da classe à qual pertence a exceção, e Objeto da Exceção é a identificação do objeto para execução das tarefas pertinentes ao tratamento que está sendo realizado.

```
public class ExemploTratamento {  
    public static void main(String[] args) {  
        try {  
            // conjunto de instruções que podem  
            // gerar uma ou mais exceções  
        }  
  
        catch ( ArithmeticException e ) {  
            // trata exceções do tipo ArithmeticException  
            // ( é subclasse de RuntimeException )  
        }  
  
        catch ( RuntimeException e ) {  
            // trata outras exceções RuntimeException  
            // ( é subclasse de Exception )  
        }  
  
        catch ( Exception e ) {  
            // trata todas as demais exceções da classe Exception  
        }  
    }  
}
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

I Hierarquia de classes para o Tratamento de Exceções

Existe uma hierarquia a ser obedecida ao se realizar um tratamento de exceções. As exceções de menor nível hierárquico (subclasses) devem preceder as classes de maior hierarquia (superclasses) quando da criação das cláusulas catch. Isso é importante e lógico, porque, se uma classe de maior hierarquia estiver na frente de uma de menor hierarquia, a ordem de análise não permitirá que a captura chegue à classe de menor hierarquia, sendo tratada pela classe de maior hierarquia.

O compilador Java evita que se coloque uma classe de maior hierarquia à frente de uma de menor de menor hierarquia em um conjunto de cláusulas catch. Sendo assim, o compilador gera um erro e não compila situações, como no exemplo a seguir:

```

public class ExemploTratamento {
    public static void main(String[] args) {
        try {
            // conjunto de instruções que podem
            // gerar uma ou mais exceções</p>
        }

        catch (Exception e) {
            // trata todas as demais exceções da classe Exception
        }
    }
}

```

Unreachable catch block for RuntimeException. It is already handled by the catch block for Exception

```

        catch (RuntimeException e) {
            // trata outras exceções RuntimeException
            // ( é subclasse de Exception )
        }
    }
}

```

Multiple markers at this line

- Unreachable catch block for ArithmeticException. It is already handled by the catch block for RuntimeException
- Unreachable catch block for ArithmeticException. It is already handled by the catch block for Exception

```

        catch (ArithmeticException e) {
            // trata exceções do tipo ArithmeticException
            // ( é subclasse de RuntimeException )
        }
    }
}

```

Notas

- 1 Como Exception é superclasse de RuntimeException, a cláusula Exception irá capturar toda e qualquer exceção, inclusive as exceções de RuntimeException. Dessa forma, uma RuntimeException nunca será capturada;

- 2 Da mesma forma, como RuntimeException é superclasse de ArithmeticException, a cláusula RuntimeException irá capturar toda e qualquer exceção de execução, inclusive as exceções de ArithmeticException. Dessa forma, uma ArithmeticException nunca será capturada;

A figura a seguir demonstra a hierarquia para as principais classes. Entretanto, como são muitas as classes de tratamento de exceções, não seria possível apresentar de forma legível a hierarquia completa.

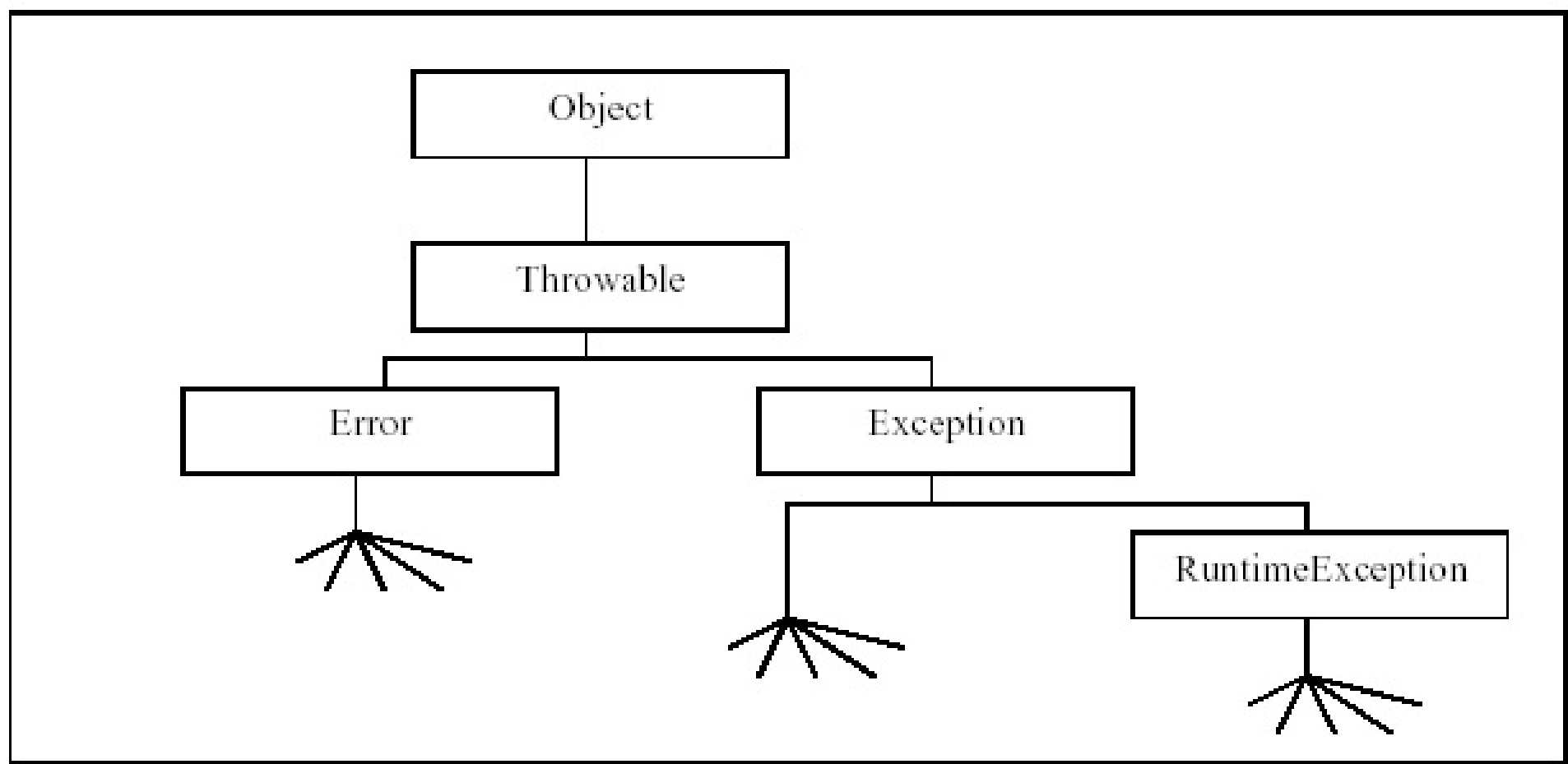


Figura 1: Hierarquia das classes de tratamento de exceções.

Error - erros na execução da máquina Virtual Java ou falhas de hardware.

Exception - erros na execução do código Java.

RuntimeException - erros de "programação". Não é obrigatório o tratamento dessas exceções; caso fosse, a codificação Java seria muito trabalhosa.

Analisando a hierarquia e os conceitos apresentados, devemos entender que é muito importante o conhecimento da hierarquia das classes de tratamento de exceções, e que não devemos, e não podemos colocar uma classe de maior hierarquia na frente de classes de menor hierarquia na montagem do conjunto de cláusulas catch em um tratamento de exceções.

[Exemplo prático e comum na criação de um tratamento de exceções para uma aplicação](#)

Clique no botão acima.

Classe AppExemplo: faz a entrada de dois números inteiros e apresenta o resultado da divisão do primeiro (numerador) pelo segundo (denominador).

```
import java.util.Scanner;
public class AppExemplo {
    public static void main(String[] args) {
        Scanner ent = new Scanner(System.in);
        int num, deno, resultado;
        System.out.println("Digite o numerador para a divisão:");
```



```
        num = Integer.parseInt(ent.nextLine());
        System.out.println("Digite o denominador para a divisão:");
        deno = Integer.parseInt(ent.nextLine());
        resultado = num / deno;
        System.out.println("O resultado da divisão é:" + resultado);
    }
}
```

● ● ● Primeiro teste de execução, com entradas dos valores de 120 e 12:

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
12
O resultado da divisão é:10
```

● ● ● Segundo teste de execução, com entradas dos valores de 120 e 0:

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at AppExemplo.main(AppExemplo.java:12)
```

● ● ● Terceiro teste de execução, com entradas dos valores de 120 e alo (texto e não numérico):

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
ola
Exception in thread "main" java.lang.NumberFormatException: For input string: "ola"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at AppExemplo.main(AppExemplo.java:11)
```

Notas:

1

No primeiro caso, os valores de entradas estavam de acordo com o esperado. O cálculo foi realizado corretamente e o resultado 10 foi exibido;

2

No segundo caso, os valores de entrada eram válidos, mas não podemos realizar uma divisão por 0 (zero), o que gerou uma exceção. Como a aplicação não possui tratamento de exceções, a aplicação encerrou com um erro:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at AppExemplo.main(AppExemplo.java:12)
```

No terceiro caso, os valores de entradas não estavam de acordo com o esperado. Para o segundo valor foi digitado um texto, gerando uma nova exceção na entrada de dados:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "ola"
```

3

```
at java.lang.NumberFormatException.forInputString(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at AppExemplo.main(AppExemplo.java:11)
```

Observamos que, mesmo em uma pequena aplicação, duas exceções diferentes ocorreram, encerrando a aplicação de forma abrupta em função das exceções geradas e não tratadas.

O primeiro passo para realizar o tratamento de exceções é identificar as possíveis exceções que, no caso, foram identificadas no encerramento da aplicação:

- **`java.lang.ArithmeticException`**
- **`java.lang.NumberFormatException`**

Uma vez identificadas as exceções, podemos retornar à codificação e melhorar nossa aplicação, incluindo o tratamento de exceções necessário:

Classe AppExemplo: faz a entrada de dois números inteiros e apresenta o resultado da divisão do primeiro (numerador) pelo segundo (denominador) **com o tratamento de exceções**.

```
import java.util.Scanner;
public class AppExemplo {
    public static void main(String[] args) {
        Scanner ent = new Scanner(System.in);
        int num, deno, resultado;
        try {
            System.out.println("Digite o numerador para a divisão:");
            num = Integer.parseInt(ent.nextLine());
            System.out.println("Digite o denominador para a divisão:");
            deno = Integer.parseInt(ent.nextLine());
            resultado = num / deno;
            System.out.println("O resultado da divisão é:" + resultado);
        }
        catch ( ArithmeticException e ) {
            System.out.println("Exceção gerada:" + e.getMessage());
            System.out.println("Foi gerada uma exceção aritmética, divisão por zero.");
            System.out.println("O denominador não pode ser 0 ( zero ) .");
        }
        catch ( NumberFormatException e ) {
            System.out.println("Exceção gerada:" + e.getMessage());
            System.out.println("Foi gerada uma exceção de conversão de dados.");
            System.out.println("Os valores devem ser números inteiros.");
        }
        catch ( Exception e ) {
            System.out.println("Outras exceções geradas poderão ser tratadas aqui.");
        }
    }
}
```

Primeiro teste de execução, com entradas dos valores de 120 e 12:

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
12
O resultado da divisão é:10
```

Segundo teste de execução, com entradas dos valores de 120 e 0:

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
0
Exceção gerada:/ by zero
```

Foi gerada uma exceção aritmética, divisão por zero.
O denominador não pode ser 0 (zero) .

Terceiro teste de execução, com entradas dos valores de 120 e alo (texto e não numérico):

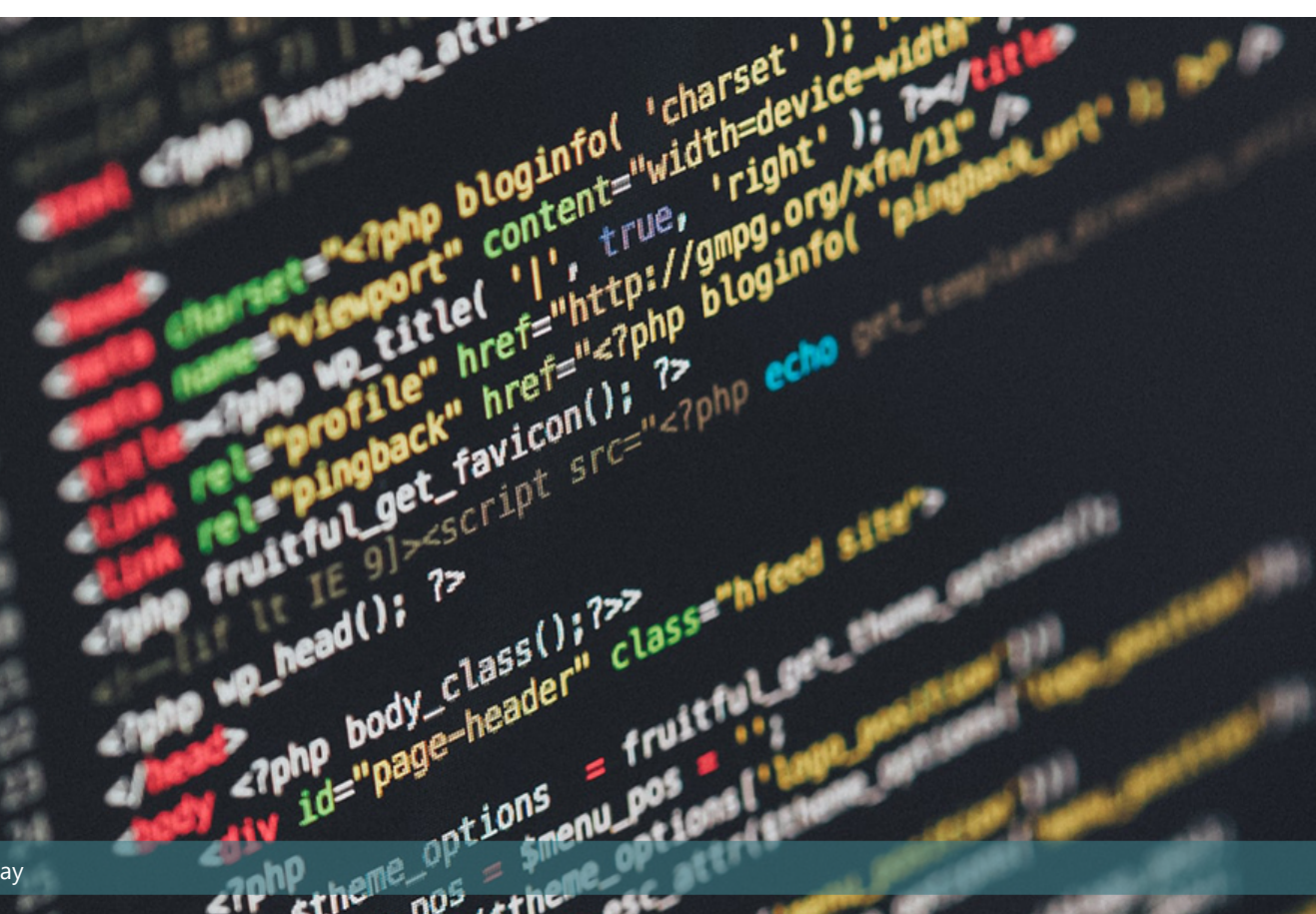
```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
Ola
Exceção gerada:For input string: "Ola"
Foi gerada uma exceção de conversão de dados.
Os valores devem ser números inteiros.
```

Nota:

Com o tratamento de exceções, a aplicação não encerrou em nenhum dos testes. Nos casos em que as exceções foram geradas (lançadas), houve a captura e, de acordo com o tipo de exceção gerada, o fluxo foi desviado para a cláusula catch que capturou.

Podemos observar que, se em uma aplicação tão simples identificamos duas exceções possíveis, podemos inferir que, em sistemas maiores e mais complexos, o tratamento de exceções é fundamental. Aplicações com acesso a banco de dados e disponibilidade para conexões por rede são exemplos práticos de que o tratamento de exceções é fundamental para o desenvolvimento de sistemas modernos e mais seguros.

No exemplo anterior, apenas realizamos avisos ao usuário, mas temos a linguagem Java inteira à disposição para programarmos trechos ou métodos específicos para resolver outros tipos de problemas, e de forma transparente, para o usuário, que não será notificado de que ações de recuperação foram executadas. Nesses casos, o usuário só será notificado caso a programação de recuperação não obtenha êxito.



I Repasses de exceções não tratadas

Como em Java podemos realizar chamadas a métodos e novos métodos podem chamar outros métodos, o tratamento de exceções pode ser realizado em vários níveis em nossas aplicações.

Vamos imaginar a seguinte situação:

a aplicação chama para a execução o método `metodo1()`, que chama para execução o método `metodo2()`, que chama para execução o método `metodo3()`. Nesse momento, temos quatro métodos em execução (`main()`, `metodo1()`, `metodo2()` e `metodo3()`) e o fluxo se encontra no método `metodo3()`, veja a seguir:

Notas:



O método `metodo3()` lança uma exceção, mas, se o método `metodo3()` não possuir tratamento de exceções para essa instrução (estar em um `try`), a exceção é propagada de volta ao ponto de chamada do método `metodo3()` no método `metodo2()`;



Se o método `metodo2()` possuir tratamento de exceções no ponto de chamada do método `metodo3()` (estar em um `try`), a exceção gerada no método `metodo3()` poderá ser tratada no método `metodo2()`. Caso contrário, a exceção será propagada de volta para o ponto de chamada do método `metodo2()` no método `metodo1()`;



Se o método `metodo1()` possuir tratamento de exceções no ponto de chamada do método `metodo2()` (estar em um `try`), a exceção gerada no método `metodo2()` poderá ser tratada no método `metodo1()`. Caso contrário, a exceção será propagada de volta para o ponto de chamada do método `metodo1()` no método `main()`;



Se o método `main()` tiver algum tratamento de exceções no ponto de chamada do método `metodo1()`, a exceção poderá ser tratada. Caso contrário, a aplicação encerrará e apresentará a linha em que foi gerada a exceção no método `metodo3()`, bem como as linhas que fizeram parte do caminho de execução até o lançamento da exceção.

Se o texto estiver um pouco difícil, veja se a figura a seguir facilita o entendimento:

Repasse de exceções:

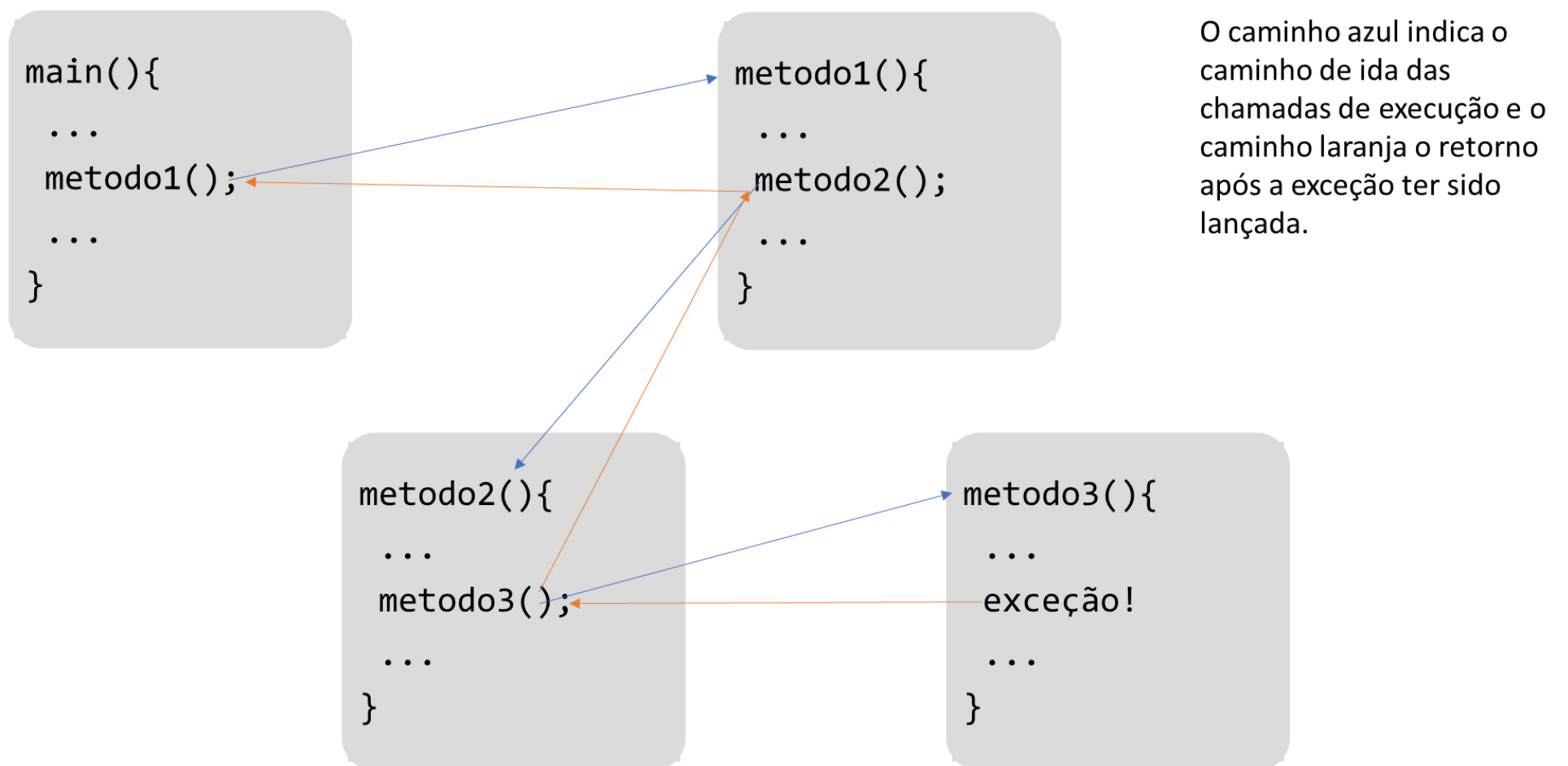


Figura 2: Caminhos de execução e repasse da exceção.

Se, em qualquer um dos métodos em execução, alguma chamada estiver em um tratamento de exceções (dentro de um `try`), a exceção poderá ser capturada e o controle da aplicação poderá ser recuperado, antes que a aplicação encerre em função da exceção. Essa característica é chamada de repasse da exceção.

Repasse da exceção é quando uma exceção não é capturada no local em que foi gerada (lançada), mas, mesmo assim, pode ser capturada em outra parte do código.

Throws - métodos podem gerar exceções

Alguns métodos na biblioteca da linguagem Java podem gerar exceções. Um método pode gerar uma ou mais exceções. Quando um método pode gerar uma exceção, devemos incluir em sua declaração essa informação. Isso fará com que esse método, ao ser utilizado, seja dentro de um tratamento de exceções (bloco `try`). Muitos métodos da biblioteca do Java possuem essa restrição, principalmente os métodos de acesso a banco de dados, arquivos, conexões de rede, dentre outros.

Um método que pode gerar uma exceção é declarado com o uso da palavra reservada throws.

Vamos transferir o código de execução da aplicação do exemplo anterior para o método metodoExcecao() da classe Excecao, mas vamos manter o tratamento de exceções na aplicação. Veja a seguir:

Classe AppTeste: faz a chamada ao método metodoExcecao() e possui o tratamento de exceções para atender aos requisitos do método, uma vez que este pode lançar duas exceções através da instrução throws. **O uso do throws em um método faz com que o mesmo deva ser usado dentro de um tratamento de exceções.**

```
import java.util.Scanner;
public class AppTeste {
    public static void main(String[] args) {
        Excecao ex = new Excecao();
        try {
            ex.metodoExcecao();
        } catch (ArithmeticException e) {
            System.out.println("Exceção gerada:" + e.getMessage());
            System.out.println("Foi gerada uma exceção aritmética, divisão por zero.");
            System.out.println("O denominador não pode ser 0 ( zero ) .");
        } catch (NumberFormatException e) {
            System.out.println("Exceção gerada:" + e.getMessage());
            System.out.println("Foi gerada uma exceção de conversão de dados.");
            System.out.println("Os valores devem ser números inteiros.");
        } catch (Exception e) {
            System.out.println("Outras exceções geradas poderão ser tratadas aqui.");
        }
    }
}
```

Classe Excecao: possui o método metodoExcecao(), que pode lançar exceções, mas não há tratamento de exceções no método, nem tão pouco na classe. **O método deverá ser usado dentro de um tratamento de exceções (bloco try) no ponto onde for chamado.**

```
import java.util.Scanner;
public class Excecao {
    public void metodoExcecao() throws ArithmeticException, NumberFormatException {
        Scanner ent = new Scanner(System.in);
        int num, deno, resultado;
        System.out.println("Digite o numerador para a divisão:");
        num = Integer.parseInt(ent.nextLine());
        System.out.println("Digite o denominador para a divisão:");
        deno = Integer.parseInt(ent.nextLine());
        resultado = num / deno;
        System.out.println("O resultado da divisão é:" + resultado);
    }
}
```

● ● ● Primeiro teste de execução, com entradas dos valores de 120 e 12:

Digite o numerador para a divisão:

120

Digite o denominador para a divisão:

12

O resultado da divisão é:10

● ● ● Segundo teste de execução, com entradas dos valores de 120 e 0:

Digite o numerador para a divisão:

120

Digite o denominador para a divisão:

0

Exceção gerada:/ by zero

Foi gerada uma exceção aritmética, divisão por zero.

O denominador não pode ser 0 (zero) .

● ● ● Terceiro teste de execução, com entradas dos valores de 120 e alo (texto e não numérico):

Digite o numerador para a divisão:

120</p>

Digite o denominador para a divisão:

01a

Exceção gerada:For input string: "01a"

Foi gerada uma exceção de conversão de dados.

Os valores devem ser números inteiros.

Notas:

- 1 Não vimos diferenças na execução, mas a técnica de proteção é diferente;
- 2 A chamada para execução do método metodoExcecao() deverá ser realizada dentro de um bloco try;
- 3 As exceções foram geradas dentro do método metodoExcecao(), mas, como não há tratamento de exceções dentro do método, as exceções foram repassadas para o método main() da aplicação;

4

Como o método `main()` da aplicação possui tratamento de exceções, as exceções geradas no método `metodoExcecao()` foram capturadas pelo bloco `try` do método `main()`, recuperando o controle da aplicação e realizando a captura e tratamento das exceções geradas.

| Throw - podemos gerar uma exceção explicitamente em nossos sistemas

Uma exceção pode ser gerada não apenas pelo sistema, mas também podemos forçar que uma exceção seja lançada. **Para forçarmos o lançamento de uma exceção, usamos a palavra reservada `throw`.** É comum associarmos uma geração de exceção a uma condição. Assim, a forma mais comum no desenvolvimento de sistemas é o uso de um `throw` dentro de uma condicional `se (if)`.

Atenção

Podemos gerar uma exceção em qualquer parte do código de uma aplicação: essa ação é explícita pelo desenvolvedor e não depende do sistema.

Vamos incluir uma nova exceção às exceções definidas no método, **chamada `IllegalArgumentException`**, e vamos inserir um teste sobre o valor do denominador. Se for igual a 0 (zero), será lançada uma exceção explícita por programação e não pelo sistema. Foi incluído ainda um tratamento de exceções específico (`IllegalArgumentException`) na aplicação, para informar que essa exceção foi gerada pelo código, acompanhe.

Classe `AppTeste2`: faz a chamada ao método `metodoExcecao()` e foi incluída uma cláusula para tratar a nova exceção: **`IllegalArgumentException`**.

```
import java.util.Scanner;

public class AppTeste2 {
    public static void main(String[] args) {
        Excecao2 ex = new Excecao2();
        try {
            ex.metodoExcecao();
        } catch (ArithmeticException e) {
            System.out.println("Exceção gerada: " + e.getMessage());
            System.out.println("Foi gerada uma exceção aritmética, divisão por zero.");
            System.out.println("O denominador não pode ser 0 ( zero ) .");
        } catch (NumberFormatException e) {
            System.out.println("Exceção gerada: " + e.getMessage());
            System.out.println("Foi gerada uma exceção de conversão de dados.");
            System.out.println("Os valores devem ser números inteiros.");
        } catch (IllegalArgumentException e) {
            System.out.println("Exceção gerada: " + e.getMessage());
            System.out.println("Foi gerada uma exceção através do código porque o ");
            System.out.println("denominador era igual a 0(zero).");
        } catch (Exception e) {
            System.out.println("Outras exceções geradas poderão ser tratadas aqui.");
        }
    }
}
```

Classe `Excecao2`: foi incluído um teste (if) para o denominador e, caso ele seja 0 (zero), será lançada uma exceção do tipo **`IllegalArgumentException`**, com a mensagem definida no lançamento da exceção. Como toda exceção gera um objeto, é necessária a palavra `new` para criar o objeto do tipo da exceção. Podemos também definir a mensagem para a exceção ao incluirmos a mensagem no construtor da exceção.

```
import java.util.Scanner;

public class Excecao2 {
    public void metodoExcecao() throws ArithmeticException,
        NumberFormatException, IllegalArgumentException {
        Scanner ent = new Scanner(System.in);
        int num, deno, resultado;
        System.out.println("Digite o numerador para a divisão:");
```

```
num = Integer.parseInt(ent.nextLine());
System.out.println("Digite o denominador para a divisão:");
deno = Integer.parseInt(ent.nextLine());
if (deno == 0)
    throw new IllegalArgumentException("Denominador = 0!");
resultado = num / deno;
System.out.println("O resultado da divisão é:" + resultado);
}
}
```

● ● ● Primeiro teste de execução, com entradas dos valores de 120 e 12:

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
12
O resultado da divisão é:10
```

● ● ● Segundo teste de execução, com entradas dos valores de 120 e 0:

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
0
Exceção gerada: Denominador = 0!
Foi gerada uma exceção através do código porque o
denominador era igual a 0(zero).
```

● ● ● Terceiro teste de execução, com entradas dos valores de 120 e alo (texto e não numérico):

```
Digite o numerador para a divisão:
120
Digite o denominador para a divisão:
01a
Exceção gerada:For input string: "01a"
Foi gerada uma exceção de conversão de dados.
Os valores devem ser números inteiros.
```

Notas:

- 1 O método metodoExcecao() passou a poder lançar 3 diferentes exceções;
- 2 A chamada para execução do método metodoExcecao() deverá ser realizada dentro de um bloco try;

3 As exceções foram geradas dentro do método `metodoExcecao()`, e são repassadas para o método `main()` da aplicação;

4 O teste sobre o valor do denominador (`deno`) é feito antes do cálculo e, caso seja 0 (zero), será lançada a exceção. O método não gerará mais uma exceção de `ArithmeticException`. Como a exceção `IllegalArgumentException` será gerada antes, o desvio pela ocorrência de uma exceção não permitirá que o cálculo seja realizado. Dessa forma, o fluxo retornará para o método `main()` e será tratado na cláusula relacionada à exceção gerada: `IllegalArgumentException`.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

O tratamento de exceções se torna mais abrangente ao podermos definir que métodos podem gerar exceções, além de podermos também gerar exceções explicitamente em nossos códigos. Essa possibilidade faz com que nossos sistemas se tornem mais protegidos e robustos contra possíveis exceções.

Não é preciso codificar pequenos trechos de código de acordo com uma exceção específica. É possível preparar trechos maiores, mesmo que com diferentes exceções sendo lançadas, uma vez que podemos utilizar várias cláusulas `catch` em um mesmo `try`.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

I Atividades

1) Concurso: COSEAC - 2015 - UFF - Técnico de Tecnologia da Informação

Em relação ao tratamento de exceções na linguagem Java, estão corretas as seguintes afirmativas, EXCETO:

- ☐ a) Pelo menos um bloco `catch` ou um bloco `finally` deve seguir imediatamente o bloco `try`.
- ☐ b) Se houver múltiplos blocos `catch` correspondentes, quando uma exceção ocorrer, somente o primeiro é executado.
- ☐ c) O tratamento de exceções processa erros síncronos, que ocorrem quando uma instrução é executada.
- ☐ d) Todas as classes de exceção do Java herdam direta ou indiretamente da classe `error`.
- ☐ e) O bloco `finally` é o meio preferido de liberar recursos para impedir vazamentos de recursos.

2) Sobre tratamento de exceções em Java, é correto afirmar que:

- I. Se ocorrer uma exceção no bloco do `try`, a execução é automaticamente desviada para o bloco `catch`.
- II. No `try`, devemos definir a exceção a ser tratada. Quando definimos uma exceção, estamos tratando também todas as suas subclasses.
- III. O `e`, mostrado na linha do `catch` (`catch(Exception e)`), é referência à exceção que ocorreu, mas não é possível acessar informações sobre essa exceção.

Assinale a alternativa correta:

- ☐ a) Somente a afirmativa I é verdadeira.
- ☐ b) Somente a afirmativa II é verdadeira.

- ☐ c) Somente a afirmativa III é verdadeira.
 - ☐ d) Somente as afirmativas I e II são verdadeiras.
 - ☐ e) Somente as afirmativas II e III são verdadeiras.
-

Notas

abruptamente¹

Término causado por algum erro.

Referências

DEITEL, Paul. **Java: como programar** (Biblioteca Virtual). 10a ed. São Paulo: Pearson, 2017.

Próxima aula

- Tratamento de dados homogêneos.

Explore mais

Leia o texto:

- Tratamento de exceções.

Disponível em:

DEITEL, Paul. Java: como programar (Biblioteca Virtual). 10. ed. São Paulo: Pearson, 2017. [Páginas: 201 – 202].