

Artigo

Invista em você! Saiba como a DevMedia pode ajudar sua carreira.



Trabalhando com Exceções em Java

Veja este artigo conceitos e práticas das exceções, saiba como funciona o processo de tratamento, captura e personalização. Saiba mais sobre as hierarquias e algumas práticas com métodos da classe Throwable.



Anotar



Marcar como concluído

Introdução

Podemos destacar que a exceção é um evento não esperado que ocorre no sistema quando está em tempo de execução (Runtime). Geralmente quando o sistema captura alguma exceção o fluxo do código fica interrompido.

Para conseguir capturar uma **exceção**, é preciso fazer antes o tratamento. O uso dos tratamentos é importante nos sistemas porque auxilia em falhas como: comunicação, leitura e escrita de arquivos, entrada de dados inválidos, acesso a elementos fora de índice, entre outros.

Classificação

O uso das exceções em um sistema é de extrema importância, pois ajuda a detectar e tratar possíveis erros que possam acontecer. Entretanto, na linguagem Java existem dois tipos de exceções, que são:

- **Implícitas:** Exceções que não precisam de tratamento e demonstram serem contornáveis. Esse tipo origina-se da subclasse `Error` ou `RuntimeException`.
- **Explícitas:** Exceções que precisam ser tratadas e que apresentam condições incontornáveis. Esse tipo origina do modelo `throw` e necessita ser declarado pelos métodos. É originado da subclasse `Exception` ou `IOException`.

Existe também a formação de erros dos tipos `throwables` que são:

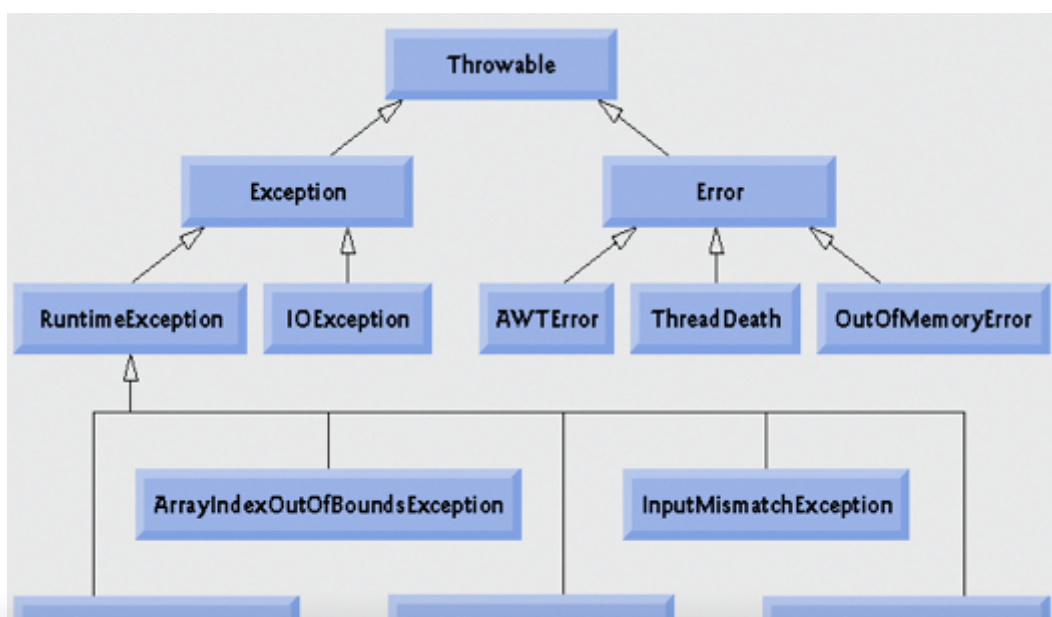
- **Checked Exception:** Erros que acontecem fora do controle do programa, mas que devem ser tratados pelo desenvolvedor para o programa funcionar.

Classe	Objetivo	Eventos
Error	Classificada como exceção que não pode ser tratada pela aplicação.	
Exception	Trata todas as exceções da aplicação que podem ser tratadas e capturadas.	I/O Exception ou aplicar uma divisão por zero resultando na exceção ArithmeticException
RuntimeException	Resgata as exceções lançadas pela máquina virtual (JVM).	Referência nula (NullPointerException)

Figura 1: Exceções das classes.

Hierarquia de exceções

No Java, todas as classes de exceção herdam direta ou indiretamente da classe **Exception**, formando uma hierarquia demonstrada na figura 2.



`Throwable` , onde mostra as situações em que a aplicação pode querer capturar e realizar um tratamento para conseguir realizar o processamento.

- **Error (`java.lang.Error`)** – Também é raiz das classes originárias da classe `Throwable` , indicando as situações em que a aplicação não deve tentar tratar, como ocorrências que não deveriam acontecer.

Existe uma diferença entre “**Erro (Error)**” e “**Exceção (Exception)**”. O “Erro” é algo que não pode mais ser tratado, ao contrário da “Exceção” que trata seus erros, pois todas as subclasses de `Exception` (menos as subclasses `RuntimeException`) são exceções e devem ser tratadas. Os erros da classe `Error` ou `RuntimeException` são erros e não precisam de tratamento, por esse motivo é usado o **try/catch** e/ou propagação com **throw/throws**.

Blocos try/catch/finally

O bloco `try` tenta processar o código que está dentro, sendo que se ocorrer uma exceção, a execução do código pula para a primeira captura do erro no bloco `catch` . O uso do `try` serve para indicar que o código está tentando realizar algo arriscado no sistema.

O bloco `catch` trata a exceção lançada. Caso a exceção não seja esperada, a execução do código pula para o próximo `catch`, se existir. Portanto, se nenhum do bloco `catch` conseguir capturar a exceção, dependendo o tipo que for, é causada a interrupção ao sistema, lançando a exceção do erro. Um exemplo do uso desse bloco é visto em transações de `Rollback` , onde são utilizados para que a

sendo obrigatório sua inserção na sequência **try/catch**. É usado em ações que sempre precisam ser executadas independente se gerar erro. Um exemplo é o fechamento da conexão de um banco de dados.

Praticamente, o uso dos blocos try/catch se dá em métodos que envolvem alguma manipulação de dados, bem como:

- CRUD no banco de dados;
- Índices fora do intervalo de array;
- Cálculos matemáticos;
- I/O de dados;
- Erros de rede;
- Anulação de objetos;
- Entre outros;

Listagem 1: Capturando exceções

```
1  import java.util.InputMismatchException;
2  import java.util.Scanner;
3
4  public class ExemploDivisao {
5
6      public static int calculaQuociente(int numerador, int denomi
7          return numerador / denominador;
8      }
9
10     public static void main(String[] args) {
11         Scanner sc = new Scanner(System.in);
12         boolean continua = true;
```

```
20         int denominador = sc.nextInt();
21
22         int resultado = calculaQuociente(numerador, denominador);
23         System.out.println("Resultado: "+resultado);
24
25         continua = false;
26
27     } catch (InputMismatchException erro1) {
28         System.err.println("Não é permitido inserir letras, insira um número");
29         sc.nextLine(); //descarta a entrada errada do usuário
30     } catch (ArithmeticException erro2){
31         System.err.println("O número do divisor deve ser diferente de zero");
32     } finally{
33         System.out.println("Execução do Finally!");
34     }
35 }while(continua);
36 }
37 }
```

No exemplo da listagem 1, foi utilizado no método a palavra `throws`. Essa é uma questão que gera alguns conflitos, pois além dessa cláusula, existe também o `throw`. Então qual a diferença entre as instruções **throws** e **throw**? Qual momento é necessário usar uma e não a outra?

Cláusulas `throw/throws`

As cláusulas `throw` e `throws` podem ser entendidas como ações que propagam exceções, ou seja, em alguns momentos existem exceções que não podem ser tratadas no mesmo método que gerou a exceção. Nesses casos, é necessário

```
1 public static int calculaQuociente(int numerador, int denominador) {
2     return numerador / denominador;
3 }
```

Na listagem 2 foi usado um pedaço do código da listagem 1, onde mostra o uso do `throws` no método `calculaQuociente`.

Portanto, entende-se que a cláusula `throws` declara as exceções que podem ser lançadas em determinado método, sendo uma vantagem muitas vezes para outros desenvolvedores que mexem no código, pois serve para deixar de modo explícito o erro que pode acontecer no método, para o caso de não haver tratamento no código de maneira correta.

Enquanto isso, a cláusula `throw` cria um novo objeto de exceção que é lançada. A listagem 3 mostra que é criada um exceção `IllegalArgumentException`.

Listagem 3: Usando o `throw`.

```
1 public class Exemplo_Throw {
2
3     public static void saque(double valor) {
4         if(valor > 400) {
5             IllegalArgumentException erro = new IllegalArgumentException(
6                 "Valor retirado da conta: R$" + valor);
7             throw erro;
8         } else {
9             System.out.println("Valor retirado da conta: R$" + valor);
10        }
11    }
12 }
```

Métodos para captura de erros

A classe **Throwable** oferece alguns métodos que podem verificar os erros reproduzidos, quando gerados para dentro das classes. Esse tipo de verificação é visualizado no rastro da pilha (stracktrace), que mostra em qual linha foi gerada a exceção. Abaixo estão descritos os principais métodos que podem ser tratados no bloco catch para visualizar em que momento foi gerado o erro.

- `printStrackTrace` – Imprime uma mensagem da pilha de erro encontrada em um exceção.
- **`getStrackTrace`** – Recupera informações do stracktrace que podem ser impressas através do método `printStrackTrace`.
- `getMessage` – Retorna uma mensagem contendo a lista de erros armazenadas em um exceção no formato String.

Na listagem 4, a variável idade, como mostrado, espera um tipo inteiro. Para simular o erro, basta rodar esse código inserindo uma letra/palavra qualquer. Mais detalhes da geração do erro estão ilustrados na figura 3.

Listagem 4: Usando printStrackTrace

```
1 | import java.util.InputMismatchException;  
2 | import java.util.Scanner;  
3 |
```



```
11
12     System.out.println(idade);
13 } catch (InputMismatchException e) {
14     e.printStackTrace();
15 }
16 }
17 }
```

```
Digite a idade: teste
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 excecoes.Exemplo_PrintStrackTrace.main(Exemplo\_PrintStrackTrace.java:12)
```

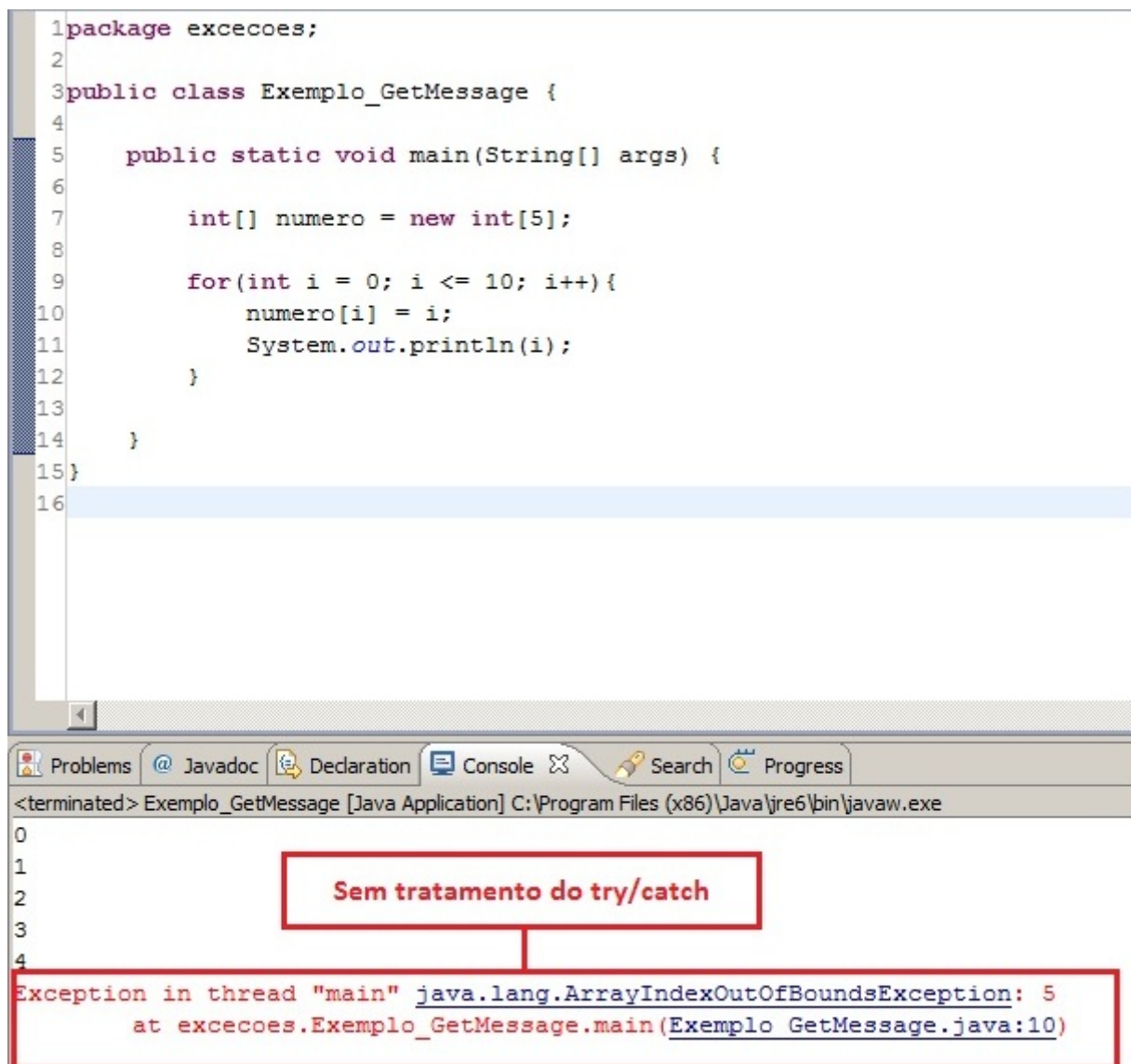
Figura 3: Visualização do erro.

Na listagem 5, se tenta acessar os elementos fora do índice, o que ocasiona o erro.

Listagem 5: Usando getMessage

```
1 public class Exemplo_GetMessage {
2
3     public static void main(String[] args) {
4         try {
5             int[] numero = new int[5];
6
7             for(int i = 0; i <= 10; i++){
8                 numero[i] = i;
9                 System.out.println(i);
10            }
```

A figura 4 demonstra em maiores detalhes um tratamento com o método `getMessage`.



The screenshot shows an IDE window with a Java file named `Exemplo_GetMessage.java`. The code is as follows:

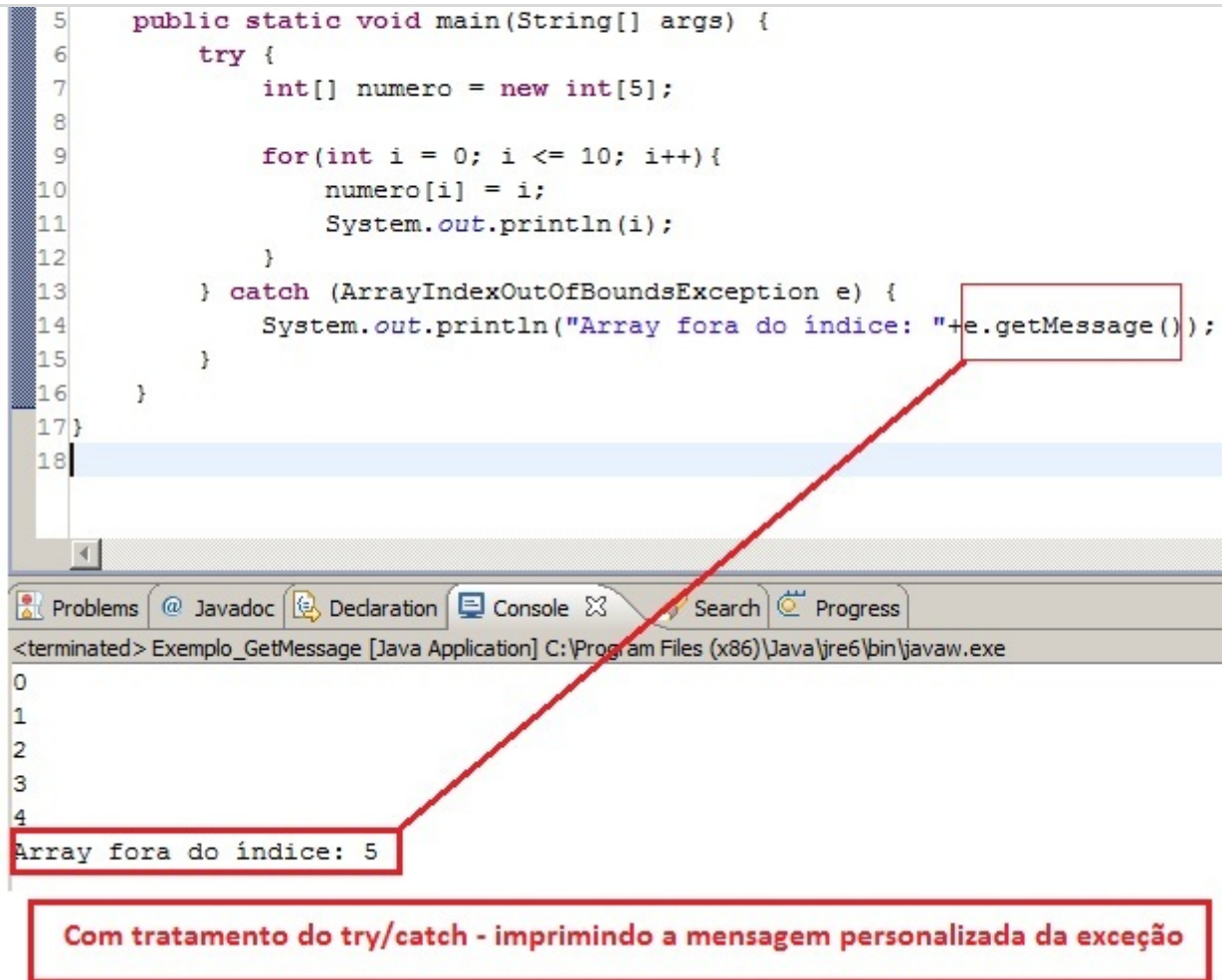
```
1 package excecoes;
2
3 public class Exemplo_GetMessage {
4
5     public static void main(String[] args) {
6
7         int[] numero = new int[5];
8
9         for(int i = 0; i <= 10; i++){
10             numero[i] = i;
11             System.out.println(i);
12         }
13
14     }
15 }
16
```

The IDE's console window at the bottom shows the following error message:

```
<terminated> Exemplo_GetMessage [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe
0
1
2
3
4
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
    at excecoes.Exemplo_GetMessage.main(Exemplo_GetMessage.java:10)
```

Two red boxes highlight the error details. The top box, labeled "Sem tratamento do try/catch", points to the `main` method. The bottom box highlights the full exception message, including the stack trace pointing to line 10 of the file.

Figura 4: Visualização do código sem tratamento de exceção personalizada.



```
5 public static void main(String[] args) {
6     try {
7         int[] numero = new int[5];
8
9         for(int i = 0; i <= 10; i++){
10             numero[i] = i;
11             System.out.println(i);
12         }
13     } catch (ArrayIndexOutOfBoundsException e) {
14         System.out.println("Array fora do índice: "+e.getMessage());
15     }
16 }
17
18
```

Problems Javadoc Declaration Console Search Progress

<terminated> Exemplo_GetMessage [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe

0
1
2
3
4
Array fora do índice: 5

Com tratamento do try/catch - imprimindo a mensagem personalizada da exceção

Figura 5: Visualização do código com tratamento de exceção personalizada.

Conclusão

O objeto `System.err` é um fluxo de erro padrão, sendo utilizado para exibir erros de um programa. A saída de mensagens tem como destaque a cor vermelha na visualização do console. Quando utilizado esse objeto, pode ser enviado para um arquivo de log, enquanto um fluxo de saída padrão `System.out` envia para exibir na tela.