

[< VOLTAR](#)

Gerando e Interceptando Exceções

Erros durante a execução de um programa, sejam eles causados por má operação do usuário, por codificação incorreta, por eventos não previstos, entre diversos outros fatores, podem ser tratados durante a execução do programa, evitando-se assim que o programa encerre inesperadamente e o usuário perca tudo que foi feito. Veremos neste tópico como tratar exceções em Java, ou seja, o famoso try..catch.

NESTE TÓPICO

- > Antes de programar, o que são exceções em Java?
- > Exceções de pré-compilação e pós-compilação
- > O try..catch
- > Mas quais são as possíveis exceções em Java
- > Criando suas próprias exceções



Antes de programar, o que são exceções em Java?

Quando um programa está sendo executado, espera-se que ele execute todas as instruções de forma coerente e linear, certo? Bem, infelizmente nem sempre isso ocorre, pois, milhares de fatores podem influenciar o fluxo de execução de uma aplicação. Os erros mais comuns são:

- **Erro de implementação, ou de lógica**, quando o código possui algum erro casado pelo programador
- **Erro de operação do usuário**, quando o operador do programa o opera de forma incorreta, por exemplo, inserindo um texto num local onde somente valores inteiros são permitidos, inserindo datas em formato inválido, abrindo alguma janela antes dos dados estarem prontos para serem exibidos etc.
- **Erro no acesso a recursos de memória**, quando, por exemplo tenta-se acessar um array além de sua capacidade ou um item que não existe dentro deste vetor

Em Java, quando um erro ocorre dizemos que uma exceção foi “levantada” e, algumas exceções podem acarretar na parada total do programa.

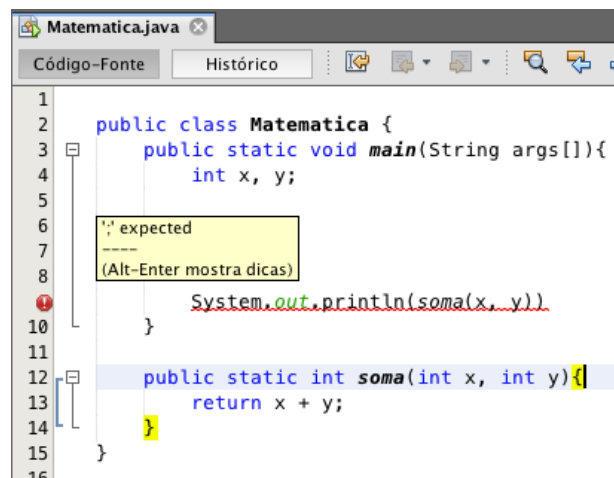
Mas antes de aprendermos a implementar mecanismos de tratamento de exceções, precisamos aprender quais são os principais erros que podemos tratar durante a programação, por isso, alguns exemplos de **códigos incorretos** serão exibidos abaixo, com suas devidas explicações.

Exceções de pré-compilação e pós-compilação

Uma das grandes vantagens de utilizarmos uma IDE poderosa, como o NetBeans, é que ele mostra para o programador erros de sintaxe, atribuições incorretas etc., antes mesmo do código ser executado. Isso ocorre, pois o NetBeans pré-compila seu código enquanto você o digita, ou seja, antes de seu programa ser compilado inteiramente, o NetBeans possui um mecanismo de pré-compilação.

Isso quer dizer que, se seu código não estiver compilando, erros irão aparecer diretamente nele, em formatos de alertas e erros. A IDE coloca avisos em vermelho sob o código defeituoso.

Veja abaixo, por exemplo, a imagem de um erro bastante comum: Quando esquecemos um “;” (ponto e vírgula), como o NetBeans mostra a mensagem para você:



Erro de pré-compilação mostrado na IDE Netbeans

Para ver a mensagem de erro, é preciso deixar o ponteiro sobre a linha destacada ou clicar no ícone de atenção que sobrepõe o número da linha. No exemplo acima, o erro informado pela IDE foi “; expected”, ou seja, a IDE esperada um “;” em algum lugar nessa linha e não encontrou. Note a exclamação na linha 9 do código.

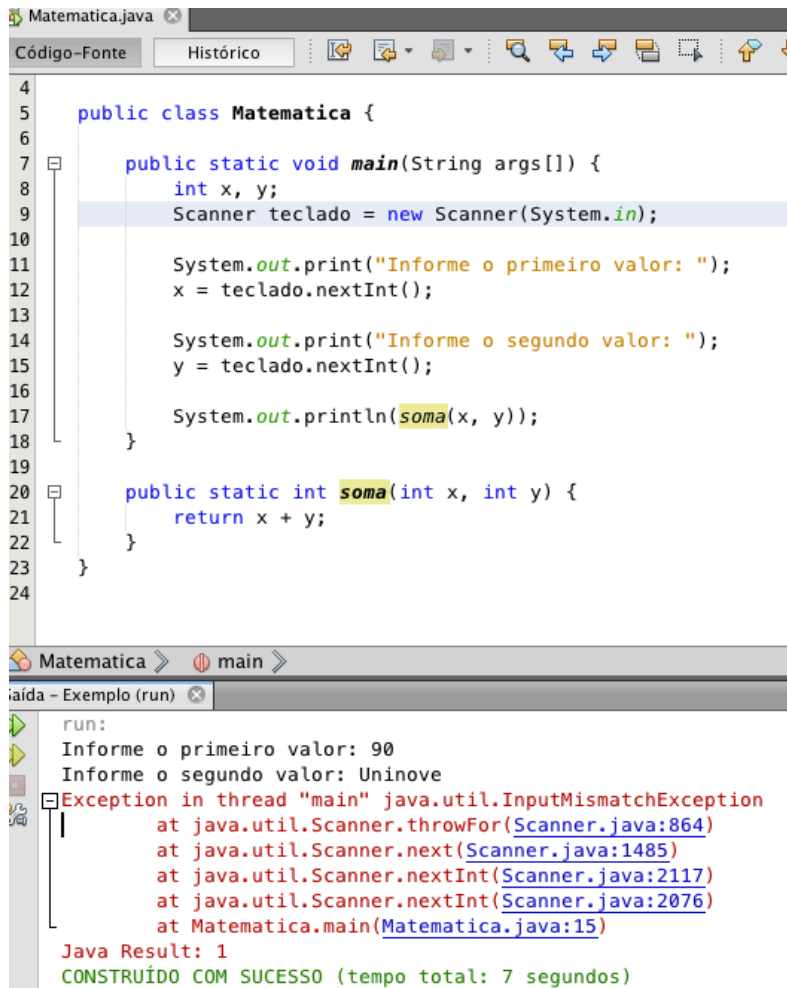
As vezes a própria IDE propõe uma solução para o erro de pré-compilação. Isso ocorre quando ao invés de uma interrogação de alerta, ela mostra um ícone de uma lâmpada com uma pequena interrogação, como visto abaixo. No exemplo abaixo o erro foi a falta de importação da classe *Scanner* para a classe local:

Erro de pré-compilação mostrado na IDE Netbeans e com sugestão de correção

Neste caso, se você clicar no ícone sobre a linha 6, diversas opções serão mostradas para sanar o erro automaticamente como, por exemplo, adicionar automaticamente a importação ao código.

Já erros de pós-compilação são aqueles que ocorrem com o programa em execução, ou seja, erros que não foram possíveis de serem previstos no código. Neste caso as mensagens de erro serão exibidas no console de execução do programa. Veja, abaixo, um exemplo de erro de tipo incompatível que ocorreu no console de execução:





The screenshot shows the NetBeans IDE with a Java file named 'Matematica.java'. The code defines a class 'Matematica' with a 'main' method and a static 'soma' method. The 'main' method prompts the user for two integers, reads them using 'Scanner', and prints their sum. The 'soma' method returns the sum of two integers. Below the code editor, the 'Run' console shows the output of the program. It displays the prompts and the user input '90' and 'Uninove'. However, an 'Exception in thread "main" java.util.InputMismatchException' is thrown at line 15 of 'Matematica.java', where 'nextInt()' is called. The console also shows 'Java Result: 1' and 'CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)'.

```

4
5 public class Matematica {
6
7     public static void main(String args[]) {
8         int x, y;
9         Scanner teclado = new Scanner(System.in);
10
11         System.out.print("Informe o primeiro valor: ");
12         x = teclado.nextInt();
13
14         System.out.print("Informe o segundo valor: ");
15         y = teclado.nextInt();
16
17         System.out.println(soma(x, y));
18     }
19
20     public static int soma(int x, int y) {
21         return x + y;
22     }
23 }
24
Matematica > main >
Matematica - Exemplo (run)
run:
Informe o primeiro valor: 90
Informe o segundo valor: Uninove
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at Matematica.main(Matematica.java:15)
Java Result: 1
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)

```

Erro de pós-compilação sendo exibido no Console do NetBeans



Note que o erro é mostrado em vermelho e há dicas de onde ele ocorreu, neste caso, na linha 15 da classe “Matemática”.

O try..catch

O try..catch serve para desviar o fluxo de execução em caso de erros de pós-compilação, ou seja, erros que ocorrem com o programa em execução.

A sintaxe de try..catch é:

```

1.  //...
2.  try {
3.      <bloco de código que será tratado>
4.  } catch ( <exceção 1 que será tratada> ) [
5.      <bloco de código caso a exceção 1 ocorra>
6.  ] catch ( <exceção 2 que será tratada> ) [
7.      <bloco de código caso a exceção 2 ocorra>
8.  ] catch ( <exceção N que será tratada> ) [
9.      <bloco de código caso a exceção N ocorra>
10. } finally { //Este bloco é opcional
11.     <bloco de código que será executado SEMPRE (se houver ou não erro), depois dos
        blocos acima>
12. }
13. //...

```

Note que na sintaxe exposta acima existe, também, o bloco “*finally*”. Ele serve para conter um bloco de código que será executado sempre ao final do tratamento das exceções, se elas ocorreram ou não. Este bloco é opcional e normalmente é usado quando estamos executando ações em bancos de dados, onde é preciso abrir e fechar a conexão. O fechamento da conexão pode ocorrer no bloco *finally*, pois ela deve ocorrer se houve ou não dispar de algum erro.

Para ficar mais claro, vamos implementar um programa que faça a divisão de dois números informados pelo usuário. Não se esqueça que se damos a liberdade para o usuário digitar, então temos que cuidar para que ele digite apenas números, certo? Nosso tratamento de exceção entrará aí. Lembre-se, também, que não existe divisão por 0 na matemática, ou seja, se o dividendo for 0, o programa deve informar ao usuário que isso não pode acontecer, ou teremos um problema de erro catastrófico no programa.

Então teremos o tratamento de duas exceções: Para o tipo de entrada e para a divisão por zero. O código, então, ficará assim:

```
1. import java.util.InputMismatchException; //Algumas exceções precisam ter a classe de
   tratamento importada
2. import java.util.Scanner;
3.
4. public class MatematicaComTry {
5.
6.     public static void main(String args[]) {
7.
8.         //Declaração das variáveis locais
9.         int a, b, resultado;
10.        Scanner teclado = new Scanner(System.in);
11.
12.        try { //Bloco que será tratado
13.
14.            //Leitura das variáveis
15.            System.out.print("Informe o 1º valor: ");
16.            a = teclado.nextInt();
17.
18.            System.out.print("Informe o 2º valor: ");
19.            b = teclado.nextInt();
20.
21.            //Calcula a divisão e imprime:
22.            resultado = a / b;
23.            System.out.println("O resultado da divisão é: " + resultado);
24.
25.        } catch (ArithmeticException AE){ //Exceção de divisão por 0
26.            System.out.println("Divisão por zero não existe!");
27.        } catch (InputMismatchException IME){ //Exceção de tipo informado é diferente
   e de inteiro
28.            System.out.println("O valor informado não é inteiro!");
29.        }
30.
31.        System.out.println("\n\n\tO programa pode continuar...!");
32.    }
33. }
```



E veja a execução deste código nos dois casos: Se o usuário tenta realizar uma divisão por 0 (zero) ou se ele entra com um texto no lugar do inteiro:

```
run:
Informe o 1º valor: 10
Informe o 2º valor: 0
Divisão por zero não existe!
```

```

O programa pode continuar...!
CONSTRUÍDO COM SUCESSO (tempo total: 4 segundos)
|
```

Resultado 1 da execução do código acima, com erro tratado

```
run:
Informe o 1º valor: 10
Informe o 2º valor: Vinte
O valor informado não é inteiro!
```

```

O programa pode continuar...!
CONSTRUÍDO COM SUCESSO (tempo total: 5 segundos)
```

Resultado 2 da execução do código acima, com erro tratado

Note que para cada bloco de exceções tratadas no exemplo acima, há uma variável que representa a exceção. Essa variável pode ser acessada a qualquer momento dentro do bloco de tratamento e contém diversas informações que podem auxiliar o desenvolvedor a corrigir o problema.

Por exemplo, se mesmo assim você quiser imprimir o erro no console para ver exatamente qual linha ele ocorreu, você pode acrescentar o comando `IME.printStackTrace()`; que a pilha de erro será impressa no console (no caso da variável `IME` quando estamos tratando o erro do tipo incompatível no exemplo acima).



Alguns métodos bastante úteis dessas variáveis são:

- **`printStackTrace()`**: Imprime no console a pilha completa do erro, incluindo o número da linha envolvida;
- **`getStackTrace()`**: Retorna para qualquer variável a pilha de exceção
- **`getMessage()`**: Retorna para qualquer variável a mensagem de erro envolvida na exceção.

Mas quais são as possíveis exceções em Java

Bem, para programar desvios com exceções, você precisa conhecer as principais exceções em Java a que erro cada uma delas está ligada. Contudo, há milhares de exceções possíveis que o Java pode disparar. Isso quer dizer que aqui serão apresentadas as principais e mais comuns:

- **Exception**: Essa é a exceção genérica que você pode usar para tratar absolutamente qualquer problema

- **ArrayIndexOutOfBoundsException:** Tentativa de acesso à posição inexistente de um array
- **ClassCastException:** Tentativa de efetuar um cast (conversão) em uma referência que não é classe ou subclasse do tipo desejado
- **IllegalArgumentException:** Argumento formatado de forma diferente do esperado pelo método
- **IllegalStateException:** O estado do ambiente não permite a execução da operação desejada
- **NullPointerException:** Acesso a objeto que é referenciado por uma variável cujo valor é nulo, normalmente a variáveis ou objetos não inicializados
- **NumberFormatException:** Tentativa de converter uma String inválida em número
- **ArrayIndexOutOfBoundsException:** Tentativa de acesso à posição inexistente no array
- **StackOverflowError:** Normalmente acontece quando existe uma chamada recursiva muito profunda
- **ArithmeticException:** Ocasionalada quando há divisão por 0 (conforme exemplo acima) ou qualquer outro problema de ordem aritmética



Criando suas próprias exceções

Isso mesmo, você leu certo: Você pode criar suas próprias mensagens de erro no Java. Essa é uma das maravilhas do mundo Java orientado a objetos. Para isso, você deve usar o *throw*.

Para exemplificar, imagine que você deve desenvolver uma calculadora que trabalhe apenas com números positivos, ou seja, não é possível subtrair dois valores sendo o segundo maior (como um saque numa conta bancária, você não pode sacar mais do que tem na conta). O programa, então, pode ficar assim:

```
1. import java.util.Scanner;
2.
3. public class MatematicaDeInteiros {
4.
5.     public static void main(String args[]) {
6.
7.         float x, y, resultado;
8.         Scanner teclado = new Scanner(System.in);
9.
10.        System.out.println("Calculadora de valores positivos");
11.
12.        System.out.print("\t Valor 1: ");
13.        x = teclado.nextFloat();
14.
15.        System.out.print("\t Valor 2: ");
16.        y = teclado.nextFloat();
17.
18.        try {
19.            if (y > x) { //Se o valor a ser subtraído for menor, sobre a exceção
20.                throw new IllegalArgumentException("Y não pode ser subtraído de X, p
ois é maior que X");
21.            } else {
22.                resultado = x - y;
23.                System.out.println("Resultado: " + resultado);
24.            }
25.        } catch (IllegalArgumentException e) {
26.            System.out.println(e.getMessage()); //Imprime o erro
27.        }
28.    }
29. }
```

E o resultado da execução deste código pode ser visto abaixo:

```
run:
Calculadora de valores positivos
Valor 1: 150
Valor 2: 190
Y não pode ser subtraído de X, pois é maior que X
CONSTRUÍDO COM SUCESSO (tempo total: 9 segundos)
```



Execução do código com throw

Quiz

Exercício Final

Gerando e Interceptando Exceções

INICIAR ➤

Referências

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman



Avalie este tópico



 ANTERIOR


Arrays e Coleções de Dados

 Índice

[Biblioteca](#)
(<https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/>)
Portal Uninove
(<http://www.uninove.br>)
Mapa do Site

Ajuda?
PRÓXIMO
(<https://ava.uninove.br/cursos/>)

Herança e Polimorfismo



© Todos os direitos reservados

