



# Exceções



## Exceções na linguagem de programação

Uma exceção Java é um objeto que descreve uma condição excepcional (ou seja, erro) que ocorreu em um trecho de código. Quando surge uma condição excepcional, um objeto que representa essa exceção é criado e lançado no método que causou o erro. Esse método pode optar por manipular a exceção em si ou transmiti-la. De qualquer forma, em algum momento, a exceção é capturada e processada.

Os programas que você escreve podem gerar muitos tipos de possíveis exceções, como quando você faz o seguinte:

- Você emite um comando para ler um arquivo de um disco, mas o arquivo não existe lá.
- Você tenta gravar dados em um disco, mas o disco está cheio ou não formatado.
- Seu programa solicita a entrada do usuário, mas o usuário insere dados inválidos.
- O programa tenta dividir um valor por 0, acessar uma matriz com um subscrito muito grande ou calcular um valor muito grande para o tipo de variável da resposta.

Esses erros são chamados de exceções porque, presumivelmente, não são ocorrências usuais; eles são “excepcionais”. As técnicas orientadas a objetos para gerenciar esses erros compreendem o grupo de métodos conhecido como tratamento de exceção.



O tratamento de exceções funciona transferindo a execução de um programa para um manipulador de exceção apropriado quando ocorre uma exceção. Caso um programa após uma operação de divisão tenha o denominador igual a zero, como por exemplo 20/0, então teremos a condição de exceção como no print abaixo:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at exceptiondemo.DivideExceptionDemo.doDevide(DivideExceptionDemo.java:20)
    at exceptiondemo.DivideExceptionDemo.main(DivideExceptionDemo.java:16)
```

Tratamento de exceções:

Existem duas maneiras de lidar com a exceção: primeiro, pegue a exceção e execute a ação corretiva ou apresente a exceção ao método de chamada e force o método de chamada a tratá-la.

- No programa acima, a execução é inesperada e termina em uma condição de erro no caso do denominador ser zero. Podemos evitar isso manipulando exceções usando um bloco try-catch. Vamos atualizar o programa para manipulação de exceções. Aqui, escreveremos códigos propensos a exceção dentro do bloco try (bloco protegido) e o bloco catch seguirá o bloco try.

```
import java.util.Scanner;
```

```
public class ManipulaDivideException{
```

```
    public static void main(String[]args){
```

```
        Scanner inputDevice=new Scanner(System.in);
```

```
        System.out.print("Digite o primeiro número (numerador): ");
```

```
        int numerador=inputDevice.nextInt();
```

```
        System.out.print("Digite o segundo número (denominador): ");
```

```
        int denominador=inputDevice.nextInt();
```

new

```
ManipulaDivideException().fazerDivisao(numerador,denominador);
```

```
    }
```

```
    public void fazerDivisao(int a,int b){
```

```
        try{
```

```
float resultado= a/b;
```



```
System.out.println("Resultado da divisão de "+ a + "/" + b  
+ " = "+ resultado);
```

```
}catch(ArithmeticException e){
```

```
System.out.println("O Programa de Condição de Exceção  
está terminando");
```

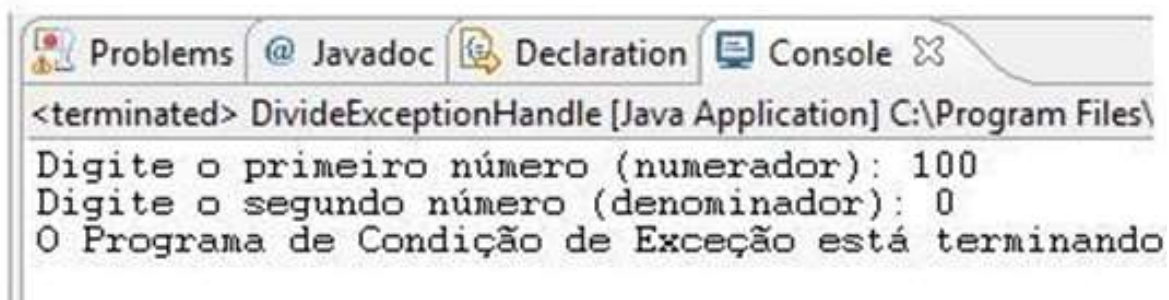
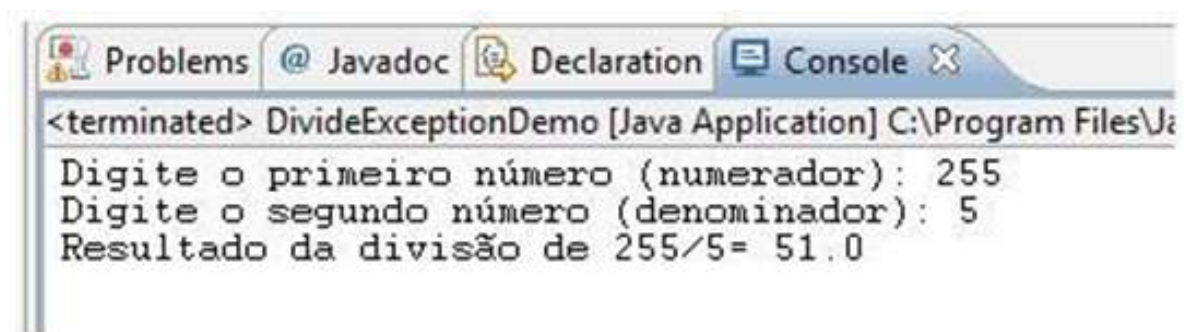
```
}
```

```
}
```

```
}
```

Resultado:

Saídas baseadas na combinação de entrada do usuário:



Quando um método Java lança uma exceção, para indicar que, como parte da assinatura do método, a palavra-chave 'throws' deve ser usada seguida por uma exceção. Isso significa que o responsável pela chamada deste método deve lidar com a exceção dada na cláusula throws. Pode haver várias exceções declaradas como lançadas por um método. Se o chamador desse método não manipular a exceção, ele se propaga para um nível superior na pilha de chamadas do método para o chamador anterior e da mesma forma até atingir a base da pilha de chamadas do método, que será o sistema de tempo de execução do Java. Para essa abordagem, usamos a palavra-chave throws na declaração do método, que instruirá o compilador a lidar com a exceção usando o bloco try-catch. Quando adicionamos a palavra-chave throws na declaração do método de divisão, o erro de tempo de compilação será visto como abaixo,

```
import java.util.Scanner;
```

```
public class DivideExceptionThrows{
```

```
    public static void main(String[] args){
```

```
        Scanner inputDevice=new Scanner(System.in);
```

```
        System.out.print("Digite o primeiro número (numerador):  
(numerador): ");
```

```
        int numerador=inputDevice.nextInt();
```

```
        System.out.print("Digite o segundo número (numerador):  
(denominador): ");
```

```
        int denominador=inputDevice.nextInt();
```

```
        try{
```

new

DivideExceptionThrows().fazerDivisao(numerador,denominador);



}catch(Exception e){

System.out.println("O Programa de Condição de Exceção  
está terminando “);

}

}

public void fazerDivisao(int a,int b)throwsException{

float resultado= a/b;

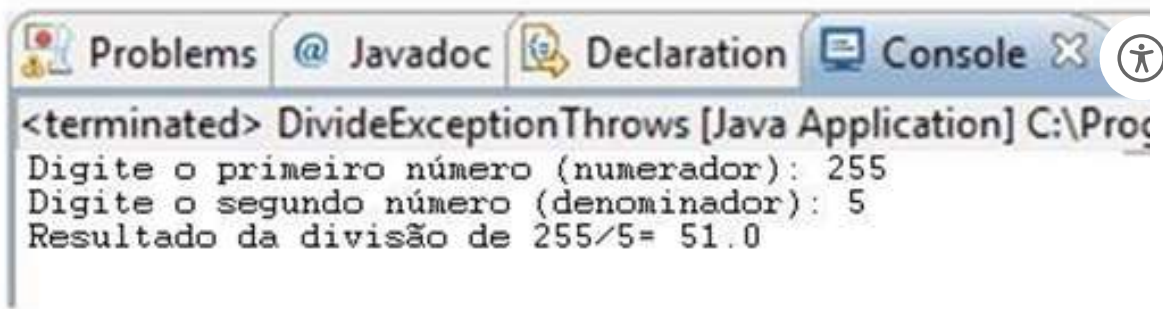
System.out.println(“Resultado da divisão de “+ a +”/”+b  
+”= “+resultado);

}

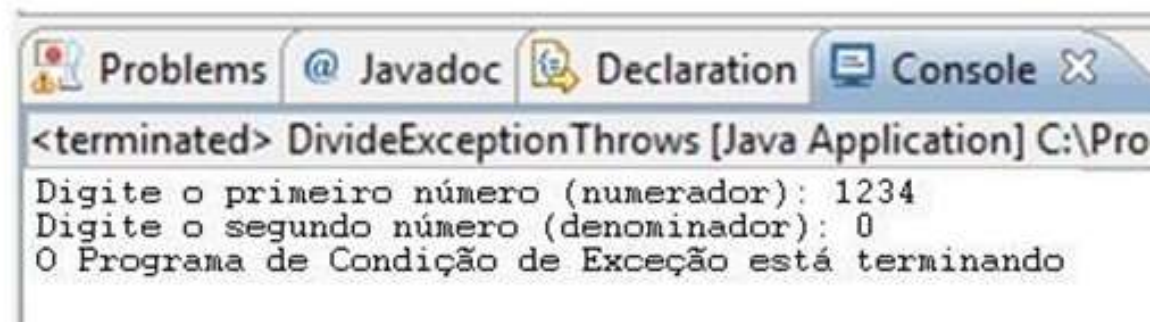
}

Como você pode ver, podemos cercar o código com o bloco try-catch ou podemos jogá-lo novamente para ser tratado chamando o método. Nesse caso, estamos chamando um método do método main (), portanto, se lançarmos novamente a exceção, ela será tratada pela JVM. Vamos atualizar o código e ver a saída com base na combinação de entradas

Saídas baseadas na combinação de entrada do usuário:



```
<terminated> DivideExceptionThrows [Java Application] C:\Pro  
Digite o primeiro número (numerador): 255  
Digite o segundo número (denominador): 5  
Resultado da divisão de 255/5= 51.0
```



```
<terminated> DivideExceptionThrows [Java Application] C:\Pro  
Digite o primeiro número (numerador): 1234  
Digite o segundo número (denominador): 0  
O Programa de Condição de Exceção está terminando
```

Bloco Try-catch aninhado:

A instrução try pode ser aninhada. Ou seja, uma instrução try pode estar dentro do bloco de outra tentativa. Sempre que uma instrução try é inserida, o contexto dessa exceção é pressionado na pilha. Se uma instrução try interna não tiver um manipulador de captura para uma exceção específica, a pilha será desenrolada e os manipuladores de captura da próxima instrução try serão inspecionados para uma correspondência. Isso continua até que uma das instruções catch seja bem-sucedida ou até que todas as instruções try aninhadas sejam esgotadas. Se nenhuma instrução catch corresponder, o sistema de tempo de execução Java tratará a exceção. Abaixo está a sintaxe do bloco try-catch aninhado.

```
public class NestedTryblockDemo{
```

```
    public static void main(String[]args){
```

```
        try{
```

```
            // Algum código que pode lançar Exception
```

```
try{
```



```
// Algum código que pode lançar Exception
```

aritmética

```
try{
```

```
// Algum código que pode lançar
```

Exception de formato numérico

```
}catch(NumberFormatException n){
```

```
// Tratamento de exceção de formato numéricp
```

```
}
```

```
}catch(ArithmeticException a){
```

```
// Tratamento de exceção aritmética
```

```
}
```

```
}catch(Exception e){
```

```
//exceções gerais
```

```
}
```


```
}
```

```
}
```

Uso do bloco finally

Quando você tem ações que deve executar no final de uma sequência de tentativa ... captura, você pode usar um bloco finally. O código dentro de um bloco finally é



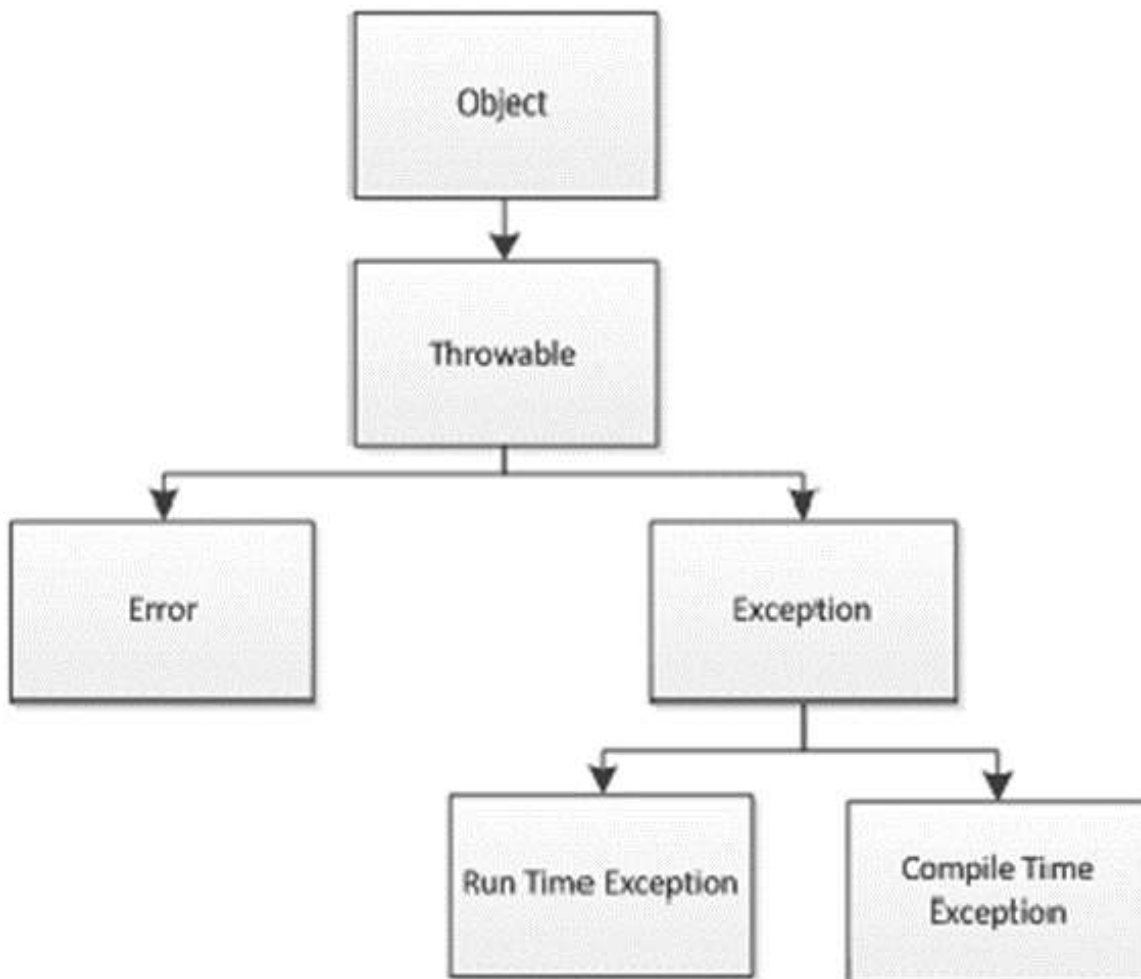
executado independentemente de o bloco de tentativas anterior identificar uma exceção. Geralmente, você usa um bloco finalmente para executar tarefas  limpeza que devem ocorrer, se ocorreram exceções ou não e se foram capturadas quaisquer exceções ocorridas ou não. Em um aplicativo em que a conexão com o banco de dados, os arquivos estão sendo operados, precisamos fechar esses recursos em condições excepcionais e normais.

```
public class TryCatchFinally{  
  
    public void Demo(){  
  
        try{  
  
            // declaração  
  
        }catch(Exception e){  
  
            // ações que ocorrem se uma exceção foi lançada  
  
        }finally{  
  
            // ação que ocorre se o bloco de captura é executado  
  
        }  
  
    }  
  
}
```

## Tipos de exceções

Java é uma linguagem de programação orientada a objetos. A exceção é um objeto criado no momento da condição de exceção / erro que será lançado do

programa e interromperá a execução normal do programa. A hierarquia de objetos de exceções Java é a seguinte:



Todos os tipos de exceção são subclasses da classe interna Throwable. Portanto, Throwable está no topo da hierarquia de classes de exceção. Imediatamente abaixo de Throwable, existem duas subclasses que particiona exceções em duas ramificações distintas. Um ramo é liderado por Exceção. Esse clássico era usado para condições excepcionais que os programas do usuário deveriam capturar. Essa também é a classe que você subclassificará para criar seus próprios tipos de exceção personalizados. Há uma subclasse importante de Exception, chamada RuntimeException. Exceções desse tipo são definidas automaticamente para os programas que você escreve e incluem itens como divisão por zero e indexação de matriz inválida.

A outra ramificação é encimada por Erro, que define exceções que não se espera que sejam capturadas em circunstâncias normais pelo seu programa. Exceções tipo Error são usadas pelo sistema Java Runtime para indicar erros relacionados ao próprio ambiente de tempo de execução. O estouro de pilha é um exemplo desse erro. Este capítulo não tratará de exceções do tipo Erro, porque elas geralmente são criadas em resposta a falhas catastróficas que geralmente não podem ser tratadas pelo seu programa.

As exceções do Java podem ser categorizadas em dois tipos:


- Exceções verificadas
- Exceções não verificadas

Geralmente, as exceções verificadas estão sujeitas à captura ou especificam um requisito, o que significa que exigem captura ou declaração. Este requisito é opcional para exceções não verificadas. O código que usa uma exceção verificada não será compilado se a regra de captura ou especificação não for seguida.

Exceções não verificadas vêm em dois tipos:

- Erros
- Exceções de tempo de execução

## Exceções checked

As exceções checked são do tipo que os programadores devem antecipar e dos quais os programas devem poder se recuperar. Todas as exceções de Java  exceções checked, exceto as das classes Error e RuntimeException e suas subclasses.

Uma exceção checked é uma exceção com a qual o código-fonte Java deve lidar, capturando-o ou declarando-o lançado. As exceções checked geralmente são causadas por falhas fora do próprio código - recursos ausentes, erros de rede e problemas com threads vêm à mente. Isso pode incluir subclasses de FileNotFoundException, UnknownHostException etc.

Exceções verificadas populares:

Nome	Descrição
IOException	Ao usar a exceção relacionada ao fluxo de entrada / saída de arquivo
SQLException.	Ao executar consultas no banco de dados relacionadas à sintaxe SQL
DataAccessExcepti on	Exceção relacionada ao acesso a dados / banco de dados
ClassNotFoundException	Lançada quando a JVM não consegue encontrar uma classe de que precisa, devido a um erro na linha de comandos, um problema no caminho da classe ou um arquivo .class ausente
InstantiationExcepti on	Tente criar um objeto de uma classe ou interface abstrata.

Abaixo um exemplo de programa de leitura, o arquivo mostra como a exceção checked deve ser tratada. A imagem abaixo mostra um erro em tempo de compilação devido à exceção checked (FileNotFoundException e IOException) relacionada à operação do arquivo. O IDE sugere que precisamos incluir nosso código no bloco try-catch ou podemos usar a palavra-chave throws na declaração do método

Atualizaremos a declaração do método com throws keyword e o método de chamada (método principal) precisará lidar com essa exceção. Durante a execução do programa, podemos encontrar dois tipos de problemas (1) O arquivo está ausente ou não está presente, o que estamos tentando ler (2) O usuário não tem permissão de leitura no arquivo ou o arquivo está bloqueado por outro usuário. Como esperamos dois tipos diferentes de exceções, temos que capturar as duas exceções ou podemos ter um bloco catch que está capturando uma exceção de superclasse. O código abaixo mostra a sintaxe de vários blocos de captura.

```
import java.io.FileInputStream;
```

```
import java.io.FileNotFoundException;
```

```
import java.io.IOException;
```

```
public class CheckedExceptionDemo{
```

```
    public static void main(String[] args){
```

```
        // Abaixo, a lista de filmes de leitura e impressão
```

```
        String filename="teste.txt";
```

```
        try{
```

String fileContent = new



```
CheckedExceptionDemo().readFile(filename);
```

```
System.out.println(fileContent);
```

```
}catch(FileNotFoundException e){
```

```
System.out.println("Arquivo:" + filename + " não encontrado,  
por favor cheque o nome do arquivo ");
```

```
}catch(IOException e){
```

```
System.out.println("O arquivo não possui permissão de  
leitura, por favor, verifique a permissão");
```

```
}
```

```
}
```

```
public String readFile(String filename) throws  
FileNotFoundException, IOException {
```

```
FileInputStream fin;
```

```
int i;
```

```
String s = "";
```

```
fin = new FileInputStream(filename);
```

```
// caracteres de leitura até que o EOF seja encontrado
```

```
do {
```

```
i = fin.read();
```

```
if(i != -1) s = s + (char) i + '\n';
```



```
}while(i != -1);
```

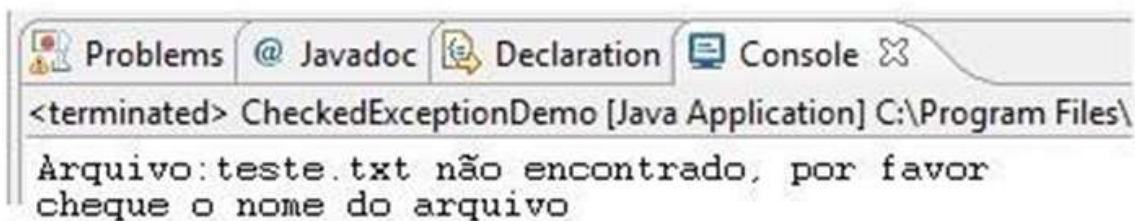
```
fin.close();
```

```
return s;
```

```
}
```

```
}
```

Saída: se test.txt não for encontrado:



## Exceções unchecked

Exceções unchecked herdam da classe `Error` ou da classe `RuntimeException`. Muitos programadores acham que você não deve lidar com essas exceções em seus programas porque representam o tipo de erros dos quais não se pode razoavelmente esperar que os programas se recuperem enquanto o programa estiver em execução.

Quando uma exceção unchecked geralmente é causada pelo uso indevido do código - passando um argumento nulo ou incorreto.

Exceções unchecked mais comuns:



Nome	Descrição
NullPointerException	Lançado ao tentar acessar um objeto com uma variável de referência cujo valor atual é nulo
ArrayIndexOutOfBoundsException	Lançada ao tentar acessar uma matriz com um valor de índice inválido (negativo ou além do comprimento da matriz)
IllegalArgumentException	Lançado quando um método recebe um argumento formatado de forma diferente do que o método espera.
IllegalStateException	Lançado quando o estado do ambiente não corresponde à tentativa de operação, por exemplo, usando um scanner fechado.
NumberFormatException	Lançado quando um método que converte uma String em um número recebe uma String que não pode ser convertida.
ArithmeticException	Erro aritmético, como dividir por zero.

Veremos como exemplo, um programa que terá a idade do usuário como entrada e concederá acesso se a idade for superior a 18 anos. Aqui, a entrada do usuário é esperada em forma numérica, se a entrada do usuário for outro alfabético, nosso programa terminará na condição de exceção (`InputMismatchException`). Essa exceção ocorre no tempo de execução. Podemos decidir lidar com isso programaticamente, mas não é obrigatório. As exceções de tempo de execução são boas para lidar com o uso do bloco `try-catch` e evitam situações de erro.

```
import java.util.Scanner;
```

```
public class RunTimeExceptionDemo{
```



```
public static void main(String[] args){
```



```
    // Lendo a entrada do usuário
```

```
    Scanner inputDevice=new Scanner(System.in);
```

```
    System.out.print("Digite a idade (valor numérico): ");
```

```
    int idade =inputDevice.nextInt();
```

```
    if(idade >18){
```

```
        System.out.println("Autorizado para ver a página.");
```

```
        //Outra lógica de negócios
```

```
    }else{
```

```
        System.out.println("Não autorizado para ver a página.");
```

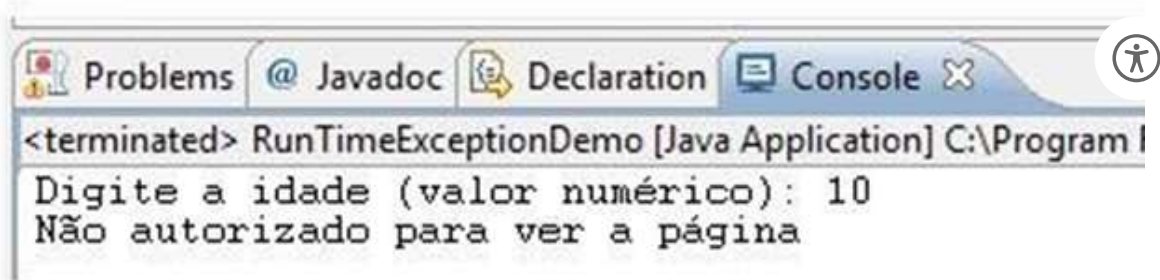
```
        // Outro código relacionado a logout
```

```
    }
```

```
}
```

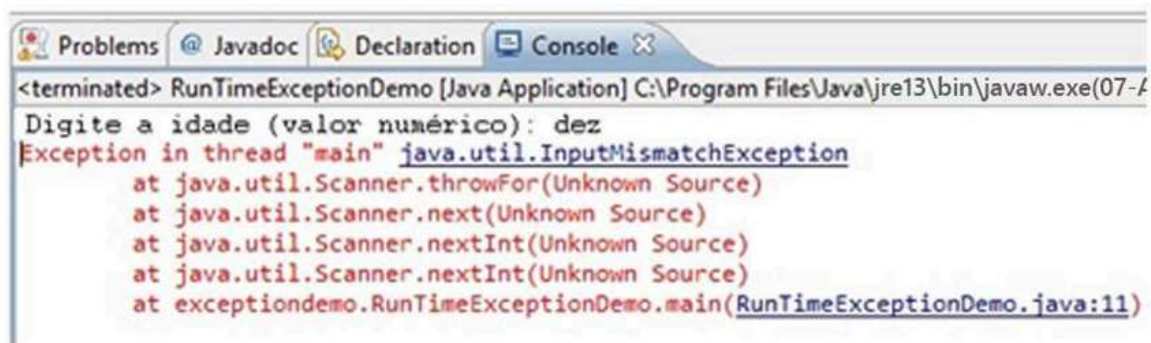
```
}
```

Resultado:



```
<terminated> RunTimeExceptionDemo [Java Application] C:\Program I
Digite a idade (valor numérico): 10
Não autorizado para ver a página
```

Se o usuário digitar um valor não numérico, o programa termina em condição de erro / excepcional.



```
<terminated> RunTimeExceptionDemo [Java Application] C:\Program Files\Java\jre13\bin\javaw.exe(07-4
Digite a idade (valor numérico): dez
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 exceptiondemo.RunTimeExceptionDemo.main(RunTimeExceptionDemo.java:11)
```

## Atividade extra

Vídeo: “Tratamento de exception”

Link: [https://www.youtube.com/watch?v=WND6\\_h3NcSw](https://www.youtube.com/watch?v=WND6_h3NcSw)



BARNES, D. J.; KOLLING, M. **Programação orientada a objetos com java: uma introdução prática usando o bluej**. 4.ed. Pearson: 2009.

FELIX, R. (Org.). **Programação orientada a objetos**. Pearson: 2017.

MEDEIROS, L. F. de. **Banco de dados: princípios e prática**. Intersaberes: 2013;

ORACLE. Java Documentation, 2021. **Documentação oficial da plataforma Java**. Disponível em: < <https://docs.oracle.com/en/java/> >.

**Ir para exercício**