

The slide features a light gray background with several decorative purple elements: a large white circle in the top-left corner, a solid purple circle in the top-left, a purple horizontal bar in the top-right, a small purple circle in the bottom-right, a large purple circle in the bottom-right, and a purple horizontal bar in the bottom-left.

# Programação II: Exceções

---


Prof<sup>a</sup>. Tainá Isabela



# Exceções

As exceções atuam como uma indicação de um problema que ocorreu durante a execução de um programa.


O tratamento de exceções permite criar aplicativos que podem resolver ou tratar exceções em tempo de execução.





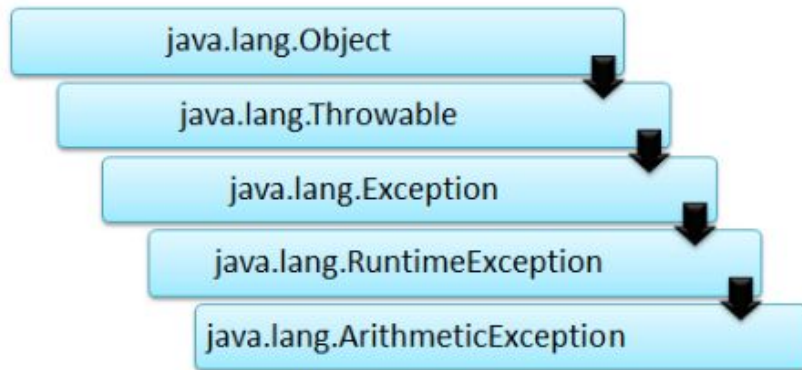
# Classificação

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.
  - **Unchecked (Runtime):** Erros que podem ser evitados se forem tratados e analisados pelo desenvolvedor.
  - **Error:** Usado pela JVM que serve para indicar se existe algum problema de recurso do programa, tornando a execução impossível de continua
- 

# Exceções


Dentro das exceções existe uma hierarquia de classes para tratamento de exceções.





# Classificação

O uso das exceções em um sistema é de extrema importância, pois ajuda a detectar e tratar possíveis erros que possam acontecer.

- **Implícitas:** Exceções que não precisam de tratamento e demonstram serem contornáveis.
  - **Explícitas:** Exceções que precisam ser tratadas e que apresentam condições incontornáveis. Esse tipo origina do modelo throw e necessita ser declarado pelos métodos.
- 

```
public class DivideByZeroNoExceptionHandling
{
    // demonstra o lançamento de uma exceção quando ocorre uma divisão por zero
    public static int quotient(int numerator, int denominator)
    {
        return numerator / denominator; // possível divisão por zero
    }

    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Please enter an integer numerator: ");
        int numerator = scanner.nextInt();
        System.out.print("Please enter an integer denominator: ");
        int denominator = scanner.nextInt();

        int result = quotient(numerator, denominator);
        System.out.printf(
            "%nResult: %d / %d = %d%n", numerator, denominator, result);
    }
} // fim da classe DivideByZeroNoExceptionHandling
```

```

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

public class DivideByZeroWithExceptionHandling
{
    // demonstra o lançamento de uma exceção quando ocorre uma divisão por zero
    public static int quotient(int numerator, int denominator)
        throws ArithmeticException
    {
        return numerator / denominator; // possível divisão por zero
    }

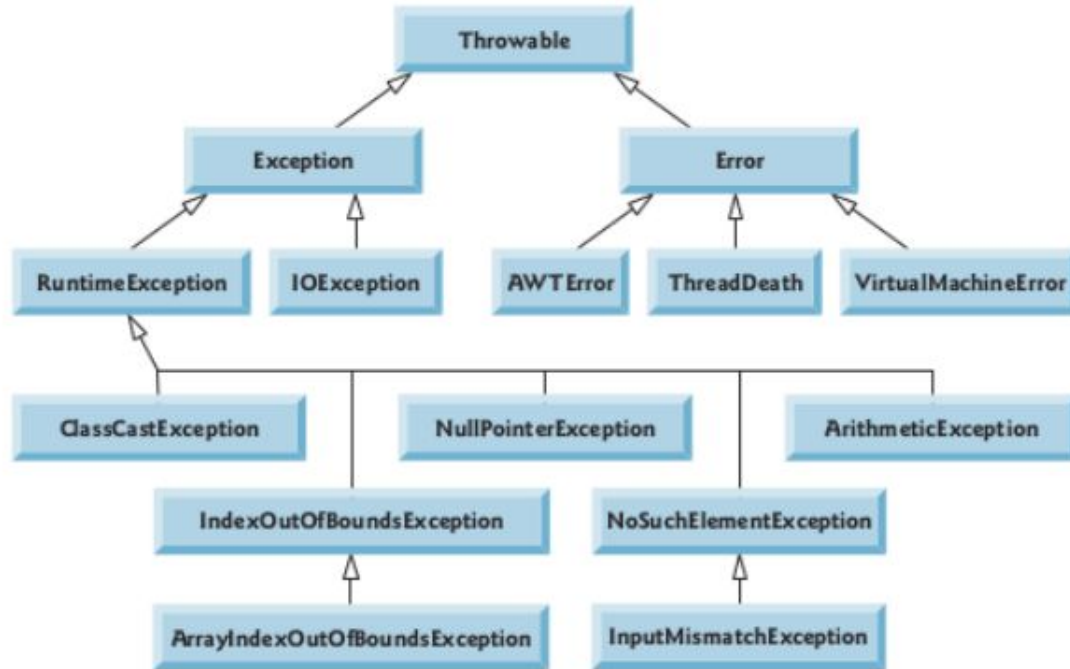
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        boolean continueLoop = true; // determina se mais entradas são necessárias

        do
        {
            try // lê dois números e calcula o quociente
            {
                System.out.print("Please enter an integer numerator: ");
                int numerator = scanner.nextInt();
                System.out.print("Please enter an integer denominator: ");
                int denominator = scanner.nextInt();

                int result = quotient(numerator, denominator);
                System.out.printf("%nResult: %d / %d = %d%n", numerator,
                    denominator, result);
                continueLoop = false; // entrada bem-sucedida; fim do loop
            }
            catch (InputMismatchException inputMismatchException)
            {
                System.err.printf("%nException: %s%n",
                    inputMismatchException);
                scanner.nextLine(); // descarta entrada para o usuário tentar de novo
                System.out.printf(
                    "You must enter integers. Please try again.%n%n");
            }
            catch (ArithmeticException arithmeticException)
            {
                System.err.printf("%nException: %s%n", arithmeticException);
                System.out.printf(
                    "Zero is an invalid denominator. Please try again.%n%n");
            }
        } while (continueLoop);
    }
} // fim da classe DivideByZeroWithExceptionHandling

```

# Hierarquia de Exceções em Java








# Hierarquia de Exceções


Todos os tipos de exceção que são subclasses diretas ou indiretas da classe `RuntimeException` (pacote `java.lang`) geralmente são causadas por defeitos no código do seu programa.

- *`NullPointerException`*
  - *`ArrayIndexOutOfBoundsException`*
  - *`ClassCastException`*
  - *`ArithmeticException`*
- 



# Hierarquia de Exceções

As classes que são herdadas direta ou indiretamente da classe `Error` são problemas tão sérios que seu programa não deve nem mesmo tentar lidar com eles.






# 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 propagar a exceção para um nível acima na pilha.






# Cláusulas throw/throws

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.






# 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.





# Métodos para captura de erros

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

# Bloco Finally

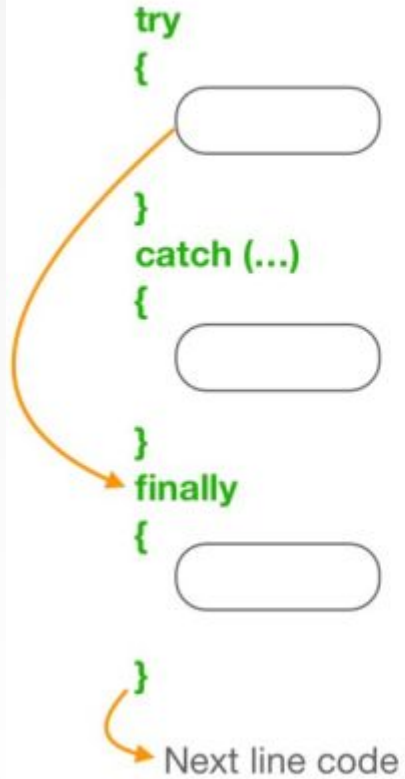
- É uma cláusula opcional;

Sempre será colocado após o último bloco catch.

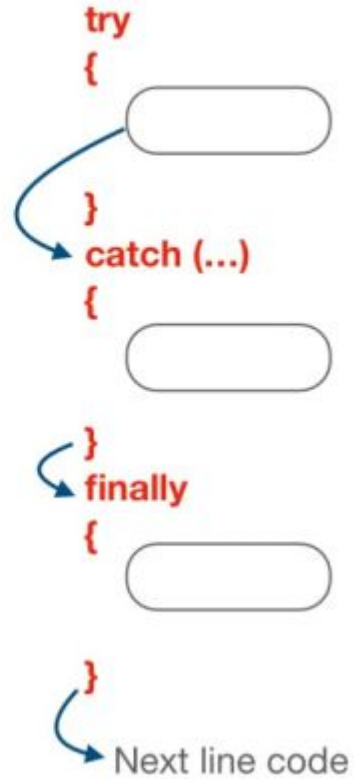
Caso não existam blocos catch, será logo após o bloco try.

O bloco finally será executado independente do lançamento ou não de uma exceção.

## Without Exception



## With Exception





# Momento Questionário



# Atividades

1. Escreva um programa que leia dois números inteiros e divida o primeiro pelo segundo. Use try-catch para tratar `ArithmeticException` (divisão por zero). Exiba uma mensagem amigável caso ocorra erro.
2. Implemente um programa que contenha: Um método `calcularRaiz(int numero)` que lance uma exceção (`IllegalArgumentException`) caso o número seja negativo. A cláusula `throws` na assinatura do método para indicar a possibilidade da exceção. No método `main`, capture a exceção com try-catch e mostre uma mensagem adequada ao usuário.
3. Crie um programa que: Solicite ao usuário um número inteiro e tente converter uma `String` para `int`. Utilize try-catch para tratar `NumberFormatException` (ex.: se o usuário digitar letras). Utilize um bloco `finally` para exibir uma mensagem do tipo: "Encerrando programa...", garantindo que essa linha sempre seja executada, independentemente de erros.