Aula 13

# Tratamento de exceções

Programação III

Prof. Augusto César Oliveira
augusto.oliveira@unicap.br

UNIVERSIDADE CATÓLICA DE PERNAMBUCO

# Na aula passada…

- Compreender a importância das coleções;
- Conhecer os principais tipos de coleções;
- Aprender a criar e inicializar coleções;
- Inserir, recuperar e remover elementos em coleções;
- Ordenar elementos em coleções.

# O objetivo da aula de hoje…

- Compreender os diferentes tipos de erros em programação;
- Compreender o que é uma exceção e como ela é usada para representar erros durante a execução do programa;
- Conhecer a hierarquia de classes de exceções;
- Utilizar as construções `try` e `catch` para capturar e tratar exceções;
- Criar e lançar exceções personalizadas.

# 1.
# Tipos de erros

Tratamento de exceções

# Tipos de erros existentes em programação

1. Erros de sintaxe:
   - Surge quando as **regras da linguagem não são seguidas**;
   - São detectados pelo compilador, **em tempo de compilação**.

2. Erros de lógica:
   - Surgem quando o programador codifica uma lógica algorítmica incorreta, mesmo achando que está correta.
   - Provoca resultados errados e são descobertos ao testar o programa, **em tempo de execução**.

3. Exceções:
   - Ocorrem quando o programa está rodando e o ambiente detecta alguma **operação inválida** somente **durante a execução**.

# Exemplos de erros

1. Divisão por zero;
2. Acessar posições inválidas de um array;
3. Acessar referência nula (null);
4. Atribuir tipos inválidos.

# 2. Exceções

Tratamento de exceções

# Definição de exceção (exception)

- **Exceção é um evento** que ocorre durante a execução de um programa, **interrompendo o fluxo de execução normal das instruções**.

- Sempre que uma exceção é lançada, o fluxo de execução é interrompido, retornando ao método chamador recursivamente até encontrar alguém que trata a exceção.

- Caso o programador não tenha tratado a exceção, a mesma irá provocar o **término abrupto do programa**.

- Exceções são objetos!! (quase tudo em Java é objeto).

# Fluxo de exceção

```
1   public class Main {
2
3      public static void main(String[] args) {
4         metodo1();
5         System.out.println("passei do metodo 1");
6      }
7
8      private static void metodo1() {
9         metodo2();
10        System.out.println("passei do metodo 2");
11     }
12
13     private static void metodo2() {
14        int numero = 20/ 0;
15     }
16  }
```

# Fluxo de exceção

```
public static void main(String[] args) {
    metodo1();
    System.out.println("passei do metodo 1");
}
```

# Fluxo de exceção

```
public static void main(String[] args) {
    metodo1();
    System.out.println("passei do metodo 1");
}
```

```
private static void metodo1() {
    metodo2();
    System.out.println("passei do metodo 2");
}
```

# Fluxo de exceção

```
public static void main(String[] args) {
    metodo1();
    System.out.println("passei do metodo 1");
}
```

```
private static void metodo1() {
    metodo2();
    System.out.println("passei do metodo 2");
}
```

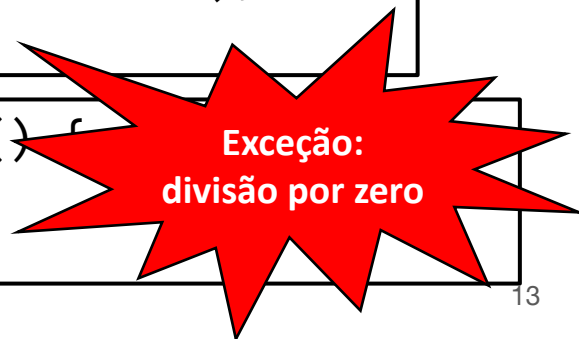
```
private static void metodo2() {
    int numero = 20/ 0;
}
```

# Fluxo de exceção

```
public static void main(String[] args) {
    metodo1();
    System.out.println("passei do metodo 1");
}
```

```
private static void metodo1() {
    metodo2();
    System.out.println("passei do metodo 2");
}
```

```
private static void metodo2() {
    int numero = 20/ 0;
}
```

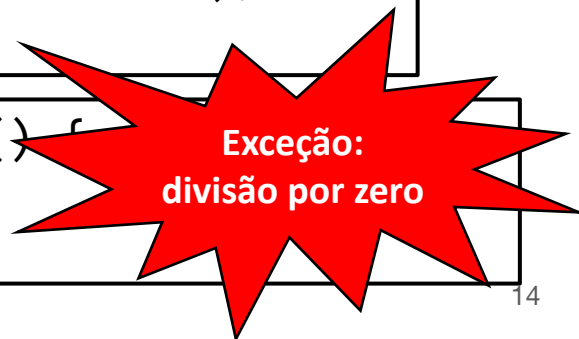


# Fluxo de exceção

```
public static void main(String[] args) {
    metodo1();
    System.out.println("passei do metodo 1");
}
```

```
private static void metodo1() {
    metodo2();
    System.out.println("passei do metodo 2");
}
```

```
private static void metodo2() {
    int numero = 20/ 0;
}
```

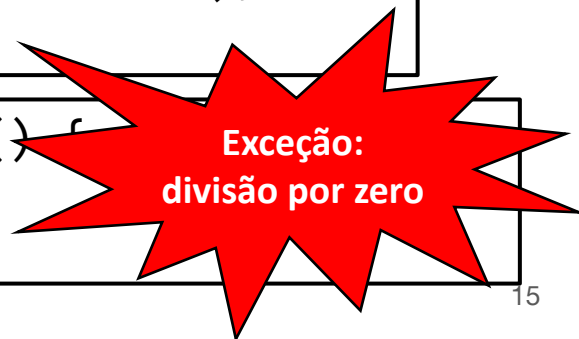


# Fluxo de exceção

```
public static void main(String[] args) {
    metodo1();
    System.out.println("passei do metodo 1");
}
```

```
private static void metodo1() {
    metodo2();
    System.out.println("passei do metodo 2");
}
```

```
private static void metodo2() {
    int numero = 20/ 0;
}
```



# Exception: mensagem de erro

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Main.metodo2(Main.java:14)
at Main.metodo1(Main.java:9)
at Main.main(Main.java:4)
```

```
1   public class Main {
2
3       public static void main(String[] args) {
4           metodo1();
5           System.out.println("passei do metodo 1");
6       }
7
8       private static void metodo1() {
9           metodo2();
10          System.out.println("passei do metodo 2");
11      }
12
13      private static void metodo2() {
14          int numero = 20/ 0;
15      }
16  }
```

# Exception: mensagem de erro

**Tipo da exceção**

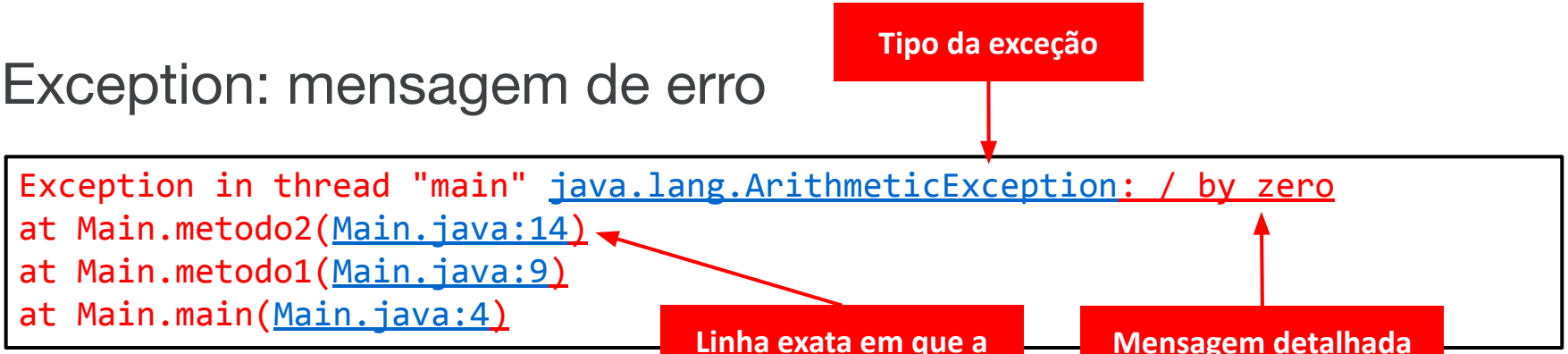
```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Main.metodo2(Main.java:14)
at Main.metodo1(Main.java:9)
at Main.main(Main.java:4)
```

```
1   public class Main {
2
3       public static void main(String[] args) {
4           metodo1();
5           System.out.println("passei do metodo 1");
6       }
7
8       private static void metodo1() {
9           metodo2();
10          System.out.println("passei do metodo 2");
11      }
12
13      private static void metodo2() {
14          int numero = 20/ 0;
15      }
16  }
```

# Exception: mensagem de erro

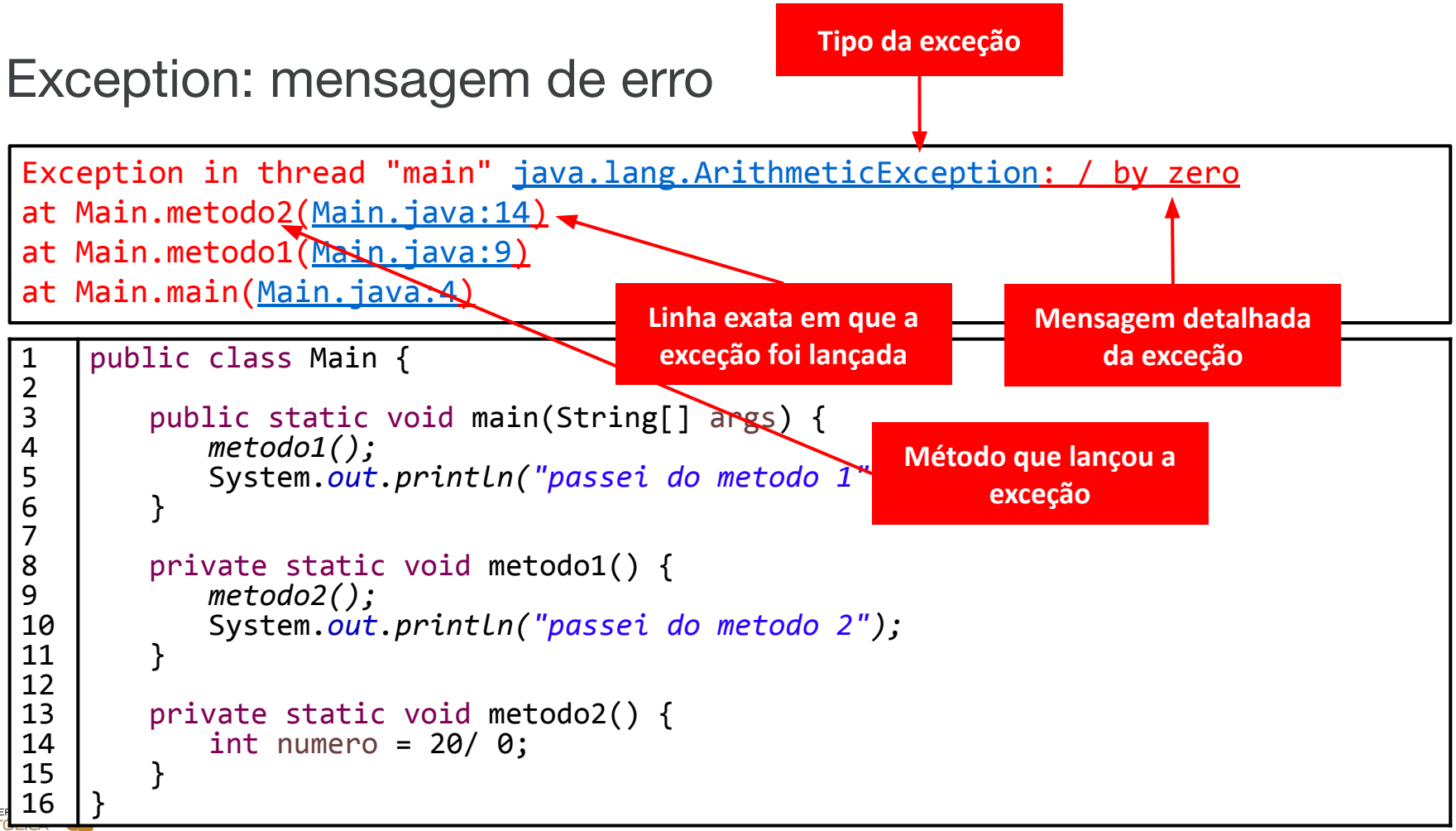

```
1   public class Main {
2
3       public static void main(String[] args) {
4           metodo1();
5           System.out.println("passei do metodo 1");
6       }
7
8       private static void metodo1() {
9           metodo2();
10          System.out.println("passei do metodo 2");
11      }
12
13      private static void metodo2() {
14          int numero = 20/ 0;
15      }
16  }
```

# Exception: mensagem de erro

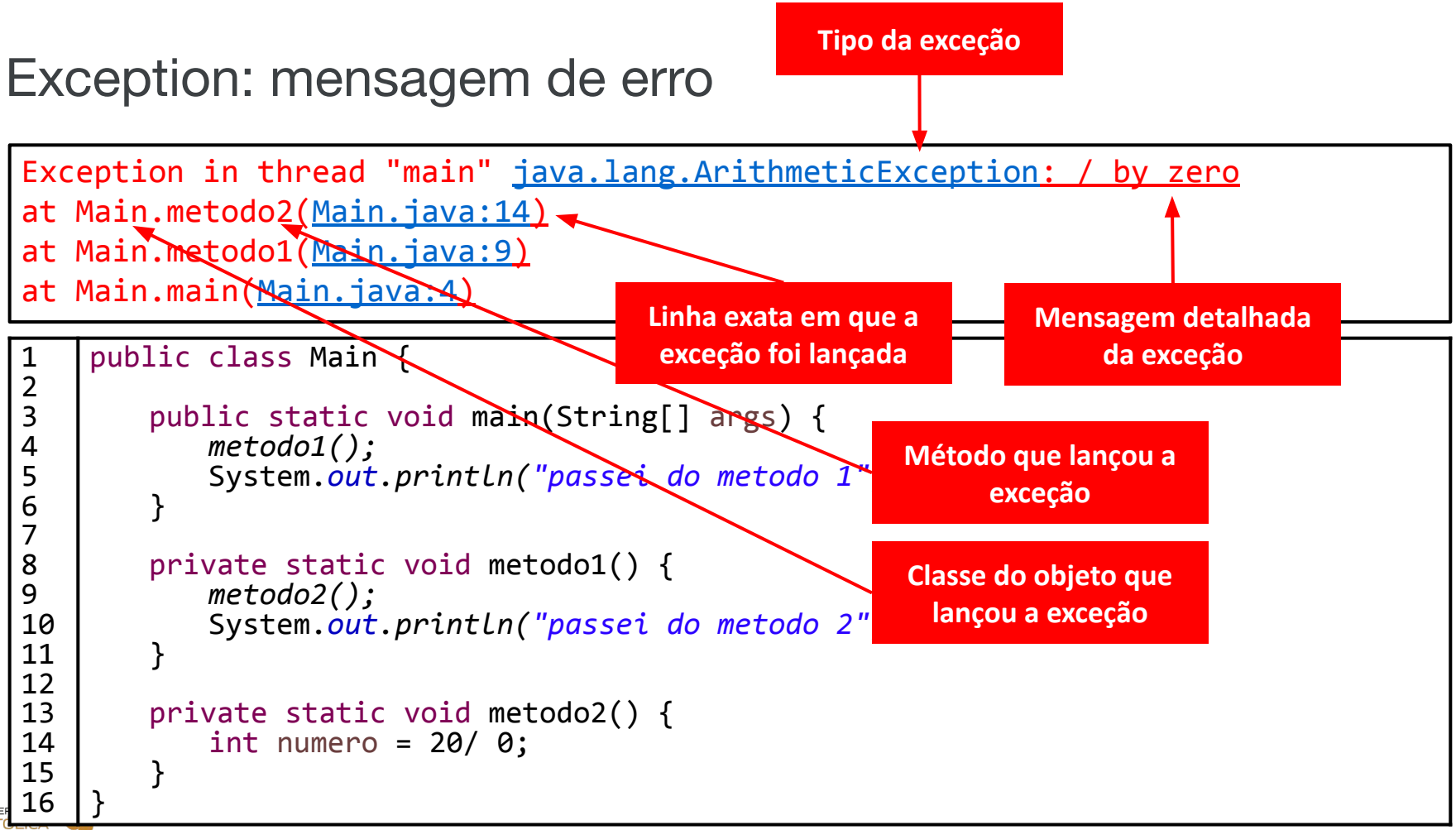



```
1   public class Main {
2
3       public static void main(String[] args) {
4           metodo1();
5           System.out.println("passei do metodo 1");
6       }
7
8       private static void metodo1() {
9           metodo2();
10          System.out.println("passei do metodo 2");
11      }
12
13      private static void metodo2() {
14          int numero = 20/ 0;
15      }
16  }
```

# Exception: mensagem de erro

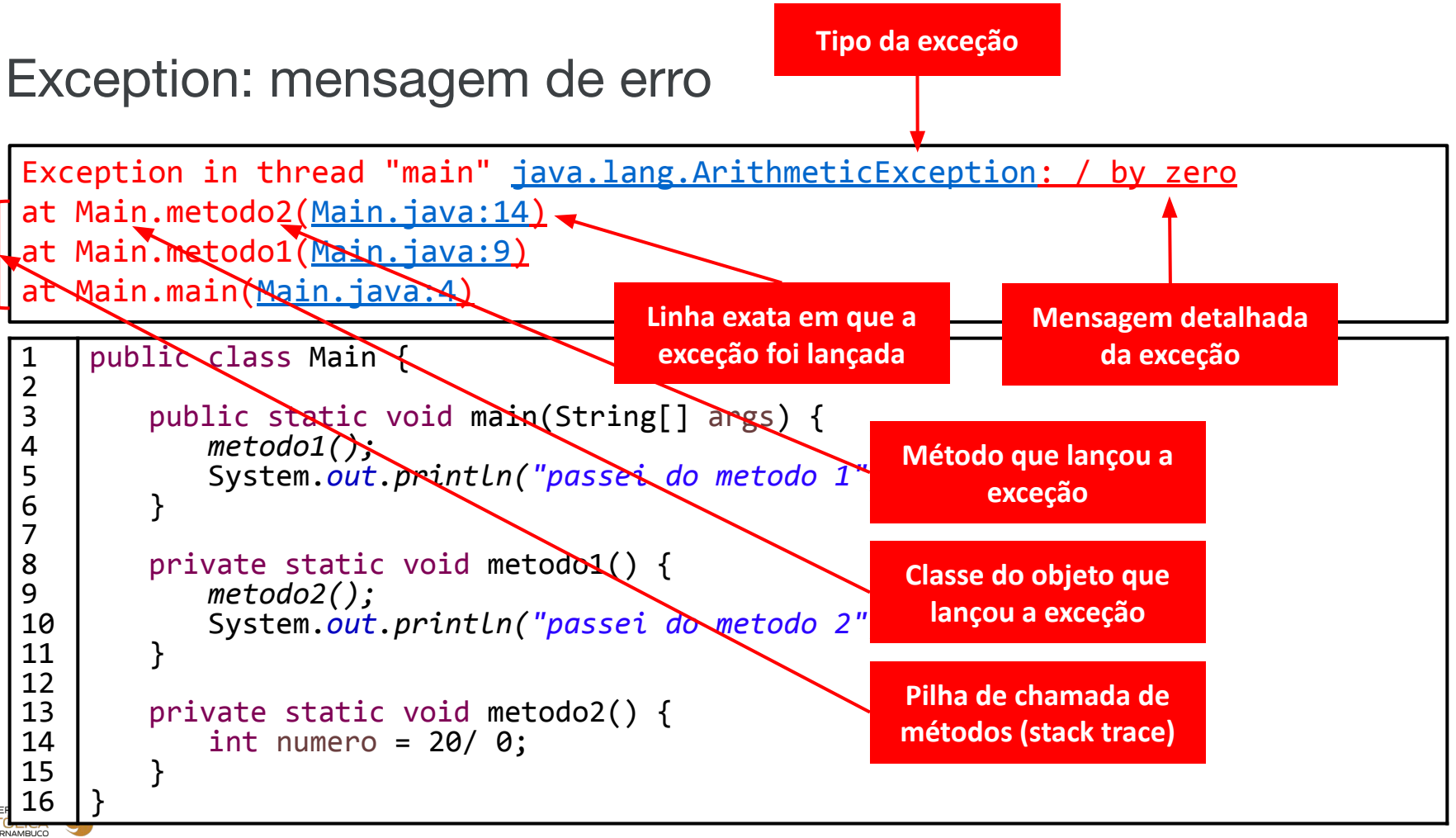


# Exception: mensagem de erro




```
1   public class Main {
2
3       public static void main(String[] args) {
4           metodo1();
5           System.out.println("passei do metodo 1"
6       }
7
8       private static void metodo1() {
9           metodo2();
10          System.out.println("passei do metodo 2"
11      }
12
13      private static void metodo2() {
14          int numero = 20/ 0;
15      }
16  }
```

# Exception: mensagem de erro



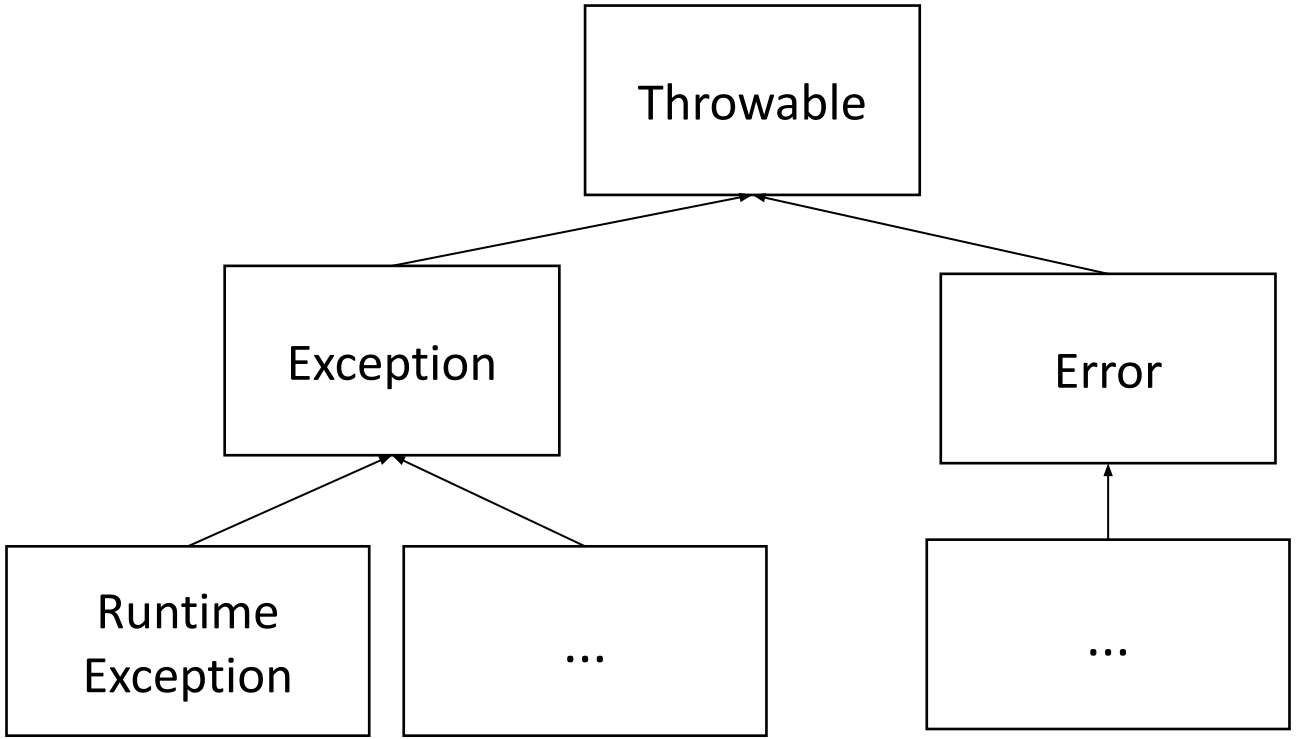
# Fluxo de exceção

Mensagens de erro foram feitas para **SEREM LIDAS**!!! Não tenha medo!

# 3.
# Hierarquia de exceções

Tratamento de exceções

# Hierarquia de exceções



# Error

- **Lançados pela JVM** e representados por objetos que herdam **classe** `Error`.
- Objetos que herdam da **classe** `Error` descrevem **erros internos de sistema**, como por exemplo estouro de memória.
- Esses tipos de erros **são raros de acontecer** e quando acontecem **não há muito o que possa** ser feito a não ser mostrar o erro na tela e parar a execução do programa.

# RuntimeExceptions (exceções não checadas)

- **Se é uma exceção não checada, a culpa é sua!**

- São causadas por erros de programação, por exemplo:
  - `NullPointerException`;
  - `ArrayIndexOutOfBounds`.

- Objetos que herdam da classe **RuntimeException**.

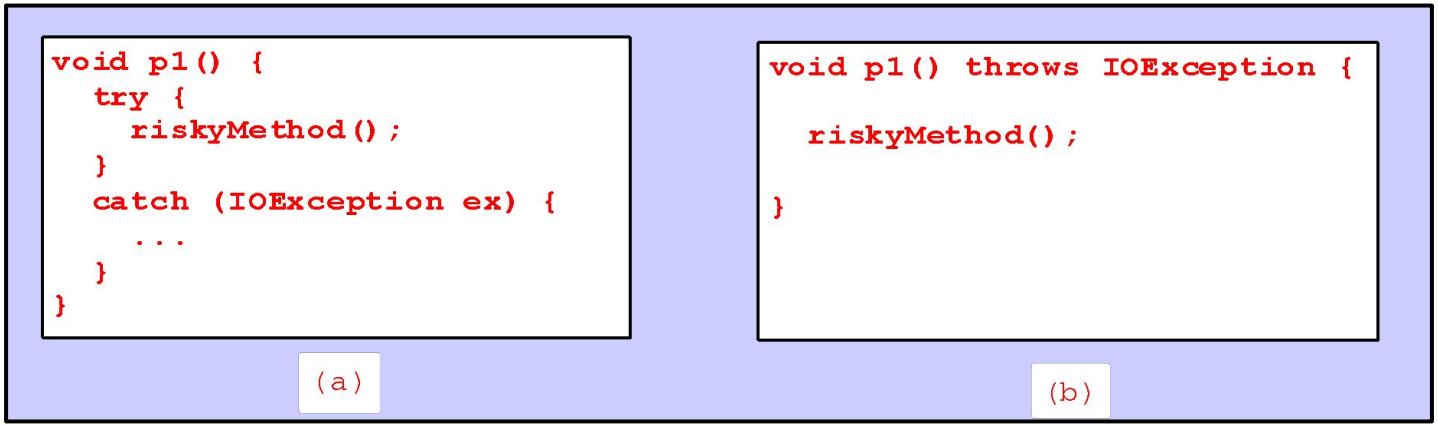
- São também conhecidas como **exceções não checadas**, pois não obrigam o programador a tratá-las de forma explícita.

# Exceções checadas

- Objetos que **herdam da classe** `Exception`, mas NÃO de `RuntimeException`.
- São conhecidas como exceções checadas, pois exigem que o programador faça um **tratamento explícito** quando chamar um método que lance uma exceção.
- Para saber **se um método lança exceção**, deve-se olhar na documentação do mesmo.
- Caso o programador não trate uma exceção checada, haverá **erro de compilação**.

# Como lidar com exceções em Java?

- É possível lidar com exceções de **três formas**:
  a. Tratando-as (bloco `try/catch`)
  b. Relançando-as (`throws`)
  c. Ignorando (exceções não checadas)



# E o que é tratamento de exceção?

- O tratamento de exceções em Java é um mecanismo que **detecta e responde a uma exceção**.

- A resposta a uma exceção **permite que o programador possa tomar ações apropriadas** ao invés de deixar o programa simplesmente acabar abruptamente.

# 4.
# try/catch

Tratamento de exceções

# Estrutura do tratador de exceções

```
try {
  // bloco de codigo que pode levantar uma exceção
}
catch (Exception1 exc) {
  // bloco de codigo para tratar a Exception1
}
catch (Exception2 | Exception3 exc) {
// bloco de codigo para tratar a Exception2 e a Exception3
}
catch (ExceptionN exc) {
  // bloco de codigo para tratar a ExceptionN
}
```

# Estrutura do tratador de exceções

```
try {
  // bloco de codigo que pode levantar uma exceção
}
catch (Exception1 exc) {
  // bloco de codigo para tratar a Exception1
}
catch (Exception2 | Exception3 exc) {
// bloco de codigo para tratar a Exception2 e a Exception3
}
catch (ExceptionN exc) {
  // bloco de codigo para tratar a ExceptionN
}
```

A partir do Java 7

# Exemplo: uso do bloco try/catch

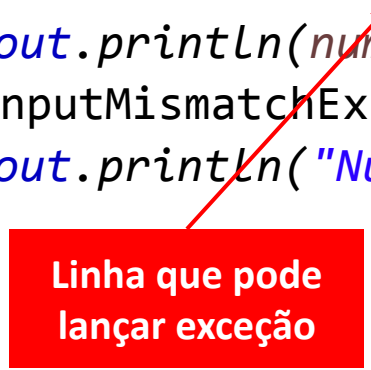
```
public class Main {
   public static void main(String[] args) {
    Scanner leitor = new Scanner(System.in);
    System.out.print("Digite um numero inteiro: ");

      try {
         int numero = leitor.nextInt();
         System.out.println(numero);
      } catch (InputMismatchException e) {
         System.out.println("Número inválido!");
      }
   }
}
```

# Exemplo: uso do bloco try/catch

```
public class Main {
   public static void main(String[] args) {
    Scanner leitor = new Scanner(System.in);
    System.out.print("Digite um numero inteiro: ");

      try {
         int numero = leitor.nextInt();
         System.out.println(numero);
      } catch (InputMismatchException e) {
         System.out.println("Número inválido!");
      }
   }
}
```



# Exemplo: uso do bloco try/catch

```
public class Main {
    public static void main(String[] args) {
     Scanner leitor = new Scanner(System.in);
     System.out.print("Digite um numero inteiro: ");

       try {
          int numero = leitor.nextInt();
          System.out.println(numero);
       } catch (InputMismatchException e) {
          System.out.println("Número inválido!");
       }
    }
}
```

**Trecho que NÃO deve ser executado caso uma exceção seja lançada**

**Linha que pode lançar exceção**

# Exemplo: uso do bloco try/catch

```
public class Main {
   public static void main(String[] args) {
    Scanner leitor = new Scanner(System.in);
    System.out.print("Digite um numero inteiro: ");

      try {
         int numero = leitor.nextInt();
         System.out.println(numero);
      } catch (InputMismatchException e) {
         System.out.println("Número inválido!");
      }
   }
}
```

Trecho que NÃO deve ser executado caso uma exceção seja lançada

Trata somente InputMismatchException

Linha que pode lançar exceção

# Exemplo: uso do bloco try/catch

```
public class Main {
   public static void main(String[] args) {
    Scanner leitor = new Scanner(System.in);
    System.out.print("Digite um numero inteiro: ");

      try {
         int numero = leitor.nextInt();
         System.out.println(numero);
      } catch (InputMismatchException e) {
         System.out.println("Número inválido!");
      }
   }
}
```

Trecho que NÃO deve ser executado caso uma exceção seja lançada

Trata somente `InputMismatchException`

Linha que pode lançar exceção

Tratamento da exceção

# Exemplo: uso do bloco try/catch

```
public class Main {
   public static void main(String[] args) {
    Scanner leitor = new Scanner(System.in);
    System.out.print("Digite um numero inteiro: ");

      try {
         int numero = leitor.nextInt();
         System.out.println(numero);
      } catch (Exception e) {
         System.out.println("Número inválido!");
      }
   }
}
```

**Por polimorfismo, vai capturar qualquer tipo de exceção, já que todas as exceções herdam de Exception**

# Obtendo informações das exceções

- Alguns métodos úteis:
  - `toString()`: retorna uma descrição sucinta da exceção;
  - `getMessage()`: retorna uma descrição detalhada da exceção;
  - `printStackTrace()`: imprime a “stacktrace” no console.

- Exemplo do que é impresso ao chamar o `printStackTrace()`:

```
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 Main.main(Main.java:10)
```

# Exemplo: obtendo informações das exceções

```
public class Main {
   public static void main(String[] args) {
    Scanner leitor = new Scanner(System.in);
    System.out.print("Digite um numero inteiro: ");

      try {
         int numero = leitor.nextInt();
         System.out.println(numero);
      } catch (InputMismatchException e) {
        e.printStackTrace();
      }
   }
}
```

**Imprimindo a stacktrace da exceção**

# Bloco finally

- **É executado sempre**, independente de ter havido exceção ou não.

```
try {
    // codigo que pode lançar execção
} catch (Exception e) {
    e.printStackTrace();
} finally {
    System.out.println("SEMPRE É EXECUTADO");
}
```

# 5.
# Lançando exceções

Tratamento de exceções

# Quando lançar exceções?

- O programador **deve lançar uma exceção** quando um código escrito encontrar um erro somente durante a execução do programa, **e que também não poderia ser evitado através teste simples**.

- Para lançar uma exceção é preciso:
  1. Criar um **objeto** de uma classe que **herde de Exception**;
  2. Lançar esse objeto de exceção usando a palavra-chave `throw`;
  3. Declarar que o método pode lançar uma exceção usando a palavra-chave `throws`, **caso a exceção seja checada**.

# Exemplo: lançando exceções

```
public class Circulo {
private double raio;

public void setRaio(double novoRaio) {
    if (novoRaio >= 0) {
     this.raio = novoRaio;
    } else {
     throw new IllegalArgumentException("O raio não pode ser negativo.");
    }
}
}
```

**Lançando uma exceção**

# Exemplo: lançando exceções

```
public static void main(String[] args) {
   Circulo circulo = new Circulo();
   circulo.setRaio(-10);
}
```

**Chamada ao método que lança a exceção**

```
Exception in thread "main" java.lang.IllegalArgumentException:
O raio não pode ser negativo.
at Circulo.setRaio(Circulo.java:10)
at Circulo.main(Circulo.java:16)
```

**Mensagem da exceção**

# Criando exceções customizadas

- É possível criar exceções customizadas, caso as exceções definidas pela API do Java **não atendam as necessidades do sistema**.

- Para criar uma exceção customizada é preciso:
  1. Criar uma **classe que herde de** `Exception` caso a exceção seja checada;
  2. Criar uma **classe que herde de** `RuntimeException`, caso a exceção seja não-checada;
  3. **É uma boa prática** definir dois construtores:
     - Um sem parâmetros;
     - Um que recebe uma string para definir a mensagem da exceção.

# Exemplo: declaração de exceção customizada

```
public class RaioInvalidoException extends Exception {

    public RaioInvalidoException() {

    }

    public RaioInvalidoException(String mensagem) {
        super(mensagem);
    }
}
```

# Exemplo: lançando uma exceção customizada

```
public class Circulo {

    private double raio;

    public void setRaio(double novoRaio) throws RaioInvalidoException {
        if (novoRaio >= 0) {
            this.raio = novoRaio;
        } else {
            throw new RaioInvalidoException("O raio não pode ser negativo.");
        }
    }
}
```

# Exemplo: tratando uma exceção customizada

```
public static void main(String[] args) {
    try {
        Circulo circulo = new Circulo();
        circulo.setRaio(-10);
    } catch (RaioInvalidoException e) {
        e.printStackTrace();
    }
}
```

```
RaioInvalidoException: O raio não pode ser negativo.
at Circulo.setRaio(Circulo.java:10)
at Circulo.main(Circulo.java:17)
```

# 6. Considerações finais

Tratamento de exceções

# O que aprendemos hoje?

- Compreender os diferentes tipos de erros em programação;
- Compreender o que é uma exceção e como ela é usada para representar erros durante a execução do programa;
- Conhecer a hierarquia de classes de exceções;
- Utilizar as construções `try` e `catch` para capturar e tratar exceções;
- Criar e lançar exceções personalizadas.

# Próxima aula…

-

# 7.
# Exercício de fixação

Teams

Exercício de fixação

# Tratamento de exceções

- **Link da atividade**: clique aqui.

Aula 13

# Tratamento de exceções

Programação III

Prof. Augusto César Oliveira
augusto.oliveira@unicap.br