

Objetivo:

- Discutir em detalhes o mecanismo de tratamento de exceções com o intuito de que o aluno possa utilizá-lo em seus programas.
- **Aprenderemos:**
 - a lançar exceções
 - a diferença entre exceções verificadas e não verificadas
 - a capturar exceções
 - quando e onde capturar uma exceção

Introdução

- Problemas podem ocorrer quando os programas são executados. Ex:
 - erros cometidos pelo programador (escrever numa posição de memória do vetor que não existe)
 - problemas nos ambientes externos ao da execução do programa (tentar ler um arquivo que não existe)
- Os programas devem ser capazes de lidar com possíveis problemas
- Existem duas ações importantes:
 - **detecção**
 - **recuperação**

Introdução

- Nos programas o ponto da detecção geralmente está dissociado do ponto de recuperação.

```
public static void main(String[] args) {  
    Rectangle rectangle = new Rectangle();  
    int a = Integer.parseInt(args[0]);  
    int b = Integer.parseInt(args[1]);  
    int area = rectangle.calculate(a,b);  
    System.out.println(" area of a rectangle is  : " + area);  
}
```



parseInt: detectei que o string não pode ser transformado num inteiro.

Introdução



- Nos programas o ponto da detecção geralmente está dissociado do ponto de recuperação.

```
public static void main(String[] args) {  
    Rectangle rectangle = new Rectangle();  
    int a = Integer.parseInt(args[0]);  
    int b = Integer.parseInt(args[1]);  
    int area = rectangle.calculate(a,b);  
    System.out.println(" area of a rectangle is : " + area);  
}
```



Mas não tenho informação suficiente para
decidir o que fazer...

Introdução

- O que fazer?
 - Pedir para o usuário tentar de novo
 - Informar o problema e sair do programa
- A lógica para executar essas ações é completamente independente do processamento normal que esses métodos fazem.
- O tratamento de exceções fornece um mecanismo para passar o controle do **ponto de detecção de erro**  para um **tratador competente de recuperação**. 

Exemplo sem tratamento de exceções

```
1. import java.util.Scanner;
2.
3. public class DivideByZeroNoExceptionHandling
4. {
5.     public static int quotient( int numerator, int denominator )
6.     {
7.         return numerator / denominator;
8.     } // fim de método quotient
9.
10.    public static void main( String args[] )
11.    {
12.        Scanner scanner = new Scanner( System.in ); // scanner para entrada
13.        System.out.print( "Enter an integer numerator: " );
14.        int numerator = scanner.nextInt(); 🍌
15.        System.out.print( "Enter an integer denominator: " );
16.        int denominator = scanner.nextInt() 🍌
17.        int result = quotient( numerator, denominator );
18.        System.out.printf(
19.            "\nResult: %d / %d = %d\n", numerator, denominator, result );
20.    } // fim de main
21.} // fim da classe DivideByZeroNoExceptionHandling
```

Continuação Exemplo

Execução 1: Divisão bem sucedida!

```
Enter an integer numerator: 77  
Enter an integer denominator: 7  
Result: 77/7 = 11
```

Continuação Exemplo

Execução 2:Usuário insere o valor 0 como denominador...



```
Enter an integer numerator: 100
Enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: /by zero
at
  DivideByZeroNoExceptionHandling.quotient(DivideByZeroNoExceptionHandling
    . java:7)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
  java:17)
```

Rastreamento de pilha:

- nome da exceção (**java.lang.ArithmeticException**)
- o problema que ocorreu (**/by zero**)
- o caminho de execução que resultou na exceção (pilha de chamadas de método)

Continuação Exemplo

Execução 2:Usuário insere o valor 0 como denominador...



```
Enter an integer numerator: 100
Enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: /by zero
at
    DivideByZeroNoExceptionHandling.quotient(DivideByZeroNoExceptionHandling
        . java:7)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
    java:17)
```

- Cada linha contém o nome da classe e o método, seguido pelo nome do arquivo e da linha.
- A linha superior da cadeia de chamadas indica o **ponto de lançamento** – ponto inicial onde a exceção ocorre. No exemplo a linha 7 do método **quotient**

Alguns Conceitos

- As exceções são **lançadas** quando um método **detecta** um problema e é incapaz de tratá-lo.
- Quando uma exceção ocorre dentro de um método, o método cria um **objeto do tipo exceção**.



Continuação Exemplo

Execução 3: Usuário insere a string “Hello” como denominador...



```
Enter an integer numerator: 100
Enter an integer denominator: Hello
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 DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
java:16)
```

- Informa a ocorrência de uma **InputMismatchException** (pacote **java.util**)
- A exceção foi detectada na linha 16 do método **main**.
- No lugar do nome do arquivo e linha, aparece o texto Unknown Source. A JVM não tem acesso ao código-fonte no local da exceção

Continuação Exemplo

- Nas execuções 2 e 3, quando as exceções ocorrem e os rastreamentos são exibidos, o programa também se fecha.
- Com tratamento de exceções o programa pode continuar mesmo que uma exceção tenha ocorrido!

Exceção

- Uma exceção é uma indicação de um **problema** que ocorre durante a **execução** de um programa.
- O tratamento de exceções permite aos programadores criar aplicativos que podem lidar com os possíveis problemas.
- Resultado: produtos de software mais **robustos e tolerantes a falhas!**
- O tratamento de exceções reduz a possibilidade de que erros sejam negligenciados.
- Mesclar a lógica do programa com o tratamento de exceções pode dificultar a leitura, modificação, manutenção e a depuração dos programas

Exemplo sem tratamento de exceções

```
1. import java.util.Scanner;
2.
3. public class DivideByZeroNoExceptionHandling
4. {
5.     public static int quotient( int numerator, int denominator )
6.     {
7.         return numerator / denominator;
8.     } // fim de método quotient
9.
10.    public static void main( String args[] )
11.    {
12.        Scanner scanner = new Scanner( System.in ); // scanner para entrada
13.        System.out.print( "Enter an integer numerator: " );
14.        int numerator = scanner.nextInt(); 🐸
15.        System.out.print( "Enter an integer denominator: " );
16.        int denominator = scanner.nextInt() 🐸
17.        int result = quotient( numerator, denominator );
18.        System.out.printf(
19.            "\nResult: %d / %d = %d\n", numerator, denominator, result );
20.    } // fim de main
21.} // fim da classe DivideByZeroNoExceptionHandling
```

O que fazer quando nextInt lança uma exceção
InputMismatchException?

Exemplo sem tratamento de exceções

```
1. import java.util.Scanner;
2.
3. public class DivideByZeroNoExceptionHandling
4. {
5.     public static int quotient( int numerator, int denominator )
6.     {
7.         return numerator / denominator;
8.     } // fim de método quotient
9.
10.    public static void main( String args[] )
11.    {
12.        Scanner scanner = new Scanner( System.in ); // scanner para entrada
13.        System.out.print( "Enter an integer numerator: " );
14.        int numerator = scanner.nextInt(); 🍌
15.        System.out.print( "Enter an integer denominator: " );
16.        int denominator = scanner.nextInt(); 🍌
17.        int result = quotient( numerator, denominator );
18.        System.out.printf(
19.            "\nResult: %d / %d = %d\n", numerator, denominator, result );
20.    } // fim de main
21.} // fim da classe DivideByZeroNoExceptionHandling
```

Permitir entrar novos dados

DICA:

Em um método que está pronto para tratar um tipo específico de exceção, coloque as instruções que podem causar a exceção dentro de um bloco *try* e o tratador da exceção dentro de um bloco *catch*

Exemplo com tratamento de exceções

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

    try{
        System.out.print( "Enter an integer numerator: " );
        int numerator = scanner.nextInt();
        System.out.print( "Enter an integer denominator: " );
        int denominator = scanner.nextInt();
        int result = quotient( numerator, denominator );
        System.out.printf(
            "\\nResult: %d / %d = %d\\n", numerator, denominator, result );
    }
    catch ( InputMismatchException inputMismatchException )
    {
        System.err.printf( "\\nException: %s\\n", inputMismatchException );
        scanner.nextLine(); // descarta entrada
        System.out.println("You must enter integers. Try again \\n" );
    } // fim de catch

} // fim de main
```

Exemplo com tratamento de exceções

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

    try{
        System.out.print( "Enter an integer numerator: " );
        int numerator = scanner.nextInt();
        System.out.print( "Enter an integer denominator: " );
        int denominator = scanner.nextInt();
        int result = quotient( numerator, denominator );
        System.out.printf(
            "\\nResult: %d / %d = %d\\n", numerator, denominator );
    }
    catch ( InputMismatchException inputMismatchException )
    {
        System.err.printf( "\\nException: %s\\n",
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again.") );
        // fim de catch
    }

} // fim de main
```

tratador
competente de
recuperação



Exemplo com tratamento de exceções



```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "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;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\\nException: %s\\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \\n" );
        } // fim de catch
    }while (continueLoop);
} // fim de main
```

Sintaxe: Instrução try

<pre>try{ código que pode gerar exceções }</pre>	bloco try
<pre>catch (classeExceção objetoExceção) { código de tratamento da exceção }</pre>	bloco catch
<pre>catch (classeExceção objetoExceção) { código de tratamento da exceção }</pre>	
<pre>finally{ código a ser executado sempre }</pre>	bloco finally (discutido depois)

Propósito: Executar uma ou mais instruções que podem gerar exceções. Se ocorrer um determinado tipo de exceção, interrompe a execução dessas instruções e vai para a cláusula catch correspondente. Se não ocorrer nenhuma exceção, pula as cláusulas catch. Em todos os casos, executa a cláusula finally, se houver alguma presente.

Instrução try

- Os métodos invocados no bloco *try*, podem criar objetos da classe *Exception* sinalizando condições de exceção. 
- O bloco *catch* (também chamado de *handler* de exceção) *captura* (i.e., recebe) e *trata* uma exceção. 
- Pelo menos um bloco *catch* ou um bloco *finally* deve seguir imediatamente a um bloco *try*.

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "Enter an integer denominator: " );
            int denominator = scanner.nextInt();
            int result = quotient( numerator, denominator );
            System.out.printf(
                "\\nResult: %d / %d = %d\\n", numerator, denominator,
                continueLoop=false;
            )
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\\nException: %s\\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \\n" );
        } // fim de catch
    }while (continueLoop);
} // fim de main
```

O que fazer se
o
denominador
é zero?

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "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;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\\nException: %s\\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \\n" );
        } // fim de catch
    }while (continueLoop);
} // fim de main
```

Permitir entrar novos dados

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "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;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\nException: %s\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \n" );
        } // fim de catch
        catch ( ArithmeticException arithmeticException )
        {
            System.err.printf( "\nException: %s\n", arithmeticException );
            System.out.println("Zero is an invalid denominator.Try again.\n");
        } // fim de catch
    }while (continueLoop);
} // fim de main
```


Instrução try



- Todo bloco *catch* especifica um **parâmetro de exceção** que identifica o tipo (classe) de exceção
- Quando ocorrer uma exceção num bloco *try*, o bloco *catch* executado é aquele cujo tipo de parâmetro **corresponde à exceção que ocorreu**.
- O parâmetro de exceção permite ao bloco *catch* interagir com um objeto de exceção capturado
 - Ex: invocar o método **toString** da exceção capturada, que exibe informações básicas sobre a exceção.

Instrução try: erros comuns

- É um erro de sintaxe colocar código entre um bloco *try* e seus blocos *catch*
- É um erro de sintaxe especificar uma lista de parâmetros de exceção no bloco *catch* .
- É um erro de compilação capturar o mesmo tipo de exceção em dois blocos *catch* diferentes de uma única instrução *try*.

Instrução try: fluxo de controle

```
try{  
    instrução  
    instrução  
    instrução  
    ...  
}  
catch (classeExceção objetoExceção) {  
    código de tratamento da exceção  
}  
catch (classeExceção objetoExceção) {  
    código de tratamento da exceção  
}  
catch (classeExceção objetoExceção) {  
    código de tratamento da exceção  
}  
finally{  
    código a ser executado sempre  
}
```

Instrução try: fluxo de controle

- Após a exceção ser tratada, o controle do programa **não retorna ao ponto de lançamento**. O bloco *try* expirou.
 - as variáveis locais do bloco também foram perdidas.
- O controle é retomado depois do último bloco *catch*.
 - Isso é conhecido como **modelo de terminação de tratamento de exceções**

Instrução try: nome do parâmetro do bloco catch

- Programadores Java costumam usar somente a letra “**e**” como o nome
- Parâmetros de exceção podem ser nomeados com base no seu tipo

• Ex: tipo

InputMismatchException

ArithmeticException

nome

InputMismatchException

ar

Para que?

lembra ao programador do tipo de exceção em tratamento

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in );
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "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;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\nException: %s\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \n" );
        } // fim de catch
        catch ( ArithmeticException arithmeticException )
        {
            System.err.printf( "\nException: %s\n", arithmeticException );
            System.out.println("Zero is an invalid denominator.Try again.\n");
        } // fim de catch
    }while (continueLoop);
} // fim de main
```

tenho que informar que esse método pode lançar uma ArithmeticException!!

Exemplo com tratamento de exceções

```
import java.util.Scanner;

public class DivideByZeroNoExceptionHandling
{
    public static int quotient( int numerator, int denominator ) throws
ArithmeticException
    {
        return numerator / denominator;
    } // fim de método quotient

    continua...
}
```

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "Enter an integer denominator: " );
            int denominator = scanner.nextInt();
            int result = quotient( numerator, denominator );
            System.out.printf("%d\nResult: %d / %d = %d\n", numerator,denominator,result);
            continueLoop=false;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\nException: %s\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \n" );
        } // fim de catch
        catch ( ArithmeticException arithmeticException )
        {
            System.err.printf( "\nException: %s\n", arithmeticException );
            System.out.println("Zero is an invalid denominator.Try again.\n");
        } // fim de catch
    }while (continueLoop);
} // fim de main
} // fim da classe DivideByZeroNoExceptionHandling
```


Sintaxe: Especificação de exceção

especificadorAcesso tipoRetorno nomeMetodo(tipoParametro
nomeParametro, ...) **throws** classeExceção, classeExceção, ...

Exemplo:

```
public void read(BufferedReader in) throws IOException
```

Propósito: Informar as exceções que esse método pode lançar. Os clientes do método são informados assim de que o método pode lançar essas exceções e de que elas deverão ser tratadas.

Exemplo:

Execução 2:Usuário insere o valor 0 como denominador...



```
Enter an integer numerator: 100  
Enter an integer denominator: 0
```

```
Exception in thread "main" java.lang.ArithmeticException: /by zero  
Zero is an invalid denominator.Try again
```

```
Enter an integer numerator: 77  
Enter an integer denominator: 7
```

```
Result: 77/7 = 11
```

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "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;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\nException: %s\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \n" );
        } // fim de catch
        catch ( ArithmeticException arithmeticException )
        {
            System.err.printf( "\nException: %s\n", arithmeticException );
            System.out.println("Zero is an invalid denominator.Try again.\n");
        } // fim de catch
    }while (continueLoop);
    } // fim de main
} // fim da classe DivideByZeroNoExceptionHandling
```

Exemplo com tratamento de exceções

```
import java.util.Scanner;

public class DivideByZeroNoExceptionHandling
{
    public static int quotient( int numerator, int denominator ) throws
        ArithmeticException
    {
        return numerator / denominator;
    } // fim de método quotient

    continua...
```

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "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;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\nException: %s\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \n" );
        } // fim de catch

        catch ( ArithmeticException arithmeticException )
        {
            System.err.printf( "\nException: %s\n", arithmeticException );
            System.out.println("Zero is an invalid denominator.Try again.\n");
        } // fim de catch

    }while (continueLoop);
} // fim de main
} // fim da classe DivideByZeroNoExceptionHandling
```

Exemplo:

Execução 3: Usuário insere a string “Hello” como denominador...



```
Enter an integer numerator: 100  
Enter an integer denominator: Hello
```

```
Exception: java.util.InputMismatchException  
You must enter integers. Try again
```

```
Enter an integer numerator: 77  
Enter an integer denominator: 7
```

```
Result: 77/7 = 11
```

Exemplo com tratamento de exceções

```
public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner para entrada
    boolean continueLoop=true;
    do{
        try{
            System.out.print( "Enter an integer numerator: " );
            int numerator = scanner.nextInt();
            System.out.print( "Enter an integer denominator: " );
            int denominator = scanner.nextInt();
            int result = quotient( numerator, denominator );
            System.out.printf("%d / %d = %d\n", numerator, denominator, result);
            continueLoop=false;
        }
        catch ( InputMismatchException inputMismatchException )
        {
            System.err.printf( "\nException: %s\n", inputMismatchException );
            scanner.nextLine(); // descarta entrada
            System.out.println("You must enter integers. Try again \n" );
        } // fim de catch
        catch ( ArithmeticException arithmeticException )
        {
            System.err.printf( "\nException: %s\n", arithmeticException );
            System.out.println("Zero is an invalid denominator.Try again.\n");
        } // fim de catch
    }while (continueLoop);
} // fim de main
} // fim da classe DivideByZeroNoExceptionHandling
```

Resumo:

- **Exceção** = indicação de problema durante a execução

```
try{  
    código que pode gerar exceções  
}  
catch (classeExceção objetoExceção) {  
    código de tratamento da exceção  
}
```

```
especificadorAcesso tipoRetorno nomeMetodo(tipoParametro  
nomeParametro, ...) throws classeExceção, classeExceção, ...
```


Exercício 1: incluir tratamento de exceções

```
import java.util.Scanner;

public class DivideByZeroVector
{
    public static void main( String args[] )
    {
        int [] list=new int[4];
        int x=10;
        Scanner scanner = new Scanner( System.in ); // scanner para entrada
        System.out.print( "Enter an integer denominator: " );
        int y = scanner.nextInt();
        int result = x/y;
        list[result]=result;
    } // fim de main
} // fim da classe
```

Exercício 2: imprime ``Dentro do bloco try``?

```
public class TrataExcecao
{
    public static void main (String [] args)
    {
        try
        {
            int vetor[] = new int[100];
            vetor[100] = 1;
            System.out.println("Dentro do bloco try...");
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Ocorreu a excecao: " + e.getMessage());
        }
        System.out.println("Após o tratamento de exceções...");
    }
}
```

nome feio