

# Programação Avançada II

---

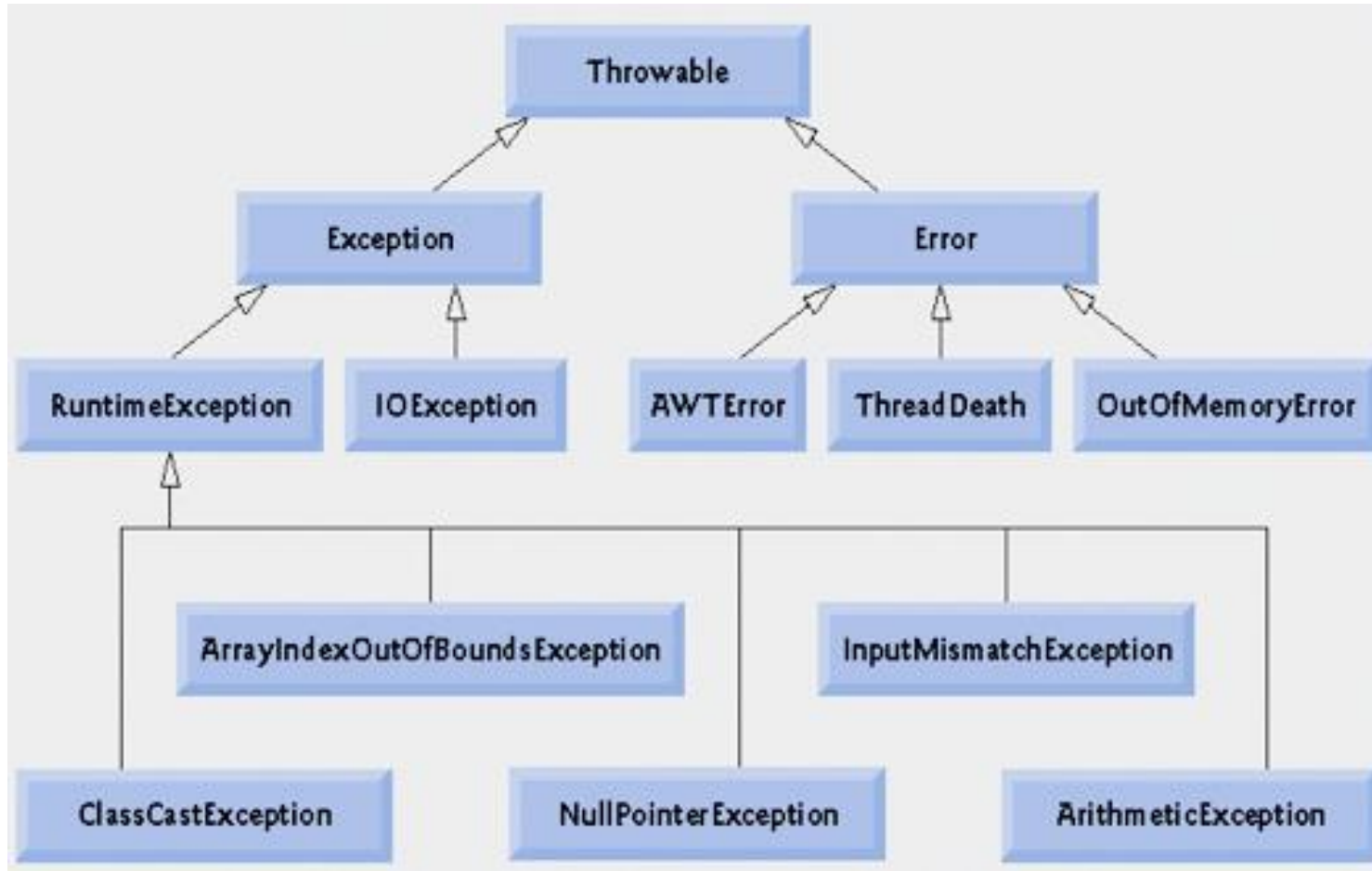
PROFESSOR ISAAC

---

# Tratamento de exceções



# Hierarquia dos Erros e Exceções



Fonte: <https://www.devmedia.com.br/trabalhando-com-excecoes-em-java/27601>

# Classe Error

---

Usada pela Máquina Virtual Java (Java Virtual Machine - JVM) para indicar se existe problemas no programa

Não permite tratamento e a execução do programa é interrompida

Exemplos:

- StackOverflowError
- OutOfMemoryError

# Exceções

---

Na linguagem Java existe um mecanismo que permite o tratamento de exceções em tempo de execução do programa.

O Java possibilita tratar e as exceções para evitar a interrupção do programa.

As exceções são “lançadas” de volta para o fluxo de execução do programa.

# Classe Exception

---

A classe Exception é a forma mais genérica de representar uma exceção em Java.

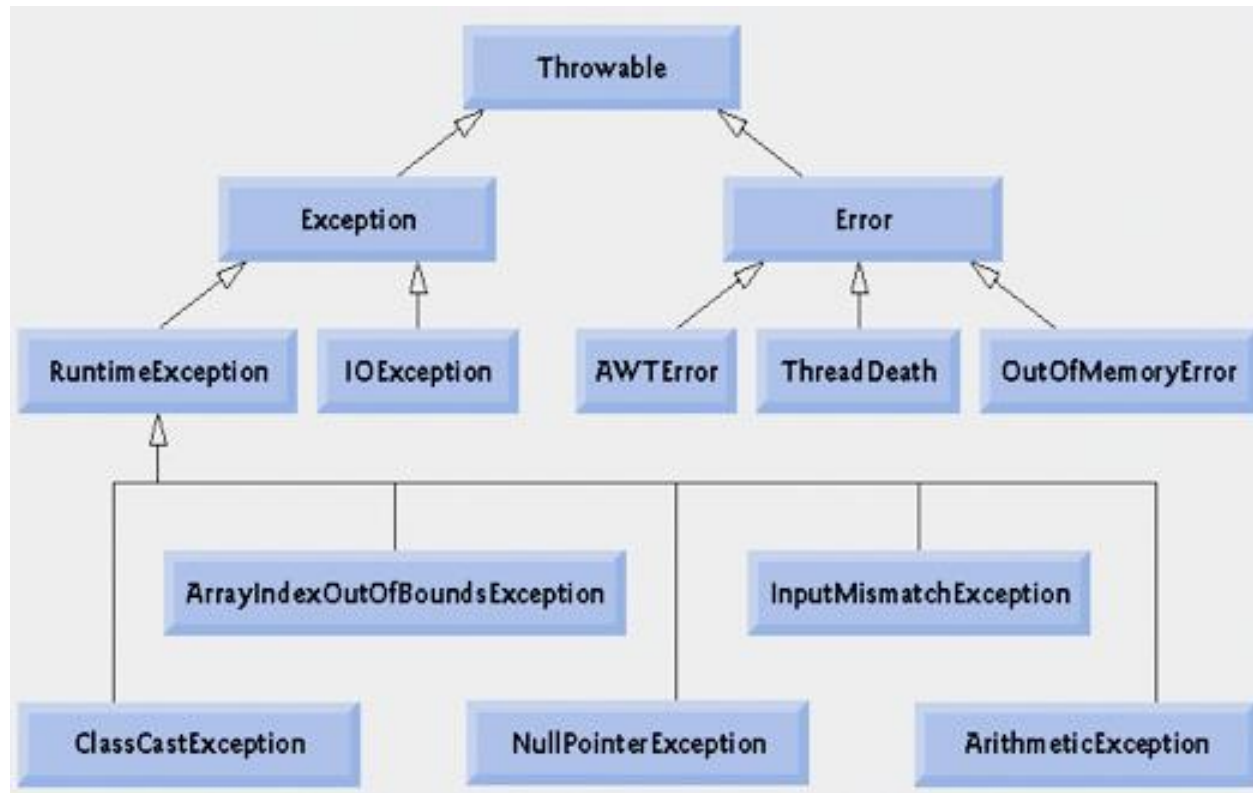
Todas as exceções em Java pode ser tratadas por objetos do tipo Exception.

Existem outras classes que tratam exceções específicas, mas todas são classes filhas da Exception, como por exemplo, IOException, SQLException, etc.

**IMPORTANTE:** deve-se usar a classe mais específica possível para tratar a exceção.

# Tipos de Exceções

---



<https://docs.oracle.com/javase/10/docs/api/java/lang/package-tree.html>

# Bloco try/catch

---

As exceções e os erros “lançados” de volta para o fluxo de execução do programa pode ser tratados pelo bloco try/catch.

O try delimita o trecho do código que pode ocorrer a exceção e o catch “pega” a exceção “lançada”.

Sintaxe:

```
try{  
    // trecho onde a exceção pode ocorrer  
} catch ( TipoExceção exc){  
    // comandos a executar se a exceção ocorrer  
}
```



# Exemplo de exceção

```
6 public static int divisao(int numerador, int denominador){  
7     return numerador / denominador;  
8 }  
9  
10 public static void main(String[] args) {  
11     // TODO code application logic here  
12  
13     try {  
14         int resultado = divisao(5, 1);  
15         System.err.println("Resultado da divisão: "+ resultado);  
16     } catch (ArithmeticException ex) {  
17         System.err.printf("Exception: %s\n", ex);  
18         System.err.println("Zero é um denominador invalido");  
19     }  
20 }  
21 }
```

run:

Resultado da divisão: 5

BUILD SUCCESSFUL (total time: 0 seconds)

# Exemplo da exceção Aritmética

```
4 public class Principal {  
5  
6     public static int divisao(int numerador, int denominador){  
7         return numerador / denominador;  
8     }  
9  
10    public static void main(String[] args) {  
11        // TODO code application logic here  
12  
13        try {  
14            int resultado = divisao(5, 0);  
15            System.err.println("Resultado da divisão: "+ resultado);  
16        } catch (ArithmeticException ex) {  
17            System.err.printf("Exception: %s\n", ex);  
18            System.err.println("Zero é um denominador invalido");  
19        }  
20    }  
21 }
```

```
run:  
Exception: java.lang.ArithmeticException: / by zero  
Zero é um denominador invalido  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Tratando as exceções

---

O programa deve tratar as exceções para evitar a interrupção do programa. Para manter o programa executando mesmo que as exceções ocorram, deverá haver o tratamento para tais exceções.

Com o tratamento de exceções, um programa pode continuar executando (em vez de encerrar) depois de lidar com um problema.

# Não tratando as exceções

---

As exceções não tratadas são aquelas que não possuem nenhum bloco *catch*, são exceções não capturadas.

As exceções não tratadas fazem o programa ser interrompido durante a execução no momento que ocorrer o problema.

# Exemplo da exceção Aritmética

```
11 public static void main(String[] args) {  
12     // TODO code application logic here  
13     Scanner teclado = new Scanner(System.in);  
14     try {  
15         int numerador = 5;  
16         System.out.println("Digite um número inteiro:");  
17         int denominador = teclado.nextInt();  
18         int resultado = divisao(numerador, denominador);  
19         System.err.println("Resultado da divisão: "+ resultado);  
20     } catch (ArithmeticException ex) {  
21         System.err.printf("Exception: %s\n", ex);  
22         System.err.println("Zero é um denominador invalido");  
23     }  
24     System.out.println("Terminando o programa");  
25 }  
26 }
```

trvcatch.Principal > divisao >

run:  
Digite um número inteiro:  
0  
Exception: java.lang.ArithmeticException: / by zero  
Zero é um denominador invalido  
Terminando o programa  
BUILD SUCCESSFUL (total time: 3 seconds)

# Exemplo da exceção sem tratamento

```
11 public static void main(String[] args) {  
12     // TODO code application logic here  
13     Scanner teclado = new Scanner(System.in);  
14     try {  
15         int numerador = 5;  
16         System.out.println("Digite um número inteiro:");  
17         int denominador = teclado.nextInt();  
18         int resultado = divisao(numerador, denominador);  
19         System.err.println("Resultado da divisão: "+ resultado);  
20     } catch (ArithmeticException ex) {  
21         System.err.printf("Exception: %s\n", ex);  
22         System.err.println("Zero é um denominador invalido");  
23     }  
24     System.out.println("Terminando o programa");  
25 }  
26 }
```

trvcatch.Principal > divisao >

run:  
Digite um número inteiro:  
a

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 trycatch.Principal.main(Principal.java:17)  
/home/fei/.cache/netbeans/8.2/executor-snippets/run.xml:53: Java returned: 1  
BUILD FAILED (total time: 3 seconds)

# Tratamento de múltipla exceções

---

O Java permite o tratamento de mais de uma exceção que tenha ocorrido no try.  
Para cada exceção deve ser escrito um catch.

Exemplo:

```
try{  
    // trecho onde a exceção pode ocorrer  
} catch ( SQLException exc1) {  
    // comandos a executar se a exceção ocorrer  
} catch ( Exception exc2) {  
    // comandos a executar se a exceção ocorrer  
}
```

# Exercício

---

Capture a segunda exceção e realize um tratamento.



# Resposta do exercício

```
11 public static void main(String[] args) {  
12     // TODO code application logic here  
13     Scanner teclado = new Scanner(System.in);  
14     try {  
15         int numerador = 5;  
16         System.out.println("Digite um número inteiro:");  
17         int denominador = teclado.nextInt();  
18         int resultado = divisao(numerador, denominador);  
19         System.err.println("Resultado da divisão: " + resultado);  
20     } catch (InputMismatchException ex1) {  
21         System.err.printf("Exception: %s\n", ex1);  
22         System.err.println("Você deveria entrar com um número inteiro");  
23     } catch (ArithmeticException ex2) {  
24         System.err.printf("Exception: %s\n", ex2);  
25         System.err.println("Zero é um denominador invalido");  
26     }  
27 }  
28 }
```

```
run:  
Digite um número inteiro:  
a  
Exception: java.util.InputMismatchException  
Você deveria entrar com um número inteiro  
BUILD SUCCESSFUL (total time: 2 seconds)
```

# Exemplo de tratamento de múltiplas exceções

```
public static void main(String[] args) {  
    // TODO code application logic here  
    Scanner teclado = new Scanner(System.in);  
    try {  
        int numerador = 5;  
        System.out.println("Digite um número inteiro:");  
        int denominador = teclado.nextInt();  
        int resultado = divisao(numerador, denominador);  
        System.err.println("Resultado da divisão: "+ resultado);  
    } catch (InputMismatchException ex1) {  
        System.err.printf("Exception: %s\n", ex1);  
        System.err.println("Você deveria entrar com um número inteiro");  
    } catch (ArithmeticException ex2) {  
        System.err.printf("Exception: %s\n", ex2);  
        System.err.println("Zero é um denominador invalido");  
    }  
}
```

trvcatch.Principal > divisao >

```
run:  
Digite um número inteiro:  
0  
Exception: java.lang.ArithmeticException: / by zero  
Zero é um denominador invalido  
BUILD SUCCESSFUL (total time: 3 seconds)
```

# Bloco finally

---

Pode ser usado com o try/catch, mas é opcional.

O trecho de código delimitado pelo finally será sempre executado.

Sintaxe:

```
try{  
    // trecho onde a exceção pode ocorrer  
} catch ( SQLException exc) {  
    // comandos a executar se a exceção ocorrer  
} finally {  
    // comandos sempre executados, com ou sem exceção  
}
```

# Exemplo de bloco finally

```
3 public class Principal {
4     public static int divisao(int numerador, int denominador){
5         return numerador / denominador;
6     }
7     public static void main(String[] args) {
8         // TODO code application logic here
9         try {
10             int resultado = divisao(5, 0);
11             System.err.println("Resultado da divisão: "+ resultado);
12         } catch (ArithmeticException ex) {
13             System.err.printf("Exception: %s\n", ex);
14             System.err.println("Zero é um denominador invalido");
15         } finally {
16             System.err.println("Essa linha de código sempre será executada");
17         }
18     }
19 }
```

run:  
Exception: java.lang.ArithmeticException: / by zero  
Zero é um denominador invalido  
Essa linha de código sempre será executada  
BUILD SUCCESSFUL (total time: 0 seconds)

# Exemplo de bloco finally

```
3 public class Principal {  
4     public static int divisao(int numerador, int denominador){  
5         return numerador / denominador;  
6     }  
7     public static void main(String[] args) {  
8         // TODO code application logic here  
9         try {  
10             int resultado = divisao(5, 1);  
11             System.err.println("Resultado da divisão: "+ resultado);  
12         } catch (ArithmeticException ex) {  
13             System.err.printf("Exception: %s\n", ex);  
14             System.err.println("Zero é um denominador invalido");  
15         } finally {  
16             System.err.println("Essa linha de código sempre será executada");  
17         }  
18     }  
19 }
```

```
run:  
Resultado da divisão: 5  
Essa linha de código sempre será executada  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Resumo do Bloco try/catch/finally

---

Caso **ocorra** exceção no trecho de código do try:

- O código depois da linha com erro não será executado.
- Os comandos do catch serão executados.

Caso **NÃO ocorra** exceção no trecho de código do try:

- Todos os comandos do try serão executados.
- Os comandos do catch NÃO serão executados.

Nos dois casos os comandos do finally serão executados.

# Resumo do Bloco try/catch/finally

---

Se uma exceção que ocorre em um bloco try não puder ser capturada, o programa pula o restante do bloco try e prossegue para o bloco finally, assim, após o finally ocorrerá a falha de execução no programa.

# Exemplo de bloco finally com exceção não tratada

```
10 public static void main(String[] args) {
11     // TODO code application logic here
12     Scanner teclado = new Scanner(System.in);
13     try {
14         int numerador = 5;
15         System.out.println("Digite um número inteiro:");
16         int denominador = teclado.nextInt();
17         int resultado = divisao(numerador, denominador);
18         System.out.println("Resultado da divisão: "+ resultado);
19     } catch (ArithmeticException ex) {
20         System.err.printf("Exception: %s\n", ex);
21         System.err.println("Zero é um denominador invalido");
22     } finally {
23         System.out.println("Essa linha de código sempre será executada");
24     }
25     System.out.println("Terminando o programa");
26 }
27 }
```

Output - TryCatch (run) X



Digite um número inteiro:



a

Essa linha de código sempre será executada



Exception in thread "main" java.util.InputMismatchException



```
at java.base/java.util.Scanner.throwFor(Scanner.java:939)
at java.base/java.util.Scanner.next(Scanner.java:1594)
at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
at trycatch.Principal.main(Principal.java:16)
```



# Exercício

---

Trate a exceção do exemplo anterior.

# Tratando a exceção dentro do método

---

```
public static int divisao (int numerador, int denominador){  
    try{  
        return numerador / denominador;  
    }catch(ArithmeticException ex2){  
        System.err.printf("Exception: %s\n", ex2);  
        System.err.println("Zero é um denominador invalido");  
        return 0;  
    }  
}
```

# Tratando a exceção dentro do método

```
public static void main(String[] args) {  
    // TODO code application logic here  
    Scanner teclado = new Scanner(System.in);  
    int numerador = 5;  
    System.out.println("Digite um número inteiro:");  
    int denominador = teclado.nextInt();  
    int resultado = divisao(numerador, denominador);  
    System.err.println("Resultado da divisão: "+ resultado);  
}
```

trowsandthrow.Principal > divisao >

run:

Digite um número inteiro:

0

Exception: java.lang.ArithmeticException: / by zero

Zero é um denominador invalido

Resultado da divisão: 0

BUILD SUCCESSFUL (total time: 2 seconds)

# Tratando a exceção dentro do método

```
public static void main(String[] args) {  
    // TODO code application logic here  
    Scanner teclado = new Scanner(System.in);  
    int numerador = 5;  
    System.out.println("Digite um número inteiro:");  
    int denominador = teclado.nextInt();  
    int resultado = divisao(numerador, denominador);  
    System.err.println("Resultado da divisão: "+ resultado);  
}
```

trowsandthrow.Principal > divisao >

run:

Digite um número inteiro:

a

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 trowsandthrow.Principal.main([Principal.java:29](#))

[/home/fei/.cache/netbeans/8.2/executor-snippets/run.xml:53](#): Java returned: 1

BUILD FAILED (total time: 1 second)

# Exercício

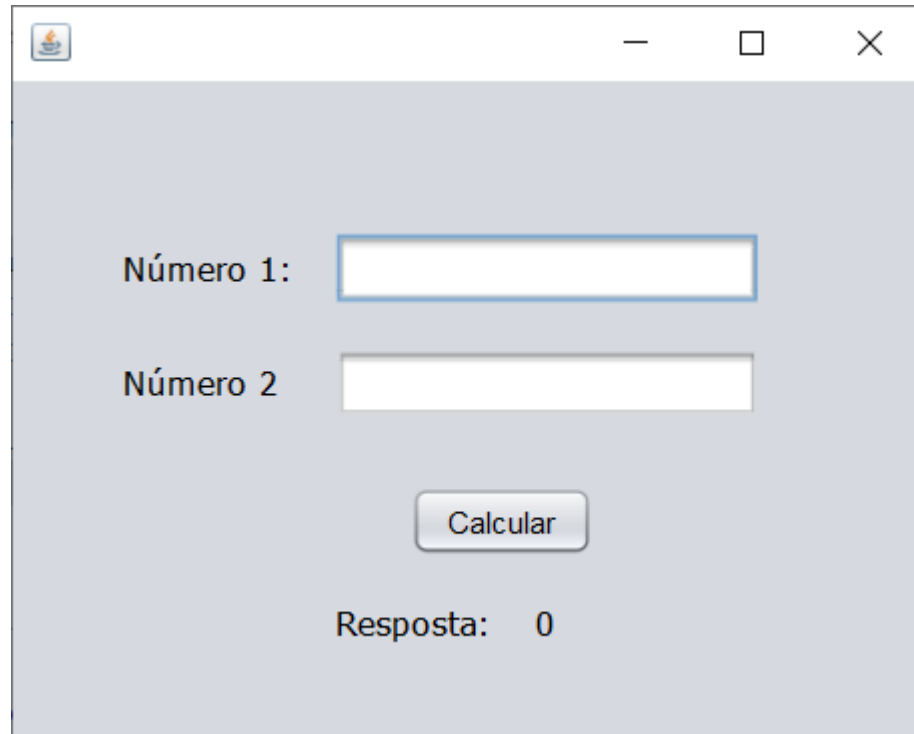
---

Realize o tratamento da divisão dentro do método.

# Exercício

---

Realize o tratamento das exceções do projeto de uma Janela que Divide 2 números.



The image shows a Java Swing window titled "Divisor" with a standard Mac OS-style title bar (red, yellow, and green buttons). The window has a light gray background and contains the following elements:

- A label "Número 1:" followed by a text input field.
- A label "Número 2" followed by a text input field.
- A button labeled "Calcular" (Calculate) with a light blue gradient.
- A label "Resposta:" followed by the text "0".

# Palavra chave throws

Alguns momentos, pode ocorrer de o programador não querer realizar controle sobre uma exceção, isto é, não desejar tratar um erro. A linguagem Java permite ao programador que um erro seja descartado, mesmo que ele ocorra

```
public static ArrayList leArquivo() throws IOException {  
    FileReader arquivo = new FileReader("impares1.txt");  
    BufferedReader br = new BufferedReader(arquivo);  
  
    ArrayList<String> impares = new ArrayList<>();  
    String str;  
    while((str = br.readLine()) != null)  
        impares.add(str);  
    return impares;  
}
```

# Palavra chave throws

Na verdade, ao usar a cláusula throws, o erro não é descartado e sim postergado. Essa Ele foi postergado e deveria ser tratado em outra classe.

```
public static ArrayList leArquivo() throws IOException {  
    FileReader arquivo = new FileReader("impares1.txt");  
    BufferedReader br = new BufferedReader(arquivo);  
  
    ArrayList<String> impares = new ArrayList<>();  
    String str;  
    while((str = br.readLine()) != null)  
        impares.add(str);  
    return impares;  
}
```



# Palavra chave throws

```
22  public static ArrayList leArquivo() throws IOException {
23      FileReader arquivo = new FileReader("impares.txt");
24      BufferedReader br = new BufferedReader(arquivo);
25
26      ArrayList<String> impares = new ArrayList<>();
27      String str;
28      while((str = br.readLine()) != null)
29          impares.add(str);
30      return impares;
31  }
32
33  public static void main(String[] args) throws IOException {
34      ArrayList<String> lista = leArquivo();
35      System.out.println(lista);
36  }
37  }
```

Output - arquivoThrows2 (run) x



run:

[Número = 1, Número = 3, Número = 5, Número = 7, Número = 9, Número = 11, Número = 13, Número = 15, Número = 17, Número = 19]  
BUILD SUCCESSFUL (total time: 0 seconds)

# Palavra chave throws

```
22  public static ArrayList leArquivo() throws IOException {
23      FileReader arquivo = new FileReader("impares1.txt");
24      BufferedReader br = new BufferedReader(arquivo);
25
26      ArrayList<String> impares = new ArrayList<>();
27      String str;
28      while((str = br.readLine()) != null)
29          impares.add(str);
30      return impares;
31  }
32
33  public static void main(String[] args) throws IOException {
34      ArrayList<String> lista = leArquivo();
35      System.out.println(lista);
36  }
37  }
```

Output - arquivoThrows2 (run) x

run:

```
Exception in thread "main" java.io.FileNotFoundException: impares1.txt (O sistema não pode encontrar o arquivo especificado)
    at java.base/java.io.FileInputStream.open0(Native Method)
    at java.base/java.io.FileInputStream.open(FileInputStream.java:216)
```

# Palavra chave throw

---

A instrução throw é utilizada para disparar uma exceção, isto é, ela pode forçar que uma determinada exceção ocorra.

Usada quando se quer “lançar” uma exceção em alguma situação específica.

Sintaxe:

```
throw new Exception("Senha inválida!");
```

# Exemplo de throws e throw

---

```
public static boolean verificaIdade(int idade) throws Exception {  
    if(idade < 18){  
        throw new Exception("Idade menor que 18, não permitido");  
    }  
    return false;  
}
```

# Exemplo de throws e throw

```
public class Principal {  
  
    public static boolean verificaIdade(int idade) throws Exception {  
        if(idade < 18){  
            throw new Exception("Idade menor que 18, não permitido");  
        }  
        return false;  
    }  
  
    public static void main(String[] args) {  
        int idade = Integer.parseInt(JOptionPane.showInputDialog("Digite a idade"));  
  
        try {  
            verificaIdade(idade);  
        } catch (Exception ex) {  
            System.out.println("Mensagem: " + ex.getMessage());  
        }  
    }  
}
```

# Exemplo5 de throws e throw – classe calculo

---

```
public class Calculo {  
    public int divisao (int numerador, int denominador)  
    throws Exception {  
        if(denominador ==0){  
            throw new Exception("Denominador não pode ser zero");  
        } else {  
            return numerador / denominador;  
        }  
    }  
}
```

---

# Exemplo5 de throws e throw

```
public class Principal {  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Calculo calc1 = new Calculo();  
        Scanner teclado = new Scanner(System.in);  
        int numerador = 5;  
        System.out.println("Digite um número inteiro:");  
        int denominador = teclado.nextInt();  
        try {  
            int resultado = calc1.divisao(numerador, denominador);  
            System.out.println("Resultado da divisão: "+ resultado);  
        } catch (Exception ex) {  
            System.out.printf("Exception: "+ ex.getMessage() + "\n");  
        } finally {  
            System.out.println("Essa linha de código sempre será executada");  
        }  
    }  
}
```