Definição e tratamento de exceções para sistemas com *threads*

Programação Orientada a Objetos

2023-03-21

- Geralmente quando acontece algum erro, Java emite uma mensagem, e termina a execução
- Outro jeito de tratar erros é usando try e catch
- O sistema não será encerrado caso ocorra algum erro
- É possível usar código para tratar o erro

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

leitura de diversos fluxos, incluindo teclado

```
public class LeitorTeclado {

  public void lerDados() {
    try {
      Scanner in = new Scanner(System.in);
      int numero = in.nextInt();
      System.out.println(numero);
      in.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    LeitorTeclado l = new LeitorTeclado();
    l.lerDados();
  }
}
```

entrada de nros inteiros

em caso de exceção

```
$ javac LeitorTeclado.java
$ java LeitorTeclado
a
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 LeitorTeclado.lerDados(LeitorTeclado.java:8)
        at LeitorTeclado.main(LeitorTeclado.java:18)
$ java LeitorTeclado
123
123
```

exceção
(vide próximo slide)

linhas do código
fonte da bilbioteca

o que causou
a origem

Quadro 2.10 | Exceções comuns em códigos Java

Exceção	Descrição
`java.lang.ArrayIndexOutOfBoundsException`	Quando se tenta acessar uma posição de vetor ou matriz que não existe.
`java.lang.ArithmeticException`	Gerado na divisão de um número **`int`** por zero.
`java.lang.IllegalArgumentException`	Enviado quando se passa argumentos errados para um método.
`java.io.FileNotFoundException`	Quando se tenta fazer a leitura ou escrita em um arquivos que não existe.

```java
import java.io.*;
public class ControleArquivos {
  public void escreveArquivo(String caminhoArquivo) {
    FileWriter fw = null;
    File f = new File(caminhoArquivo);
    try {
      fw = new FileWriter(f);
      fw.write(65);
      fw.close();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (fw != null)
        try {
          fw.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
    }
  }
  public static void main(String[] args) {
    ControleArquivos l = new ControleArquivos();
    l.escreveArquivo("arquivoTeste.txt");
  }
}
```

bloco onde o erro pode ocorrer

executa caso erro

sempre executa

ControleArquivos.java — KWrite

```java
import java.io.*;
public class ControleArquivos {
  public void escreveArquivo(String caminhoArquivo) {
    FileWriter fw = null;
    File f = new File(caminhoArquivo);
    try {
      fw = new FileWriter(f);
      fw.write(65);
      fw.close();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (fw != null)
        try {
          fw.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
    }
  }
  public static void main(String[] args) {
    ControleArquivos l = new ControleArquivos();
    l.escreveArquivo("arquivoTeste.txt");
  }
}
```

(slide anterior)

java : bash — Konsole

```
$ javac ControleArquivos.java
$ java ControleArquivos
$ cat arquivoTeste.txt ; echo
A
$
```

- Try
  - executa as tarefas pertinentes à abertura de recursos necessários (arquivos, conexões de rede e outros)
- Catch
  - avalia o que ocorreu e se cria logs para informar qual é o problema
- Finally
  - fecha os recursos utilizados, em alguns casos sendo necessários outros blocos para garantir que todos os recursos sejam utilizados

Fonte: geeksforgeeks.org

Closeable

```
import java.io.*;
class Resource {
 public static void main(String s[]) {
  try (Demo d = new Demo(); Demo1 d1 = new Demo1()) {
   d.show();
   d1.show1();
   int x = 10 / 0;
  } catch (ArithmeticException e) {
   System.out.println(e); } } }
class Demo implements Closeable {
 void show() {
  System.out.println("inside show"); }
 public void close() {
  System.out.println("close from demo"); } }
class Demo1 implements Closeable {
 void show1() {
  System.out.println("inside show1"); }
 public void close() {
  System.out.println("close from demo1"); } }
```

vide saída no próximo slide

divisão por zero

close

```
$ javac Resource.java
$ java Resource
inside show
inside show1
close from demo1
close from demo
java.lang.ArithmeticException: / by zero
$
```

throws

- throws indica uma exceção personalizada
  - ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.

throws

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
public class EscritorArquivo {
  private String caminhoArquivo;
  private FileWriter fw;
  private File f;
  public EscritorArquivo(String pCaminhoArquivo) {
    caminhoArquivo = pCaminhoArquivo; }
  public void escreveArquivo() throws IOException {
    f = new File(caminhoArquivo);
    fw = new FileWriter(f);
    fw.write(65);
    if (fw != null)
      fw.close(); }
  public static void main(String[] args) {
    EscritorArquivo s = new EscritorArquivo("arquivo.txt");
    try {
      s.escreveArquivo();
    } catch (IOException e) {
      e.printStackTrace(); } } }
```

se ocorrer um erro,
lança IOException

```
$ javac EscritorArquivo.java
$ java EscritorArquivo
$ cat arquivo.txt ; echo
A
$
```

saída da execução

quando não coloca o throw:

```
$ javac EscritorArquivo.java
EscritorArquivo.java:12: error: unreported exception IOException;
must be caught or declared to be thrown
   fw = new FileWriter(f);
        ^
EscritorArquivo.java:13: error: unreported exception IOException;
must be caught or declared to be thrown
   fw.write(65);
           ^
EscritorArquivo.java:15: error: unreported exception IOException;
must be caught or declared to be thrown
     fw.close(); }
              ^
EscritorArquivo.java:20: error: exception IOException is never
thrown in body of corresponding try statement
   } catch (IOException e) {
     ^
4 errors
$
```

throws

```
import java.util.Scanner;
public class LeitorTeclado3 {
  public void lerDados() throws java.util.InputMismatchException {
      Scanner in = new Scanner(System.in);
      int numero = in.nextInt();
      System.out.println(numero);
      in.close();
  }
  public static void main(String[] args) {
    LeitorTeclado3  l = new LeitorTeclado3();
    l.lerDados();
  }
}
```

```
$ javac LeitorTeclado3.java
$ java LeitorTeclado3
1
1
$ java LeitorTeclado3
a
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 LeitorTeclado3.lerDados(LeitorTeclado3.java:6)
        at LeitorTeclado3.main(LeitorTeclado3.java:13)
$
```

throw new Exception()

```
import java.io.File;
import java.util.Scanner;
public class LeitorArquivo {
  private String caminhoArquivo;
  public LeitorArquivo(String pCaminhoArquivo) {
    caminhoArquivo = pCaminhoArquivo; }
  public void lerArquivo() throws java.lang.Exception {
    File f = new File(caminhoArquivo);
    if (f.exists() == false) {
     throw new Exception("Arquivo nao encontrado"); }
    Scanner sc = new Scanner(f);
    String dados = sc.next();
    System.out.println(dados); }
  public static void main(String[] args) {
    LeitorArquivo le = new LeitorArquivo("arquivo2.txt");
    try {
      le.lerArquivo();
    } catch (Exception e) {
      e.printStackTrace(); } } }
```

possibilita definir uma exceção personalizada

tratamento do erro

caso ocorra um erro na leitura do arquivo

usando um arquivo que não existe:

```
$ javac EscritorArquivo.java
$ javac LeitorArquivo.java
$ java LeitorArquivo
java.lang.Exception: Arquivo nao encontrado
        at LeitorArquivo.lerArquivo(LeitorArquivo.java:11)
        at LeitorArquivo.main(LeitorArquivo.java:19)
```

quando o arquivo existe:

```
$ javac LeitorArquivo.java
$ java LeitorArquivo
A
$
```

Em sistemas paralelos

Quadro 2.15 | Exceções geradas por um sistema paralelo

Exceções geradas por threads	Descrição
`IllegalArgumentException`	Quando é feito um `Thread.sleep` (valor), o valor deve estar entre 0 e 999999.
`InterruptedException`	Quando uma thread é interrompida. Esse processo pode ocorrer quando se tenta parar a thread.
`SecurityException`	É possível criar grupos de threads. Essa exceção é gerada quando a thread não pode ser inserida em um certo grupo.