Programação Orientada e Objetos II

Anhanguera


AVALIE
SUA PROFISSÃO
QUANDO APARECER EM SEU PORTAL UMA AVALIAÇÃO SOBRE SEU CURSO, RESPONDA:
NOTAS
9 OU 10
SIGNIFICA QUE VOCÊ INDICA
NOTAS
7 OU 8
SIGNIFICA QUE VOCÊ NÃO INDICA
Anhanguera

Anhanguera

Definição e tratamento de exceções para sistemas com threads

Os recursos para tratamento de exceções são muito importantes para produzir sistemas de qualidade e são mecanismos que a própria linguagem propicia para detectar o local em que um problema ocorreu. As formas de tratar esses erros geram blocos em que, caso ocorra alguma falha, a própria linguagem de programação informa quais linhas geraram o problema e qual problema foi gerado (DEITEL; DEITEL, 2016).

O processo de detecção de falhas em sistemas que possuem apenas uma linha de execução já é trabalhoso e se torna ainda mais complexo em códigos que possuem diversas linhas de execução (threads). Com isso, utilizar as diversas formas de tratamento de exceção que o Java possui é importante para detectar o problema. Imagine que o sistema em que você está trabalhando será apresentado em 15 minutos e, por um motivo desconhecido, ele para de funcionar. Com o sistema de tratamento de exceção é possível descobrir em qual linha o problema foi gerado e qual a mensagem de erro relacionada ao problema (FURGERI, 2015).

Em passos anteriores já nos deparamos com esse tratamento de exceção, por exemplo, no Quadro 2.8, que apresenta um código simples que lê números do teclado. A linha 1 importa a classe Scanner, que faz a leitura de diversos fluxos e que, no caso, estamos utilizando para ler o teclado (System.in) na linha 8, e na linha 9 é utilizado o comando int a = in.nextInt(); para ler números inteiros. Repare que o bloco try está entre as linha 7 e 12 e, em sequência, temos o bloco catch. O try determina que o código enclausurado pelas chaves será redirecionado para o bloco catch, caso alguma linha gere alguma exceção. O comando e.printStackTrace(); imprime na saída padrão do sistema a sequência que gerou a exceção (no caso, o console, mas essas mensagens devem ser direcionadas para um arquivo de texto central ou um banco de dados).

Anhanguera

```
1.  import java.util.Scanner;
2.
3.  public class LeitorTeclado {
4.
5.      public void lerDados()
6.      {
7.          try {
8.              Scanner in = new Scanner(System.
    in);
9.              int numero = in.nextInt();
10.             System.out.println(numero);
11.             in.close();
12.         }catch(Exception e)
13          {
14.             e.printStackTrace();
15.         }
16.     }
17.
18.     public static void main(String[] args) {
19.         LeitorTeclado l = new LeitorTeclado();
20.         l.lerDados();
21.     }
22. }
```

O código do Quadro 2.8 lê números do terminal e os imprime na tela. Todavia, caso o usuário insira um dado que não seja um número, o sistema gerará uma exceção. O Quadro 2.9 apresenta o resultado de uma execução de quando o usuário insere um texto, no caso, a palavra teste, e o sistema não está preparado para esse tipo de entrada. O erro gerado está especificado na linha 2 java.util. InputMismatchException que, traduzindo, significa Entrada incompatível. Esse erro é gerado pois o sistema espera um número inteiro e o usuário inseriu um texto. A linha 8, do Quadro 2.9, mostra o local que originou o erro, ou seja, na linha 20 do método main da classe LeitorTeclado (Quadro 2.8). A linha 7 (Quadro 2.8) indica o que causou o erro, ou seja, a linha 9, do método lerDados da classe LeitorTeclado (Quadro 2.9). Se você voltar ao código do Quadro 2.8, a linha 9 é exatamente onde a entrada do teclado é lida e atribuída à variável inteira numero. As demais linhas do Quadro 2.9 indicam os locais na classe Scanner que geram o erro. Nesse caso, o erro foi realmente gerado por uma entrada errada do usuário. Todavia, como é possível ver o código-fonte do Java, existe a chance de analisar o erro e verificar se o SDK está com defeitos (nainstalação do JDK é possível instalar o código-fonte).

```
1.  Teste
2.  java.util.InputMismatchException
3.       at java.util.Scanner.throwFor(Scanner.
    java:864)
4.       at java.util.Scanner.next(Scanner.java:1485)
```

```
5.       at java.util.Scanner.nextInt(Scanner.
    java:2117)
6.       at java.util.Scanner.nextInt(Scanner.
    java:2076)
7.       at LeitorTeclado.lerDados(LeitorTeclado.
    java:9)
8.       at LeitorTeclado.main(LeitorTeclado.java:20)
```

No Quadro 2.10 são descritos alguns tipos de exceção comuns que podem ocorrer em diversos cenários.

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.

Esse tipo de tratamento deve ser usado na maioria dos pontos de uma aplicação, porém, devemos nos atentar à execução das linhas. Quando ocorrer uma exceção, utilizaremos como exemplo uma implementação do método escreveArquivo() no Quadro2.11. Se uma falha for lançada na linha 11, as linhas 12 e 13 não serão executadas e a sequência irá para o bloco catch. Com isso, o recurso nunca será “fechado”, pois a instrução para fechar está na linha 13 e esta só será executada em uma sequência sem erros. Para evitar esse problema, é possível utilizar o bloco fi nally, que sempre executará, não importando se o método gerou erro ou se executou um return. O fi nally não executará somente se a aplicação for terminada (HORSTMANN, 2016). Assim, é possível fazer o tratamento final de qualquer evento, como nas linhas de 16 a 23. Veja que, mesmo dentro do fi nally, é necessário utilizar um try e um catch por imposição do método. Porém, isso garante que sempre se tentará executar o procedimento para finalizar ou tratar algo sem importar se houveram erros a serem tratados (WINDER, 2009).

Anhanguera

```
1.  import java.io.*;
2.
3.  public class ControleArquivos
4.  {
5.
6.      public void escreveArquivo(String
    caminhoArquivo)
7.      {
8.          FileWriter fw = null;
9.          File f = new File(caminhoArquivo);
10.         try {
11.             fw = new FileWriter(f);
12.             fw.write(10);
13.             fw.close();
14.         }catch (IOException e) {
15.             e.printStackTrace();
16.         }finally {
17.             if(fw != null)
18.                 try {
19.                     fw.close();
20.                 } catch (IOException e) {
21.                     e.printStackTrace();
22.                 }
23.             }
24.         }
25.     public static void main(String[] args) {
26.         ControleArquivos l = new
    ControleArquivos();
27          l.escreveArquivo("c:\\arquivoTeste.txt");
28      }
29  }
```

Com essa maneira de tratar os erros, acabamos por determinar uma regra:

1. No bloco try se executa as tarefas pertinentes à abertura de recursos necessários (arquivos, conexões de rede e outros).
2. No catch se avalia o que ocorreu e se cria logs para informar qual é o problema;
3. No fi nally se fecha os recursos utilizados, em alguns casos sendo necessários outros blocos para garantir que todos os
recursos sejam utilizados.

Com base nessas regras, os próprios desenvolvedores do Java criaram um mecanismo de tratamento para condicioná-las em um próprio comando da linguagem, a partir do Java 9.

Anhanguera

```
1.  public void escreveArquivo()
2.  {
3.          f = new File(caminhoArquivo);
4.          try(FileWriter fw = new
5.  FileWriter(f);)
6.           {
7.                  fw.write(10);
8.           }catch (IOException e) {
9.                 e.printStackTrace();
10.         }
    }
```

Existem diversas classes que implementam a interface Closeable e que podem aproveitar essa nova abordagem de tratamento de exceção, tais como:

- ServerSocket: classe que controla os sockets da camada transmission control protocol (TCP).
- DatagramSocket: classe que controla o envio de pacote pelo protocolo user datagram protocol (UDP).
- Connection: controla as conexões com os sistemas de gerenciamento de banco de dados.

Tratamento personalizado de exceções

Existem outras formas de fazer o tratamento de exceção relacionado à centralização e ao controle do local onde os tratamentos de erros são implementados.

Alguns métodos, quando utilizados, obrigam a implementaçãodo tratamento de erros. Isso ocorre porque esses métodos utilizam a cláusula throws, recurso que faz com que o método “lance” uma exceção ou mais, que serão especificadas pelo programador, caso ocorra algum problema.

No exemplo do Quadro 2.13 é apresentada uma classe que faz a leitura de um arquivo. Nesse tipo de situação, podem ocorrer erros ao ler o arquivo, por exemplo, ele pode não ser encontrado. Para evitar que o programa pare sua execução, na linha 12, o método escreveArquivo() foi escrito de modo a lançar uma exceção do tipo IOExeption, ou seja, ele invocará a classe = usada para esse tipo específico de erro. A classe só será invocada caso na linha 23 ocorra um erro, que será capturado na linha 24, especificamente pela classe usada no throws. Implementar dessa forma permite centralizar o tratamento das exceções, além de personalizar cada ação.

Anhanguera

```java
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(10);

        if(fw != null)
            fw.close();
    }

    public static void main(String[] args) {
        EscritorArquivo s = new EscritorArquivo("dados1/arquivo.txt");
        try {
            s.escreveArquivo();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Outro ponto interessante sobre o uso da cláusula throws é a possibilidade de o programador criar suas próprias exceções. Isso é feito por meio do comando throw new Exception(). Veja que, no código do Quadro 2.14, linha 9, o método lerArquivo() lança uma exceção da classe Exception. Caso ocorra um erro na leitura do arquivo, será exibida a mensagem Arquivo não encontrado na linha 22. Isso acontece porque o erro será tratado na linha 14, pelo comando throw new Exception(“Arquivo nao encontrado”);.

Anhanguera

```
1.  import java.io.File;
2.  import java.util.Scanner;

3.  public class LeitorArquivo {

4.      private String caminhoArquivo;

5.      public LeitorArquivo(String pCaminhoArquivo)
6.      {
7.          caminhoArquivo = pCaminhoArquivo;
8.      }
9.      public void lerArquivo() throws
10. java.lang.Exception
11.     {
12.         File f = new File(caminhoArquivo);
13.         if(f.exists() == false)
14.         {
15.             throw new Exception("Arquivo nao
16. encontrado");
17.         }
18.         Scanner sc = new Scanner(f);
            String dados = sc.next();
19.         System.out.println(dados);
        }
20.     public static void main(String[] args) {

21.         LeitorArquivo le = new
22. LeitorArquivo("dados1/arquivo.txt");
23.         try {
24.             le.lerArquivo();
25.         } catch (Exception e) {
26.             e.printStackTrace();
            }
        }
    }
```

Em sistemas paralelos temos exceções que devem ser tratadas devido à exceção de diversas threads. Veja no Quadro 2.15 algumas exceções que poderão ocorrer caso uma thread seja interrompida ou receba parâmetros incorretos.

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.

Portanto, o sistema de tratamento de erros é essencial para garantir o funcionamento caso algum evento ocorra de forma incorreta, além de assegurar que o sistema terá maiores chances de manutenção, evitando problemas maiores em caso de falhas ou problemas.

O tratamento de exceção é responsável pelo controle de fluxos alternativos que um software pode ter. Certos eventos podem ser considerados como errados e devem ser tratados para que o caminho principal do sistema seja executado. Para isso, a linguagem Java utiliza algumas palavras reservadas para fazer esse controle.
Quais opções representam palavras reservadas do tratamento de exceção do Java?

a) try, catch e error.
b) try, catch e fi nally.
c) throw, throws e bit.
d) int, main e go.
e) try, catch e throwt.

O Java possui algumas formas para o tratamento de uma exceção. Esses métodos ou construtores são modelados para enviar uma exceção que pode ocorrer, como uma conexão via rede que não está disponível ou ainda valores fora do intervalo permitido.

Qual das palavras reservadas do Java força o tratamento de exceção em método ou construtor?

a) throw.
b) throws.
c) try.
d) catch.
e) error.