5

TRATAMENTO DE EXCEÇÕES

Paradigmas de Programação

LEI - ISEP

Luiz Faria, adaptado de Donald W. Smith (TechNeTrain.com)

Objetivos

- Lançar (*throw*) e capturar (*catch*) exceções
- Implementar programas que propaguem exceções do tipo *checked exceptions*

2

Conteúdos

- Como tratar exceções

3

Exceções - Enquadramento

- Anteriormente já fazíamos tratamento de exceções:
  - Se um ficheiro de entrada para um Scanner não existe, ocorre uma FileNotFoundException quando o objeto Scanner é construído
  - O construtor PrintWriter pode gerar esta exceção se não conseguir abrir o ficheiro para escrita
    - Se o nome do ficheiro é ilegal ou se o utilizador não possui permissões para criar um ficheiro

4

Exceções - Enquadramento

- Adicionávamos duas palavras a qualquer método que use File I/O

```
public static void main(String[] args) throws
   FileNotFoundException
```

  - Até aprendermos como efetuar o nosso próprio tratamento de exceções

5

Package `java.io`

- Os diferentes tipos de exceções estão definidos no package `java.io`
  - Colocar as diretivas `import` no início do ficheiro de código que fará uso de File I/O e de exceções

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class LineNumberer
{
   public void openFile() throws FileNotFoundException
   {
      . . .
   }
}
```

6

Exemplo: Total.java (1)

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.io.PrintWriter;
4  import java.util.Scanner;
5
6  /**
7     This program reads a file with numbers, and writes the numbers to another
8     file, lined up in a column and followed by their total.
9  */
10 public class Total
11 {
12    public static void main(String[] args) throws FileNotFoundException
13    {
14       // Prompt for the input and output file names
15
16       Scanner console = new Scanner(System.in);
17       System.out.print("Input file: ");
18       String inputFileName = console.next();
19       System.out.print("Output file: ");
20       String outputFileName = console.next();
21
22       // Construct the Scanner and PrintWriter objects for reading and writing
23
24       File inputFile = new File(inputFileName);
25       Scanner in = new Scanner(inputFile);
26       PrintWriter out = new PrintWriter(outputFileName);
```

A cláusula throws

7

Exemplo: Total.java (2)

```
28       // Read the input and write the output
29
30       double total = 0;
31
32       while (in.hasNextDouble())
33       {
34          double value = in.nextDouble();
35          out.printf("%15.2f\n", value);
36          total = total + value;
37       }
38
39       out.printf("Total: %8.2f\n", total);
40
41       in.close();
42       out.close();
43    }
44 }
```

8

Tratamento de Exceções

- Existem dois aspetos a ter em conta para lidar com erros de *run-time*:
  1) Detecção de Erros
     Perante uma situação de erro podemos 'lançar' (throw) uma exceção

     Usar a instrução throw para assinalar uma exceção

```
if (amount > balance)
{
  // O que fazer?
}
```

  2) Tratamento de Erros
     Tarefa mais complexa – é necessário 'capturar' (*catch*) cada exceção possível e reagir a ela de forma apropriada
- O tratamento de erros recuperáveis pode ser feito:
  - Simplesmente encerrando o programa
  - De forma amigável: corrigindo o erro

9

Sintaxe para lançamento (*Throwing*) de uma Exceção

- Quando se lança uma exceção, estamos a lançar um objeto de uma classe de exceção
  - Escolher a classe de exceção de forma apropriada
  - Podemos passar uma String descritiva para a maioria dos objetos de exceção

A maioria dos objetos de exceção podem ser construídos com uma mensagem de erro

É construído um objeto de exceção e lançado de seguida

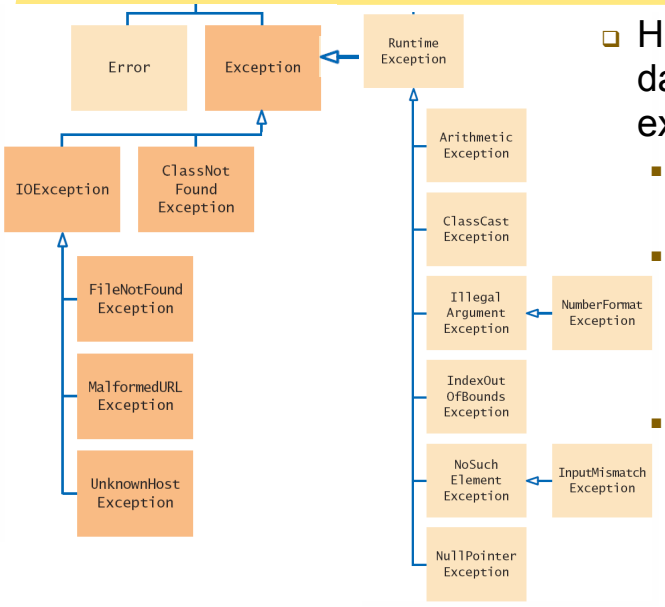
```
if (amount > balance)
{
   throw new IllegalArgumentException("Amount exceeds balance");
}
balance = balance - amount;
```

Esta instrução não será executada quando a exceção é lançada

Quando uma exceção é lançada, o fluxo normal do programa é terminado

10

Classes de Exceção



- Hierarquia parcial das classes de exceção
  - Em cima as mais genéricas
  - Em baixo as mais específicas
  - As mais escuras são *Checked exceptions*

11

Captura (*Catching*) de Exceções

- As exceções que são lançadas devem ser capturadas 'caught' algures no programa

Colocar no interior de um bloco 'try' as chamadas a métodos que podem lançar exceções

```
try
{
   String filename = . . .;
   Scanner in = new Scanner(new File(filename));
   String input = in.next();
   int value = Integer.parseInt(input);
   . . .
}
catch (IOException exception)
{
   exception.printStackTrace();
}
catch (NumberFormatException exception)
{
   System.out.println("Input was not a number");
}
```

FileNotFoundException

NoSuchElementException

NumberFormatException

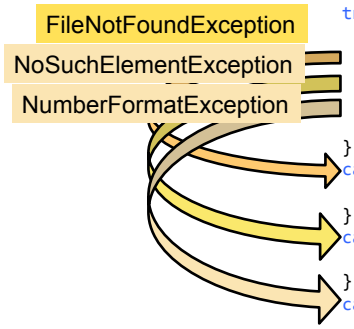
Definir blocos 'catch' para cada possível exceção

No bloco 'catch' é usual nomear o parâmetro de exceção como 'e' ou 'exception'

12

Captura de Exceções

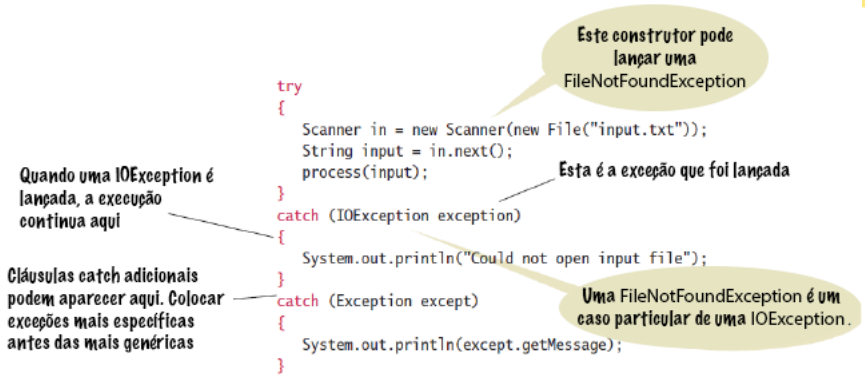
- Quando uma exceção é detectada, a execução 'salta' imediatamente para a primeira instrução do bloco `catch` correspondente
  - `IOException` tem correspondência com `FileNotFoundException` (`FileNotFoundException` é descendente de `IOException`)




```
try {
    String filename = ...;
    Scanner in = new Scanner(new File(filename));
    String input = in.next();
    int value = Integer.parseInt(input);
    ...
}
catch (IOException exception) {
    exception.printStackTrace();
}
catch (NoSuchElementException exception) {
    System.out.println("File contents invalid.");
}
catch (NumberFormatException exception) {
    System.out.println("Input was not a number");
}
```

13

Sintaxe da Captura de Exceções



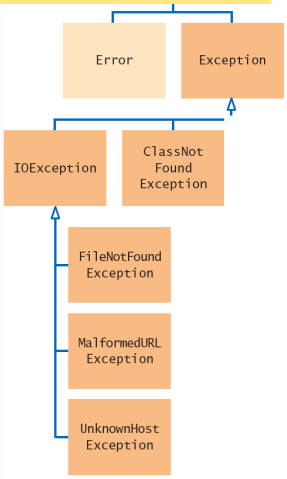
- Algumas alternativas para o tratamento de exceções:
  - Apenas informar o utilizador acerca do problema
  - Dar ao utilizador nova oportunidade para corrigir um erro de entrada
  - Imprimir uma '*stack trace*' mostrando a sequência de métodos invocados

```
exception.printStackTrace();
```

14

Checked Exceptions

- Throw/catch aplica-se a três tipos de exceções:
  - ***Error***: Erros internos (ex: OutOfMemoryError)
    - Raras, não consideradas aqui
  - ***Unchecked***: Descendentes de RuntimeException
    - Devidas a erros no programa (ex: IndexOutOfBoundsException)
    - O compilador **não verifica** se as tratamos
  - ***Checked***: Todas as outras exceções
    - Não são da responsabilidade do programador (ex: IOException)
    - O compilador **verifica** (***checks***) se as tratamos; se não forem tratadas o programa não compila
    - Representadas a escuro no diagrama ao lado



Checked exceptions devem-se a circunstâncias que o programador não pode evitar

15

Sintaxe da Cláusula `throws`

- Em alternativa ao tratamento local de *checked exceptions*, é possível transmitir a exceção ao método que invocou o método onde é detectada a exceção
- Os métodos que poderão gerar estas exceções e que não as tratem devem ser assim declarados:

```
public static String readData(String filename)
    throws FileNotFoundException, NumberFormatException
```

É necessário especificar todas as *checcked exceptions* que este método possa lançar

É possível incluir *unchecked exceptions*

  - Declarar todas as ***checked*** *exceptions* que o método pode lançar
  - É possível também indicar ***unchecked*** *exceptions*

16

Sintaxe da Cláusula `throws` (cont.)

- Se um método trata internamente uma *checked exception*, não precisa de lançar (*throw*) a exceção
  - O método não necessita de declarar a cláusula `throws`
- Declarando exceções na cláusula `throws` "passa a tarefa" de tratamento ao método que o chama ou a outra ainda acima

17

Cláusula `finally`

- `finally` é uma cláusula opcional num bloco `try/catch`
  - Usada quando é necessário tomar alguma ação num método, independentemente da exceção ser lançada
    - O bloco *finally* é executado em ambos os casos

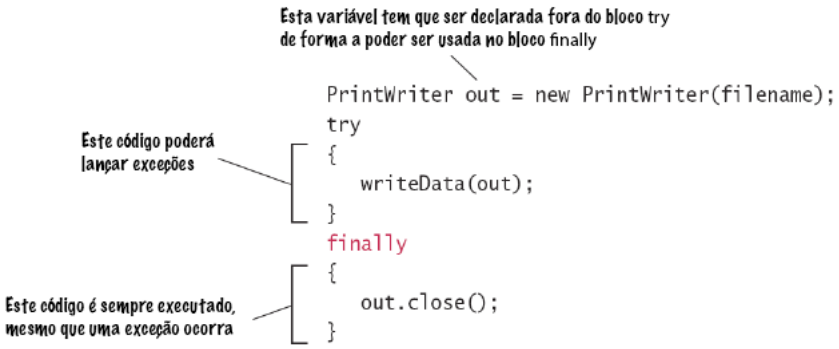
```
public void printOutput(String filename) throws IOException
{
   PrintWriter out = new PrintWriter(filename);
   try
   {
      writeData(out);    // O método pode lançar uma exceção I/O
   }
   finally
   {
      out.close();
   }
}
```

Assim que o bloco `try` é alcançado, as instruções numa cláusula `finally` são executas, quer a exceção seja ou não lançada

18

Sintaxe da Cláusula `finally`

- O código contido no bloco `finally` é sempre executado assim que o bloco `try` seja alcançado



19

Como usar `try/catch`

- Lançar cedo
  - Quando um método detecta um problema que não pode resolver, é melhor lançar uma exceção em vez de tentar efectuar uma correção imperfeita
- Capturar tarde
  - Pelo contrário, um método deve apenas capturar uma exceção se o método pode realmente remediar a situação
  - Caso contrário, o melhor remédio é simplesmente propagar a exceção ao método anterior (origem da chamada), permitindo o tratamento da exceção de forma adequada

20

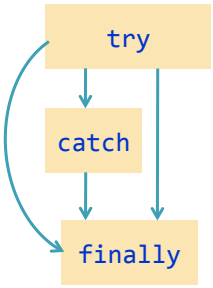
Como usar `try/catch`

- Não "silenciar" exceções
  - Quando se invoca um método que lança uma *checked exception* e não foi especificado um *handler*, o compilador gera um erro
  - Poderá ser tentador escrever um bloco catch vazio para 'silenciar' o compilador e voltar ao código mais tarde - **Má prática**
    - As exceções foram criadas para reportar problemas a um *handler* competente
    - Construir um *handler* incompetente apenas esconde uma situação de erro que poderá vir a ser grave

21

Como usar `try/catch`

- Não usar `catch` e `finally` no mesmo bloco `try`
  - A cláusula `finally` é executada sempre que o bloco *try* é terminado por uma das seguintes 3 formas:
    1. Após a execução da última instrução contida no bloco `try`
    2. Após a execução da última instrução de uma cláusula `catch`, se este bloco *try* capturou uma exceção
    3. Quando uma exceção foi lançada no bloco `try` e não foi capturada

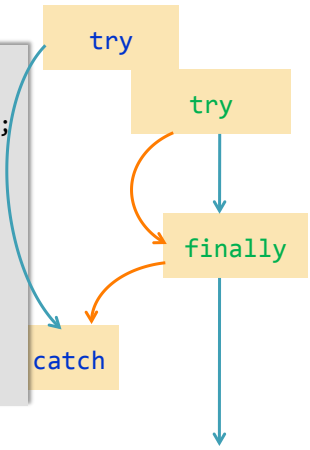


22

Como usar `try/catch` (cont.)

- Será melhor usar duas (*nested*) cláusulas *try* para controlar o fluxo

```
try
{
  PrintWriter out = new PrintWriter(filename);
  try
  {      // Escrever saída    }
  finally
  {  out.close(); }  // Fechar recursos
}
catch (IOException exception)
{
  // Tratar a exceção
}
```



23

Sumário: Exceções (1)

- Para assinalar uma condição excecional, usar a instrução `throw` para lançar um objeto de exceção
- Quando uma exceção é lançada, o processamento continua com o tratamento da exceção
- Colocar as instruções que poderão causar uma exceção dentro de um bloco `try` e o *handler* dentro de uma cláusula `catch`
- As *checked exceptions* devem-se a circunstâncias externas que o programador não pode evitar
  - O compilador verifica se o programa trata estas exceções

24

Sumário: Exceções (2)

- Adicionar uma cláusula `throws` a um método que pode lançar `uma` *checked exception*
- Sempre que se entra num bloco `try`, as instruções numa cláusula `finally` são sempre executadas, independentemente de uma exceção ser lançada ou não
- Lançar uma exceção assim que o problema seja detetado
- Capturar a exceção apenas quando o problema poder ser resolvido
- Durante a concepção de um programa, identificar que tipos de exceções poderão ocorrer
- Para cada exceção, é necessário decidir qual a parte do programa que terá competência para a tratar

25