Java

Tratamento de Exceções

Exceções

- Noção
  - São acontecimentos
  - Ocorrem
    - Em tempo de execução do programa // runtime
- Provocados
  - Por erros de execução do programa

Exceções – Exemplo1 (1)

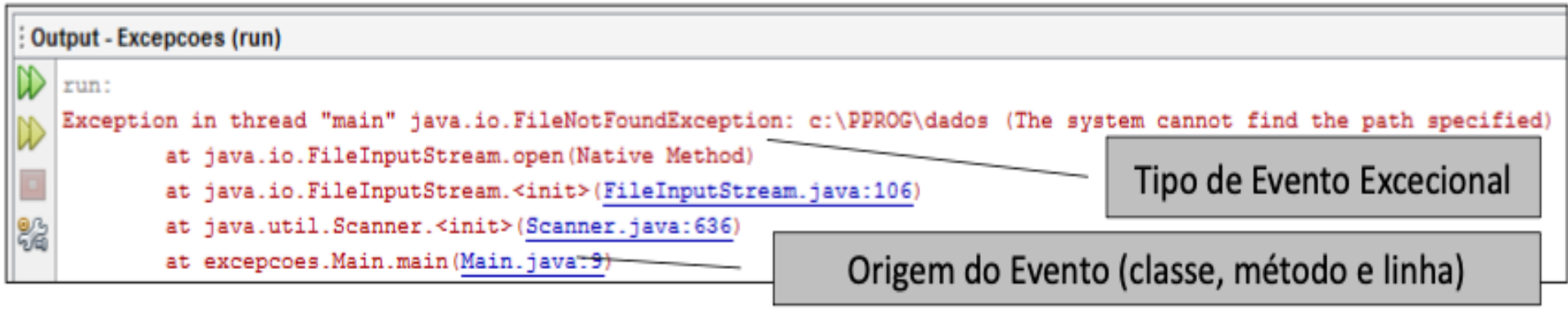
- Abertura ficheiro inexistente

```
7   public class Main {
8       public static void main(String[] args) throws FileNotFoundException {
9           Scanner ficheiroDeDados = new Scanner(new File("c:/PPROG/Dados"));
10      }
11  }
```

- Durante a execução:
  - Ficheiro inexistente ⇒ Erro de execução ⇒
    Evento excecional FileNotFoundException ⇒
    Programa termina abruptamente

Exceções – Exemplo1 (2)

- Na saída:
  - Tipo de evento excecional
  - Origem do evento
  - Sequência de métodos envolvidos (pela ordem inversa)



Exceções – Exemplo2 (1)

- Índice de array fora dos limites

```
5   public class Main {
6       public static void main(String[] args) {
7           Scanner ler = new Scanner(System.in);
8
9           String[] nomes = new String[2];
10          for (int i = 0; i <= nomes.length; i++) {
11              System.out.println("Indique um nome:");
12              nomes[i] = ler.nextLine();
13          }
14      }
15  }
```

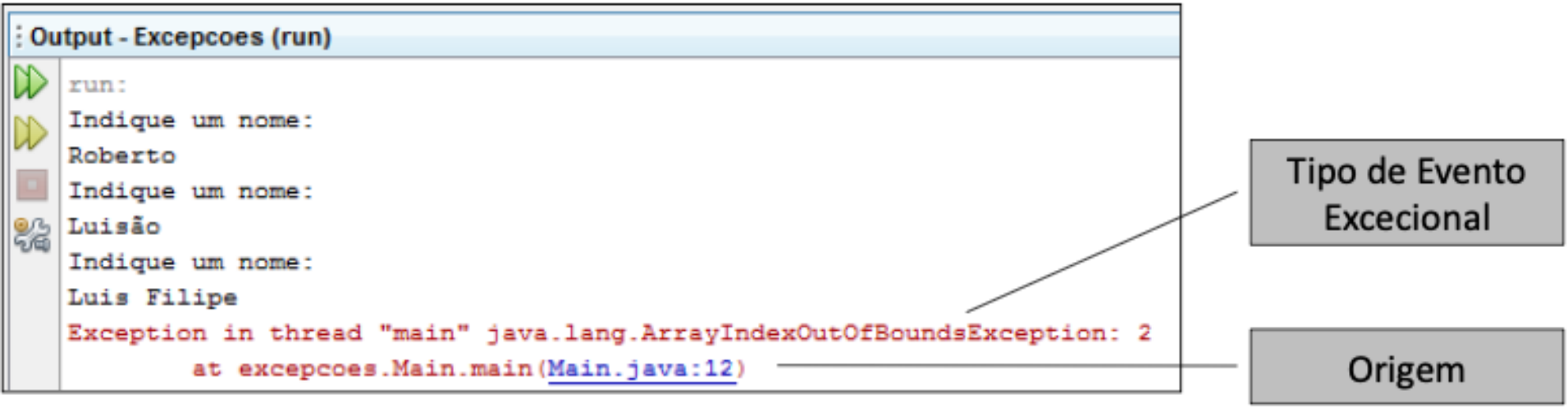
- Durante a execução:
  - Índice fora dos limites ⇒ Erro de execução ⇒

    Evento excecional ArrayIndexOutOfBoundsException ⇒

    Programa termina abruptamente

Exceções – Exemplo2 (2)

- Na saída:
  - Tipo de evento excecional
  - Origem do evento
  - Sequência de métodos envolvidos (pela ordem inversa)



Eventos excecionais

- São indesejados
  - Provocam terminação abrupta do programa
    - Podem provocar perda de trabalho
- Causas
  - Erros de programação
    - Outro exemplo: Divisão por zero
- Podem ser evitados

Eventos excecionais

- Circunstâncias externas ao programa
  - Erros nos dados de entrada fornecidos **// dados inválidos**
    - Utilizador
    - Ficheiros **// ficheiro inexistente**
    - Rede
  - Limitações físicas do hardware **// disco/memória cheio**
  - Erros de dispositivos **// falha rede, erro impressora, …**

- Conclusão
  - **Não é possível** evitar completamente…

    Eventos Excecionais!

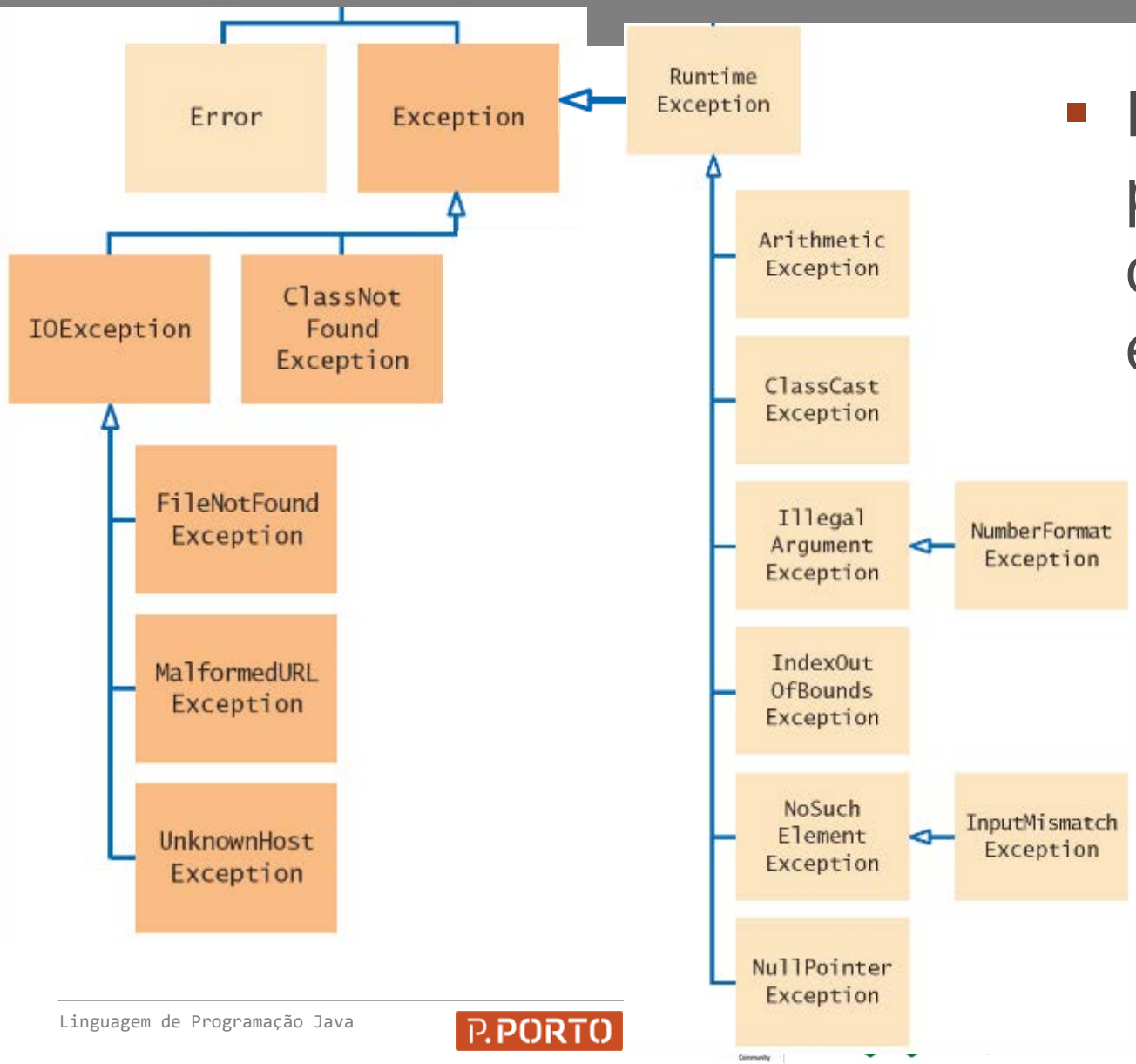
Package java.io

- 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
   {
      . . .
   }
}
```

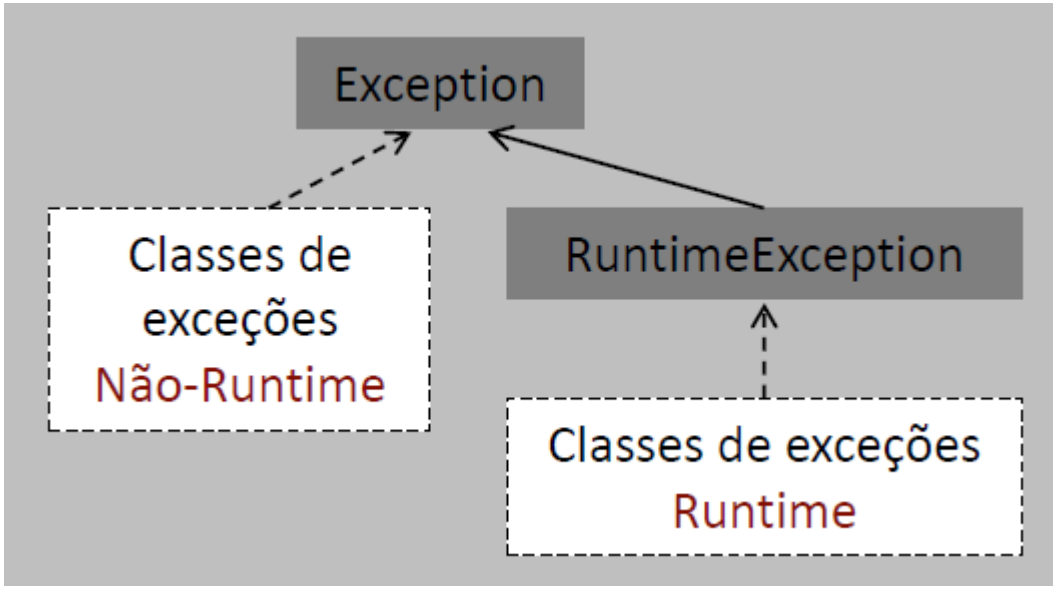
Classes das Exceções Nativas



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

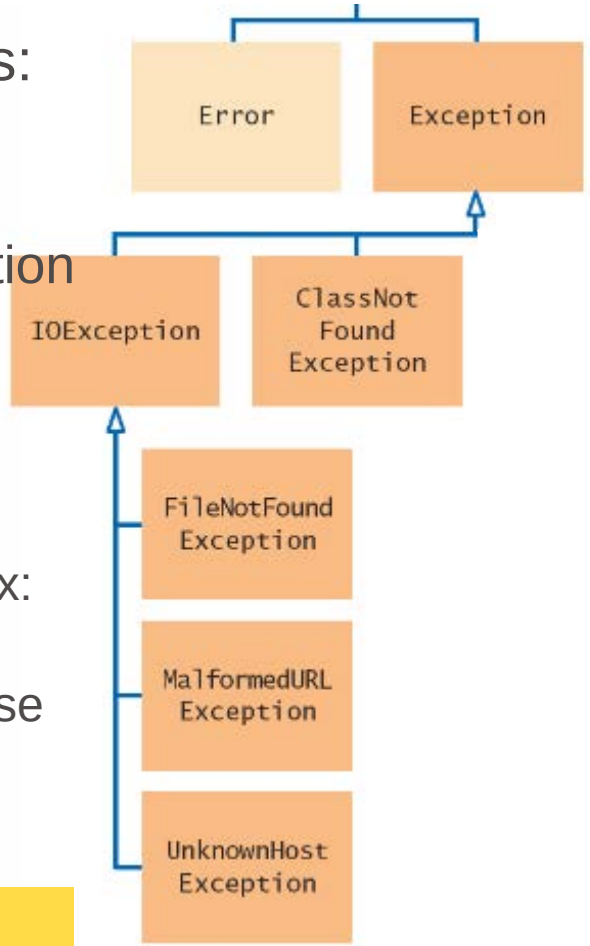
Tipos de Classes de Exceções

- Runtime (*unchecked*)
- Não runtime (*checked*)



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

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*

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

Cláusula finally

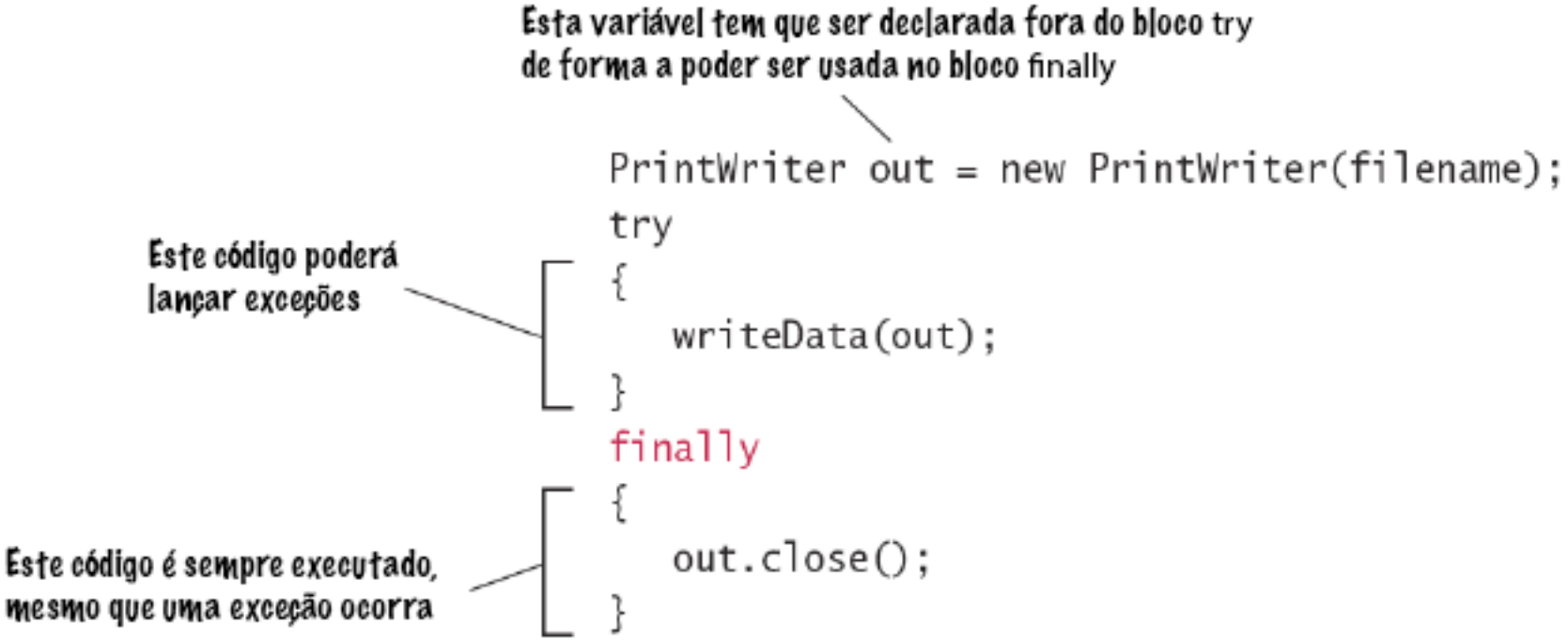
- `finally` é uma cláusula opcional num bloco `try/catch`
  - Usada quando é necessário tomar alguma ação num método, independentemente da exceção ser lançada
    - O bloco *finally* é executado em ambos os casos

```
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

Sintaxe da Cláusula finally

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




```
PrintWriter out = new PrintWriter(filename);
try
{
   writeData(out);
}
finally
{
   out.close();
}
```

Como usar try/catch

- Lançar cedo
  - Quando um método deteta um problema que não pode resolver, é melhor lançar uma exceção em vez de tentar efetuar 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

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

Como usar try/catch

- **try {...}**
  - Define
    - Bloco de código suscetível de gerar erros de execução
- **catch(Tipo_de_exceção parâmetro){...}**
  - Define
    - Tipo de exceção a capturar
    - Parâmetro recebe a exceção capturada
    - Bloco de código para tratar esse tipo de exceção
- **finally {...}**
  - Opcional
  - Complementa try
    - Bloco de código suscetível de gerar erros de execução
    - Usado sempre que try seja executado

Como usar try/catch

- Um bloco try, múltiplos catch e, opcionalmente, um finally // por esta ordem

```
try {
      // código susceptível de gerar erros de execução
} catch ( Tipo_exceção_1  parâmetro ) {
      // código para tratar qualquer Tipo_exceção1 ou subtipo
}
 ...
} catch ( Tipo_exceção_N  parâmetro ) {
      // código para tratar qualquer Tipo_exceçãoN ou subtipo
} finally {
      // código executado quando try é usado
}
```

1 try

Múltiplos catch

- Para diferenciar tratamentos
- Exceção ocorrida em try é capturada pelo primeiro bloco catch do seu tipo ou de uma sua superclasse
- Primeiros mais específicos
- Últimos mais genéricos
- Tipo Exception captura tudo (deve ser o último)

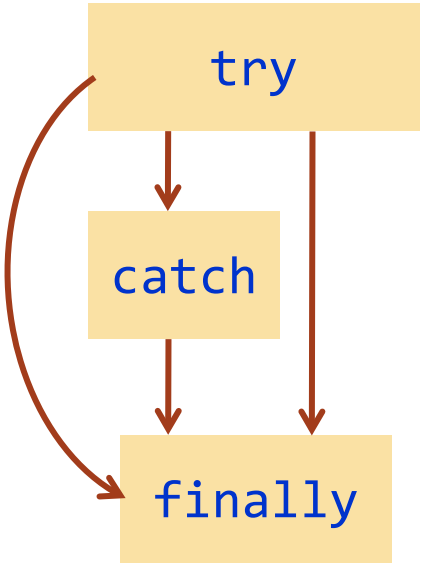
1 finally

- Opcional

- **Torna programa legível**
  - Separa claramente código de exceções do código principal

Como usar try/catch

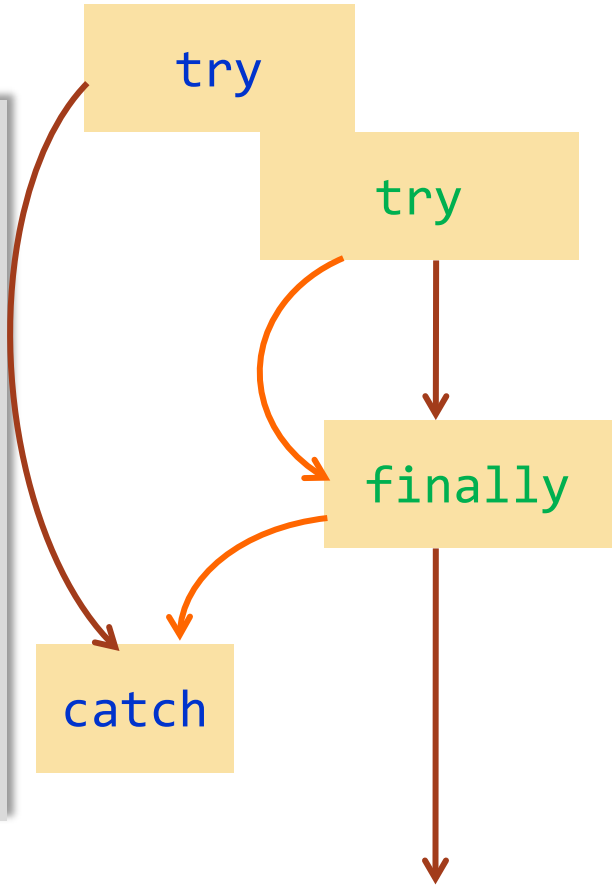
- Não usar `catch` e `finally` no mesmo bloco `try`
  - A cláusula `finally` é executada sempre que o bloco *try* é terminado por uma das seguintes 3 formas:
    1. Após a execução da última instrução contida no bloco `try`
    2. Após a execução da última instrução de uma cláusula `catch`, se este bloco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



Como usar try/catch

- 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
}
```



Captura e Tratamento – Exemplo 1

- Leitura de dados do utilizador simples e mensagem de erro com informação **não enviada** pela exceção

```
6   public class Main {
7       public static void main(String[] args) {
8           Scanner ler = new Scanner(System.in);
9           boolean invalido = true;
10          do {
11              try {
12                  System.out.println("Indique um número inteiro:");
13                  int num = Integer.parseInt(ler.next());
14                  invalido = false;
15              } catch (Exception e) {
16                  System.out.println("Número Inválido!!");
17              }
18          } while (invalido);
19      }
20  }
21
```

Tipo Exception captura qualquer exceção

```
Output - Excepcoes (run)
run:
Indique um número inteiro:
1a
Número inválido !!
Indique um número inteiro:
11
```

Captura e Tratamento – Exemplo 2

- Leitura de dados do utilizador simples e mensagem de erro com informação **enviada** pela exceção

```
6   public class Main {
7       public static void main(String[] args) {
8           Scanner ler = new Scanner(System.in);
9           boolean invalido = true;
10          do {
11              try {
12                  System.out.println("Indique um número inteiro:");
13                  int num = Integer.parseInt(ler.next());
14                  invalido = false;
15              } catch (Exception e) {
16                  System.out.printf("Exception %s %n",e.getMessage());
17              }
18          } while (invalido);
19      }
20  }
21
```

```
Output - Excepcoes (run)
run:
Indique um número inteiro:
1a
Exception For input string: "1a"
Indique um número inteiro:
1
```

mensagem de erro veiculada pela exceção capturada

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

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