AULA TEÓRICA 3

Tema 4. Excepções

- Tratamento de excepções

Excepções

Uma excepção é um evento que ocorre durante a execução de um programa que interfere no fluxo normal das instruções deste programa.

Em Java, a ocorrência de erros durante a execução de um programa não significa necessariamente que o programa termina.

A linguagem possui um mecanismo para indicar partes críticas num programa e recuperar eventuais erros ocorridas nestas partes, sem parar a execução do programa.

Este mecanismo é designado por **excepção** (*Exception*).

Ao executar um programa é frequente que ocorram situações inesperadas.

Por exemplo:

- um utilizador que escreve uma letra quando se espera um inteiro;
- um ficheiro que o programa necessita abrir não existe;
- tentar calcular a raiz quadrada de um número negativo;
- dividir pelo zero

Exemplo:

```
public class Zero {
  ...
   int a = 10,b = 0;
   System.out.println(a/b);
 ...
```

execução:

```
java.lang.ArithmeticException: / by zero at Zero.main(Zero.java:xx)
```

Onde xx: número da linha onde ocorreu o erro.

Erro e Excepção em Java

Erro

- Um **Erro** (*Error*) em Java corresponde a uma situação para a qual **nenhuma recuperação** é já possível.
- Descreve erros internos e a exaustão de recursos durante a execução do programa.
- Pouco se pode fazer se um erro interno desses ocorrer, além de notificar o utilizador e tentar finalizar o programa adequadamente.
- Essas situações são bastante raras.

Excepção

- Uma **Excepção** (*Exception*) corresponde a uma situação para a qual a recuperação **é possível**.
- É um sinal gerado (lançado) pela máquina virtual de Java em tempo de execução do programa, indicando a ocorrência de um erro recuperável.
- A captura e o tratamento de excepções contribui para a proclamada robustez do código dos programas Java.

Lançamento de excepção

✓Quando ocorre um erro recuperável dentro de um método, este cria um objecto da classe **`Exception`** e passa este objecto para o sistema de execução do Java (*runtime*) - **lança uma Excepção.**

✓Este objecto contém informações sobre a excepção (seu tipo e o estado do programa quando o erro ocorreu).

✓A partir deste momento, o sistema de execução do Java responsabiliza-se por encontrar o código que trate o erro ocorrido.

✓O sistema passa a procurar o código capaz de tratar a excepção.

✓A lista de “candidatos” para este tratamento vem da pilha de chamadas de métodos que antecederam o método que lançou a excepção.

✓O sistema de execução do Java “percorre” a pilha de chamadas, e começa com o próprio método onde ocorreu o erro, na busca de um método que possua um **gestor de Excepção (*`catch`*)** adequado.

Captura de Excepção

✓Se a excepção não for tratada e chegar à função `main`, o programa será interrompido com uma mensagem de erro.

✓Um “gestor de excepção” é considerado adequado quando a excepção que ele manipula é do mesmo tipo da excepção lançada.

✓Quando ele é encontrado, recebe o controle do programa para que possa tratar o erro ocorrido.

✓Em outras palavras, diz-se que ele “capturou” a excepção (*catch the exception*).

✓Se nenhum dos métodos pesquisados pelo sistema de execução possui um gestor de excepções adequado, então o programa Java em questão é inesperadamente encerrado.

Tratamento de Excepções

A linguagem Java permite a descrição de situações de excepção através da utilização de 5 palavras - chave correspondentes a cláusulas especiais, a saber:

```
try, catch, finally, throw, throws
```

Sintaxe:

```
try {
   //instruções que podem levar ao aparecimento de uma excepção
   . . . .
   } catch (Excepcao1 ex1) {
   // instruções a executar se ocorrer uma excepção de tipo Excepcao1
   . . . .
   } catch (Excepcao2 ex2) {
   // instruções a executar se ocorrer uma excepção de tipo Excepcao2
   . . . .
   } finally {
   //o bloco é opcional, se existe, executado sempre
   . . . .
   }
```

Em primeiro lugar, executam-se as instruções incluídas no bloco `try`. Se não for gerado qualquer erro a execução prossegue para instrução seguinte ao bloco `catch`. No caso de ocorrer um erro (excepção), os blocos `catch` são verificados, procurando-se um que se aplique ao tipo de excepção que surgiu. Se não houver, o programa irá parar com erro.

Um bloco `try` pode ser acompanhado por vários blocos `catch`, quando a(s) instrução(ões) incluída(s) no `try` pode(m) gerar mais do que um tipo de excepção e se pretender dar um tratamento diferenciado a cada uma dessas excepções.

O comando `finally` contém código a ser executado, independente de outros comandos - é opcional, mas quando presente, é sempre executado, após o término do bloco `try` ou de um `catch` qualquer.

A ordem em que as cláusulas `catch` aparecem, importa. Por esta razão, as excepções mais genéricas devem ser tratadas após as mais específicas.

`throw` é usado para lançar uma excepção em algum lugar do código.
Ex: `throw new IllegalArgumentException("Valor inválido");`

No caso de usar `try-catch` não é necessário indicar as palavras reservadas `throws IOException`.

Classificação das Excepções

RuntimeException

- Ocorre porque houve um erro de programação.
- Conversão explícita de tipo (*cast*)
- Acesso a elemento de uma tabela além dos limites.
- Acesso de ponteiro nulo.

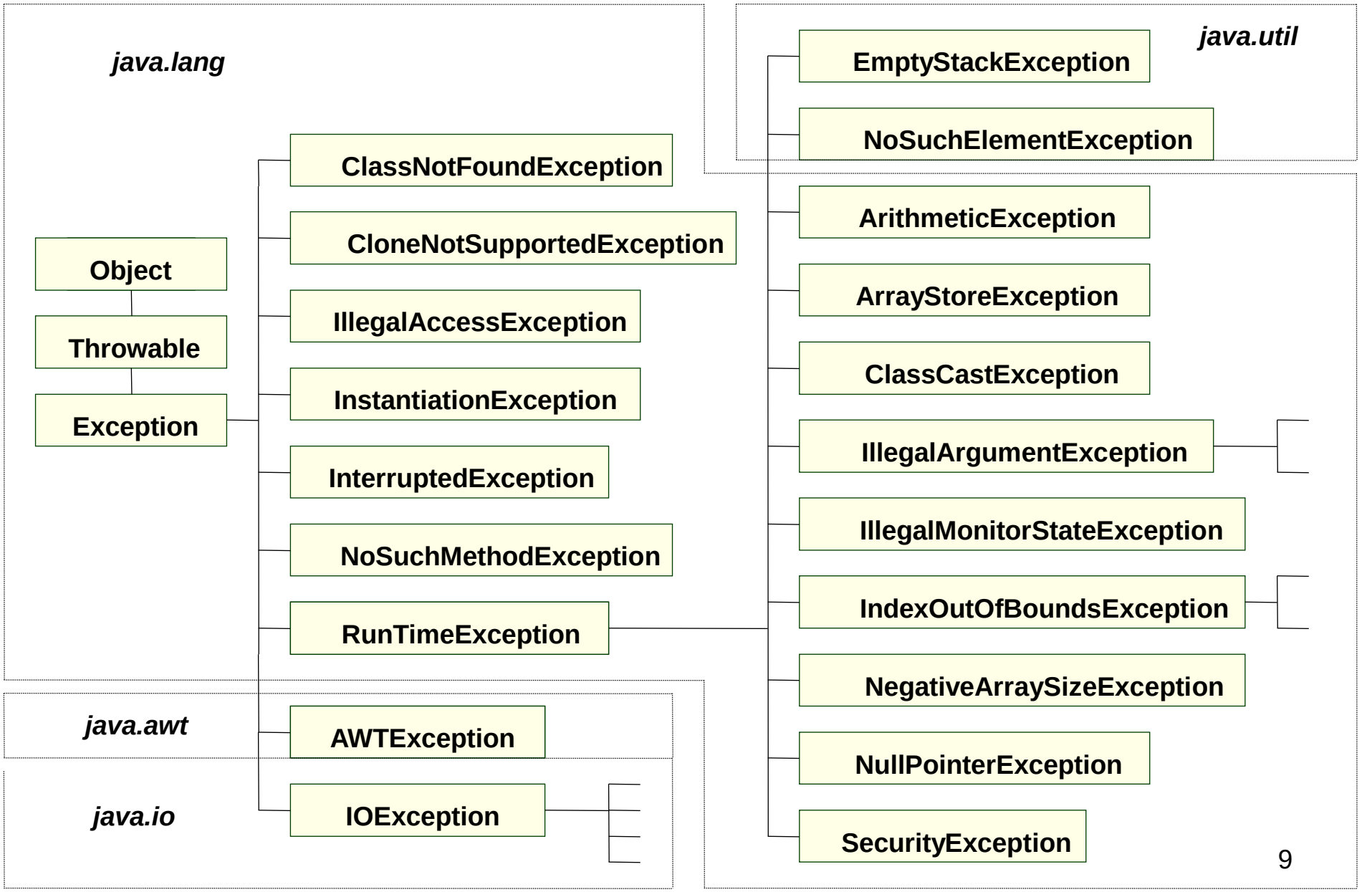
IOException

• Tentar ler além do final de um ficheiro

• Tentar abrir um URL incorrecto

•Tentar encontrar um objecto Class através de uma string que não denota uma classe existente.

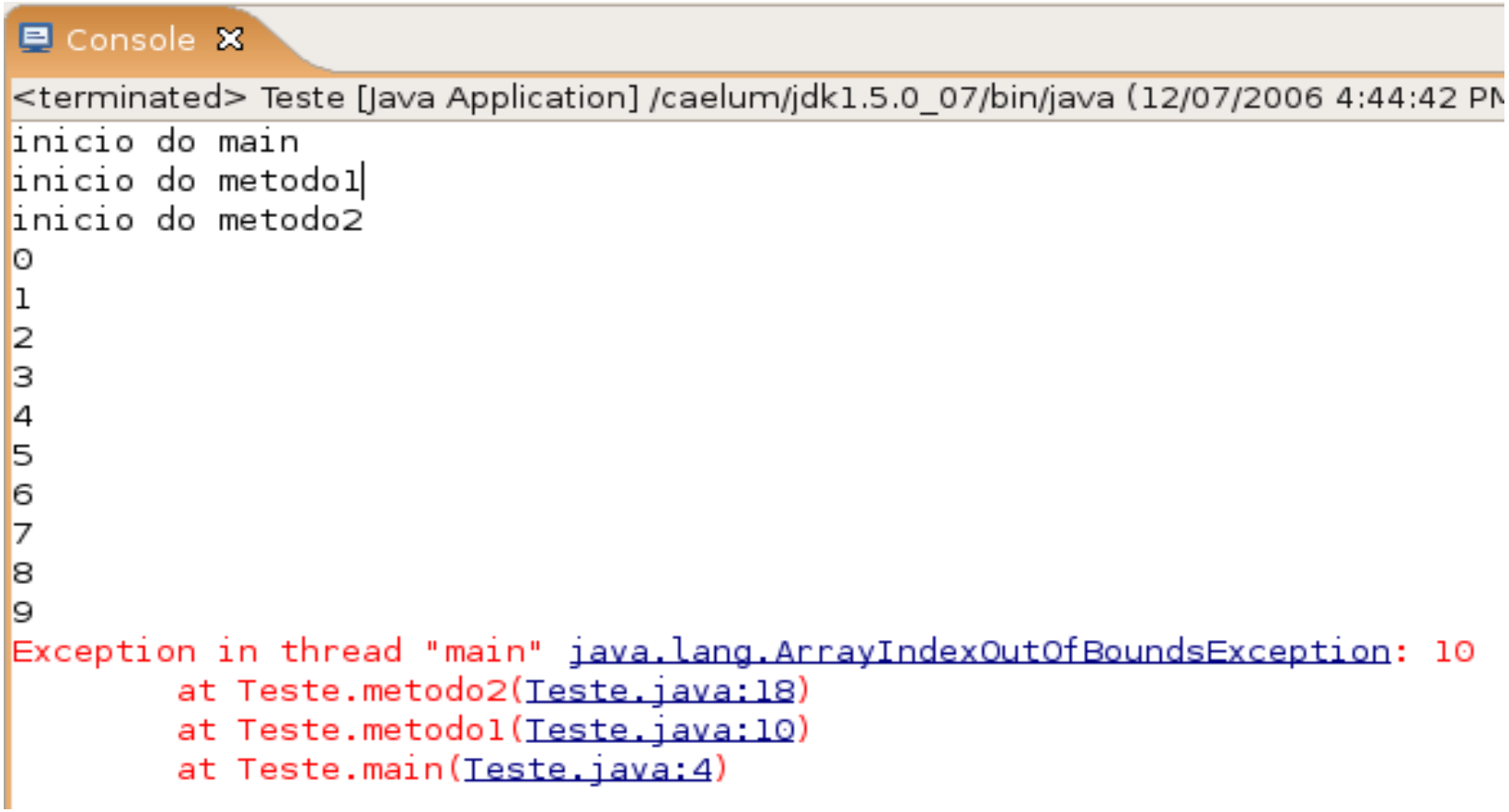
Classes de Excepção em Java



Exemplo (sem tratamento de excepção)

```
public class TesteErro {
  public static void main(String[] args) {
  System.out.println("inicio do main");
  metodo1();
  System.out.println("fim do main");
}
  public static void metodo1() {
    System.out.println("inicio do metodo1");
    metodo2();
    System.out.println("fim do metodo1");
  }
  public static void metodo2() {
   System.out.println("inicio do metodo2");
    int[] array = new int[10];
    for(int i = 0; i <= 15; i++) {
      array[i] = i;
      System.out.println(i);
    }
   System.out.println("fim do metodo2");
  }
}
```

O programa termina inesperadamente com seguinte erro:



Alguns exemplos de possíveis soluções:

a) . . .

```
try {
  for(int i = 0; i <= 15; i++) {
    array[i] = i;
    System.out.println(i);
  }
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("erro: " + e); }
```

b)
```
. . .
for(int i = 0; i <= 15; i++) {
  try {
    array[i] = i;
    System.out.println(i);
  } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("erro: " + e); }
}
```

c)
```
. . .
System.out.println("inicio do metodo1");
try {
  metodo2();
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("erro: " + e); }
System.out.println("fim do metodo1");
```

d)
```
. . .
System.out.println("inicio do main");
try {
  metodo1();
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Erro : "+e); }
System.out.println("fim do main");
```

Qual das variantes é a melhor?

A variante A é a melhor!

A partir do momento que uma excepção foi capturada, a execução volta ao normal a partir daquele ponto.

Existem dois tipos de excepções: ***unchecked*** e ***checked .***

1) Quando o Java não obriga a dar o `try/catch` chamamos tais excepções de ***unchecked*** (em outras palavras, o compilador não **checa** se você está tratando excepções).

2) Um outro tipo (***checked***), obriga a quem chama o método ou construtor a tratar o erro.

Existe uma grande tentação de sempre passar o erro para frente para “outros” tratarem dele.

Não há uma regra para decidir em que momento do seu programa você vai tratar determinada excepção. Isso vai depender de em que ponto você tem condições de tomar uma decisão em relação àquele erro.

Exemplo de validação de um inteiro lido do teclado:

```
public int validarInt(int a, int b, String str){
  int x = 0;
  do {
    System.out.println("Introduza " + str);
    try { //o método parseInt() pode gerar excepção
      x = Integer.parseInt(br.readLine());
    } catch (NumberFormatException f) {
       System.out.println("O valor introduzido nao é um inteiro!");
      }
      catch (IOException k) {
         System.out.println(k.printStackTrace());
      }
   if(x < a || x > b)
      System.out.println("Valor invalido! Tente novamente.");
  } while(x < a || x > b);
  return x;
}
```

Referência bibliográfica:

António José Mendes; Maria José Marcelino.
"Fundamentos de programação em Java 2". FCA. 2002.

Elliot Koffman; Ursula Wolz.
"Problem Solving with Java". 1999.

F. Mário Martins;
"Programação Orientada aos objectos em Java 2", FCA, 2000,

John Lewis, William Loftus;
"Java Software Solutions: foundation of program design", **2nd edition, Addision-Wesley**

John R. Hubbard.
"Theory and problems of programming with Java". Schaum's Outline series. McGraw-Hill.

H. Deitel; P. Deitel.
"Java, como programar". 4 edição. 2003. Bookman.

Rui Rossi dos Santos.
"*Programando em Java 2– Teoria e aplicações"*. Axcel Books. 2004

Apostilas CAELUM