Parte 7 – Ficheiros e Exceções

Fernando Barros, Karima Castro, Luís Cordeiro,
Marília Curado, Nuno Pimenta

Os conteúdos desta apresentação baseiam-se nos materiais produzidos por António José Mendes para a unidade curricular de Programação Orientada a Objetos.

Quaisquer erros introduzidos são da inteira responsabilidade dos autores.

Exceções

- No acesso a ficheiros podem ocorrer vários problemas
  - Ficheiro não existe
  - Não tem permissões para escrever no ficheiro
  - Falta de espaço para continuar a escrever no ficheiro
- É necessário gerir os erros que podem ocorrer
- O JAVA permite tratar exceções:
  - Gerar exceções
  - Propagar exceções
  - Tratar exceções

Tipos de exceções

- Exceções do JAVA: exceções ou erros gerados pela JVM ou por métodos da biblioteca do JAVA
  - NullPointerException, IOException, ArrayIndexOutOfBoundsException, ClassCastException, FileNotFound, etc.
- Exceções defindas pelo utilizador
  - Deve extender a class Exception (Throwable)
  - Útil para gerir situações de exceção ou erros próprios da aplicação

Tratamento de exceções

- **throws IOException** no cabeçalho de um método serve para indicar ao compilador que ele pode gerar ou propagar um erro do tipo IOException
- **throw** permite gerar uma exceção
- Bloco **try/catch** permite identificar e capturar exceções, indicando o procedimento a seguir caso a exceção ocorra

Tratamento de exceções

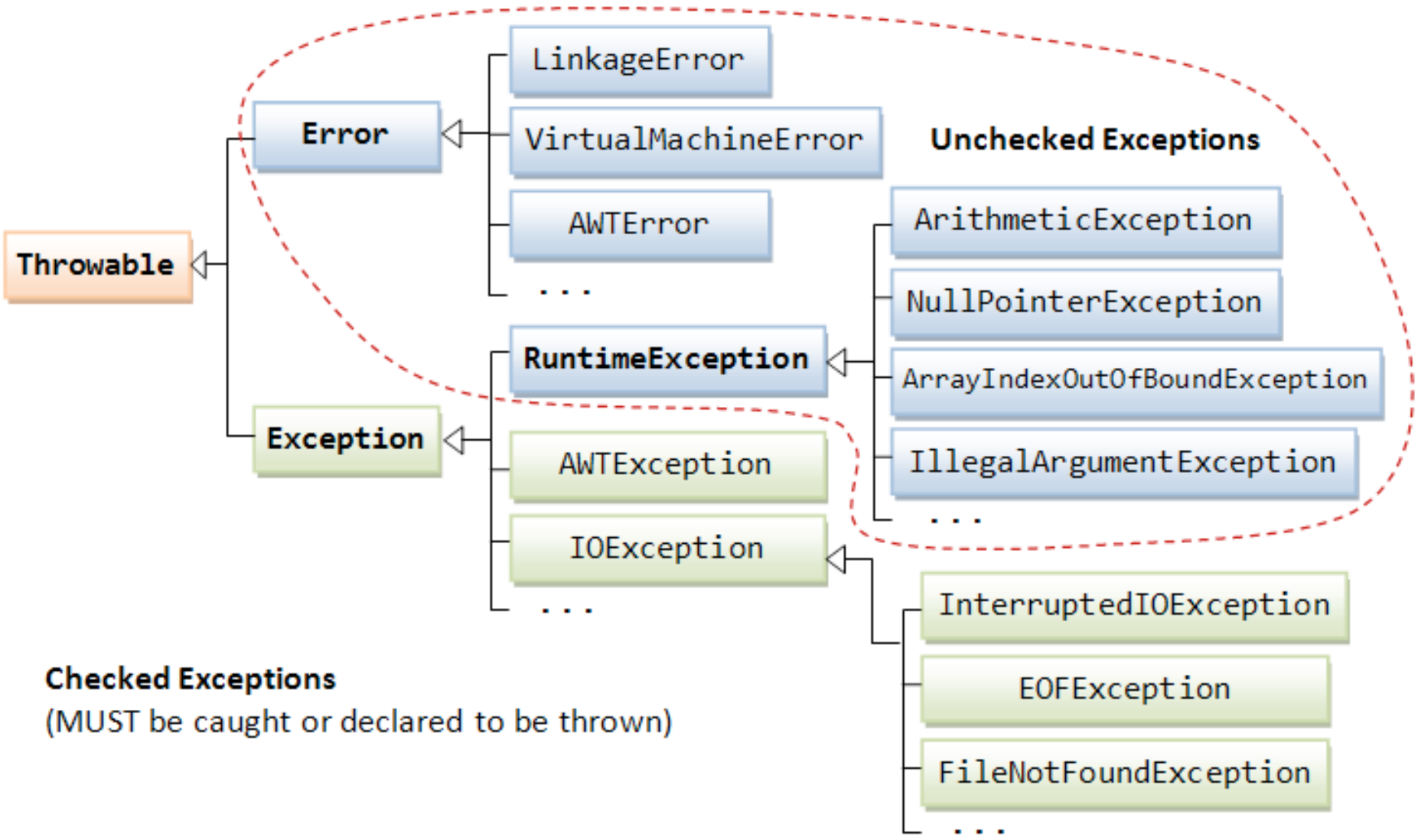
- Qualquer método que contenha instruções potencialmente geradoras de erros ou que chame métodos que possam propagar erros deve:
  - Declarar a possibilidade de propagar erros (através da inclusão de throws no seu cabeçalho)
  - Tratar as eventuais exceções, utilizando try/catch
- O tratamento de exceções deve ser feito em algum ponto do programa
  - O tratamento feito pelo sistema operativo consiste em terminar o programa abruptamente

Bloco try/catch

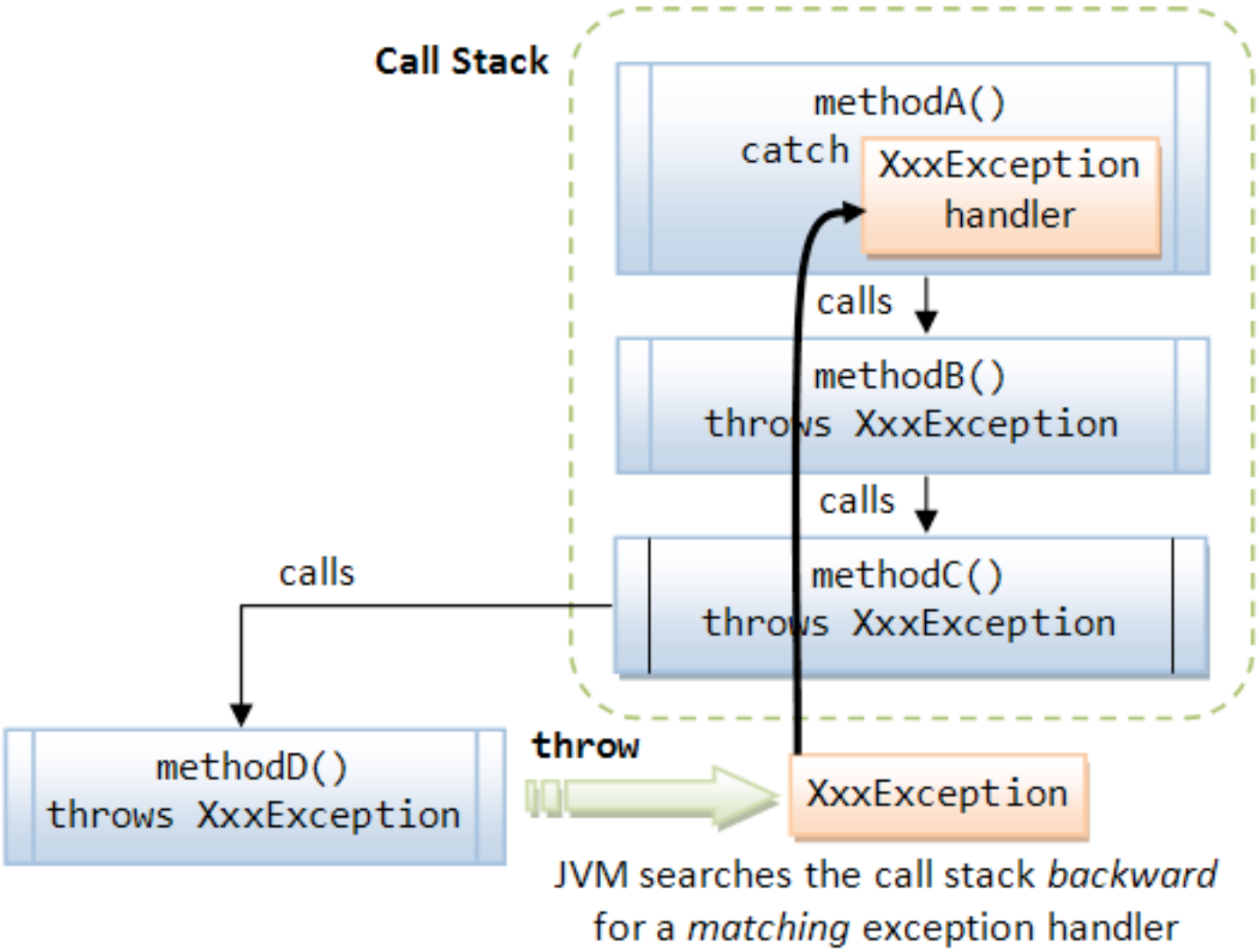
```
try {
   // Código que faz a ação desejada,
   // mas pode gerar um erro
} catch (ExceptionType1 e1) {
   // Instruções a executar se ocorrer
   // uma exceção do tipo ExceptionType1
} catch (ExceptionType2 e2) {
   // Instruções a executar se ocorrer
   // uma exceção do tipo ExceptionType2
}
```

Exemplo

```
Scanner sc = new Scanner(System.in);
try {
    System.out.print("Introduza o numero: ");
    String text = sc.nextLine();
    int num = Integer.parseInt(text);
    System.out.println("Numero: " + num);
} catch(NumberFormatException e) {
    System.out.println("Erro a converter texto em inteiro.");
}
```



https://www3.ntu.edu.sg/home/ehchua/programming/java/J5a_ExceptionAssert.html



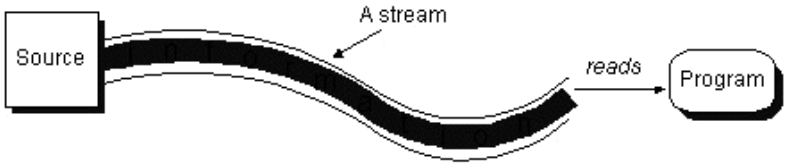
https://www3.ntu.edu.sg/home/ehchua/programming/java/J5a_ExceptionAssert.html

Armazenar dados

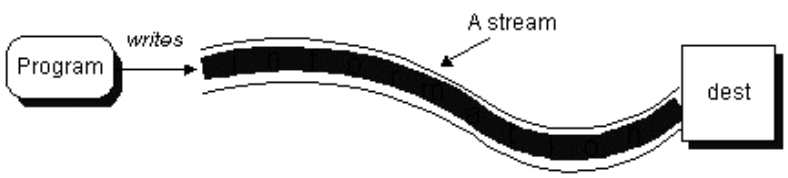
- Armazenar dados em variáveis, objetos e coleções é temporário (memória central do computador)
- Os ficheiros são utilizados para armazenar dados de longa duração (dados persistentes)
- Manipular ficheiros faz parte dos mecanismos I/O (input/output) do JAVA

Streams

- Objeto que gere a entrega ou recolha de dados de uma origem de dados
- Estrutura intermediária entre a origem e o destino dos dados



Input stream
Stream para leitura de dados



Output stream
Stream para escrita de dados

Ficheiros

- Todos os dados armazenados em ficheiros são zeros e uns (binário)
- **Ficheiros de texto**
  - Os bits representam caracteres (ASCII)
  - Fácil interpretação por humanos
- **Ficheiros binários**
  - Os bits representam informação codificada (não ASCII)
  - Fácil interpretação por computadores
  - Maior performance/eficiência

Ficheiros em JAVA

- O Java inclui diversas classes destinadas a manipular ficheiros (incluídas na package java.io)
- A classe *File* serve para modelar ficheiros em disco
- A criação de um objeto File não garante a criação de um ficheiro correspondente em disco, serve apenas para o representar logicamente
- Se o ficheiro correspondente existir, o objeto *File* fornece alguns métodos para o manipular

File

```
File f = new File("temp.txt");
```

- Principais métodos:
  - canRead: indica se o ficheiro pode ser lido
  - canWrite: indica se o ficheiro pode ser escrito
  - exists: indica se o ficheiro existe ou não no sistema de ficheiros
  - isFile: indica se o objeto corresponde a um ficheiro
  - isDirectory: indica se o objeto corresponde a uma diretoria
  - delete: remove o ficheiro do sistema de ficheiros
  - list: devolve uma lista com o nome dos ficheiros de uma diretoria
  - mkdir: cria uma diretoria

Buffering

- Os acessos I/O são feitos em bytes assim que possível
- São efetuados os acessos de leitura/escrita de/para disco com o mínimo atraso
- Estas operações têm um impacto elevado nos discos
- Acessos *buffered*
  - leitura/escrita em conjuntos (*chunks*) de dados
  - Há algum atraso na leitura/escrita dos bytes
  - As operações provocam menos impacto nos discos

Ficheiros de texto

- **FileReader/FileWriter**
  - Assim que a *stream* é construída o ficheiro de texto é criado se não existia antes ou os seus conteúdos removidos se o ficheiro já existia
  - Leitura ou escrita carácter a caracter

```
FileReader frd = new FileReader(new File(nomeDoFicheiro));
FileWriter fwt = new FileWriter(new File(nomeDoFicheiro));
```

- **BufferedReader/BufferedWriter**
  - Estas classes têm métodos convenientes para a leitura e escrita de linhas de caracteres, tornando-se, por isso, mais convenientes para a manipulação de ficheiros

```
BufferedReader fR = new BufferedReader(frd);
BufferedWriter fW = new BufferedWriter(fwt);
```

- Ficheiros de texto envolve quatro operações: **abertura, leitura, escrita e fecho**

Escrever ficheiro de texto

```
File f = new File("ficheiro.txt");
try {
    FileWriter fw = new FileWriter(f);
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write("Ola");
    bw.newLine();
    bw.write("mundo!");
    bw.close();
} catch (IOException ex) {
    System.out.println("Erro a escrever no ficheiro.");
}
```

Representação lógica do ficheiro

Estruturas para escrita *buffered*

Escrever para o ficheiro

Fechar o ficheiro

Ler ficheiro de texto

```
File f = new File("ficheiro.txt");
if (f.exists() && f.isFile()) {
    try {
        FileReader fr = new FileReader(f);
        BufferedReader br = new BufferedReader(fr);
        String line;
        while((line = br.readLine()) != null)
            System.out.println(line);
        br.close();
    } catch (FileNotFoundException ex) {
        System.out.println("Erro a abrir ficheiro de texto.");
    } catch (IOException ex) {
        System.out.println("Erro a ler ficheiro de texto.");
    }
} else {
    System.out.println("Ficheiro não existe.");
}
```

Verificar ficheiro

Estruturas para leitura *buffered*

Ler todas as linhas do ficheiro

Fechar o ficheiro

Ficheiros de objetos

- **FileInputStream/FileOutputStream**
  - Recolhe os dados do fluxo de entrada e reorganiza-os de forma a reconstruir os objetos
  - Organiza os dados dos objetos de modo a serem enviados sequencialmente para o ficheiro

```
FileInputStream is = new FileInputStream(new
File(nomeDoFicheiro));
FileOutputStream os = new FileOutputStream(new
File(nomeDoFicheiro));
```

- **ObjectInputStream/ObjectOutputStream**
  - Só objetos *Serializable* podem ser enviados para ficheiro
  - Permitir a conversão de um objeto para uma série de bytes

```
ObjectInputStream ois = new ObjectInputStream(is);
ObjectOutputStream oos = new ObjectOutputStream(os);
```

Objeto *Serializable*

```
public class Person implements Serializable {
    private String name;
    private int age;
    private int cc;
    public Person(String name, int age, int cc) {
        this.name = name;
        this.age = age;
        this.cc = cc;
    }
    public String toString() {
        return "Pessoa " + name + " com idade " + age + "
e CC " + cc;
    }
}
```

Escrever ficheiro de objetos

```
File f = new File("ficheiro.obj");
Person person = new Person("Luis", 30, 111223212);
try {
    FileOutputStream fos = new FileOutputStream(f);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(person);
    oos.close();
} catch (FileNotFoundException ex) {
    System.out.println("Erro a criar ficheiro.");
} catch (IOException ex) {
    System.out.println("Erro a escrever para o ficheiro.");
}
```

Ler ficheiro de objetos

```
File f = new File("ficheiro.obj");
try {
    FileInputStream fis = new FileInputStream(f);
    ObjectInputStream ois = new ObjectInputStream(fis);
    Person person = (Person)ois.readObject();
    System.out.println(person);
    ois.close();
} catch (FileNotFoundException ex) {
    System.out.println("Erro a abrir ficheiro.");
} catch (IOException ex) {
    System.out.println("Erro a ler ficheiro.");
} catch (ClassNotFoundException ex) {
    System.out.println("Erro a converter objeto.");
}
```