Entrada de dados do teclado (Scanner)
Ficheiros

Classe Scanner

- Scanner ler dados do:
  - Teclado (interacção com o utilizador)
  - String (parsing)
  - Ficheiro (armazenamento permanente)

Importação de classes

```
import java.util.Scanner;
public class ... {
    ...
    public static void main(String[] args) {
        Scanner scanner = new Scanner(...);
        ...
    }
}
```

Classe Scanner (teclado)

- Leitura do teclado

```
Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();
```

  - O programa bloqueia até que o utilizador escreva algo (na consola) e pressione *enter*
  - A linha introduzida é guardada num objecto String

    (Referenciado por line no exemplo acima)

Classe Scanner (ficheiro)

- Leitura de ficheiro

```
Scanner scanner = new Scanner(new File("file.txt"));

String line = scanner.nextLine();

int i = scanner.nextInt();
```

Classe Scanner (String)

```
String sentence = “um dois    tres quatro cinco  “;
Scanner scanner = new Scanner(sentence);
int n = 0;
String inverted = “”;

while(scanner.hasNext()) {
    n++;
    String token = scanner.next();
    inverted = token + “ ” + inverted;
}
System.out.println(n + “ palavras”);
System.out.println(“Invertida: “ + inverted);
```

```
> 5 palavras
> Invertida: cinco quatro tres dois um
```

Ficheiros

TEXTUAIS

- Legíveis
- Extensos
- Formato (quase) universal
- Formatos padronizados:
  - XML
  - SGML

BINÁRIOS

- Ilegíveis por humanos
- Compactos
- Formato pode depender da arquitectura da máquina
- Formatos padronizados:
  - ASN.1
  - Object Serialization Stream Protocol (Java)

Classes para acesso a ficheiros de texto

- `Scanner`
  - Para leitura
  - Usada anteriormente para ler do teclado
  - Estabelece *fluxo* (interno) de entrada do ficheiro

- `PrintWriter`
  - Para escrita
  - Interface semelhante à de `System.out`
  - Estabelece *fluxo* (interno) de saída para ficheiro

- `File`
  - Representa ficheiros

Objecto que liga ao ficheiro e o permite ler como uma sequência de caracteres.

Excepções de entrada e saída

- Que acontece quando
  - ficheiro não existe?
  - tipo dos dados pedido não corresponde ao conteúdo a ler?
  - …
- É lançada uma *excepção* ou fica registado um erro
- Excepções suportadas por *mecanismo de excepções do Java*

Excepções de entrada com `Scanner`

- `IOException`
  - `FileNotFoundException`
- `RuntimeException`
  - `InputMismatchException`
  - `NoSuchElementException`
  - `IllegalStateException`

Faz parte da lógica do programa

Erro de programação! A possibilidade de leitura tem de ser verificada *a priori*!

Excepções de saída com `PrintWriter`

- `IOException`
  - `FileNotFoundException` – Tentativa falhada de estabelecimento de fluxo de saída para ficheiro

Faz parte da lógica do programa

- `RuntimeException`
  - nada

Não há excepções relacionadas com erros de programação! O sucesso da escrita tem de ser verificado *a posteriori*!

Exemplo de leitura: abertura

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import static java.lang.System.out;
…
try {
    final Scanner fileScanner =
        new Scanner(new File("My file.txt"));
    …
} catch (final FileNotFoundException exception) {
    out.println("File was not found. Sorry!");
}
…
```

O Scanner cria um fluxo de entrada de caracteres entre o ficheiro e si mesmo.

Tem de se lidar com possibilidade de ficheiro não existir!

Exemplo de leitura: leitura e fecho

```
…
try {
    if (fileScanner.hasNextInt()) {
        final int numberOfCars = fileScanner.nextInt();
        …
    } else {
        out.println("Ops!");
    }
} …
```

É um erro de programação se a leitura falhar, sendo por isso lançada uma `RuntimeException`.

Tem de se lidar explicitamente com possibilidade de ficheiro não conter dados no formato esperado!

Exemplo de escrita: abertura

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import static java.lang.System.out;

…
try {
    final PrintWriter fileWriter =
        new PrintWriter(new File("My new file.txt"));
    …
} catch (final FileNotFoundException exception) {
    out.println("Error creating file. Sorry!");
    …
}
…
```

O `PrintWriter` cria um *fluxo* de saída de caracteres entre si mesmo e o ficheiro.

Tem de se lidar com a possibilidade de não se conseguir ligar ao ficheiro para escrita (e.g., directório não existe, ficheiro já existe e não se pode escrever sobre ele, etc.!

Exemplo de escrita: escrita e fecho

```
…
try {
    fileWriter.println(20);
    …
    if (fileWriter.checkError())
        out.println("Error writing to file.");
} finally {
    fileWriter.close();
}
…
```

Tem de se lidar explicitamente com a possibilidade de ocorrerem erros durante a escrita no ficheiro!

Uma excepção não apanhada é propagada ao longo da pilha de invocações. Como o *fluxo* que liga um `PrintWriter` a um ficheiro *tem sempre de se fechar*, usa-se o bloco `finally`.

Canais de Objectos

- Mantêm as relações entre Objectos
- Escrevem em formato binário
- São muito simples de usar
- ObjectOutputStream
- ObjectInputStream

Canais de Objectos

```
public class Aula implements Serializable {

  String nome = null;
  int n_presenças = 0;

  Disciplina disciplina = null;

  public Aula(String nome, int n, Disciplina d) {
     this.nome = nome;
     n_presenças = n;
     disciplina = d;
  }
// …
```

A interface Serializable

- Classes com objectos persistentes = serializable
- Variáveis não static (e não transient) e qualquer objecto são serializáveis
- A interface Serializable não obriga à implementação de qualquer método; Permitem usar ObjectStreams

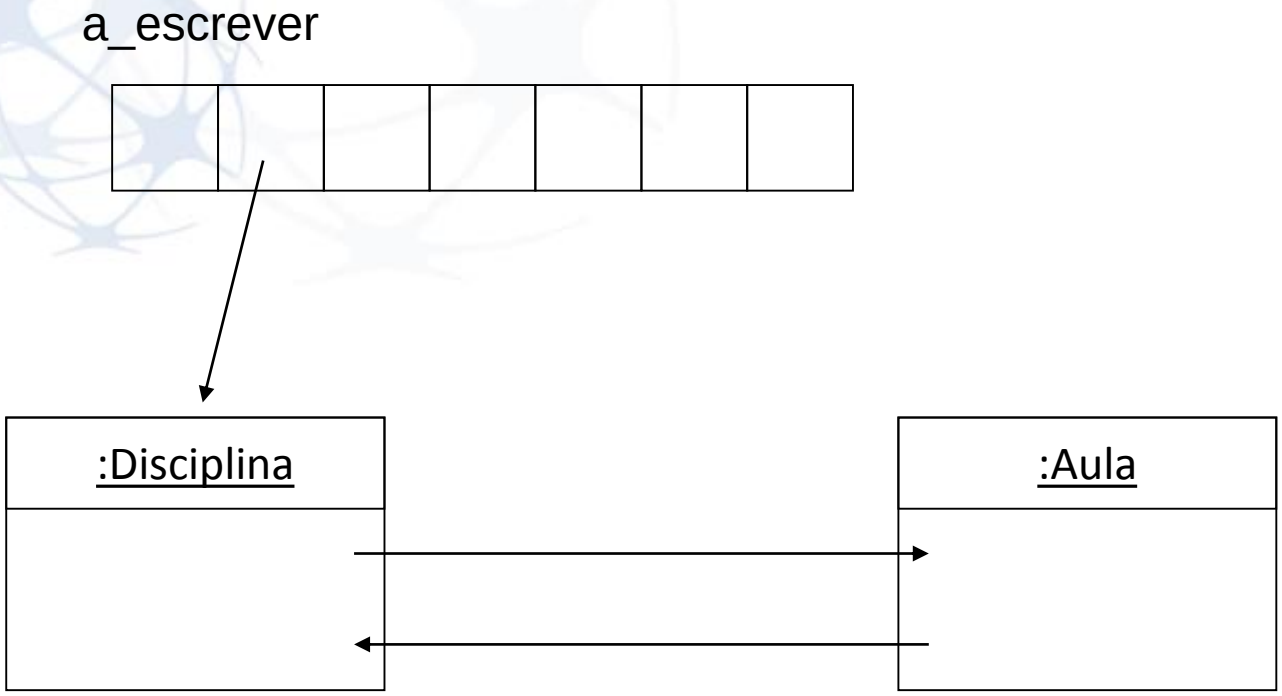
Canais de Objectos

```
public class Disciplina implements Serializable {


private String nome = null;
private int média_presenças = 0;
private Aula aula = null;

public Disciplina(String nome, String nome_aula, int n, int m)
  {
   this.nome = nome;
   média_presenças = m;
   aula = new Aula(nome_aula, n, this);
  }


// …
```

Canais de Objectos

```
public static void main(String[] args) {
  Disciplina[] a_escrever = new Disciplina[10];
  Disciplina[] a_ler = null;

  for (int i = 0; i != a_ler.length; ++i) {
     a_escrever[i] = new Disciplina(…);
  }
  // …
```

Canais de Objectos



Canais de Objectos

```
public static void main(String[] args) {
   // …
try {
  ObjectOutputStream o = new    ObjectOutputStream(new
      FileOutputStream("dat.dat"));
  o.writeObject(a_escrever);
  o.close();
} catch { //… }

try {
  ObjectInputStream in = new ObjectInputStream(new
                          FileInputStream("dat.dat"));
  a_ler = (Disciplina[]) in.readObject();
  in.close();
} catch { //… }
```

Cópia, profunda e integral com apenas UMA instrução de escrita e outra de leitura.

Canais Pré-Definidos

- `System.in`
- `System.out`
- `System.err`

Acesso para acrescentar informação

```
File dados = new File("info");

try {

  FileWriter f = new FileWriter(dados, true);
  f.write('A');
  f.close();

} catch (IOException e) {

  // ...

}
```

Referências

- Java2 Platform API, Scanner, http://download.oracle.com/javase/6/docs/api/java/util/Scanner.html
- Java 2 Platform API, FileWriter, http://download.oracle.com/javase/1.4.2/docs/api/java/io/FileWriter.html
- Y. Daniel Liang, "Introduction to Java Programming" 7th Ed. Prentice-Hall, 2010.

Sumário

- Entrada de dados do teclado (Scanner)
- Ficheiros