

Leitura e Escrita em Ficheiros

Material didáctico elaborado pelas diferentes equipas de
Introdução à Programação

Luís Caires (Responsável), Armanda Rodrigues, António Ravara, Carla Ferreira, Fernanda Barbosa, Fernando Birra, Jácome Cunha, João Araújo, Miguel Goulão, Miguel Pessoa Monteiro, e Sofia Cavaco.

Mestrado Integrado em Engenharia Informática FCT UNL

Estação Meteorológica

Recorde a Estação Meteorológica

- Objectivo
 - Manipular valores de temperaturas ao longo do tempo.
- Descrição
 - Numa estação meteorológica regista-se valores de temperaturas (valores reais).
- Funcionalidades
 - É sempre possível registar um dado valor de temperatura.
 - É necessário saber sempre o número de temperaturas registadas, poder consultar os valores médio, máximo e mínimo das temperaturas registadas.
 - Quando é criada, não existem temperaturas registadas.

Recorde a Estação Meteorológica

- Interface (classe WeatherStation)

```
void sampleTemperature(double temp)
```

Registar a amostra temp na estação

```
int numberTemperatures()
```

Consultar o número de temperaturas registadas até ao momento

```
double getMaximum()
```

Consultar a máxima temperatura observada até ao momento

```
pre: numberTemperatures() > 0
```

```
double getMinimum()
```

Consultar a mínima temperatura observada até ao momento

```
pre: numberTemperatures() > 0
```

```
double getAverage()
```

Consultar a média das temperaturas observadas até ao momento

```
pre: numberTemperatures() > 0
```

Estação Meteorológica

(Nova interação com o utilizador)

- Interação com o utilizador
 - O utilizador pode registar um conjunto de temperaturas. Para finalizar o registo de temperaturas deve escrever um valor não numérico.
 - No final, caso se tenha feito o registo de pelo menos uma temperatura, deve ser apresentado o valor médio, máximo e mínimo das temperaturas registadas.

Estação Meteorológica – Classe Main

```
public class Main {  
    /**  
     * Programa principal  
     */  
    public static void main(String[] args) {  
        WeatherStation ws = Main.readFromKeyboard();  
        Main.writeToConsole(ws);  
    }  
  
    /**  
     * Métodos auxiliares da Main  
     * Cria uma estação, recebe dados, gera relatório.  
     */  
    private static WeatherStation readFromKeyboard() {...}  
    private static void writeToConsole(WeatherStation ws) {...}  
}
```

Estação Meteorológica – Classe Main

```
public class Main {  
    /**  
     * Programa principal  
     */  
    public static void main(String[] args) {  
        WeatherStation ws = Main.readFromKeyboard();  
        Main.writeToConsole(ws);  
    }  
  
    /**  
     * Métodos auxiliares da Main  
     * Cria uma estação, recebe dados, gera relatório.  
     */  
    private static WeatherStation readFromKeyboard() {...}  
    private static void writeToConsole(WeatherStation ws) {...}  
}
```

Normalmente será boa ideia irmos criando métodos auxiliares, em vez de colocar todo o código na classe com o programa principal.

Estação Meteorológica – Classe Main

```
public class Main {  
    /**  
     * Programa principal  
     */  
    public static void main(String[] args) {  
        WeatherStation ws = Main.readFromKeyboard();  
        Main.writeToConsole(ws);  
    }  
  
    /**  
     * Métodos auxiliares da Main  
     * Cria uma estação, recebe dados, gera relatório.  
     */  
    private static WeatherStation readFromKeyboard() {...}  
    private static void writeToConsole(WeatherStation ws) {...}  
}
```

Opcional: realçar os métodos que pertencem à Main e usar uma variável temporária (ws)

Estação Meteorológica – Classe Main

```
public class Main {  
    /**  
     * Programa principal  
     */  
    public static void main(String[] args) {  
        writeToConsole(readFromKeyboard());  
    }  
}
```

Simplificando um pouco... temos código equivalente

```
/**  
 * Métodos auxiliares da Main  
 * Cria uma estação, recebe dados, gera relatório.  
 */  
private static WeatherStation readFromKeyboard() {...}  
private static void writeToConsole(WeatherStation ws) {...}  
}
```

Estação Meteorológica – Classe Main

```
/**
 * Cria uma estação meteorológica e lê uma sequência de temperaturas,
 * armazenando-as na estação meteorológica. Finalmente, devolve a
 * estação meteorológica com as temperaturas registadas.
 * @return a estação meteorológica, após lidas as temperaturas.
 */
private static WeatherStation readFromKeyboard() {
    double temp;
    Scanner in = new Scanner(System.in);
    WeatherStation ws = new WeatherStation();

    System.out.print("Temperatura (carácter não numérico para fim): ");
    while ( in.hasNextDouble() ) {
        temp=in.nextDouble();
        in.nextLine();
        ws.sampleTemperature(temp);
        System.out.print("Temperatura (carácter não numérico para fim): ");
    }
    in.close();
    return ws;
}
```

Estação Meteorológica – Classe Main

```
/**
 * Cria uma estação meteorológica e lê uma sequência de temperaturas,
 * armazenando-as na estação meteorológica. Finalmente, devolve a
 * estação meteorológica com as temperaturas registadas.
 * @return a estação meteorológica, após lidas as temperaturas.
 */
private static WeatherStation readFromKeyboard() {
    double temp;
    Scanner in = new Scanner(System.in);
    WeatherStation ws = new WeatherStation();

    System.out.print("Temperatura (carácter não numérico para fim): ");
    while ( in.hasNextDouble() ) {
        temp=in.nextDouble();
        in.nextLine();
        ws.sampleTemperature(temp);
        System.out.print("Temperatura (carácter não numérico para fim): ");
    }
    in.close();
    return ws;
}
```

O ciclo vai decorrer enquanto o utilizador inserir valores que são aceites como double.

Cada valor inserido é adicionado à estação!

No final, não esquecer de devolver a estação preenchida!

Estação Meteorológica – Classe Main

```
/**
 * Escreve na consola algumas estatísticas sobre a
 * estação meteorológica.
 * @param ws A estação meteorológica.
 */
private static void writeToConsole(WeatherStation ws) {
    System.out.println("Estatísticas");
    System.out.println("Temperatura máxima: " + ws.getMaximum());
    System.out.println("Temperatura mínima: " + ws.getMinimum());
    System.out.printf("Média: %.2f\n", ws.getAverage());
}
```

Estação Meteorológica – Classe Main

```
/**
 * Escreve na consola algumas estatísticas sobre a
 * estação meteorológica.
 * @param ws A estação meteorológica.
 */
private static void writeToConsole(WeatherStation ws) {
    System.out.println("Estatísticas");
    System.out.println("Temperatura máxima: " + ws.getMaximum());
    System.out.println("Temperatura mínima: " + ws.getMinimum());
    System.out.printf("Média: %.2f\n", ws.getAverage());
}
```

A operação `printf` permite escrever uma `String` pré-formatada.

O primeiro argumento tem o modelo de `String` a escrever

- Dentro desse modelo, existem “locais” especiais a preencher com dados

- Neste exemplo, queremos introduzir um número real com duas casas decimais. `%.2f` vai ser substituído pelo valor devolvido por `ws.getAverage()`, apresentando-o apenas com duas casas decimais.

- Por exemplo, se `ws.getAverage()` retornar `12.648`, a `String` será:

- “Média: 12.65”

- `\n` simboliza a mudança de linha, no final da `String`

Estação Meteorológica – Classe Main

```
/**  
 * Escreve na consola algumas estatísticas sobre a  
 * estação meteorológica.  
 * @param ws A estação meteorológica.  
 */  
private static void writeToConsole(WeatherStation ws) {  
    System.out.println("Estatísticas");  
    System.out.println("Temperatura máxima: " + ws.getMaximum());  
    System.out.println("Temperatura mínima: " + ws.getMinimum());  
    System.out.printf("Média: %.2f\n", ws.getAverage());  
}
```

E se não houver temperaturas registadas?

É violada a pré-condição de `getAverage`: `numberTemperatures() > 0`

O cálculo da média faz uma divisão por 0 e o programa aborta!

Para cumprir a pré-condição, devemos usar a operação `numberTemperatures()` da `WeatherStation` num `if` para prevenir contra essa situação.

Estação Meteorológica – Classe Main

```
/**
 * Escreve na consola algumas estatísticas sobre a
 * estação meteorológica.
 * @param ws A estação meteorológica.
 */
private static void writeToConsole(WeatherStation ws) {
    if (ws.numberTemperatures() > 0) {
        System.out.println("Estatísticas");
        System.out.println("Temperatura máxima: " + ws.getMaximum());
        System.out.println("Temperatura mínima: " + ws.getMinimum());
        System.out.printf("Média: %.2f\n", ws.getAverage());
    }
    else
        System.out.println("Não há temperaturas registadas!");
}
```

E se os dados estivessem
num ficheiro ?

Leitura de ficheiros

Escrita em ficheiros

Informação geral sobre ficheiros

- Na generalidade das linguagens de programação – não apenas orientadas ao objecto (Java, Smalltalk, Eiffel, C#, Python, Ruby) mas também linguagens procedimentais mais antigas (Fortran, Basic, Pascal, C, PL/1, etc), existem operações de **abertura** e **fecho** dum ficheiro, para se obter uma variável que representa o ficheiro e através da qual é processado.
- Há vários modos de abertura:
 - Para leitura – apenas se pode ler e o ficheiro não é modificado, o que até permite que o ficheiro possa ser lido por mais do que um programa ao mesmo tempo. A abertura **falha** se o ficheiro não for encontrado.
 - Para escrita – apenas se pode escrever. Caso o ficheiro não exista, é criado vazio. Caso já exista, o conteúdo anterior perde-se.
 - Para acrescento – semelha a escrita, mas conteúdo anterior não se perde e a escrita é (geralmente) feita a seguir ao conteúdo anterior.
 - Para escrita e leitura – modo mais complexo, que permite a um programa comutar entre escrita e leitura, conforme seja conveniente.

Ficheiros de texto

- Ficheiros de texto são ficheiros cujo conteúdo, se for directamente escrito na consola (ou para uma impressora), será entendido por humanos.
- A organização em linhas (*new line*) é própria dos ficheiros de texto
- Ficheiros de texto não possuem qualquer formatação adicional
- Para teste, há várias maneiras fáceis de criar ficheiros de texto:
 - com um editor de texto simples, e.g. Bloco de Notas, Notepad++
 - como saída dos seus programas
- No Eclipse, podemos criar ficheiros de texto para usar nos nossos projectos, seleccionando o projecto no Package Explorer e fazendo:
 - File->New->Untitled text file
- Tipicamente, os ficheiros de texto são gravados com a extensão txt
- **Não use** processadores de texto (e.g. Word) para criar ficheiros de texto simples. Processadores de texto introduzem muita informação para além do texto, para dar suporte aos detalhes de formatação dos documentos.

Informação geral sobre ficheiros

- Na generalidade das linguagens de programação, o teclado e a consola são representados como **ficheiros de texto** – mesmo quando a linguagem fornece instruções próprias para os usar.
 - O teclado é um ficheiro de texto aberto para leitura
 - A consola é um ficheiro de texto aberto para escrita
- Nesta disciplina, o processamento de ficheiros foca-se nos ficheiros de texto. São considerados os modos de **leitura** e **escrita** (conteúdo anterior perde-se e ficheiro é criado vazio caso não exista).
- A linguagem Java usa classes diferentes para representar ficheiros abertos para leitura (e.g., `FileReader`) e escrita (e.g., `PrintWriter`). Serão essas as classes usadas – em conjunto com `Scanner` nos casos de leitura.
- Em linguagens OO como Java, a criação do objecto que representa o ficheiro (i.e., o construtor) efectua a abertura do ficheiro.

Ler ficheiros de texto

- A forma mais simples de ler texto é usar `Scanner`
- Para ler de um ficheiro do disco, contrói-se um objecto da classe `FileReader` (do pacote `java.io`). Depois, usa-se o `FileReader` para construir um objecto `Scanner`

```
FileReader reader = new FileReader("input.txt");
Scanner in = new Scanner(reader);

...
in.close();
```

- Utilizam-se os métodos de `Scanner` para ler dados:
 - `next`, `nextLine`, `nextInt`, e `nextDouble`
- Simplificando:

```
Scanner in = new Scanner(new FileReader("input.txt"));

...
in.close();
```

Escrever ficheiros de texto

- Para escrever num ficheiro, constrói-se um objecto da classe `PrintWriter` (também `java.io`)

```
PrintWriter out = new PrintWriter("output.txt");
```

- Se o ficheiro já existe, o seu conteúdo é apagado antes das novas acções de escrita. Se o ficheiro não existir, é criado um ficheiro vazio
- Usam-se `print`, `println` e `printf` para escrever num objecto `PrintWriter`:

```
out.println(29.95);  
out.println(new Rectangle(5, 10, 15, 25));  
out.println("Hello, World!");
```

- Terminado o processamento do ficheiro, este deve ser fechado: `out.close();` Senão, o output arrisca-se a não ser todo escrito no ficheiro.

Programa principal - Main

```
public class Main {  
  
    private static WeatherStation readFromFile() {...}  
    private static void writeToFile(WeatherStation ws) {...}  
  
    /**  
     * Programa principal  
     */  
    public static void main(String[] args){  
        Main.writeToFile(Main.readFromFile());  
    }  
  
}
```

Leitura de ficheiros de texto

```
/*
 * Cria uma estação meteorológica e lê uma sequência de temperaturas,
 * de um ficheiro chamado sample.txt armazenando-as na estação
 * meteorológica. Finalmente, devolve a estação meteorológica com as
 * temperaturas registadas.
 * @return a estação meteorológica, após lidas as temperaturas.
 */
private static WeatherStation readFromFile() {
    Scanner in = new Scanner(new FileReader("sample.txt"));
    WeatherStation ws = new WeatherStation();
    while ( in.hasNextDouble() ) {
        double temp = in.nextDouble();
        in.nextLine();
        ws.sampleTemperature(temp);
    }
    in.close();
    return ws;
}
```

E se o ficheiro `sample.txt` não existir?

Ooops! Isto assim não funciona. Nem sequer conseguimos compilar. Obtemos a seguinte mensagem de erro:

Unhandled exception type FileNotFoundException
Isto acontece porque o construtor de `FileReader` pode lançar esta exceção, se o ficheiro não existir. Quando um método lança uma exceção, delega a sua resolução no método chamador.

Leitura de ficheiros de texto

```
/**
 * Cria uma estação meteorológica e lê uma sequência de temperaturas,
 * de um ficheiro chamado sample.txt armazenando-as na estação
 * meteorológica. Finalmente, devolve a estação meteorológica com as
 * temperaturas registadas.
 * @return a estação meteorológica, após lidas as temperaturas.
 * @throws FileNotFoundException
 */
private static WeatherStation readFromFile() throws FileNotFoundException {
    Scanner in = new Scanner(new FileReader("sample.txt"));
    WeatherStation ws = new WeatherStation();
    while ( in.hasNextDouble() ) {
        double temp = in.nextDouble();
        in.nextLine();
        ws.sampleTemperature(temp);
    }
    in.close();
    return ws;
}
```

A declaração `throws FileNotFoundException` sinaliza que este método pode não se executar correctamente se o ficheiro `sample.txt` não existir, e que nesse caso o método aborta e lança uma excepção do tipo `FileNotFoundException`.
Hmmm... Não foi precisamente por isso que a nossa versão original de `readFromFile()` não compilou?

Escrita de ficheiros de texto

```
/**
 * Escreve na consola algumas estatísticas sobre a estação meteorológica.
 * @param ws A estação meteorológica.
 * @throws FileNotFoundException
 */
private static void writeToFile(WeatherStation ws) throws FileNotFoundException {
    PrintWriter out = new PrintWriter("statistics.txt");
    if (ws.numberTemperatures() > 0) {
        out.println("Estatísticas");
        out.println("Temperatura maxima: " + ws.getMaximum());
        out.println("Temperatura mínima: " + ws.getMinimum());
        out.printf ("Média: %.2f\n", ws.getAverage());
    } else {
        out.println("Não há temperaturas registadas.");
    }
    out.close();
}
```

A declaração `throws FileNotFoundException` sinaliza que este método pode não se executar correctamente se o ficheiro `statistics.txt` não puder ser escrito, e que nesse caso o método aborta e lança uma excepção do tipo `FileNotFoundException`.

Escrita de ficheiros de texto

```
/**
 * Escreve na consola algumas estatísticas sobre a estação meteorológica.
 * @param ws A estação meteorológica.
 * @throws FileNotFoundException
 */
private static void writeToFile(WeatherStation ws) throws FileNotFoundException {
    PrintWriter out = new PrintWriter("statistics.txt");
    if (ws.numberTemperatures() > 0) {
        out.println("Estatísticas");
        out.println("Temperatura máxima: " + ws.getMaximum());
        out.println("Temperatura mínima: " + ws.getMinimum());
        out.printf ("Média: %.2f\n", ws.getAverage());
    } else {
        out.println("Não há temperaturas registadas.");
    }
    out.close();
}
```

Está tudo resolvido? Ainda não! Agora, quer `readFromFile` quer `writeToFile` podem lançar a excepção `FileNotFoundException`. Os métodos que as invocam têm de poder lidar com a excepção de algum modo.

Programa principal - Main

```
public class Main {  
    /**  
     * @throws FileNotFoundException  
     */  
    public static void main(String[] args) throws FileNotFoundException {  
        Main.testFiles();  
    }  
    /**  
     * Método auxiliar de teste da WeatherStation com ficheiros  
     * Cria uma estação, recebe dados, gera relatório.  
     * @throws FileNotFoundException  
     */  
    private static void testFiles() throws FileNotFoundException {  
        WeatherStation ws = Main.readFromFile();  
        Main.writeToFile(ws);  
    }  
    private static WeatherStation readFromFile()  
        throws FileNotFoundException {...}  
    private static void writeToFile(WeatherStation ws)  
        throws FileNotFoundException {...}  
}
```

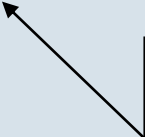
O que fazer com a exceção?

```
public class Main {  
    /**  
     * @throws FileNotFoundException  
     */  
    public static void main(String[] args) throws FileNotFoundException {  
        Main.testFiles();  
    }  
    ...  
}
```

O ideal seria tentar usar o ficheiro. Se conseguir, excelente. Caso contrário, falhar graciosamente.

Devemos “apanhar a exceção”

```
public class Main {  
  
    ...  
    public static void main(String[] args) {  
        try {  
            Main.testFiles();  
        } catch (FileNotFoundException e) {  
            System.out.println("Problemas nos ficheiros");  
        }  
    }  
}
```



Se o acesso ao ficheiro
Falhar, o código executado
Será este

Podemos proteger o nosso código com a instrução **try...catch**!
Nesse caso, o método `main` já não tem de re-lançar a exceção.
A clausula **throws** desaparece do `main`, mas mantém-se nos
métodos privados de leitura e escrita!

A construção `try... catch`

- A construção `try... catch` permite apanhar as exceções a sua sintaxe (simplificada) é:

```
try {  
    //bloco de instruções que pode lançar a exceção  
} catch (ExceptionType name) {  
    //instruções a executar caso ocorra a exceção  
}
```

- Por exemplo:

```
try {  
    //bloco de instruções que pode lançar a exceção  
} catch (FileNotFoundException e) {  
    System.out.println("Ficheiro não acessível");  
}
```

Leitura e escrita de ficheiros

- Em resumo:
 - A leitura e escrita em ficheiros de texto é feita de modo muito semelhante ao que já fazia na consola
 - Na leitura, inicializamos o `Scanner` com um objecto do tipo `FileReader`, em vez de `System.in`
 - Na escrita, usamos um objecto do tipo `PrintWriter`, em vez de `System.out`
 - Como o ficheiro de leitura, ou de escrita, podem não estar acessíveis, temos de prever essa possibilidade com a excepção `FileNotFoundException`
 - Além da utilização de ficheiros, vimos também uma primeira abordagem ao mecanismo de excepções do Java
 - Por agora, o tratamento dado a excepções é delegar (através da declaração `throws`) a sua resolução na operação que chama a operação onde a excepção ocorre e propagar essa delegação até ao programa principal
 - Mais à frente estudará detalhadamente o mecanismo de tratamento de excepções e formas mais sofisticadas de lidar com elas.

Agenda de Contactos

Agenda de contactos

- Objectivo
 - Manipular uma agenda de contactos.
- Interacção com o utilizador
 - ...
 - A informação dos contactos existentes no início do programa encontra-se num ficheiro de texto.
 - No final da execução do programa deverá ser escrito em ficheiro de texto toda a informação referentes aos contactos, para posterior uso.



Escrita da agenda em ficheiro

- O que é necessário definir na classe Main?
- Constante com o nome do ficheiro que vai guardar os contactos

```
public static final String FILE = "contacts.txt";
```
- Método de escrita do ficheiro, que vai chamar os métodos de iteração do ContactBook

```
public static void writeContactBook (ContactBook cb, String file) throws FileNotFoundException
```
- Chamada ao método de escrita do ficheiro, antes do final do método `main`
- O método `main` necessita de referência a `throws FileNotFoundException` na assinatura

Adicionar à classe Main

```
public static void writeContactBook(ContactBook cBook,  
    String file) throws FileNotFoundException {  
    PrintWriter pw = new PrintWriter(file);  
    Contact c = null;  
    pw.println(cBook.getNumberOfContacts());  
    cBook.initializeIterator();  
    while (cBook.hasNext()) {  
        c = cBook.next();  
        pw.println(c.getName());  
        pw.println(c.getPhone());  
        pw.println(c.getEmail());  
    }  
    pw.close();  
}
```

`pw` é a referência que nos permite escrever no ficheiro de dados. Temos de ter o cuidado de **abrir o ficheiro**, **escrever nele**, e **no final fechar o ficheiro**.

Adicionar à classe Main

```
public static void writeContactBook(ContactBook cBook,  
    String file) throws FileNotFoundException{  
    PrintWriter pw = new PrintWriter(file);  
    Contact c = null;  
    pw.println(cBook.getNumberOfContacts());  
    cBook.initializeIterator();  
    while (cBook.hasNext()) {  
        c = cBook.next();  
        pw.println(c.getName());  
        pw.println(c.getPhone());  
        pw.println(c.getEmail());  
    }  
    pw.close();  
}
```

`null` é a referência nula, o “zero” das referências a objectos. Estamos a especificar que ainda não temos o contacto `c`.

Atenção:
Em cada iteração do ciclo apenas invocamos o `next()` uma vez, caso contrário avançaríamos mais que um contacto.

Adicionar à classe Main

```
public class Main {  
    public static final String FILE = "contacts.txt";  
    ...  
    public static void main(String[] args) throws FileNotFoundException {  
        Scanner in = new Scanner(System.in);  
        ContactBook cBook = new ContactBook();  
        String comm = getCommand(in);  
        while (!comm.equals(QUIT)) {  
            if (comm.equals(ADD_CONTACT))  
                addContact(in, cBook);  
            else  
                System.out.println(WRONG_COMM);  
            comm = getCommand(in);  
        }  
        writeContactBook(cBook, FILE);  
        System.out.println(BYE);  
        in.close();  
    }  
    ...  
}
```

Ler a agenda de ficheiro

- Em `ContactBook`:
 - Não é preciso desenvolver mais nada.
- Na Classe `main`:
 - Vamos usar a constante `FILE`;
 - Vamos usar método estático `addContact()`;
 - Que por sua vez usa o `addContact()` do `ContactBook`;
 - Vamos desenvolver o método
 - `public static void readContactBook(ContactBook cBook, String file) throws FileNotFoundException`
 - Este método vai chamar o método `addContact()`

Ler a agenda de ficheiro

```
public static void readContactBook(ContactBook cBook,  
    String file) throws FileNotFoundException {  
  
    System.out.println("Reading Contacts File...");  
    FileReader fich = new FileReader(file);  
    Scanner fin = new Scanner(fich);  
    int cont = fin.nextInt();  
    fin.nextLine();  
    for(int i=0; i<cont; i++){  
        addContact(fin, cBook);  
    }  
    fin.close();  
}
```



O Scanner usa
um FileReader

Ler a agenda de ficheiro

```
public class Main {
    private static final String FILE = "contacts.txt";
    ...
    public static void main(String[] args) throws FileNotFoundException{
        Scanner in = new Scanner(System.in);
        ContactBook cBook = new ContactBook();
        readContactBook(cBook, FILE);
        String comm = getCommand(in);
        while (!comm.equals(QUIT)) {
            if (comm.equals(ADD_CONTACT))
                addContact(in, cBook);
            else
                System.out.println(WRONG_COMM);
            comm = getCommand(in);
        }
        writeContactBook(cBook, FILE);
        System.out.println(BYE);
        in.close();
    }
}
```


Ler a agenda de ficheiro

```
public class Main {  
    private static final String FILE = "contacts.txt";  
    ...  
    public static void main(String[] args) throws FileNotFoundException{  
        Scanner in = new Scanner(System.in);  
        ContactBook cBook = new ContactBook();  
        readContactBook(cBook, FILE);  
        String comm = getCommand(in);  
        while (!comm.equals(QUIT)) {  
            if (comm.equals(ADD_CONTACT))  
                addContact(in, cBook);  
            else  
                System.out.println(WRONG_COMM);  
            comm = getCommand(in);  
        }  
        writeContactBook(cBook, FILE);  
        System.out.println(BYE);  
        in.close();  
    }  
}
```

Ao correr, o ideal é tentar usar o ficheiro. Se conseguir, excelente. Caso contrário, falhar “graciosamente”.

Escrever a agenda de ficheiro

```
public class Main {  
    private static final String FILE = "contacts.txt";  
    ...  
    public static void main(String[] args) throws FileNotFoundException{  
        Scanner in = new Scanner(System.in);  
        ContactBook cBook = new ContactBook();  
        readContactBook(cBook, FILE);  
        String comm = getCommand(in);  
        while (!comm.equals(QUIT)) {  
            if (comm.equals(ADD_CONTACT))  
                addContact(in, cBook);  
            else  
                System.out.println(WRONG_COMM);  
            comm = getCommand(in);  
        }  
        writeContactBook(cBook, FILE);  
        System.out.println(BYE);  
        in.close();  
    }  
}
```

Ao correr, o ideal é tentar usar o ficheiro. Se conseguir, excelente. Caso contrário, falhar “graciosamente”.

Devemos “apanhar a exceção”

```
public class Main {  
    public static final String FILE = "contacts.txt";  
    ...  
    public static void main(String[] args) {  
        try {  
            Scanner in = new Scanner(System.in);  
            ContactBook cBook = new ContactBook();  
            readContactBook(cBook, FILE);  
            String comm = getCommand(in);  
  
            while (!comm.equals(QUIT)) {  
                if (comm.equals(ADD_CONTACT))  
                    addContact(in, cBook);  
                else System.out.println(WRONG_COMM);  
                comm = getCommand(in);  
            }  
            writeContactBook(cBook, FILE);  
            System.out.println(BYE);  
            in.close();  
        } catch (FileNotFoundException e) { ... }  
    }  
}
```

Podemos proteger o nosso código com a instrução **try...catch**! Nesse caso, o método `main` já não tem de re-lançar a exceção. A clausula `throws` desaparece do `main`, mas mantém-se em `readContactBook` e `writeContactBook`!

Se o acesso ao ficheiro falhar, o código executado Será este

Recapitulando, a construção `try... catch`

- A construção `try... catch` permite apanhar as exceções. A sua sintaxe (simplificada) é:

```
try {  
    //bloco de instruções que pode lançar a exceção  
} catch (ExceptionType name) {  
    //instruções a executar caso ocorra a exceção  
}
```

- Por exemplo:

```
try {  
    //bloco de instruções que pode lançar a exceção  
} catch (FileNotFoundException e) {  
    System.out.println("Ficheiro não acessível");  
}
```

Por agora, é suficiente...

- A forma de tratar excepções descrita aqui é suficiente no contexto de IP
- Em Programação Orientada pelos Objectos, a segunda cadeira de programação da LEI, estudará em detalhe o mecanismo de excepções do Java

Estrutura da aplicação

Interface com o utilizador

```
public class Main {
    ...
    public static void main(...) {
        ContactBook cBook;
        ...
    }
    private static void addContact(...) {...}
    private static void deleteContact(...) {...}
    private static void getPhone(...) {...}
    private static void getEmail(...) {...}
    private static void setPhone(...) {...}
    private static void setEmail(...) {...}
    private static void readContactBook(...)
        throws FileNotFoundException {...}
    private static void writeContactBook(...)
        throws FileNotFoundException {...}
}
```

Classes do domínio

```
public class ContactBook {
    public boolean hasContact(...) {...}
    public int getNumberOfContacts() {...}
    public void addContact(...) {...}
    public int getPhone(...) {...}
    public String getEmail(...) {...}
    public void deleteContact(...) {...}
    public void setPhone(...) {...}
    public void setEmail(...) {...}
    public void initializeIterator() {...}
    public boolean hasNext() {...}
    public Contact next() {...}
}
```

```
public class Contact{
    public Contact(...) {...}
    public String getName() {...}
    public int getPhone() {...}
    public String getEmail() {...}
    public void setPhone(int phone) {...}
    public void setEmail(String email) {...}
}
```