

#### METODOLOGIA

- › Interprete o documento calmamente e com atenção.
- › Acompanhe a execução do exercício no seu computador.
- › Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- › Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- › Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

#### CONTEÚDO PROGRAMÁTICO

1. [Escrita de dados em Ficheiros de texto](#)
2. [Criar e remover ficheiros](#)
3. [Leitura de dados de Ficheiros de texto](#)
4. [Ficheiros de dados estruturados](#)

## 1. Escrita de dados em Ficheiros de texto

Os programas que temos vindo a criar até este exercício, trabalham exclusivamente com informação em memória, ou seja, com dados inseridos e processados no momento. Para que um programa tenha uma aplicação útil, terá de ser capaz de armazenar informação, algo que acontece em quase todas as aplicações. Existem duas formas de armazenar dados: em ficheiros ou em bases de dados.

Ao longo deste exercício iremos conhecer alguns dos mecanismos que o Java disponibiliza para manipulação de ficheiros.

› **Crie** uma nova classe com o nome `EscritaFicheiro.java`

› **Insira** o seguinte código:

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class EscritaFicheiro {
    public static void main (String[] args) {

        try {
            PrintWriter pw = new PrintWriter("EscritaDados.txt");
            pw.println("Este é um exemplo de escrita de dados ");
            pw.println("em ficheiros ");
            pw.println("de texto");
            pw.println("Linha 4");
            pw.println("mais texto");
            pw.flush();
            pw.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("Erro : " + e.getMessage());
        }
    }
}
```

› **Compile e execute** a classe

› **Procure** o ficheiro de texto criado no computador, na pasta raiz do seu projeto do Eclipse

› **Abra** o ficheiro `EscritaDados.txt` e **analise** o seu conteúdo

📄 As principais classes de manipulação de ficheiros em Java estão no package `java.io` onde a classe `PrintWriter` disponibiliza mecanismos de criação e escrita de dados em ficheiros de texto.

📄 A invocação do método `flush` é essencial, caso contrário os dados não serão escritos no ficheiro. O método `close` é muito importante, pois as aplicações que escrevem muitos dados em ficheiros podem levar o sistema a uma rutura de recursos.

› **Execute** a classe várias vezes

📄 Este exemplo de escrita em ficheiros na prática não é muito útil porque, de cada vez que é executado, escreve os dados por cima dos dados anteriores.

Vamos melhorar o nosso exemplo:

› **Altere** o código que se encontra destacado:

```
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class EscritaFicheiro {
```

```
public static void main (String[] args) {  
    try {  
        FileWriter fw = new FileWriter("EscritaDados.txt", true);  
        PrintWriter pw = new PrintWriter(fw);  
        pw.println("Este é um exemplo de escrita de dados ");  
        pw.println("em ficheiros ");  
        pw.println("de texto");  
        pw.println("Linha 4");  
        pw.println("mais texto");  
        pw.flush();  
        pw.close();  
    }  
    catch (FileNotFoundException e) {  
        System.out.println("Erro : " + e.getMessage());  
    }  
    catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
}
```

1

› **Compile e teste** a classe

› **Execute** esta classe várias vezes

› **Abra** o ficheiro `EscritaDados.txt` e analise o seu conteúdo

📄 Neste exemplo foi adicionado o objeto `FileWriter`, que permite indicar o modo de escrita de junção ou `append`. Este modo escreve os dados sempre a seguir ao que já estiver escrito no ficheiro e é definido no **segundo parâmetro de construção do objeto 1**.

## 2. Criar e remover ficheiros

O Java disponibiliza outros mecanismos de manipulação de ficheiros além da simples escrita ou leitura de dados.

› **Modifique** na classe `EscritaFicheiro` o código que se encontra realçado:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class EscritaFicheiro {
    public static void main (String[] args) {

        File ficheiro = new File("EscritaDados.txt");

        try {
            if(ficheiro.exists()){
                ficheiro.delete();
            }
            else{
                FileWriter fw = new FileWriter("EscritaDados.txt", true);
                PrintWriter pw = new PrintWriter(fw);
                pw.println("Este é um exemplo de escrita de dados ");
                pw.println("em ficheiros ");
            }
        }
    }
}
```

```

        pw.println("de texto");
        pw.println("Linha 4");
        pw.println("mais texto");
        pw.flush();
        pw.close();
    }
}
catch (FileNotFoundException e) {
    System.out.println("Erro : " + e.getMessage());
}
catch (IOException e) {
    e.printStackTrace();
}
}
}

```

› **Compile e execute** a classe várias vezes

📄 Neste exemplo, o ficheiro é removido caso já exista. No bloco seguinte é criado e reescrito novamente.

Esta remoção é feita através da classe `File`. Esta disponibiliza vários métodos para manipulação de ficheiros e pastas. No exemplo anterior foi utilizado o método `delete` e o método `exists`. Vamos ver agora como é possível listar os ficheiros na diretoria corrente:

› **Adicione** um método à classe:

```

public static void listFiles() {
    try {
        System.out.println("Ficheiros na diretoria corrente:");
    }
}

```

```
File fList = new File(".");
String [] listaFicheiros = fList.list();

for (int i = 0; i < listaFicheiros.length; i++) {
    System.out.println(listaFicheiros[i]);
}
catch(Exception e) {
    System.out.println("Erro: " + e.getMessage());
}
}
```

› **Acrescente** no final do método `main` a instrução:

```
listFiles();
```

› **Teste** esta alteração

📄 Observe a listagem dos ficheiros da diretoria. Executando o programa várias vezes, são mostrados ficheiros diferentes devido à criação e remoção do ficheiro `EscritaDados.txt`.

Esta linha seguinte cria um objeto `File` que aponta para a diretoria corrente:

```
File fList = new File(".");
```

A linha a seguir obtém uma lista de textos (array de Strings) que representa os nomes dos ficheiros dessa diretoria:

```
String [] listaFicheiros = fList.list();
```

› **Altere** a instrução:

```
File fList = new File(".");
```

Para:

```
File fList = new File("..");
```

› **Compile** e **execute** a classe

📄 Neste exemplo, estamos a listar os ficheiros e pastas que existem no diretório acima daquele onde o programa é executado (indicado pelo caminho `..`).

📄 Se pretendêssemos listar os ficheiros que estão numa diretoria abaixo da corrente seria feito através do caminho `pasta/`.

### 3. Leitura de dados de Ficheiros de texto

Agora que já sabemos como escrever dados em ficheiros, vamos trabalhar a leitura.

› **Crie** uma nova classe com o nome `LeituraFicheiro.java`



› **Insira** o seguinte código:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class LeituraFicheiro {
    public static void main(String[] args) {

        File ficheiro = new File("EscritaDados.txt");

        try{
            BufferedReader br = new BufferedReader(new FileReader(ficheiro));
            String texto = br.readLine();

            if (texto == null){
                System.out.println("Ficheiro vazio!");
                return;
            }

            System.out.println("Conteudo do ficheiro:");

            while (texto != null) {
                System.out.println(texto);
                texto = br.readLine();
            }

            br.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("Ficheiro não existente");
        }
        catch (IOException e) {
            System.out.println("Erro de leitura do ficheiro");
        }
    }
}
```

```
}  
}  
}
```

#### › **Teste** e **execute** a classe

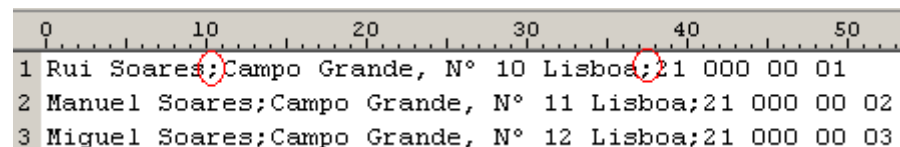
📄 Confirme a existência do ficheiro `EscritaDados.txt` e a visualização do seu conteúdo na consola.

#### › **Execute** a classe depois de apagar o ficheiro de forma a ver a mensagem de conteúdo vazio

## 4. Ficheiros de dados estruturados

Normalmente a escrita em ficheiros de dados, para ser coerente e para que seja possível armazenar informação de forma útil, tem de seguir um determinado padrão. No nosso caso vamos utilizar o padrão CSV, abreviatura para *Comma Separated Values*, que contém valores separados pelo caractere ";". Isto significa que em cada linha, a que chamamos "registo", o ";" separa os vários campos de dados. A interpretação deste tipo de informação é facilitada pelo uso do método "split" da classe String que nos permite separar um texto em vários blocos através de um caractere por nós especificado.

Temos a seguir um exemplo de um ficheiro com esta informação:



```
0 10 20 30 40 50  
1 Rui Soares;Campo Grande, N° 10 Lisboa;21 000 00 01  
2 Manuel Soares;Campo Grande, N° 11 Lisboa;21 000 00 02  
3 Miguel Soares;Campo Grande, N° 12 Lisboa;21 000 00 03
```

 Existem mais formatos para ficheiros de texto, cada um com as suas especificidades assim como: FLF, XML, PRN, etc.

Vamos aplicar esta estruturação do tipo CSV para guardar e ler informação relativa à classe `Pessoa`. O primeiro passo é criar dois métodos, um para escrita no ficheiro e outro para apresentação de dados no ecrã.

› **Adicione** o método `escreverFicheiro` à classe `Pessoa`:

```
public void escreverFicheiro(PrintWriter pw){
    pw.write(nome+ ";");
    pw.write(morada+ ";");
    pw.write(telefone + ";");

    SimpleDateFormat fmt = new SimpleDateFormat("dd-MM-yyyy");

    pw.write(fmt.format(datanascimento) + "\n");
}
```

Este recebe um `PrintWriter`, objeto de suporte à escrita em ficheiros e apenas escreve os campos da classe nesse objeto. O campo data serve-se do objeto `SimpleDateFormat` para ser escrito com o formato correto.

› **Adicione** o método `toString` à mesma classe:

```
public String toString(){
    SimpleDateFormat fmt = new SimpleDateFormat("dd-MM-yyyy");
```

```

return "nome:" + nome +
      "\nmorada:" + morada +
      "\ntelefone:" + telefone +
      "\ndata nascimento:" + fmt.format(datanascimento);
}

```

Este método gera uma String representante da informação da `Pessoa`. Repare como esta é partida em várias linhas através do caractere `\n`.

Já possuindo os métodos de suporte na classe `Pessoa` falta agora a construção de uma classe que utilize objetos deste tipo para fazer algumas operações. Iremos simular isto com uma lista de pessoas e as respetivas opções para adicionar, remover e listar pessoas. Para além destas, será possível gravar a lista de pessoas num ficheiro, assim como carregá-la a partir do mesmo.

› **Crie** a classe `MainPessoa.java` e **adicione** o seguinte código:

```

import java.io.*
import java.text.ParseException;
import java.util.*

public class MainPessoa {
    private static Scanner teclado;

    public static void main(String[] args) {
        ArrayList<Pessoa> listapessoas = new ArrayList<Pessoa>();
        teclado = new Scanner(System.in);

        int opcao = 6;

        do {
            escreveMenu();

            try {
                opcao = teclado.nextInt();
            }

```

```

    catch(Exception e){
        teclado.nextLine();
        System.out.println("Por favor insira um numero");
        opcao = 0;
    }

    switch(opcao){
        case 1: adicionarPessoa(listapessoas);break;
        case 2: //removerPessoa(listapessoas);break;
        case 3: listarPessoas(listapessoas);break;
        case 4: carregarPessoas(listapessoas);break;
        case 5: gravarPessoas(listapessoas);break;
        case 6: System.out.println("A sair...");break;
        default: System.out.println("Opcao invalida, escolha outra");break;
    }
}
while (opcao !=6);
}
}

```

📄 Após esta alteração, podemos verificar que existem vários erros no código, pois os métodos não existem mas vão ser implementados com o decorrer do módulo. Repare que aplicámos o tratamento de exceções para a leitura do número inserido no ecrã.

Com esta alteração criamos o ciclo que apresenta as opções do menu e as respetivas ações para cada uma dessas opções. Começemos pela escrita do menu no ecrã, que é o método mais simples:

➤ **Adicione** o método `escreveMenu`:

```

public static void escreveMenu(){
    System.out.println("1 - Adicionar Pessoa");
    System.out.println("2 - Remover Pessoa");
    System.out.println("3 - Listar Pessoas");
}

```

```
System.out.println("4 - Carregar Pessoas do ficheiro");  
System.out.println("5 - Escrever Pessoas no ficheiro");  
System.out.println("6 - Sair");  
}
```

› **Adicione** agora outro método na mesma classe:

```
public static void adicionarPessoa(ArrayList<Pessoa> lista){  
    try {  
        teclado.nextLine();  
        System.out.println("Insira o seu nome");  
        String nome = teclado.nextLine();  
        System.out.println("Insira a morada");  
        String morada = teclado.nextLine();  
        System.out.println("Insira o telefone");  
        long telefone = teclado.nextLong();  
        System.out.println("Insira a sua data de nascimento (dd-mm-aaaa)");  
        String data_nasc = teclado.next();  
  
        Pessoa p = new Pessoa(nome, morada, telefone, data_nasc);  
        lista.add(p);  
    }  
    catch (Exception e) {  
        System.out.println("Dados incorretos!Não é possível de adicionar a Pessoa");  
    }  
}
```

Este método pede todos os dados ao utilizador e cria um objeto do tipo `Pessoa` com esses dados:

```
Pessoa p = new Pessoa(nome, morada, telefone, data_nasc);
```

Após a criação do objeto é feita a adição na lista de pessoas através do método `add`:


```
lista.add(p);
```

➤ **Adicione** o método `listarPessoas` na classe `MainPessoa`:

```
public static void listarPessoas(ArrayList<Pessoa> lista){  
    for (Pessoa p: lista){  
        System.out.println(p.toString());  
    }  
}
```

Note como o código é bastante compacto e diferente do que temos visto nos ciclos for até este ponto. Este ciclo está a tirar partido de uma possibilidade diferente da construção de ciclos. Este tipo de construções também existe em muitas outras linguagens e tem o nome de *foreach* (no Java chama-se "*enhanced for loop*"). O objetivo é apenas percorrer toda a lista e efetuar operações com cada um dos seus elementos.

A sintaxe desta construção é a seguinte:



```
for (Objeto nome_variavel: lista_objetos)  
{  
    instruções  
}
```

[2] - O tipo de objeto que se vai percorrer na lista.

[3] - Variável que irá conter cada um dos objetos da lista.

[4] - A lista dos objetos.

Dentro do ciclo podemos utilizar a **variável** 3 para efetuar operações sobre o objeto. O ciclo percorre a lista e para cada um dos objetos que esta

contém chama o método `toString`.

O nosso método `listarPessoas` poderia ter sido escrito da seguinte forma:

```
public static void listarPessoas(ArrayList<Pessoa> lista){
    for (int i = 0; i < lista.size(); ++i){
        Pessoa p = lista.get(i);
        System.out.println(p.toString());
    }
}
```

› **Comente** as **linhas** que neste momento **têm erro** no método `main` e **teste** a classe

📄 Com os métodos até agora adicionados deve ser possível adicionar pessoas e listá-las no ecrã.

Cada vez que sai do programa, a lista de pessoas começa vazia e não temos, de momento, maneira de voltar a carregá-la. Para evitar isto falta-nos garantir a persistência de dados em ficheiro. Comecemos pelo método que grava:

› **Adicione** o seguinte método à classe:

```
public static void gravarPessoas(ArrayList<Pessoa> lista) {
    try{
        PrintWriter pw = new PrintWriter("Pessoas.csv");
        for (Pessoa p: lista){
            p.escreverFicheiro(pw);
        }

        pw.flush();
    }
}
```



```

        pw.close();
    }
    catch (Exception e){
        System.out.println("Erro de escrita no ficheiro");
    }
}

```

- › **Remova o comentário** deste método no bloco `main` e **teste-o**

Verifique o conteúdo do ficheiro `Pessoas.csv` e confirme que a estrutura deste consiste com o formato mencionado atrás: pessoas separadas por linhas e em cada linha os dados separados pelo caractere ";".

Para a escrita é usado o método `escreverFicheiro` da classe `Pessoa`. Para finalizar, vamos implementar o método `carregarPessoas`.

- › **Adicione** o método `carregarPessoas` à classe:

```

public static void carregarPessoas(ArrayList<Pessoa> lista) {
    lista.clear();
    File ficheiro = new File("Pessoas.csv");

    try{
        BufferedReader br = new BufferedReader(new FileReader(ficheiro));
        String texto = br.readLine();

        while (texto != null) {
            String[] dadosficheiro = texto.split(";");
            String nome = dadosficheiro[0];
            String morada = dadosficheiro[1];
            long telefone = Long.parseLong(dadosficheiro[2]);
            String texto_data = dadosficheiro[3];
            Pessoa pessoalida = new Pessoa(nome, morada, telefone, texto_data);
        }
    }
}

```

```

        lista.add(pessoalida);
        texto = br.readLine();
    }

    br.close();
}

catch (FileNotFoundException e) {
    System.out.println("Ficheiro não existente");
}

catch (IOException e) {
    System.out.println("Erro de leitura do ficheiro");
}

catch (ParseException e) {
    System.out.println("Erro na leitura das datas do ficheiro");
}
}

```

Começamos por apagar o estado da lista através do método `clear`. Isto para que não sejam adicionadas várias pessoas repetidas numa situação de carregamentos consecutivos.

```
lista.clear();
```

Após isto, é efetuado um ciclo de leitura linha a linha sobre o ficheiro. Cada linha possui a informação de uma pessoa. Para cada linha é então aplicado o método `split` para a dividir em várias **Strings**, através do caratere ";":

```
String[] dadosficheiro = texto.split(";");
```

Cada uma destas **Strings** resultantes representa um campo da pessoa. Os campos que não são do tipo **String** têm que ser convertidos para o tipo correto, como no caso do telefone:

```
long telefone = Long.parseLong(dadosficheiro[2]);
```

📄 O campo data nascimento não é convertido, pois isso é tratado diretamente no método construtor de `Pessoa`.

Após ter todas as informações da pessoa resta apenas construí-la e adicioná-la à lista:

```
Pessoa pessoalida = new Pessoa(nome, morada, telefone, texto_data);  
lista.add(pessoalida);
```

### › **Compile e teste** a classe

📄 Confirme que os métodos de gravação e leitura de ficheiro funcionam corretamente.