Módulo 2 - Ficheiros

Objetivos

- Conseguir modelar com classes um sistema de informação de pequena dimensão.
- Saber utilizar vetores redimensionáveis (java.util.ArrayList).
- Entender a relação entre o programa e os ficheiros do Sistema Operativo.
- Dominar a leitura e escrita de ficheiros de texto.
- Saber fazer o tratamento de exceções.
- Realizar os exercícios desta semana fazendo a compilação/execução "manual", i.e. não utilizando o Eclipse para esse efeito, tal como no módulo anterior.

Classe java.util.ArrayList

Os vetores em Java são inicializados fornecendo o seu comprimento, o qual é definitivo, não sendo possível "esticar" ou "encolher" o vetor. Nalguns casos, quando sabemos à partida qual o espaço necessário, esta característica não consiste num problema. Porém, existem várias situações onde o espaço necessário não consegue ser determinado à partida, o que vai obrigar o programador a tratar de alocação de novos vetores. Dado que esta tarefa obriga a código repetitivo, existe a estrutura de dados java.util.ArrayList que detém um vetor que é automaticamente redimensionado (alocando vetores maiores/menores quando necessário).

Um vetor de Strings é inicializado da seguinte forma:

```
String[] array = new String[10];
```

Por sua vez, um ArrayList de String é inicializado desta forma:

```
ArrayList<String> list = new ArrayList<String>(10);
```

Sendo que `String` é opcional, pois consegue ser inferido do tipo da variável **`list`**. O inteiro `10` dado no argumento indica a capacidade inicial do vetor, que é o valor por omissão caso se utilize o construtor sem argumentos. A forma mais usual de inicializar objetos ArrayList é `new ArrayList<>()`. Atenção, não confundir este valor com capacidade máxima, pois não é possível especificá-la.

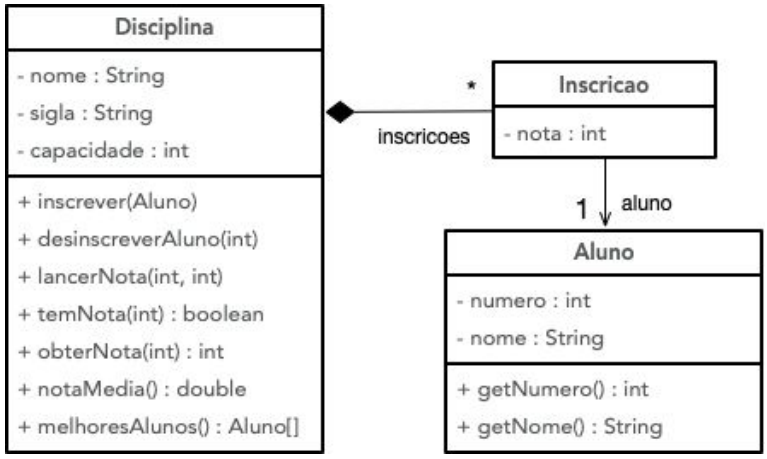
A lógica de acesso aos elementos é semelhante aos vetores, com índices de base zero. A tabela seguinte faz o paralelo entre a manipulação de vetores e ArrayList.

	Vetores	ArrayList
Inicialização	`array = new String[5]` `next = 0`	`list = new ArrayList<>(5)`
Inserção (no final)	`array[next] = "a"` `next++`	`list.add("a")`
Comprimento	`array.length  // 5`	`list.size()  // 1`
Acesso (índice 0)	`array[0]  // "a"`	`list.get(0)  // "a"`
Modificação (índice 0)	`array[0] = "?"`	`list.set(0, "?")`

Remoção (índice 0)	`for(i = 0; i < next; i++)`
	`  array[i] = array[i+1]`
	`next--`
		`list.remove(0)`

Exercício 2.1 - Gestão de disciplinas

Desenvolva um sistema de gestão de disciplinas, onde se inscrevem alunos e são-lhes atribuídas notas. Cada `Disciplina` terá a sua designação completa (String), a sua sigla (String), e o número de vagas (inteiro). Cada `Aluno` terá o seu número (inteiro) e nome (String). Os alunos têm notas de 0 a 20. Considere que o mesmo aluno poderá estar inscrito em várias disciplinas, e logo, as notas do aluno não devem estar guardadas no objeto `Aluno`.



Deve ser possível realizar as seguintes operações nos objetos `Disciplina`:

- Inscrever um aluno, se houver vaga. Ao ser inscrito, o aluno fica por omissão com nota indefinida, a qual pode ser definida posteriormente.

- Desinscrever um aluno, com base no seu número (se o aluno não estiver inscrito, este método não terá efeito).
  Dica: Procurar primeiro o objeto relativo à inscrição, e depois removê-lo da lista.

- Atribuir uma nota a um aluno, com base no seu número (validar inscrição e valor da nota).

- Saber se um aluno tem nota atribuída, com base no seu número.

- Obter a nota de um aluno, com base no seu número (validar se nota foi dada).

- Saber nota média, entre as que foram lançadas.

- Saber o conjunto de alunos que tiveram a nota mais alta (pode ser vazio caso não haja inscrições).

Desenvolva um pequeno programa que permita listar todos alunos, incluindo a nota de cada aluno (quando disponível). Teste as operações desenvolvidas.

Classe java.io.File

Os ficheiros e diretórios do Sistema Operativo são representados em Java pela classe `java.io.File`. Para instanciar estes objetos é necessário fornecer um caminho (*path*), o qual pode ser absoluto ou relativo. Um caminho absoluto inclui todos os elementos desde a raiz do Sistema Operativo, ao passo que um caminho relativo inclui apenas os elementos partindo do diretório de execução. Ao utilizar o terminal, o diretório será aquele onde se lançou o processo. No seguinte exemplo, o diretório de execução seria `/Users/Viriato/Project`.

```
> cd /Users/Viriato/Project
> javac KickOutRomans.java
> java KickOutRomans
```

Para se obter uma String com o diretório de execução pode-se fazer o seguinte:

```
String execPath = System.getProperty("user.dir");
```

Pode-se criar um objeto `File` para o diretório de execução fazendo:

```
File dir = new File(execPath);
```

Atenção que a criação de objetos `File`, por si só, não corresponde à criação de ficheiros ou diretórios em disco. Estes objetos representam um hipotético elemento, que poderá não existir.

```
boolean exists = dir.exists();  // true
```

Dado que os objetos `File` representam tanto ficheiros como diretórios, existem métodos para interrogar o objeto a esse respeito.

```
boolean isFile = dir.isFile(); // false;
boolean isDir = dir.isDirectory(); // true;
```

Podemos criar um objeto `File` que representa o diretório "pai" de um elemento, e criar um objeto `File` "filho" de outro existente:

```
File parent = dir.getParentFile(); // /Users/Viriato
File child = new File(dir, "Program.class"); //...iato/Project/Program.class
```

Por fim, podemos obter todos os elementos contidos num diretório fazendo:

```
File[] children = dir.listFiles(); // {../Program.java, ../Program.class}
```

Classe java.util.Scanner

A classe `Scanner`, apresentada anteriormente para processamento de Strings e leitura do teclado, também pode ser utilizada para leitura de ficheiros de texto. Estes podem ser processados linha a linha, ou palavra a palavra. A lógica de utilização é semelhante aos outros casos, mas devido a estarmos a lidar com elementos externos ao programa (ficheiros), existem exceções que podem ocorrer. Por exemplo, o ficheiro que estamos a tentar ler pode não existir, ou pode conter erros que comprometem a sua leitura. Este tipo de exceções (detalhes mais tarde), terão que ser forçosamente tratadas no código, mediante os blocos `try-catch`. O exemplo seguinte ilustra a abertura de um ficheiro `estudantes.txt` (que terá que estar no diretório de execução), e a impressão de cada uma das suas linhas.

```
File file = new File("estudantes.txt");
try {
      Scanner scanner = new Scanner(file);
      while(scanner.hasNextLine()) {
            String line = scanner.nextLine();
            out.println(line);
       }
      scanner.close();
}
catch (FileNotFoundException e) {
       System.err.println("ficheiro nao encontrado");
}
```

Em geral, os blocos `try-catch` têm que ser escritos quando existe uma operação que pode lançar uma exceção. Neste caso, é o construtor de `Scanner` que pode eventualmente lançar a exceção `FileNotFoundException` (caso o ficheiro `estudantes.txt` não exista ou não esteja no diretório onde seria suposto estar). Dependendo do que se pretende, uma alternativa ao tratamento consiste em propagar a exceção para fora do método (caso a mesma ocorra), ficando à responsabilidade do método que a invoca de tratar a exceção com `try-catch`. Nesse caso utiliza-se `throws` da forma ilustrada no seguinte exemplo:

```
public static void printLines(File file) throws FileNotFoundException {
      Scanner scanner = new Scanner(file);
      while(scanner.hasNextLine()) {
            String line = scanner.nextLine();
            out.println(line);
       }
      scanner.close();
}
```

Exercício 2.2 - Ler alunos de ficheiro

Desenvolva uma função para criar uma lista de objetos `Aluno` a partir de um ficheiro de texto. A função poderá ser um método estático da classe Aluno. Assuma que o ficheiro de texto contém uma linha por aluno, com número, seguido de um espaço, seguido do nome (atenção que o nome poderá ser mais do que uma palavra).

Estudantes.txt

```
1 Viriato
2 Táutalo
3 Tito Lívio
4 Caio Pláucio
...
```

Teste a função desenvolvendo um programa que possa ser executado da seguinte forma:

```
> java ListStudents Estudantes.txt
> 1: Viriato
> 2: Táutalo
> 3: Tito Lívio
> 4: Caio Pláucio
> ...
```

Exercício 2.3 - Ler disciplina de ficheiro

Desenvolva uma função para criar um objeto Disciplina, partindo de uma lista de alunos e de um ficheiro de texto que contém a informação da disciplina. A função poderá ser um método estático da classe Disciplina. O nome do ficheiro sem a extensão corresponde à sigla da disciplina, a primeira linha do ficheiro corresponde ao nome da disciplina, e as linhas que se seguem contêm o número do aluno inscrito, seguido de um espaço, e a sua nota ("NA", caso a nota não esteja atribuída).

- Teste a função carregando primeiro o ficheiro de alunos, e com base nessa lista a criação da disciplina a partir do ficheiro. Experimente as operações do Exercício 2.1, para verificar que o objeto foi bem construído.

POO.txt

```
Programação Orientada para Objetos
1 20
2 16
5 NA
6 NA
9 19
```

Classe java.io.PrintWriter

Uma das formas de escrever ficheiros de texto em Java é utilizando a classe `java.io.PrintWriter`. Estes objetos podem ser criados a partir de um objeto `File`, implicando que irá ser criado (ou escrito por cima) um ficheiro neste caminho. Tal como no caso de `java.util.Scanner`, poderão ocorrer exceções, neste caso relacionadas com permissões de escrita do ficheiro. As operações para escrita são similares às disponíveis em `System.out.*`. É importante não esquecer de fechar o objeto (*close*) após a escrita do conteúdo. O seguinte exemplo produziria o ficheiro em baixo.

```
try {
      PrintWriter writer = new PrintWriter(new File("teste.txt"));
      writer.println("uma linha com quebra");
      writer.print("segunda linha sem quebra");
      writer.println(", e pronto.");
      writer.close();
}
catch (FileNotFoundException e) {
      System.err.println("problema a escrever o ficheiro");
}
```

teste.txt

```
uma linha com quebra
segunda linha sem quebra, e pronto.
```

Exercício 2.4 - Escrita de disciplina em ficheiro

Desenvolva uma função para escrever um objeto `Disciplina` num ficheiro dado como argumento do programa, de acordo com o formato explicado no Exercício 2.3. A função poderá ser um método de instância da classe Disciplina.

Exercício 2.5 - Geração de disciplina com base em lista de alunos

Desenvolva um programa para gerar uma disciplina em formato de ficheiro, fornecendo o nome completo da disciplina e o ficheiro que contém uma lista de alunos. A sigla da disciplina deverá ser derivada do nome, utilizando as letras maiúsculas do mesmo. Todos os alunos contidos no ficheiro serão inscritos à disciplina, ficando sem nota atribuída. A execução do programa como se segue irá produzir um ficheiro *POO.txt* com o conteúdo no formato apresentado no Exercício 2.3, contendo todos os alunos presentes em *Estudantes.txt*.

```
> java GenerateCourse "Programacao Orientada para Objetos" Estudantes.txt
```

Exercício 2.6 - Listagem de ficheiros

Desenvolva duas funções para auxiliar a listagem de ficheiros, com os seguintes propósitos:

1. Obter todos os ficheiros (excluindo diretórios) que estão contidos num diretório.

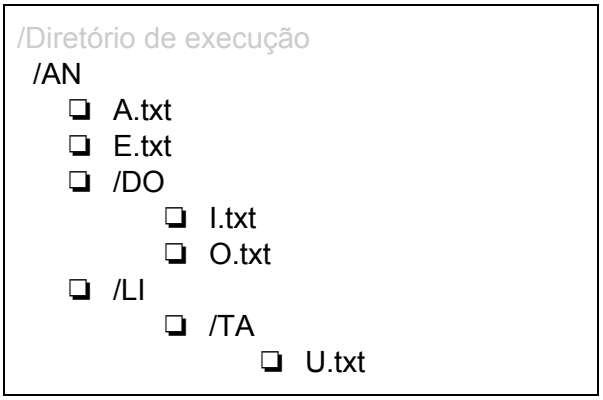
2. Uma versão do ponto (1), mas onde a procura é recursiva (sem limite de profundidade). Isto é, irão ser incluídos ficheiros contidos nos diretórios filho do diretório especificado, e assim sucessivamente.

Desenvolva um programa que possa ser executado das seguintes formas:

```
> java ListFiles AN
> /AN/A.txt
> /AN/E.txt

> java ListFiles -r AN
> /AN/A.txt
> /AN/E.txt
> /AN/DO/I.txt
> /AN/DO/O.txt
> /AN/LI/TA/U.txt
```

Estrutura de diretórios



Para implementar a função (2) será adequado ter a seguinte estrutura, onde a função principal recorre a outra auxiliar que será recursiva.

```
public static ArrayList<File> collectFiles(File dir) {
      ArrayList<File> list = new ArrayList<>();
      collectFilesRec(dir, list);
      return list;
}

private static void collectFilesRec(File f, ArrayList<File> list) {
      // ...
}
```