Módulo 1 - Strings

Objetivos

- Entender o processo de compilação e execução de um programa Java.
- Dominar a manipulação de strings (java.lang.String).
- Realizar os exercícios desta semana fazendo a compilação/execução "manual", i.e. não utilizando o Eclipse para esse efeito. Crie os ficheiros .java numa pasta fora do Eclipse. Utilize a linha de comandos para compilar e executar os programas deste módulo.

Método principal: main()

Para executar um programa de forma independente, torna-se necessário definir um método principal (vulgarmente designado de *main*), que é o que irá executar primeiro ao lançar o programa. Para executar um programa em Java é necessário indicar uma classe que contenha uma definição do main.

```
public class Program {
	public static void main(String[] args) {
		System.out.println("Bye-bye graphical interface");
	}
}
```

O argumento *args* é um vetor de String que irá conter os argumentos passados no lançamento do programa. O vetor será vazio caso não tenham sido passados argumentos.

Compilação (javac) e execução (java)

O código fonte de um programa Java consiste num conjunto de ficheiros .java. Antes de executar o programa implementado, os ficheiros terão que ser compilados para *bytecode*. Estes ficheiros *bytecode* poderão ser executados em qualquer máquina que tenha a máquina virtual do Java (JVM).

O comando de compilação, na sua forma mais elementar, é o seguinte:

```
> javac Program.java
```

Caso o ficheiro não contenha erros, será produzido o ficheiro Program.class.

Para compilar vários ficheiros, podemos utilizar o seguinte comando.

```
> javac *.java
```

Pressupondo que existe um ficheiro Program.class, resultante da compilação prévia de Program.java, o comando Java para o executar, na sua forma mais elementar, é o seguinte:

```
> java Program arg1 arg2 ... argN
```

O conteúdo do vetor de String passado no main terá o conteúdo {"arg1", "arg2", ..., "argN"}.

Exercício 1.1 - Argumentos de programa

Escreva um programa que liste os argumentos passados ao mesmo. Para imprimir valores para a consola invoque `System.out.println(...)`.

```
> java Program um dois tres
> 0: um
> 1: dois
> 2: tres
```

Dica: dado que `System.out.println(...)` é um pouco verboso, se incluir o seguinte *import*:
`import static java.lang.System.out;`
poderá escrever apenas `out.println(...)`.

Classe java.lang.String

A classe java.lang.String representa cadeias de caracteres, usualmente referidas como *strings*. Uma *string* é no fundo um vetor de caracteres (char[]), porém, dado que a manipulação direta destes vetores para propósitos comuns (p.e. concatenação) não é muito prática, estes objetos String têm o papel de facilitar essa tarefa.

A classe String, por ser tão frequentemente necessária, é importada por omissão em todos os programas (todas as classes do pacote java.lang o são; outro exemplo é java.lang.Math). A representação dos objetos String tem um tratamento especial na sintaxe da linguagem. Ao escrevermos uma sequência de caracteres entre aspas ("..."), estamos a criar um objeto String. Desta forma, as strings são normalmente inicializadas da seguinte forma:

```
String emptyString = "";
String otherString = "other";
```

A lógica de acesso ao conteúdo de uma String é similar à de um vetor de caracteres. A String tem um comprimento, e os caracteres podem ser acedidos mediante um índice (sendo o primeiro zero).

Os objetos String são imutáveis, isto é, não é possível alterar o seu estado. Os métodos disponíveis são funções que fornecem alguma informação sobre a String, ou produzem outra String com base na mesma.

Métodos comuns para acesso ao conteúdo da String:

Assinatura	**Exemplos**
`length() : int`	`"POO".length() // 3`
`charAt(int) : char`	`"POO".charAt(0) // 'P'`
`isEmpty() : boolean`	`"POO".isEmpty() // false`
`startsWith(String) : boolean`	`"POO".startsWith("PO") // true`
`endsWith(String) : boolean`	`"POO".endsWith("P") // false`
`contains(String) : boolean`	`"Objetos".contains("bje") // true`

Métodos comuns para obtenção de novas Strings:

`concat(String) : String`	`"POO".concat("!!!")  // "POO!!!"`
`substring(int) : String` `substring(int, int) : String`	`"POO".substring(1) // "OO"` `"POO!!!".substring(1, 4) // "OO!"`
`replace(String) : String`	`"POO!!!".replace("OO", "CD") // "PCD!!!"`

A operação concat(...), por ser necessária frequentemente, tem também um tratamento especial na sintaxe da linguagem, sendo permitido utilizar o operador + para fazer concatenações. Por exemplo:

```
String s = "POO";
s = s + "!!!";  // "POO!!!", equivalente a s = s.concat("!!!")
```

Comparação de igualdade de strings
Para verificar se duas strings são iguais, i.e. se têm exatamente os mesmos caracteres, deve ser sempre utilizado o método `equals`, e não o operador de igualdade (==). Por exemplo:

```
String s = "POO";
boolean t = s.equals("POO") // true
boolean f = !s.equals("POO") // false
```

Dica: Quando queremos verificar a igualdade de uma String predefinida com outra apontada por uma referência que pode estar a null, é mais prático inverter os termos: `"POO".equals(s)` Caso `s` esteja a null, será devolvido falso. Por outro lado, `s.equals("POO")` iria causar um erro (`NullPointerException`).

Exercício 1.2 - Conversão String - inteiro

Desenvolva funções para:

a) Verificar se uma String corresponde a um inteiro. Para efeitos de simplificação, considere apenas inteiros positivos.

b) Obter um inteiro a partir de uma String, assumindo que a mesma é válida (a função da alínea (a) devolve verdadeiro para o mesmo argumento).
Dica: existe uma função nas bibliotecas do Java para fazer isto (`Integer.parseInt(String)`), mas o objetivo aqui é implementar sem a utilizar.

Desenvolva um programa que possa ser executado da seguinte forma:

```
> java CheckIntegers 20 3 4 a b 11
> Encontrei 4 inteiros
> Encontrei 2 termos não inteiros
> Somatorio dos inteiros: 38
```

Classe java.util.Scanner

As bibliotecas do Java fornecem a classe java.util.Scanner para processamento de texto, podendo este ter diferentes proveniências. No caso mais simples, um objeto Scanner pode ser usado para processar uma String, iterando sobre o seu conteúdo palavra a palavra (*token*).

```
Scanner scanner = new Scanner("uma frase de exemplo");
while(scanner.hasNext()) {
      String token = scanner.next();
      out.println(token);
}
scanner.close();
```

```
> uma
> frase
> de
> exemplo
```

A classe Scanner também pode ser utilizada para ler strings do teclado, sendo que neste caso o processamento é feito linha a linha. O código seguinte lê sucessivamente linhas do teclado, bloqueando a cada nextLine(), até que seja escrito "fim" (a guarda do ciclo falha).

```
Scanner scanner = new Scanner(System.in);
String line = "";
while(!line.equals("fim")) {
      line = scanner.nextLine(); // bloqueia ate ser dado enter
      out.println("linha: " + line);
}
scanner.close();
```

Quando o objeto Scanner já não é necessário, deve-se invocar close(), em especial no caso da leitura de ficheiros (próximo módulo).

Declarações de importação

Todas as classes que queiramos utilizar que não pertençam ao pacote *java.lang*, terão que ser explicitamente importadas. Isso é feito incluindo uma declaração no topo do ficheiro .java, tal como no seguinte exemplo para a classe java.util.Scanner:

```
import java.util.Scanner;

public class ...
```

Exercício 1.3 - Filtro de palavras

Desenvolva um programa que recebe como argumentos um conjunto de palavras, as quais serão utilizadas para filtrar as frases que o utilizador insira de seguida. O programa deve repetidamente pedir frases ao utilizador e de seguida exibir as mesmas filtradas (excluindo todas as palavras passadas como argumento ao programa), até que seja dada uma frase vazia.

```
> java FilterWords nao nem
> frase: Eu nao gosto de programar nem a brincar
> filtrada: Eu gosto de programar a brincar
```

Exercício 1.4 - Contagem de caracteres

Desenvolva um programa que apresente a contagem da frequência de caracteres nos seus argumentos da seguinte forma:

```
> java CharacterStats atirei o pau ao gato
> a: 4 (25.0%)
> e: 1 (6.25%)
> g: 1 (6.25%)
> i: 2 (12.5%)
> o: 3 (18.75%)
> p: 1 (6.25%)
> r: 1 (6.25%)
> t: 2 (12.5%)
> u: 1 (6.25%)
```

Para auxílio à contagem, desenvolva uma classe para agregar as contagens de carateres das várias palavras. Como simplificação, considere que as palavras utilizam apenas os caracteres a...z (letras minúsculas, com código ASCII contíguo).

```
public class CharacterData {
	private int total;
	private int[] count;

	...

	public void addWord(String word) {
		// ...
	}
}
```

Exercício 1.5 - Avaliador de expressões

Desenvolva um programa para avaliar expressões aritméticas, onde cada operação aritmética entre dois termos está entre parêntesis. Assuma que a expressão está bem formada, contendo apenas números inteiros positivos e os operadores (+, -, x, /).

```
> java ExpressionEvaluator ( 1 + ( ( 2 + 3 ) x ( 4 x 5 ) ) )
> 101
```

Algoritmo de Dijkstra de duas pilhas:

- Criar pilha vazia para valores (val)
- Criar pilha vazia para operadores (op)
- Varrer os termos da esquerda para a direita
  - As aberturas de parêntesis "(" são ignoradas
  - Os valores são colocados no topo de val
  - Os operadores são colocados no topo de op
  - Quando ocorre um fecho de parêntesis ")"
    - Retirar dois valores (a, b) do topo de val
    - Retirar um operador do topo de op
    - Efetuar o cálculo em (b, a) tendo em conta o operador encontrado
    - Colocar o resultado no topo de val
- val conterá apenas um valor, que corresponde ao resultado final

Ilustração para `( 1 + ( ( 2 + 3 ) x ( 4 x 5 ) ) )`

(1 + ((2 + 3	)	x (4 x 5	)	)	)
val: 1 2 3 op: + +	val: 1 5 op: +	val: 1 5 4 5 op: + x x	val: 1 5 20 op: + x	val: 1 100 op: +	val: **101** op:

Para identificar e converter inteiros utilize as funções desenvolvidas no exercício anterior.
Para representar as pilhas pode utilizar a classe java.util.Stack. Exemplo:

```
Stack<Integer> intStack = new Stack<>();  // cria pilha vazia [ ]
intStack.push(1);             // coloca 1 no topo da pilha [1]
intStack.push(2);             // coloca 2 no topo da pilha [1, 2]
intStack.push(3);             // coloca 3 no topo da pilha [1, 2, 3]
int a = intStack.pop();       // retorna 3, e remove da pilha [1, 2]
int b = intStack.pop();       // retorna 2, e remove da pilha [1]
int c = intStack.pop();       // retorna 1, e remove da pilha [ ]
```