Paradigmas de Programação
Ficha de Trabalho TP1

O objectivo desta ficha de trabalho é familiarizar os alunos com a sintaxe da linguagem JAVA. Para tal, apresenta-se nesta secção um primeiro resumo das principais regras sintáticas da linguagem, ao que se segue uma secção onde são propostos alguns exercícios que os alunos deverão realizar.

1. MATERIAL DE APOIO

1.1 SINTAXE ESSENCIAL DO JAVA – PARTE 1

1.1.1 VARIÁVEIS

Em JAVA as variáveis têm que ser declaradas.

As variáveis são **válidas** dentro do bloco em que são declaradas.

A declaração de uma variável tem a forma: *tipo_dados identificador_variavel*;

Declaração e inicialização de uma variável tem a forma: *tipo_dados identificador_variavel = valor*;

Exemplos:

```
int num = 100, comp = 25;
char c = '5'; char nl = '\n';
long perímetro = 2563L;
```

Em JAVA há vários **tipos de variáveis**:

- Variáveis de instâncias
  - Correspondem aos campos da classe
  - Declaradas dentro da classe mas fora dos métodos
  - Pode explicitar-se o seu modificador de acesso/visibilidade; quando não estiver explicitado por defeito é `package-private`
  - São automaticamente inicializadas com os valores de defeito típicos do tipo de dados da variável
- Variáveis de classe
  - Declaradas dentro da classe mas fora dos métodos
  - São precedidas da palavra `static`
  - Válidas durante a execução do programa
- Variáveis locais
  - Declaradas dentro de um método (no corpo de um método)
  - Não têm modificadores de acesso/visibilidade `public` e `private`
  - Não são inicializadas por defeito; têm que ser inicializadas
- Parâmetros
  - Declaradas na assinatura dos métodos
  - Válidas durante a execução do método

1.1.2 CONSTANTES

Declaração similar às das variáveis mas precedidas da palavra `final`.

Normalmente escritas em letras maiúsculas.

Devem ser inicializadas no momento em que são declaradas.

Os seus valores não podem ser alterados por nenhuma instrução.

Exemplo: `final double PI = 3.14159F;`

Ao declarar uma constante:

- Sufixo **L** ou **l** indica constante *long*
- Sufixo **F** ou **f** indicam constante *float*
- Sufixo **D** ou **d** indicam constante *double*

1.1.3 TIPOS PRIMITIVOS

Existem **8 tipos primitivos** em Java.

Tipos primitivos são armazenados na RAM na zona de *stack.*

Cada tipo primitivo tem um **tamanho perfeitamente definido** (igual em qualquer plataforma).

Tabela 1 – Tipos primitivos em JAVA

Tipo			Contém	Valor defeito	Tamanho	Menor valor	Maior valor
Tipo Numérico	**Inteiro**	***byte***	signed integer	0	8 bits	-128	127
		short	signed integer	0	16 bits	-32768	32767
		int	signed integer	0	32 bits	-2147483648	2147483647
		long	signed integer	0	64 bits	-9223372036854775808	9223372036854775807
	Real	***float***	IEEE 754 floating-point	0,0	32 bits	±1.4E-45	±3.4E+38
		double	IEEE 754 floating-point	0,0	64 bits	-+4,9406564584124654E-324	-+1,7976931348623157E+308
Char			Unicode character	\u0000	16 bits	\u0000	\uFFFF
boolean			true ou false	false	1 bit	N.A	N.A.

Boleanos não são inteiros e não podem ser tratados como tal!!

1.1.4 TIPOS NÃO PRIMITIVOS / TIPOS OBJECTO

São todos os tipos que não são primitivos.

O valor de defeito para variáveis destes tipos é ***null***. O null não está associado a 0 e não pode ser *casted*.

Um tipo referência não pode ser *casted* para tipo primitivo. Um tipo primitivo não pode ser *casted* para tipo referência.

Variáveis do tipo objecto são manipuladas por referência (os tipos primitivos são manipulados por valor!!).

Posso ter duas variáveis diferentes a referirem o mesmo objecto.

Exemplo:

```
Button p, q;
p = new Button();
q = p;
p.setLabel("Ok");
String s = q.getLabel();
```

Exemplo:

```
Button a = new Button("Okay");
Button b = new Button("Cancel");
a = b;
```

1.1.5 OPERADORES

Os operadores existentes em Java, bem como as respectivas precedências, são indicadas na tabela seguinte:

Tabela – Operadores em Java

Precedência	Operador	Tipo de operandos	Associação	Operação
1	++, --	número	D	sinal unário
1	+, -	número	D	sinal unário
1	!	boleano	D	Negação
1	(*tipo*)	qualquer	D	*Cast*
2	*, /	número, número	E	multiplicação, divisão
2	/, %	inteiro, inteiro	E	quociente inteiro e módulo
3	+, -	número, número	E	soma e subtracção
4	>, >=	aritméticos	E	Comparação
4	instanceof	objecto, tipo	E	Comparação de tipo
5	==	primitivos	E	igual valor
5	==	objecto	E	igual valor das referências
5	!=	primitivos	E	valor diferente
5	!=	objecto	E	valor diferente das referências
6	^	boleanos	E	OU exclusivo lógico
6	|	boleanos	E	OU lógico
7	&&	boleano	E	E condicional
7	||	boleano	E	OU condicional
8	=	variável, qualquer	D	Atribuição
8	*= /= %=	variável, qualquer	D	atribuição após operação
8	+= -=	variável, qualquer	D	atribuição após operação
8	<<= >>=	variável, qualquer	D	atribuição após operação
8	>>>= &=	variável, qualquer	D	atribuição após operação
8	^= |=	variável, qualquer	D	atribuição após operação

*Nota relativa ao operador **+***:

O sinal **+ é um operador polimórfico**: pode representar a operação de adição matemática (quando ambos os operandos são numéricos) ou a concatenação de strings (quando pelo menos um operador é do tipo String). Neste caso, o operador que não for `String` é convertido para `String`. A conversão é automática para os tipos primitivos; para os tipos não primitivos a conversão tem que ser forçada pelo uso do método `toString( )`.

*Nota relativa ao operador **instanceof***:

O operador **`instanceof`** retorna `true` se o objecto do seu lado esquerdo (operando da esquerda) é instância da classe indicada do seu lado direito (operando da direita) ou de uma sua subclasse.

Exemplo:

```
Circle c1 = new Circle();
boolean b = c1 instanceof Circle;     // b vai assumir o valor de true
```

*Nota relativa ao pré e pós-incremento **de variáveis numéricas***:

A diferença entre a utilização do pré e pós incremento de variáveis é demonstrada pelo exemplo seguinte:

```
int i = 1;
System.out.println("i : " + i);
System.out.println("++i : " + ++i); // Pré-incremento
System.out.println("i++ : " + i++); // Pós-incremento
System.out.println("i : " + i);
System.out.println("--i : " + --i); // Pré-decremento
System.out.println("i-- : " + i--); // Pós-decremento
System.out.println("i : " + i);
```

cujo resultado produzido é:

```
i : 1
++i : 2
i++ : 2
i : 3
--i : 2
i-- : 2
i : 1
```

Nota relativa à divisão…

Em JAVA há 2 tipos de divisão: **divisão inteira** e **divisão decimal**.
Ambas usam o operador /.
Ser divisão inteira ou decimal depende dos tipos de operandos.
A divisão inteira produz sempre um resultado inteiro.
A divisão decimal produz sempre um resultado decimal.

Exemplos:

```
int i = 7;
int j = 2;
int k = i / j;                              // dá 3
double d = (double) i / ( double) j;        // dá 3.5
double oops = (double) ( i / j );           // dá 3.0
double shortcut = ((double) i ) / j;        // dá 3.5, j visto como double
double surprise = 1.0d / 10.0d;             // dá 0.1.
```

Nota relativa à realização de arredondamentos…

Para arredondar números de acordo com o número de casas decimais desejadas pode fazer-se:

```
double num = 3.456789;
double novoNum = (double)Math.round(num * 100) / 100;
```

*Nota relativa à **comparação de objectos***:

Supondo que tenho uma classe Aluno com dois campos, nome e número, e faço:

```
Aluno a1 = new Aluno(“Manuel”, “12345”);
Aluno a2 = new Aluno(“Manuel”, “12345”);
```

a expressão

`a1 == a2` dá valor FALSE!!!!!

Esta expressão compara as referências para os objectos e não os objectos propriamente ditos (ou seja não compara os valores dos campos dos objectos)

Para comparar mesmo os objectos a forma correcta de fazer é `a1.equals(a2)`.

1.1.6 CASTING

Por vezes tem-se um valor de determinado tipo e quer-se armazená-lo numa variável de tipo diferente.

Em Java, a conversão de tipos é aceite se for de um tipo com menor precisão para um com maior precisão.

Neste caso ocorre uma **conversão automática** de tipos.

Exemplo:

```
 int i = 10;
float x = i;
```

Se pretender, posso usar o mecanismo de casting para indicar de forma explícita uma conversão ou para "forçar" a sua ocorrência em situações em que ela não iria automaticamente ocorrer.

A indicação da cast é efetuada pela colocação do novo tipo entre parentesis.

Exemplo:

```
 int i = 200;
long l = (long)i;
long l2 = (long)300;
int i2 = (int)l2;
```

Java permite o cast de qualquer tipo primitivo para qualquer tipo primitivo, exceto de **boolean** (não se pode efectuar cast de um boleano para outro tipo).

1.1.7 COMENTÁRIOS

/* ... texto de comentário ... */

- Estes comentários não podem ser aninhados

// ... texto...

- Comentário válido até ao final da linha corrente

/** ... texto ... ***/**

- Forma de comentário especial usado pelo programa javadoc para gerar automaticamente documentação a partir do ficheiro fonte
- Não podem ser aninhados

1.2 OUTPUT EM JAVA

As instruções de escrita de JAVA são dirigidas para o dispositivo básico de saído, o monitor. O monitor é visto pelo JAVA como um "ficheiro" para o qual se escrevem caracteres. Este ficheiro especial é designado por **`System.out`**.

É possível escrever para o System.out (monitor) de uma forma muito simples utilizando o método **`println()`**. Por exemplo:

```
System.out.println("Vamos lá experimentar! \tParece que funcionou!!!\n\n");
System.out.println("O nome do aluno é: " + nome);
```

1.3 INPUT EM JAVA

Uma das classes disponíveis na API do JAVA é a classe **`Scanner`** (java.util.Scanner).

Esta classe possui métodos para ler diversos tipos de dados a partir de ficheiro e… do teclado.

O teclado está, por defeito, associado à variável **`System.in`**. Assim sendo, para que possamos ler diferentes tipos de dados a partir do teclado, temos que associar a variável que representa o teclado (System.in) a um Scanner.

De forma objectiva, aquilo que é necessário fazer para que num programa consigamos ler tipos primitivos a partir do teclado é:

a) Escrever no início do ficheiro `import java.util.Scanner;`

b) Criar um `Scanner` e associá-lo ao teclado, fazendo

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

c) Usar os métodos disponíveis em Scanner para ler os valores, por exemplo

```
String name = input.next(); // lê uma string
String line = input.nextLine(); // lê uma linha de texto terminada por \n
int x = input.nextInt(); // lê um inteiro válido
double pi = input.nextDouble(); // lê um real válido
…
```

EXERCÍCIOS

Exercício 1: Escreva um programa para calcular a média de três números, indicados pelo utilizador.

Exercício 2: Escreva um programa para calcular a área e o perímetro de um retângulo, em que as dimensões são dadas pelo utilizador.

Exercício 3: Escrever um programa que converta valores expressos em polegadas para valores expressos em centímetros.
Considere que uma polegada corresponde a 2,54 centímetros.

Exercício 4: O imposto de IVA num determinado país é de 23%. Escreva um programa em Java que aceite um preço inserido via teclado e imprima o custo do produto, o valor do imposto associado e o preço total a pagar.

Exercício 5: Dada uma determinada hora do dia, em horas, minutos e segundos, indicar quantos segundos decorreram desde o início do dia (sem efetuar validações).

Exercício 6: Dado o número de minutos decorridos desde o início do dia, calcular as horas e minutos correntes (sem efetuar validações).