2. A03. Elementos básicos dun programa Java

2.1 Introdución

Na actividade que nos ocupa aprenderanse os seguintes conceptos e manexo de destrezas:

- Identificar a sintaxe básica da linguaxe de programación Java: variables, identificadores, constantes, literais, expresións e operadores.
- Realizar conversión de tipos de datos.
- Coñecer a orde de precedencia nas expresións.
- Realizar operacións de entrada e saída de datos

2.2 Actividade

2.2.1 Datos e tipos de datos

En Java, coma en calquera outra linguaxe de programación, os programas xestionan os datos, que poden ser fixos ou variables, e que poden estar definidos por defecto ou obterse cando se executa o programa. Estes datos serán representados de distinto xeito segundo os distintos tipos de datos de que dispoña a linguaxe para poder definilos e manipulalos.

Distinguiremos entre os **tipos básicos ou primitivos de datos**, que son aqueles definidos por defecto na sintaxe da linguaxe, e os **tipos abstractos de datos**, que son aqueles que se definen a posteriori a partires dos primeiros.

Os tipos de datos primitivos están univocamente definidos en Java e son iguais en todas as plataformas. Por outra banda, os tipos de datos abstractos en Java son as *clases*, que as estudaremos na unidade 3.

2.2.2 Tipos de datos primitivos en Java

A linguaxe Java define 8 tipos de datos primitivos:

- Datos de tipo numérico
  - Números enteiros **byte, short, int e long**
  - Números en coma flotante **float e double**
- Datos de tipo carácter **char**
- Datos de tipo booleano **boolean**

Á representación do valor dun tipo de dato denomínase **literal**.

Enteiros

Os números enteiros son os que carecen de parte fraccionaria.

Dispoñemos de 4 tipos para representar números enteiros:

Tipo	Espazo en memoria (bytes)	Valor mínimo	Valor máximo
byte	1	-128	127
short	2	-32768	32767
int	4	-2147483648	2147483647
long	8	-9×10^{18}	9×10^{18}

Podemos expresar os literais enteiros en decimal (base 10), octal (base 8) ou hexadecimal (base 16). Os octais indicarémolos antepoñendo un 0 ao número e os hexadecimais antepoñendo 0x. Por exemplo:

En decimal: 255

En octal: 0377

En hexadecimal: 0xff

Os literais enteiros son de tipo **int** por defecto. Serán de tipo **long** se lle engadimos a letra **l** ou **L** ao final (a **l** minúscula non se emprega para non confundila co número 1):

255 é de tipo **int**

255L é de tipo **long**

Coma flotante

Os números en coma flotante son os que teñen parte fraccionaria.

Dispoñemos de 2 tipos para representar números fraccionarios ou reais:

Tipo	Espazo en memoria (bytes)	Valor mínimo	Valor máximo
float	4	$\pm 3.4 \times 10^{38}$	$\pm 1.4 \times 10^{-45}$
		Coma flotante de precisión simple (6 ou 7 díxitos)	
double	8	$\pm 1.8 \times 10^{308}$	$\pm 4.9 \times 10^{-324}$
		Coma flotante de precisión dobre (15 díxitos)	

Os decimais non son almacenados de forma exacta, polo que sempre hai un posible erro, por iso se fala de *precisión*: é máis preciso o tipo **double** que o **float**.

Podemos representar un literal en coma flotante como unha cadea de díxitos con punto decimal ou en notación científica ($\text{mantisa} \cdot 10^{\text{exp}}$). Por exemplo:

Cadea de díxitos: 801.3

Notación científica: 8.013E+2

Por defecto, os literais reais representar valores de tipo **double**. Serán de tipo **float** se lle engadimos a letra **f** ou **F** ao final:

8.5 é de tipo **double**

0.01F é de tipo **float**

Caracteres

En Java, o tipo para caracteres é **char**, e admite calquera dato codificado en formato UNICODE (http://www.unicode.org/).

Tipo	Espazo en memoria (bytes)	Valor
char	2	Caracteres (en Unicode)

Ademais dos caracteres estándar existen diferentes **secuencias de escape**, que son caracteres especiais precedidos por "\":

Secuencia	Nome
\b	Retroceso
\f	Salto de páxina
\n	Salto de liña
\r	Retorno de carro
\t	Tabulador
\'	Comiña simple
\"	Comiña dobre
\\	Barra inclinada (backslash)
\u????	Carácter Unicode (en hex)

Podemos representar os literais de tipo carácter empregando o valor Unicode escrito como enteiro, como carácter ou a secuencia de escape (\u????) entre comiñas. Por exemplo:

'A' ou 65

'\n' ou 10

'\u2122' ou 2122

Ambos valores son o mesmo á hora de definir o carácter.

Booleanos

Os valores booleanos ou lóxicos representan algo que pode ser verdadeiro (true) ou falso (false).

Tipo	Espazo en memoria (bit)	Valor mínimo	Valor máximo
boolean	1	Verdadeiro ou falso (true ou false)	

Podemos representar os literais booleanos con dous valores:

true, que significa verdadeiro

false, que significa falso

Pode obterse máis información dos tipos de datos primitivos na documentación oficial de Java https://docs.oracle.com/javase/specs/jls/se9/html/jls-4.html

2.2.3 Variables e identificadores

As variables son contedores que almacenan os datos que emprega un programa.

En realidade, unha variable ten 3 propiedades: unha posición de memoria para almacenar o valor; o tipo de datos almacenado na posición de memoria; e o nome empregado para referirse a esa posición de memoria.

De maneira xeral, dicimos que a variable é o nome simbólico (**identificador**) que identifica unha dirección de memoria que contén un ou máis datos, de xeito que cando empregamos o nome da variable (ou identificador) realmente estamos facendo referencia ao dato ou datos que están na memoria do ordenador.

Un nome dunha variable pode ser calquera identificador legal (una secuencia de letras e díxitos *Unicode*), sempre que respecte as seguintes normas:

- O primeiro carácter do nome debe ser unha letra (a-z,A-Z) ou os caracteres "_" ou"$".
- Non pode levar caracteres en branco
- Non pode coincidir coas palabras reservadas de Java, que son:
  - abstract
  - assert
  - boolean
  - break
  - byte
  - byteevalue
  - case
  - catch
  - char
  - class
  - const
  - continue
  - default
  - do
  - double
  - else
  - enum
  - extends
  - false
  - final
  - finally
  - float
  - for
  - goto
  - if
  - implements
  - import
  - instanceof
  - int
  - interface
  - long
  - native
  - new
  - null
  - package
  - private
  - protected
  - public
  - return
  - short
  - static
  - strictfp
  - super
  - switch
  - synchronized
  - this
  - threadsafe
  - throw
  - throws
  - transient
  - true
  - try
  - void
  - volatile
  - while

Hai que ter en conta que Java é sensible ás maiúsculas e ás minúsculas, polo que non é o mesmo "variable" que "Variable".

É recomendable establecer unha convención para os identificadores das variables:

- Comezar por letra, evitando "$" e "_".
- Empregar nomes descritivos. Así, por exemplo, nomeCliente indica que almacena o nome do cliente.
- Se está formado por una soa palabra, recoméndase poñelo en minúsculas. Se o nome está formado por dúas ou máis palabras, recoméndase poñer en maiúsculas o primeiro carácter da segunda e seguintes palabras. Exemplos: nomeCliente, númeroFactura,…

Na seguinte táboa analízase a validez dalgúns identificadores de variables:

Identificador variable	Válido	Xustificación
nome	**SI**	Empréganse minúsculas
meu nome	**NON**	Non se poden deixar espazos en branco
meuNome	**SI**	Séguese a convención das palabras compostas
1cliente	**NON**	Non pode empezar por un díxito
cliente1	**SI**	O díxito pode intercalarse no nome

número_cliente	**SI**	Pódese empregar "_", pero sería máis recomendable seguir a convención númeroCliente
CLIENTE	**SI**	Pero non é recomendable escribir en maiúsculas (só para constantes)
$cliente	**SI**	Pódese empregar "$" como primeiro carácter, pero non é recomendable
x	**SI**	Pero debería ser máis descritivo
final	**NON**	É unha palabra reservada

Realizarase a tarefa 1 "Identificadores das variables" onde o alumnado analizará a validez dos identificadores presentados.

Declaración de variables

Antes de empregar unha variable en Java hai que declarala, é dicir: indicar o nome da variable e o tipo de dato que vai a almacenar.

A sintaxe de declaración é a seguinte:

```
<tipo> identificador;
```

Onde **tipo** é o tipo de datos que almacenará a variable (texto, números enteiros,...) e **identificador** é o nome co que se coñecerá a variable. Ao final da declaración hai que empregar ";". Exemplos:

```
int número;
double prezo;
boolean recibido;
```

Tamén se poden declarar varias variables que sexan do mesmo tipo, separando por comas os identificadores das mesmas. A sintaxe é:

```
<tipo> identificador1, identificador2, …, identificadorN;
```

Exemplos:

```
int número, posición;
double prezo, cantidade;
boolean recibido, aberto;
```

A linguaxe Java fai unha comprobación estrita dos tipos de datos, de xeito que variables de tipos distintos son incompatibles. Polo tanto, as variables que declaremos en Java deben ter sempre un tipo de dato determinado. Atendendo ao seu tipo de dato podemos distinguir:

- **Variables de tipos primitivos**: as que almacenan un dato de tipo primitivo.
- **Varibles de referencia**: as que conteñen a identificación dun obxecto, que tal e como veremos na unidade 3, será obtido pola aplicación dunha *clase*.

Inicialización de variables

Despois de declarar unha variable hai que asignarlle un valor de forma explícita.

No caso de **variables de tipos primitivos**, para asignarlle o valor á variable emprégase o operador de asignación, o signo "=", seguido do valor e rematado en ";". Por exemplo:

```
int número;
número = 1;
```

Tamén podemos asignarlle o valor no momento de declarala. Por exemplo:

```
int número = 1;
```

En xeral, non se acepta asignar variables de distinto tipo, excepto cando se asignan valores de variables dun tipo primitivo a variables dun tipo primitivo superior. Por exemplo:

```
int número=1;
long dato=número;
```

Ao revés non se pode:

```
int número=1;
byte dato=número;    //erro de compilación
```

Para poder facelo hai que facer un **cast** (operación de **casting**), operación que permite converter valores dun tipo a outro. Para iso poñemos o tipo desexado entre parénteses *'(tipo)'*, diante da expresión. Faríamolo así:

```
int número=1;
byte dato = (byte)número;
```

Hai que ter en conta que se o valor asignado supera o rango da variable, o *cast* non terá sentido, pois non se poden almacenar todos os bits necesarios para representar o número. Por exemplo:

```
int número = 1000;
byte dato = (byte)número;   //Non podemos representar nun byte o valor 1000
```

No caso de **variables de referencia**, a inicialización supón a creación dun novo obxecto (nova instancia) do tipo de dato. Para iso emprégase a palabra clave **new**. Por exemplo:

```
Cliente usuario = new Cliente();
```

Nas variables de referencia profundizarase na unidade 3.

Realizarase a tarefa 2 "Uso de variables" onde o alumnado declarará e iniciará as variables necesarias para resolver os supostos que se lle propoñan.

Ámbito das variables

Toda variable ten un **ámbito**, é dicir, a parte do código onde a variable pódese usar. En realidade, este ámbito queda definido en función da posición do código onde se declarou a variable. E en función desta posición, Java define os seguintes tipos de variables:

- **Variables de clase**: son as definidas no corpo da clase (fóra de calquera método). Son variables accesibles dende calquera lugar da clase. Son variables de ámbito global.
- **Variables locais**: son as variables definidas dentro dalgún método:
  - Serán accesibles en todo o método se se declaran ao principio do mesmo.
  - Se se declaran dentro dalgunha estrutura, soamente serán accesibles dentro dese bloque.
- **Parámetros dun método**: son os datos necesarios para execución do método e que se pasan como parámetros ao mesmo. Son variables accesibles en todo o método.

O seguinte exemplo amosa o ámbito das variables:

```
/**
   * Clase ExemploVariables
*/
public class ExemploVariables{

    int variableGlobal; //variable de ámbito goblal na clase

    public void metodo1(int parametro1){
    //parámetro1 será accesible en todo o método1

      int variableLocal; //variable local accesible dentro do metodo1
      if (parametro1 > 100){
          int variableDeBloque; //variable accesible dentro deste if
          sentencias ……
      } //fin do if
    } //fin metodo1

    public void metodo2(int parametro2){
    //parámetro2 será accesible en todo o método2

      int variableLocal2; //variable local accesible dentro do metodo2
      if (parametro2 = 1){
          double variableDeBloque2; //variable accesible dentro deste if
          sentencias ……
      } //fin do if
    } //fin do metodo2
}
```

Pode obterse máis información das variables na documentación oficial de Java https://docs.oracle.com/javase/specs/jls/se9/html/jls-4.html#jls-4.4

2.2.4 Constantes

Unha constante é unha variable de só lectura, de xeito que o seu valor non pode variar. Polo tanto, esta variable estará protexida para que non cambie o seu valor durante toda a execución do programa (poderíamos dicir que non é unha variable en si, senón un valor que non cambia).

A declaración de constantes farase do mesmo xeito que a declaración das variables pero antepoñendo a palabra reservada **final**. Por exemplo:

```
final double PI=3.141591;
```

2.2.5 Operadores

Os operadores son símbolos especiais que permiten realizar operacións cos datos e devolver o resultado de ditas operacións.

A continuación analizaranse os operadores que podemos atopar en Java.

Operadores aritméticos

Java inclúe cinco operadores para realizar operacións aritméticas:

Operador	Operación
+	Suma ou concatenación de cadeas de caracteres
-	Resta ou cambio de signo
*	Multiplicación
/	División
%	Módulo (resto da división)

Hai que ter en conta o tipo de operando que esteamos a empregar:

- Se os operandos son enteiros, o resultado será un enteiro.
- Se un dos operandos é de tipo *float* ou *double*, a operación realízase en coma flotante.
- A división entre enteiros da coma resultado un enteiro e o resto pérdese.
- A división por cero dun enteiro produce unha excepción.
- A división por cero en coma flotante produce *Infinitive* ou *NaN* (Not a Number).
- O módulo soamente se pode realizar con tipos enteiros.

Exemplos:

Operación	Tipo	Resultado
9/2	int	4
9/2.0f	float	4.5f
9.0/2	double	4.5
9.0/0.0	double	+Infinity
0.0/0.0	double	NaN
12%5	Int	2

Operadores de incremento e diminución

Java inclúe operadores de incremento e diminución de variables numéricas:

Operador	Operación
++	Suma 1
--	Resta 1

Exemplo:

```
int número = 10;
número++;  //número ten o valor 11
--número;  //número volve a ter o valor 10
```

Estes operadores poden ir antes ou despois da variable, é dicir, como prefixo ou sufixo da variable. Hai que ter en conta que se se empregan en expresións, cando vai o prefixo, primeiro faise o incremento/diminución da variable e despois avalíase a expresión; pola contra, se vai coma sufixo, primeiro avalíase a expresión e logo faise o incrementase/diminución da variable. Por exemplo:

```
int número = 10;
resultado = 5 * ++número;  //o resultado é 55 e número ten o valor 11

int número = 10;
resultado = 5 * número++;  //o resultado é 50 e número ten o valor 11
```

Operadores relacionais

Os operadores relacionais son operadores de comparación que podemos empregar cos números e os caracteres.

Java inclúe seis operadores para realizar operacións relacionais:

Operador	Operación
==	Igual
!=	Distinto
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual

O resultado sempre é un booleano.

Exemplos:

Operación	Resultado
1>2	false
6>=8	false
5!=6	true
7>=7	true

Nota: En Java non se emprega o valor cero como falso e outro distinto de cero como verdadeiro.

Operadores booleanos ou lóxicos

Os operadores booleanos soamente actúan sobre operandos booleanos.

Java inclúe catro operadores para realizar operacións booleanas:

Operador	Nome	Operación	Significado
!	NOT	Negación	Cambia o valor booleano
&&	AND	'Y' lóxica	Devolve *true* se ambos operandos son *true*
||	OR	'O' lóxica	Devolve *false* se ambos operandos son *false*
^	XOR	'O' exclusiva	Devolve *true* se ambos operandos son diferentes

Recordemos as táboas da verdade:

A	B	!A	A&&B	A||B	A^B
false	false	true	false	false	false
false	true	true	false	true	true
true	false	false	false	true	true
true	true	false	true	true	false

Exemplos:

Datos	Operación	Resultado
x=5 , y=6	(x>0) && (x<y)	true
x=5 , y=6	(x>0) && (x>y)	false
x=5 , y=6	(x>0) || (x>y)	true
x=5 , y=6	(x<0) || (x<y)	false
x=5 , y=6	!((x>0) && (x>y))	true

Operadores a nivel de bit

Os operadores binarios permiten modificar bit a bit os tipos enteiros. Non se poden empregar con outro tipo de datos.

Java inclúe os seguintes operadores para realizar operacións a nivel de bit:

Operador	Nome	Operación	Significado
~	NOT	Complemento a 1	Cambia o bit
&	AND	'Y' binaria	Devolve *1* se ambos bits son 1
|	OR	'O' binaria	Devolve *0* se ambos bits son 0
^	XOR	'O' exclusiva binaria	Devolve *1* se ambos bits son diferentes
<<	Desprazamento á esquerda	Despraza os bits do primeiro operando tantas posicións á esquerda como indique o segundo operando. Os novos bits serán 0.	
>>	Desprazamento á dereita con signo	Despraza os bits do primeiro operando tantas posicións á dereita como indique o segundo operando. Os novos bits serán 1 se o primeiro operando é negativo, e serán 0 se é positivo.	
>>>	Desprazamento á dereita sen signo	Despraza os bits do primeiro operando tantas posicións á dereita como indique o segundo operando. Os novos bits serán 0. Pérdese o signo.	

Exemplos:

Operación	A nivel de bits	Resultado en bits	Resultado
2&5	0…00000010 & 0…00000101	0…00000000	0
2|5	0…00000010 | 0…00000101	0…00000111	7
2^5	0…00000010 ^ 0…00000101	0…00000111	7
2<<1	0…00000010 <<1	0..00000100	4
2>>1	0…00000010 >>1	0..00000001	1
-2>>>1	1…11111110 >>>1	01…111111	2147483647

Se non fixamos, o desprazamento á esquerda é equivalente a multiplicar por $2^{\text{desprazamento}}$, mentres que os desprazamentos á esquerda son equivalentes a dividir entre $2^{\text{desprazamento}}$ sempre que os números sexan positivos.

Operadores de asignación

Os operadores de asignación permiten asignar valores a unha variable. Cando estudamos como iniciar as variables, vimos que o operador de asignación era "=", pero haberá casos en que desexemos asignar á variable o resultado da expresión en que ela participa, como por exemplo, un incremento do seu valor. Por exemplo:

```
número = número +5;
```

é equivalente a:

```
número+=5
```

Así, os operadores de asignación permitirán reducir o código nestas expresións.

Java inclúe os seguintes operadores de asignación:

Operador	Nome
=	Asignación
+=	Incremento
-=	Diminución
*=	Multiplicación
/=	División
%=	Módulo (resto da división)
&=	AND
|=	OR
^=	XOR
<<=	Desprazamento á esquerda
>>=	Desprazamento á dereita con signo
>>>=	Desprazamento á dereita sen signo

Exemplos:

Operación	Operación equivalente
número += 5	número = número + 5
número -= 5	número = número - 5

número *= 5	número = número * 5
número /= 5	número = número / 5
número %= 5	número = número % 5
número &= 5	número = número & 5
número |= 5	número = número | 5
número ^= 5	número = número ^ 5
número <<= 5	número = número << 5
número >>= 5	número = número >> 5
número >>>= 5	número = número >>> 5

Outros operadores

En Java existen outros operadores que permiten operar sobre variables de referencia (manexar clases e obxectos), ou aplicar condicións ás expresións na mesma sentencia.

Estes operadores son:

Operador	Operación
instanceof	Identifica se un obxecto é dunha clase
this	Identifica a clase propia
new	Crea un obxecto
?:	Asgina un boolean nunha asignación condicionada

Estudarémolos nas seguintes unidades, pero enuméramolos aquí para poder incluílos na relación de precedencia que se explica a continuación.

Realizarase a tarefa 3 “Operadores en Java” onde o alumnado probará todos os exemplos indicados na tarefa comprobando os resultados obtidos.

Prioridade nas expresións

Ao igual que nas matemáticas, as operacións pódense encadear nunha expresión máis complexa. Neste caso hai que establecer a orde de prioridade na execución das distintas operacións.

A seguinte táboa amosa a orde de precedencia dos operadores de Java:

Nome	Operador
Expresións	() []
Sufixos	op++ op--
Unitarios	++op –op +op –op ~ !
Multiplicativos	* / %
Aditivos	+ -
Desprazamentos	<< >> >>>
Relacionais	< > <= >= instanceof == !=
AND binario	&
XOR binario	^
OR binario	|

AND lóxico	&&
OR lóxico	||
Condicionais	?:
Asignacións	= *= /= %= += -= &= ^= |= <<= >>= >>>=

Así por exemplo, tendo en conta a orde de precedencia, a seguinte expresión:

```
número=12+8/4
```

terá como resultado

```
número=14
```

xa que se realizará primeiro a división e despois a suma.

Pode obterse máis información sobre os operadores na documentación oficial de Java https://docs.oracle.com/javase/specs/jls/se9/html/jls-3.html#jls-3.12

Realizarase a tarefa 4 “Orde de precedencia nas expresións” onde o alumnado avaliará as expresións que se lle propoñan.

2.2.6 Fluxo estándar de datos

Os fluxos de datos estándar permiten ler ou escribir datos a través da consola.

En Java existen tres fluxos estándar de datos:

- **System.in**: permite a lectura por consola (teclado en xeral)
- **System.out**: permite escribir na consola (pantalla en xeral)
- **System.err**: permite escribir mensaxes de erro.

No primeiro programa de *Benvida* que fixemos empregamos **System.out.println** para escribir na pantalla.

```
System.out.println("Benvido á programación en Java");
```

Esta función escribe na consola calquera tipo básico de datos que lle pasemos: un texto entre comiñas dobres, expresións enteiras, booleanas, …, e ao final incorpora un salto de liña (se non queremos facer o salto de liña, empregamos **print** en vez de **println**)

Por exemplo:

```
int número=1, dato=2;
System.out.println("Aquí un texto literal");
System.out.println(número+dato);
System.out.println(dato>número);
```

E a saída por consola será:

```
Aquí un texto literal
3
true
```

Se queremos sacar varios valores á vez empregaremos o operador "+" para concatenar. Por exemplo:

```
System.out.println("A suma é " + (número+dato));
```

A saída por consola será:

```
A suma é 3
```

Nota: Recorda que operador "+" concatena cando ao menos un dos operadores é un texto.

A función **System.err** funciona como **System.out** pero amosando mensaxes de erro. Emprégase para non mesturar a saída normal coas mensaxes de erro. Así por exemplo, tanto en Netbeans como en Eclipse estas mensaxes aparecen de cor vermello na pantalla.

Por exemplo, se a sentencia é:

```
System.err.println("Non se admiten decimais");
```

Na consola do IDE aparecerá a mensaxe en vermello:

```
Non se admiten decimais
```

A función **System.in** permite a entrada de datos, pero é bastante complicado de empregar, polo que desde a versión 5.0, Java incorpora a clase *Scanner*, do paquete *java.util*, para facilitar a entrada de datos.

Polo tanto, para a lectura de datos o primeiro paso é crear un obxecto deste tipo:

```
java.util.Scanner scan = new java.util.Scanner(System.in);
```

A partires de aquí xa podemos empregar os métodos deste obxecto para realizar a lectura de datos.

Vexamos un exemplo no que realizaremos a suma de dous números enteiros introducidos por teclado:

```
/**
   * Programa que suma dous números enteiros usando a clase Scanner
   * e o método nextInt()
*/
public class OperacionSuma {

 public static void main(String[] args) {
        java.util.Scanner scan = new java.util.Scanner(System.in);

        System.out.print("\n Introduce o primeiro sumando:");
        int primeiroSumando=scan.nextInt(); //Lemos o primeiro sumando
        System.out.print("\n Introduce o segundo sumando:");
        int segundoSumando=scan.nextInt();  //Lemos o segundo sumando

    System.out.println("\n\n A suma é " + (primeiroSumando+segundoSumando));
 }
}
```

Neste exemplo, para ler os números enteiros usamos o método *nextInt()*.

Pode obterse máis información sobre os métodos da clase *Scanner* na documentación oficial de Oracle https://docs.oracle.com/javase/9/docs/api/java/util/Scanner.html

Realizarase a tarefa 5 “Fluxo estándar de datos” onde o alumnado amosará por pantalla o resultado dunha operación realizada con datos introducidos por teclado.

2.3 Tarefas

As tarefas propostas son as seguintes.

- **Tarefa 1. Identificadores de variables.** Nesta tarefa o alumnado analizará a validez dos identificadores presentados.
- **Tarefa 2. Uso de variables**. Nesta tarefa o alumnado declarará e iniciará as variables necesarias para resolver os supostos que se lle propoñan.
- **Tarefa 3. Operadores en Java.** Nesta tarefa o alumnado probará todos os exemplos indicados nesta actividade e comprobando os resultados obtidos.
- **Tarefa 4. Orde de precedencia nas expresións.** Nesta tarefa o alumnado avaliará as expresións que se lle propoñan.
- **Tarefa 5. Fluxo estándar de datos.** Nesta tarefa o alumnado amosará por pantalla o resultado dunha operación realizada con datos introducidos por teclado.

2.3.1 Tarefa 1. Identificadores de variables

Nesta tarefa o alumnado analizará a validez dos identificadores presentados.

Enunciado

Indica se os seguintes identificadores son válidos para variables en Java. Xustifica a túa resposta:

```
datos, nomeUsuario, número usuarios, 1usuario, público, private,
número_usuarios, NUMERO, $número, a
```

Solución

Identificador variable	Válido	Xustificación
datos	**SI**	Empréganse minúsculas
nomeUsuario	**SI**	Séguese a convención das palabras compostas
número usuarios	**NON**	Non se poden deixar espazos en branco
1usuario	**NON**	Non pode empezar por un díxito
público	**SI**	Empréganse minúsculas
private	**NON**	É unha palabra reservada
número_usuarios	**SI**	Pódese empregar "_", pero sería máis recomendable seguir a convención númeroCliente
NUMERO	**SI**	Pero non é recomendable escribir en maiúsculas (só para constantes)
$número	**SI**	Pódese empregar "$" como primeiro carácter, pero non é recomendable
a	**SI**	Pero debería ser máis descritivo

2.3.2 Tarefa 2. Uso de variables

Nesta tarefa o alumnado declarará e iniciará as variables necesarias para resolver o exercicio que se lle propoña.

Enunciado

Crea un proxecto en Java denominado UsoVariables, que amose por consola os valores que se indican a continuación:

- 65
- 2000
- 300000
- $5 \cdot 10^{30}$
- $3.3 \cdot 10^{200}$

Debes empregar as variables co formato mínimo que permitan almacenar o valor.

Ademais, crea unha variable tipo carácter e asígnalle o primeiro valor. Amosa por consola o valor.

Solución

Para resolver o exercicio hai que ter en conta o rango de valores de cada un dos tipos de datos en Java.

Así, unha posible solución sería:

```
/**
   * O programa UsoVariables é un exemplo para declarar e iniciar variables
amosando os seus valores por consola
*/
public class UsoVariables {

 public static void main(String[] args) {
        byte variableByte=65;
        short variableShort=2000;
        int variableInt=300000;
        float variableFloat=5e+30f;
        double variableDouble=3.3e+200;
        char variableChar=65;

        System.out.println("Primeiro valor: " + variableByte);
        System.out.println("Segundo valor: " + variableShort);
        System.out.println("Terceiro valor: " + variableInt);
        System.out.println("Cuarto valor: " + variableFloat);
        System.out.println("Quinto valor: " + variableDouble);
        System.out.println("Valor da variable char: " + variableChar);
 }
}
```

Unha vez executado, a saída por consola sería:

```
Primeiro valor: 65
Segundo valor: 2000
Terceiro valor: 300000
Cuarto valor: 5.0E30
Quinto valor: 3.3E200
Valor da variable char: A
```

Podemos observar como no caso da variable de tipo carácter amosa o carácter Unicode 65, que é o 'A'.

2.3.3 Tarefa 3. Operadores en Java

Nesta tarefa o alumnado probará todos os exemplos indicados nesta actividade e comprobará os resultados obtidos.

Enunciado

Proba todos os exemplos cos operadores proporcionados nesta actividade e comproba os teus resultados.

Solución

O alumno copiará todas as operacións dos exemplos do apartado "2.2.4 Operadores", e comprobará que coinciden os seus resultados cos proporcionados en dito apartado.

2.3.4 Tarefa 4. Orde de precedencia nas expresións

Nesta tarefa o alumnado avaliará as expresións que se lle propoñan.

Enunciado

Calcula o resultado de avaliar as seguintes expresións en Java e indica o valor final da variable *dato* se inicialmente vale 2:

- (++dato) + dato
- dato + (dato++)
- (dato > 2) && (dato++ < 10)
- (dato > 2) || (dato++ < 10)
- ! (dato++ < 2)

Solución

Para resolver o exercicio o alumno debe repasar a orde de precedencia nas expresións:

- (++dato) + dato

  O resultado é 6 e dato ten o valor 3

- dato + (dato++)

  O resultado é 4 e dato ten o valor 3

- (dato > 2) && (dato++ < 10)

  O resultado é *false* e dato ten o valor 3
- (dato > 2) || (dato-- < 10)

  O resultado é *true* e dato ten o valor 1
- ! (dato++ > 2)

  O resultado é 6 *true* e dato ten o valor 3

2.3.5 Tarefa 5. Fluxo estándar de datos

Nesta tarefa o alumnado amosará por pantalla o resultado dunha operación realizada con datos introducidos por teclado.

Enunciado

Crea un proxecto en Java denominado *SumaDecimais*, que amose por pantalla o resultado de sumar dous números en coma flotante introducidos por teclado.

Para a lectura de datos por teclado debes empregar a clase *Scanner*.

Solución

Para resolver o exercicio accedemos á páxina https://docs.oracle.com/javase/9/docs/api/java/util/Scanner.html e buscamos os métodos da clase *Scanner* que permitan facer a lectura de datos en coma flotante, e atopamos os métodos *nextFloat()* e *nextDouble()*.

Polo tanto, unha posible solución sería:

```
/**
   * Programa que suma dous números decimais usando a clase Scanner
*/
public class SumaDecimais {

 public static void main(String[] args) {
        java.util.Scanner scan = new java.util.Scanner(System.in);

        System.out.print("\n Introduce o primeiro sumando:");
        double primeiroSumando=scan.nextDouble(); //Lemos o primeiro sumando
        System.out.print("\n Introduce o segundo sumando:");
        double segundoSumando=scan.nextDouble();  //Lemos o segundo sumando

         System.out.println("\n\n A suma é " + (primeiroSumando + segundoSu-
mando));
 }
}
```

3. Materiais

3.1 Textos de apoio ou de referencia

- Especificación da linguaxe Java: http://docs.oracle.com/javase/specs/
- Java SE 9 API Documentation: https://docs.oracle.com/javase/9/docs/api/index.html
- The Java Tutorials: https://docs.oracle.com/javase/tutorial/
- A. GARCÍA Y BELTRÁN. *Programación con Java 7*. Vision Libos. Ed. 2012.
- J. SÁNCHEZ ASENJO. *Fundamentos de programación*. http://jorgesanchez.net. Licenza CC BY-NC-SA. 2008
- F. Javier Moldes. *Manual imprescindible Java SE 9*. Anaya Multimedia. Ed. 2017.
- Blog Programación Java: http://puntocomnoesunlenguaje.blogspot.com.es
- Wikipedia. https://www.wikipedia.org

3.2 Recursos didácticos

- Apuntamentos na aula virtual do centro
- Ordenador persoal, con navegador web e conexión a Internet
- Máquinas virtuais Windows/Linux
- Software necesario para a realización das tarefas.
- Proxector.