

Tema 2: Tipos de datos básicos

Introducción a la Programación
Grado en Ingeniería Informática, EPI Gijón

{jdiez,oluaces,juanjo}@uniovi.es

Departamento de Informática - Universidad de Oviedo en Gijón





Objetivos

- Diferenciar entre variables y constantes y su uso en los programas.
- Ser capaz de declarar y usar una constante o una variable, eligiendo el tipo adecuado.
- Utilizar el operador de asignación para asignar valores a variables.
- Dominar el uso de los operadores matemáticos (+, −, *, ...) en expresiones con reales y enteros.
- Entender y saber aplicar los conceptos de precedencia y asociatividad de los operadores.
- Ser capaz de imprimir datos en la consola y leer datos de teclado.



Contenidos

- | | |
|-------------------|------------------|
| 1 Variables | 6 E/S formateada |
| 2 Tipos de datos | 7 Tipos reales |
| 3 Tipos enteros | 8 Constantes |
| 4 Caracteres | 9 Asignación |
| 5 Identificadores | 10 Expresiones |



Programa: Suma dos números enteros

Enunciado

Enunciado

Realizar un programa que permita al usuario introducir dos números enteros por teclado y muestre su suma en la pantalla.

Nuevos conceptos:

- 1 Trabajar con datos enteros dentro de un programa.
- 2 Leer datos del teclado.
- 3 Realizar operaciones matemáticas con esos datos.
- 4 Imprimir información con formato en la pantalla.



Programa: Suma dos números enteros

Análisis

SumaDosNúmeros

```
+ main(String[])
```

Algoritmo 1 SumaDosNúmeros - main()

```
//PRE: el usuario escribirá dos números enteros  
Leer los números enteros del teclado  
Calcular su suma  
Mostrar la suma en pantalla  
//POS: muestra la suma de los dos enteros
```

En este caso el algoritmo es bastante simple:

- Consiste en una **secuencia de acciones**.
- Lo más importante es **el orden en el que se ejecutan**.
- Si se cambiara el orden de las acciones el programa NO produciría el resultado correcto.



Programa: Suma dos números enteros

Código fuente: SumaDosNúmeros.java

```
1 import java.util.Scanner;

3 /** Lee dos números enteros de teclado y muestra su suma en la consola
4  * @author los profesores de IP */
5 public class SumaDosNúmeros {

6     public static void main(String[ ] args) {
7         int número1; //Declaramos dos variables enteras
8         int número2;
9         //Objeto Scanner asociado con el teclado
10        Scanner teclado= new Scanner(System.in);
11        //Leemos ambos enteros
12        System.out.print("Introduce dos números: ");
13        número1=teclado.nextInt();
14        número2=teclado.nextInt();
15        //Calculamos su suma
16        int suma=número1+número2;
17        //Mostramos el resultado en la pantalla
18        System.out.printf("Su suma es %d\n",suma);
19    }
20 }
21 }
```



Variables

¿Qué son y para qué sirven?

Variable

Nombre simbólico que un programa utiliza para referenciar una posición de la memoria. En dicha posición el programa guarda un dato que necesita para su funcionamiento.

Las variables son el mecanismo que permite a los programas almacenar datos en la memoria del ordenador. Se caracterizan por

- 1 el **nombre** o **identificador**, será lo que nos permitirá acceder dentro del código del programa al dato que almacena, y
- 2 el **tipo** de valores que pueden almacenar. Cada tipo de dato puede almacenar datos diferentes y realizar ciertas operaciones.



Variables

¿Cómo se declaran?

Declaración de una variable

```
tipo nombre [= valorinicial];
```

Tiene tres elementos:

- 1 Tipo:** indica la clase de valores que guardará la variable.
- 2 Nombre o Identificador:** cómo nos referiremos a esa variable dentro del programa.
- 3 Valor inicial** (opcional¹): primer valor que toma la variable.

Declaración de varias variables del mismo tipo

```
tipo nombre1 [= valorinicial], nombre2 [= valorinicial], ...;
```

¹Los [] indican elementos opcionales de la sintaxis y no se escriben



Tipo de dato

¿Qué es?

Tipo de dato

Se define mediante el conjunto de valores de que consta y el conjunto de operaciones que se pueden realizar con ellos.

Al escoger el tipo de una variable determinamos

- el **conjunto de valores** que la variable podrá tomar, y
- las **operaciones** que podremos hacer con esa variable.

En nuestro problema concreto, necesitamos un tipo que:

- permita almacenar números enteros, y
- sumar dos números enteros.



Tipos básicos en Java

Tipos necesarios en cualquier programa

Como todos los lenguajes de programación, Java viene con un serie de tipos predefinidos que son necesarios en cualquier programa.

Enteros	<code>byte, short, int, long</code>
Reales	<code>float, double</code>
Caracteres	<code>char</code>
Lógicos	<code>boolean</code>

En nuestro programa todas las variables son enteras, y las hemos declarado como `int`, el tipo más común de los enteros:

```
8      int número1; //Declaramos dos variables enteras
9      int número2;
17     int suma=número1+número2;
```



Tipos enteros

Distintos rangos de valores

Cuatro tipos diferentes que se diferencian en su rango de valores.

Tipo	Espacio	Rango de valores
byte	1 byte	-128 .. 127
short	2 bytes	-32768 .. 32767
int	4 bytes	-2147483648 .. 2147483647
long	8 bytes	-9223372036854775808 .. 9223372036854775807

Aspectos al escoger el tipo de una variable entera:

- No se deben escoger por sistema los más grandes (**int** o **long**).
- Antes al contrario, hay que escoger el más pequeño que pueda almacenar todos los valores factibles.
- En vectores grandes y si los valores lo permiten, se puede usar **byte** o **short** en lugar de **int** para ahorrar memoria.



Operadores matemáticos

Tipos enteros

Operador

Símbolo del lenguaje que permite realizar una operación.

Tipo	Operador	Operación	Ejemplo	
Aditivos	+	suma	$7 + 4$	11
	-	resta	$4 - 7$	-3
Multiplicativos	*	multiplicación	$7 * 4$	28
	/	división entera	$7/4$	1
	%	módulo o resto	$7 \% 4$	3

Los operadores matemáticos comparten dos características:

- Son **binarios**, necesitan dos operandos (**cardinalidad**).
- Producen un **resultado entero**.

Cardinalidad de un operador

Es el número de operandos que necesita dicho operador.



El tipo carácter: char

Representación

- Java representa los caracteres mediante el tipo básico **char**.
- Codificación: **Unicode**.
- Espacio: se representa mediante **2 bytes**.
- Rango: **0 .. 65535**.
- Constantes carácter: se representan entre **comillas simples**.

Formato	Ejemplo
Símbolo	'a'
Secuencia de escape	'\n'
Octal	\141 → 'a'
Hexadecimal (Unicode)	\u0061 → 'a'

- Las secuencias de escape sirven para representar caracteres especiales.



El tipo carácter: char

Operadores

- No se debe confundir el carácter '4' con el número 4. El código del carácter '4', es el número 52.
- Como se representan con enteros, los caracteres soportan los mismos operadores matemáticos que el resto de tipos enteros.
- Sí, ¡se pueden sumar dos caracteres!
- La suma de dos caracteres tendrá como el resultado el carácter que tenga cómo código la suma de los códigos de los caracteres que intervienen en la suma.

```
char c='A'+ '0';    //c valdrá 'q'. 65+48=113='q'
```

- Las letras minúsculas están separadas de las mayúsculas por 32, una potencia exacta de 2. Eso facilita las transformaciones entre minúsculas y mayúsculas.



Identificadores

Válidos y descriptivos

Reglas fundamentales:

- 1 Debe ser un **identificador válido** dentro del lenguaje Java:
 - Esto es obligatorio.
 - Todos los lenguajes tienen reglas para restringir los identificadores que se pueden usar.
 - Ejemplo: una variable puede llamarse **apellido1**, pero no **1apellido**.
- 2 Debe ser **descriptivo**, indicando a través del nombre la utilidad de ese elemento dentro del programa:
 - No es obligatorio, ...
 - pero sí recomendable, aumenta la claridad del programa y lo hace más comprensible para otros programadores.
 - Ejemplo: si una variable va a guardar la altura de algo, en lugar de llamarla **a**, es preferible que se llame **altura**.



Identificadores válidos

Reglas de Java

- Debe comenzar con una letra del alfabeto, el carácter de subrayado (`_`), o el signo dólar (`$`).
- Es decir, no puede empezar por un dígito.
- Los siguientes caracteres, pueden ser cualquier carácter del estándar Unicode, salvo espacios en blanco. Es decir, `á`, `é`, `ó`, `ú`, `í`, o nuestra querida `ñ`, son válidos.
- No puede haber espacios en blanco.
- No pueden coincidir con una palabra reservada del lenguaje.
- Son *case-sensitive*, `Longitud` y `longitud` son identificadores diferentes.

Ejemplos:

- Válidos: `_ancho` `años` `$saldo` `perímetro` π
- No válidos: `1número` `primer valor`

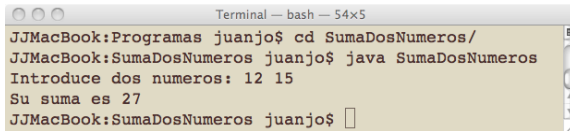


Comunicación Programa-Usuario

El usuario introduce datos, el programa devuelve resultados

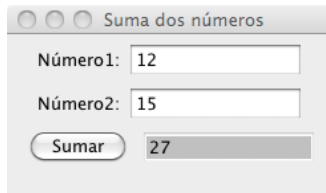
Formas de “comunicación” entre programas y usuarios:

- E/S estándar (consola): es el sistema más básico, los datos se leen directamente del teclado y se muestra en la pantalla.



```
Terminal — bash — 54x5
JJMacBook:Programas juanjo$ cd SumaDosNumeros/
JJMacBook:SumaDosNumeros juanjo$ java SumaDosNumeros
Introduce dos numeros: 12 15
Su suma es 27
JJMacBook:SumaDosNumeros juanjo$
```

- Entornos gráficos: el usuario introduce datos a través de ventanas (formularios) que tienen distintos campos.



Suma dos números

Número1:

Número2:



Clase Scanner

Leer de teclado

Para leer datos usando la clase `Scanner` necesitamos:

- 1 Incluir en nuestro programa la clase `Scanner` (directiva `import`).
- 2 Crear un objeto de la clase `Scanner`.
- 3 Asociarlo con el teclado, representado por `System.in`.

```
1 import java.util.Scanner;
10         //Objeto Scanner asociado con el teclado
11         Scanner teclado= new Scanner(System.in);
```

Una vez creado el objeto `teclado` podremos usarlo para leer nuestros datos enteros, llamando al método `nextInt()`:

```
14         número1=teclado.nextInt();
15         número2=teclado.nextInt();
```



Método printf()

Impresión con formato en consola

- El método `printf()` permite realizar una salida formateada, incluyendo tabuladores, justificaciones, etc.
- Permite escribir el contenido de variables de todos los tipos básicos.
- Tiene dos parámetros:
 - 1 Una cadena de texto que incluye instrucciones de formato, y
 - 2 la lista de las variables o expresiones que se quieren imprimir.
- Cada variable o expresión, normalmente, se inserta por orden en la posición en la que aparece información de formato en la cadena.

Ejemplos:

```
System.out.printf("Su suma es%d", suma);  
System.out.printf("\n%f \t%d", a, b);
```



Método printf()

Texto de formato

El texto de formato puede contener:

- 1 Texto normal: se imprimirá tal cual.
- 2 Especificadores de formato: implican la impresión de un dato.

formato	descripción
%d	imprime un entero en formato decimal
%f	imprime un número real en formato decimal
%c	imprime un carácter Unicode
%b	imprime "true" o "false" en función del valor booleano

- 3 Secuencias de escape: permiten imprimir caracteres especiales. Comienzan siempre por el carácter \ seguido de otro carácter.

formato	descripción
\n	salto de línea
\t	tabulador horizontal
\'	comilla doble
\\	barra invertida



Programa: Área de un círculo

Enunciado y análisis

Enunciado

Realizar un programa que permita al usuario introducir el radio de un círculo (número real) y muestre su área en la pantalla.

ÁreaCírculo

+ main(String[])

Algoritmo 2 ÁreaCírculo - main()

//PRE: el programa recibirá el radio (en formato real)

Leer el valor del radio del teclado

$\text{área} \leftarrow \text{radio}^2 * \pi$

Mostrar el área en pantalla

//POS: se muestra el área del círculo en la pantalla

Necesitamos hacer varias cosas nuevas:

- Usar números reales en lugar de enteros,
- calcular el cuadrado de un número, y
- representar la constante π .



Programa: Área de un círculo

Código fuente: ÁreaCírculo.java

```
1 import java.util.Scanner;

3 /** Lee de teclado el radio de un círculo y muestra su área en la consola
4  * @author los profesores de IP */
5 public class ÁreaCírculo {

7     public static void main(String[ ] args) {
8         final double PI=3.1416; //Constante para pi
9         double radio;           //variable para el radio
10        //Objeto Scanner asociado con el teclado
11        Scanner teclado= new Scanner(System.in);
12        //Leemos el radio
13        System.out.print("Introduce el radio: ");
14        radio=teclado.nextDouble();
15        double área;
16        área=radio*radio*PI;
17        //Mostramos el área en la pantalla
18        System.out.printf("El área es %f\n",área);
19    }
20 }
```



Tipos reales

Distintos rangos de valores

Tipo	Espacio	Rango de valores
float	4 bytes	$\pm 3.4 \times 10^{-38}$.. $\pm 3.4 \times 10^{38}$
double	8 bytes	$\pm 1.7 \times 10^{-308}$.. $\pm 1.7 \times 10^{308}$

- El tipo real más usado es **double**, al tener más precisión.
- Pero, para guardar muchos reales (vector) se debe emplear **float** si tiene la precisión suficiente.
- Para leer números reales de teclado con la clase `Scanner` se utilizan los métodos `nextFloat()` y `nextDouble()`.
- Para imprimir con `printf()` se usan los indicadores de formato `%e` o `%f`.

```

14         radio=teclado.nextDouble();
18         System.out.printf("El área es %f\n", área);

```



Operadores matemáticos

Tipos reales

Tipo	Operador	Operación	Ejemplo	
Aditivos	+	suma	$4.8 + 2.1$	6.9
	-	resta	$2.1 - 4.8$	-2.7
Multiplicativos	*	multiplicación	$4.8 * 2.1$	10.08
	/	división real	$4.8 / 2.1$	2.29
	%	módulo o resto	$4.8 \% 2.1$	0.6

- tienen **cardinalidad binaria**, necesitan dos operandos, y
- producen un **resultado real**.

Importante

No existe el operador potencia. Debes usar la multiplicación o métodos que realicen la potencia (Math.pow)



Constantes

¿Qué son y para qué sirven?

Constante

Nombre simbólico que un programa utiliza para referirse a un valor que no cambiará durante la ejecución del programa.

Se caracterizan casi por las mismas cosas que una variable:

- 1 Un **identificador** para poder referirnos a ellas en el código del programa,
- 2 el **tipo** que tiene el valor que representa la constante, y
- 3 el **valor de la constante** (no podrá cambiar). Es obligatorio que tengan un valor desde su declaración.

Importante

Una vez creada un constante, nunca se podrá cambiar su valor, como se hace con las variables



Constantes

¿Cómo se declaran?

Declaración de una constante

final tipo nombre = *valor*;

```
8      final double PI=3.1416; //Constante para pi
```

Hay dos diferencias respecto a la declaración de una variable:

- 1 Se incluye al principio la palabra reservada **final**.
- 2 El **valor** no es opcional. Podrá ser cualquiera de los valores del rango de ese tipo.

Uso: igual que las variables, se indica su nombre

```
16      área=radio*radio*PI;
```

Declaración de varias constantes del mismo tipo

final tipo nombre1 = *valor1*, nombre2 = *valor2*, ...;



Constantes sin nombre

Usar una constante sin declararla

- Es posible utilizar un valor constante sin crear una constante.

$$\text{área} = \text{radio} * \text{radio} * 3.1416;$$

- Aunque no se declaren, las constantes sin nombre tienen tipo.
- Como regla general, se suelen usar cuando se cumplen simultáneamente dos condiciones:
 - 1 Ese valor solo aparece una vez en el programa.
 - 2 La constante realmente no tiene un significado dentro del programa.
- Problemas de las constantes sin nombre:
 - 1 A veces, dan la impresión de ser números mágicos.
 - 2 Dificultan el mantenimiento. ¿Qué pasa si hay que cambiar su valor alguna vez?



Operador Asignación

Qué es y cómo se utiliza

Asignación

Expresión que permite cambiar el contenido de una variable. Hace uso del operador de asignación `=`.

Asignación

`variable = expresión;`

16 `área=radio*radio*PI;`

- El elemento de la izquierda (**left-value**), debe ser obligatoriamente una **variable**.
- El de la derecha (**right-value**), puede ser cualquier *expresión*.

Importante

El tipo de la variable de la parte izquierda y de la expresión de la parte derecha deben ser compatibles



Expresiones

¿Cómo se ejecutan?

Expresión

Es una combinación de operadores y operandos (constantes, variables y llamadas a métodos) que produce un resultado (valor) de un cierto tipo.

Varias cosas en esa definición:

- Se forman al mezclar los operadores con sus operandos.
- Todas las expresiones producen siempre un **valor**.
- Ese valor será, como es lógico, de algún **tipo**.
- Tanto el tipo como el valor dependerá de los operandos y de los operadores que se usen.

La expresión de la sentencia de asignación de la línea 16 es:

- de tipo **double**, y
- su valor dependerá del valor de la variable **radio**.



Programa: Media de dos números

Enunciado y análisis

Enunciado

Realizar un programa que permita al usuario introducir dos números enteros y muestre su media (entera) en la pantalla.

MediaDosNúmeros

```
+ main(String[])
```

Algoritmo 3 MediaDosNúmeros - main()

```
//PRE: el programa recibirá dos números enteros
Leer el valor de los dos números enteros
media ←  $\frac{\text{número1} + \text{número2}}{2}$ 
Mostrar la media en pantalla
//POS: se muestra la media en la pantalla
```

Necesitamos comprender algunas cosas nuevas:

- Cómo se ejecutan exactamente las expresiones que tienen más de un operador,
- qué son la precedencia y la asociatividad, y
- entender mejor el operador de asignación.



Programa: Media de dos números

Código fuente: MediaDosNúmeros.java

```
1 import java.util.Scanner;

3 /** Lee dos números enteros de teclado y muestra su media
4  * @author los profesores de IP */
5 public class MediaDosNúmeros {

7     public static void main(String[ ] args) {
8         int número1; //Declaramos dos variables enteras
9         int número2;
10        //Objeto Scanner asociado con el teclado
11        Scanner teclado= new Scanner(System.in);
12        //Leemos ambos enteros
13        System.out.print("Introduce dos números: ");
14        número1=teclado.nextInt();
15        número2=teclado.nextInt();
16        //Calculamos su media
17        int media;
18        media=(número1+número2)/2;
19        //Mostramos el resultado en la pantalla
20        System.out.printf("Su media es %d\n",media);
21    }
22 }
```



Precedencia

Orden de evaluación de los operadores de una expresión

Cuando en el pseudocódigo hemos escrito la expresión:

$$media \leftarrow \frac{número1 + número2}{2}$$

- Se entiende que antes de dividir, se suman los números.
- Aplicamos la sintaxis y la semántica de las expresiones matemáticas.
- Lo mismo ocurre con las expresiones dentro de un lenguaje de programación.

En nuestro programa, podríamos haber escrito:

```
media = número1+número2 / 2; //No calcula la media!
```

pero sin embargo hemos puesto unos paréntesis:

```
18 media=(número1+número2)/2;
```



Precedencia

Definición y ejemplo

Precedencia

Sirve para decidir qué operador debe aplicarse primero cuando en una expresión aparecen varios operadores de distintos grupos.

Ejemplo: si tenemos la expresión $4 + 5 * 7$,

- Aparecen dos operadores $*$ y $+$ que pertenecen a distintos grupos, uno es multiplicativo y el otro aditivo.
- El operador $*$ tiene más precedencia que el $+$.
- El resultado al evaluar la expresión será 39.
- Si quisiéramos que la suma se hiciera primero deberíamos usar paréntesis.

$$\begin{array}{rcccl} 4 & + & 5 & * & 7 \\ & & & \downarrow & \\ 4 & + & & 35 & \\ & \downarrow & & & \\ & 39 & & & \end{array}$$



Precedencia

Los paréntesis permiten cambiar el orden de evaluación

18

`media=(número1+número2)/2;`

grupo	operadores	asociatividad
agrupamiento	<i>(expresión)</i>	ID
multiplicativos	* / %	ID
aditivos	+ -	ID
asignación	= += -= *= /= %=	
	&= ^= = <<= >>= >>>=	DI

Aparecen CUATRO operadores =, (), + y / que pertenecen a distintos grupos:

- 1 Primero el agrupamiento, (), y dentro de él, la + que es su única operación,
- 2 después el operador /,
- 3 y por último la asignación =.



Asociatividad

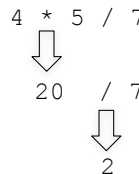
Orden de ejecución de los operadores de un mismo grupo

Asociatividad

Sirve para decidir qué operador debe aplicarse primero cuando en una expresión aparecen varios operadores del mismo grupo (con la misma precedencia).

Ejemplo: si tuviéramos la expresión $4 * 5 / 7$,

- los operadores que aparecen pertenecen al mismo grupo, son multiplicativos,
- luego tienen la misma precedencia.
- Para decidir cuál se aplicará primero debemos recurrir a la asociatividad de ese grupo.
- Los multiplicativos tiene asociatividad de izquierda a derecha (ID).
- El resultado al evaluar la expresión será 2.





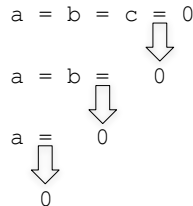
Asociatividad

Los operadores de asignación

- Todos los grupos de operadores tienen asociatividad de izquierda a derecha (ID),
- salvo, los de asignación y los unarios, que es de derecha a izquierda (DI).
- El hecho de que la asignación tenga asociatividad DI permite asignar a varias variables el mismo valor.

Ejemplo: expresión $a = b = c = 0;$

- Si la asociatividad de la asignación fuera ID, entonces a la variable a se le asignaría b , a esta c y c valdría 0.
- Al ser de DI, todas valdrán 0.



Importante

El valor que produce una asignación es el valor asignado, esto es el valor de la expresión de la derecha



Conversiones

Cambiar el tipo de una expresión

Enunciado

Realizar un programa que permita al usuario introducir dos números enteros y muestre su media (real) en la pantalla.

Debemos tener en cuenta varias reglas semánticas:

- Cuando los dos operandos de $/$ son enteros o reales, la división es respectivamente entera o real.
- ¿Qué pasaría si uno fuera real y el otro entero?
- En ese caso, la división sería real.
- ¿Por qué? El motivo son las conversiones automáticas.
- Luego, hay dos soluciones para nuestro problema:
 - 1 Cambiar ambos operandos para que sean reales.
 - 2 Cambiar uno y que el otro se convierta automáticamente.



Operador de conversión

Cambiar el tipo de una expresión

Operador de conversión

`(tipo) expresión`

- El valor de la *expresión* se convertirá (cambiará) al **tipo** indicado entre paréntesis.
- ¡Ojo! El operador de conversión es uno de los de más precedencia

No todas las siguientes expresiones producirían el resultado correcto:

```
media = (double) (número1+número2) / (double) 2;  
media = (double) (número1+número2) / 2.0;  
media = (double) ( (número1+número2) / 2 );
```

¿Por qué?



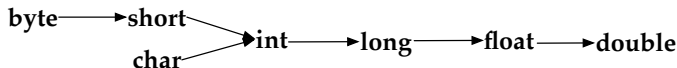
Conversiones automáticas

No producen pérdida de información

Conversión automática

Conversión que se produce implícitamente, sin que la indique el programador, cuando en una expresión sus operandos pertenecen a diferentes tipos básicos.

- En todos los operadores binarios matemáticos, relacionales o de comparación, los dos operandos deben ser del mismo tipo.
- Cuando son de tipos diferentes, el operando del tipo “menor” se convierte al tipo “mayor”.
 - El rango de valores del tipo menor está “contenido” en el rango de valores del tipo mayor.
- En una asignación, solo la parte derecha se puede promover.
- **En una conversión automática NO se pierde información.**





Conversiones automáticas

Ejemplos

```
media = (double) (número1+número2) / 2;  
media = (número1+número2) / 2.0 ;
```

En ambos casos la división será real:

- La idea es que solamente uno de los dos operandos sea **double**.
- El otro se convertirá automáticamente gracias a una conversión automática.
- En el primer caso usamos el operador de conversión para el numerador y la constante entera 2 se convertirá automáticamente a **double**.
- En el segundo, al ser el denominador una constante **double**, lo que se convierte automáticamente es el numerador.



Programa: Media (real) de dos números

Código fuente: MediaRealDosNúmeros.java

```
1 import java.util.Scanner;

3 /** Lee dos números enteros de teclado y muestra su media (real)
4  * @author los profesores de IP */
5 public class MediaRealDosNúmeros {

7     public static void main(String[ ] args) {
8         int número1; //Declaramos dos variables enteras
9         int número2;
10        //Objeto Scanner asociado con el teclado
11        Scanner teclado= new Scanner(System.in);
12        //Leemos ambos enteros
13        System.out.print("Introduce dos números: ");
14        número1=teclado.nextInt();
15        número2=teclado.nextInt();
16        //Calculamos su media
17        double media;
18        media=(número1+número2)/2.0;
19        //Mostramos el resultado en la pantalla
20        System.out.printf("Su media es %f\n",media);
21    }
22 }
```