

2. Variables

2.1 Definición y tipos de variables

Una variable es un contenedor de información. Imagina una variable como una cajita con una etiqueta que indica su nombre, una cajita en la que se puede introducir un valor. Las variables pueden almacenar valores enteros, números decimales, caracteres, cadenas de caracteres (palabras o frases), etc. El contenido de las variables puede cambiar durante la ejecución del programa, de ahí viene el nombre de “variable”.

Java es un lenguaje fuertemente tipado, es decir, es necesario declarar todas las variables que utilizará el programa, indicando siempre el nombre y el tipo de cada una.

El nombre que se le da a una variable es muy importante; intenta usar siempre nombres significativos que, de alguna forma, identifiquen el contenido. Por ejemplo, si usas una variable para almacenar el volumen de una figura, una buena opción sería llamarla `volumen`; si tienes una variable que almacena la edad de una persona, lo mejor es llamarla `edad`, y así sucesivamente.

Un programa se lee y se depura mucho mejor si los nombres de las variables se han elegido de forma inteligente.

¿Podrías averiguar qué hace la siguiente línea de código?

```
x = vIp3 * Weerty - zxc;
```

Ahora observa el mismo código pero con otros nombres de variables.

```
precioTotal = cantidad * precio - descuento;
```

Se entiende mucho mejor ¿verdad? No me cansaré de repetir lo importante que es nombrar correctamente las variables. Piensa que un programador pasa mucho más tiempo leyendo código que programando. Leer código fuente en Java bien escrito debería ser casi tan fácil y natural como leer una novela.

Escribiremos los nombres de variables en formato *lowerCamelCase*. La primera letra se escribe en minúscula y, a continuación, si se utiliza más de una palabra, cada una de ellas empezaría con mayúscula. Por ejemplo, `edadMin` es un buen nombre para una variable que almacena la edad mínima a la que se puede acceder a un sitio web. Observa que hemos usado una mayúscula para diferenciar dos partes (edad y Min). Puede que en algún libro también encuentres nombres de variables con el carácter

de subrayado (`_`) de tal forma que el nombre de la variable sería `edad_min`, pero como hemos comentado anteriormente, en este libro nos ceñimos al [estándar de Google¹](#), que exige el formato *lowerCamelCase*.

No se permiten símbolos como \$, %, @, +, -, etc. Puedes usar números en los nombres de variables pero nunca justo al principio; `5x` no es un nombre válido pero `x5` sí lo es.

No se debe poner una letra mayúscula al comienzo del nombre de una variable para no confundirla con una clase (los nombres de las clases comienzan por mayúscula).



Nombres de variables

Los nombres de las variables deben ser **significativos**, es decir, deben indicar perfectamente qué información contienen.

Enteros (`int` y `long`)

Las variables que van a contener números enteros se declaran con `int`. Veamos un ejemplo.

```
/**
 * Uso de variables enteras
 *
 * @author Luis J. Sánchez
 */
public class VariablesEnteras {
    public static void main(String[] args) {
        int x; // Declara la variable x como entera

        x = 5; // Asigna el valor 5 a la variable x

        // Muestra el valor de x
        System.out.println("El valor actual de mi variable es " + x);

        x = 7; // Asigna el valor 7 a la variable x

        // Muestra de nuevo el valor de x
        System.out.println("y ahora es " + x);
    }
}
```

Si pretendemos almacenar valores muy grandes en una variable, usaremos el tipo `long` en lugar de `int`.

¹<https://google.github.io/styleguide/javaguide.html>

Números decimales (double y float)

Usamos los tipos `double` o `float` cuando queremos (o esperamos) almacenar números con decimales en las variables. La diferencia está en la precisión, las variables de tipo `double` tienen mayor precisión que las de tipo `float`.

Ciñéndonos al estándar de Google, no daremos por válidas definiciones de variables como la siguiente (aunque funcionaría sin ningún problema).

```
double x, y;
```

Cada variable se debe definir en una línea diferente. Lo siguiente sí sería correcto.

```
double x;  
double y;
```

A continuación tienes un ejemplo completo.

```
/**  
 * Uso de variables que contienen números decimales  
 *  
 * @author Luis J. Sánchez  
 */  
public class VariablesConDecimales {  
    public static void main(String[] args) {  
        double x; // Se declaran las variables x e y  
        double y; // de tal forma que puedan almacenar decimales.  
  
        x = 7;  
        y = 25.01;  
  
        System.out.println(" x vale " + x);  
        System.out.println(" y vale " + y);  
    }  
}
```

Como puedes ver, también se pueden almacenar números enteros en variables de tipo `double`.

Cadenas de caracteres (String)

Las cadenas de caracteres se utilizan para almacenar palabras y frases. Todas las cadenas de caracteres deben ir entrecomilladas mediante el símbolo de comillas dobles (").

```
/**
 * Uso del tipo String
 *
 * @author Luis J. Sánchez
 */
public class UsoDeStrings {
    public static void main(String[] args) {
        String miPalabra = "cerveza";
        String miFrase = "¿dónde está mi cerveza?";

        System.out.println("Una palabra que uso con frecuencia: " + miPalabra);
        System.out.println("Una frase que uso a veces: " + miFrase);
    }
}
```

Un cadena de caracteres puede contener cero (cadena vacía) o más caracteres. Nos podríamos encontrar con un código como el siguiente:

```
String cadenaInicial = "";
```

En un principio, parece absurdo disponer de una cadena de caracteres que no contiene ningún carácter, pero más adelante verás la utilidad. Por ejemplo, imagina que estás programando el juego del ahorcado y necesitas tener una cadena donde vayas guardando todas las letras que va probando el jugador; en este caso, cuando empiece el juego, no se ha probado ninguna letra, por tanto, esa cadena estará inicialmente vacía y posteriormente se irá rellenando.

Como puedes ver, en una cadena de caracteres se pueden almacenar signos de puntuación, espacios y letras con tildes.

Puede que te estés preguntando ¿por qué los tipos `int`, `long`, `double` y `float` empiezan por minúscula y `String` empieza por mayúscula? La respuesta es que `String` es en realidad una clase, no un tipo primitivo. Veremos las clases en profundidad en el [capítulo 9](#) que trata sobre la programación orientada a objetos.

Caracteres (`char`)

Un carácter suelto como una letra o un signo de puntuación se puede almacenar en una variable de tipo `char`. El carácter debe ir entrecomillado utilizando las comillas simples (`'`).

Hay que tener en cuenta que no es lo mismo `"a"` que `'a'`. Aunque el contenido en ambos casos es la letra `"a"`, lo primero es una cadena de caracteres y lo segundo es un carácter. En algunos lenguajes de programación se pueden usar indistintamente las comillas simples y las dobles pero en Java tienen un significado muy distinto.

```
/**
 * Uso del tipo char
 *
 * @author Luis J. Sánchez
 */
public class UsoDeChar {
    public static void main(String[] args) {
        char letra1 = 'c';
        char letra2 = 'a';
        char letra3 = 's';
        char letra4 = 'a';

        System.out.println("letra1: " + letra1);
        System.out.println("letra3: " + letra3);
        System.out.println("todas las letras juntas: " + letra1 + letra2 + letra3 + letra4);
    }
}
```

Veamos un comportamiento curioso del tipo `char`.

```
/**
 * Correspondencia entre los tipos char e int
 *
 * @author Luis J. Sánchez
 */
public class CharComoInt {
    public static void main(String[] args) {
        char letra1 = 'a';
        char letra2 = 'b';

        System.out.println(letra1);
        System.out.println(letra2);
        System.out.println(letra1 + letra2);
        System.out.println(letra1 + "" + letra2);
    }
}
```

El resultado del programa anterior se muestra a continuación.

```
a
b
195
ab
```

Existe una correspondencia entre los tipos `char` e `int` de tal forma que la suma de dos caracteres se considera la suma de dos enteros. En el programa anterior, la suma

`letra1 + letra2` en realidad es la suma de los códigos ASCII de la letra “a” y de la letra “b” que son el 97 y el 98 respectivamente. En Java se podrían escribir cosas como `'a' + 7` sin que diera ningún error. Fíjate que en el ejemplo, para mostrar el contenido de las variables `letra1` y `letra2` de forma consecutiva se recurre al truco de pintar en medio una cadena de caracteres vacía.

2.2 Resumen de tipos primitivos

Los tipos `int`, `long`, `double` y `float` vistos anteriormente son los llamados “primitivos”. Estos tipos forman parte del propio lenguaje y están disponibles desde las primeras versiones. En Java hay 8 tipos primitivos.

TIPO	DESCRIPCIÓN	TAMAÑO	EJEMPLO
<code>boolean</code>	verdadero o falso	1 bit	<code>boolean abierto = true;</code>
<code>byte</code>	número entero	8 bits	<code>byte repeticiones = 22;</code>
<code>char</code>	carácter	16 bits	<code>char letra = 'a';</code>
<code>short</code>	número entero	16 bits	<code>short pantalones = 22;</code>
<code>int</code>	número entero	32 bits	<code>int asistentes = 22;</code>
<code>long</code>	número entero	64 bits	<code>long poblacion = 22L;</code>
<code>float</code>	número con decimales	32 bits	<code>float nota = 9.5f;</code>
<code>double</code>	número con decimales	64 bits	<code>double precio = 22.55d;</code>

2.3 Operadores aritméticos

En Java se puede operar con las variables de una forma muy parecida a como se hace en matemáticas. Los operadores aritméticos de Java son los siguientes:

OPERADOR	NOMBRE	EJEMPLO	DESCRIPCIÓN
<code>+</code>	suma	<code>20 + x</code>	suma dos números
<code>-</code>	resta	<code>a - b</code>	resta dos números
<code>*</code>	multiplicación	<code>10 * 7</code>	multiplica dos números
<code>/</code>	división	<code>altura / 2</code>	divide dos números
<code>%</code>	resto (módulo)	<code>5 % 2</code>	resto de la división entera
<code>++</code>	incremento	<code>a++</code>	incrementa en 1 el valor de la variable
<code>--</code>	decremento	<code>a--</code>	decrementa en 1 el valor de la variable

A continuación tienes un programa que ilustra el uso de los operadores aritméticos.

```
/**
 * Uso de los operadores aritméticos
 *
 * @author Luis J. Sánchez
 */
public class UsoDeOperadoresAritmeticos {
    public static void main(String[] args) {
        int x;
        x = 100;

        System.out.println(x + " " + (x + 5) + " " + (x - 5));
        System.out.println((x * 5) + " " + (x / 5) + " " + (x % 5));
    }
}
```

2.4 Asignación de valores a variables

La sentencia de asignación se utiliza para dar un valor a una variable. En Java (y en la mayoría de lenguajes de programación) se utiliza el símbolo igual (=) para este cometido. Es importante recalcar que una asignación no es una ecuación. Por ejemplo $x = 7 + 1$ es una asignación en la cual se evalúa la parte derecha $7 + 1$, y el resultado de esa evaluación se almacena en la variable que se coloque a la izquierda del igual, es decir, en la x , o lo que es lo mismo, el número 8 se almacena en x . La sentencia $x + 1 = 23 * 2$ no es una asignación válida ya que en el lado izquierdo debemos tener únicamente un nombre de variable.

Veamos un ejemplo con algunas operaciones y asignaciones.

```
/**
 * Operaciones y asignaciones
 *
 * @author Luis J. Sánchez
 */
public class Asignaciones {
    public static void main(String[] args) {
        int x = 2;
        int y = 9;

        int sum = x + y;
        System.out.println("La suma de mis variables es " + sum);

        int mul = x * y;
        System.out.println("La multiplicación de mis variables es " + mul);
    }
}
```

```
}  
}
```

2.5 Conversión de tipos (*casting*)

En ocasiones es necesario convertir una variable (o una expresión en general) de un tipo a otro. Simplemente hay que escribir entre paréntesis el tipo que se quiere obtener. Experimenta con el siguiente programa y observa los diferentes resultados que se obtienen.

```
/**  
 * Conversión de tipos  
 *  
 * @author Luis J. Sánchez  
 */  
public class ConversionDeTipos {  
    public static void main(String[] args) {  
        int x = 2;  
        int y = 9;  
        double division;  
  
        division = (double) y / (double) x;  
  
        // Descomenta la siguiente línea y observa cómo cambia el resultado.  
        // division = y / x;  
  
        System.out.println("El resultado de la división es " + division);  
    }  
}
```

Primero tenemos

```
division = (double) y / (double) x
```

`(double) y` convierte el valor de `y` en un `double`, es decir, en un número con decimales y `(double) x` hace lo propio con `x`, por tanto, la operación de división quedaría como `9.0 / 2.0`. Al tratarse de una división entre números decimales, Java entiende que el resultado debe tener también decimales y usa toda la precisión disponible en el tipo. El resultado que se guarda en la variable `division` es `4.5` que es el valor correcto.

Descomentando la línea 15, tendríamos

```
division = y / x
```

Ahora la división es $9 / 2$. Java ve una división entre dos números enteros, así que el resultado que ofrece es otro número entero después de despreciar los decimales si los hubiera. El resultado que da Java es la división entera, o sea, 4. El siguiente paso es una asignación, tendríamos `division = 4`. Como la variable `division` es de tipo `double`, el valor 4 se guarda en realidad como 4.0; es curioso ¿verdad?



Usa el *casting* para conservar los decimales en las divisiones con enteros

La división entre dos números enteros es otro número entero. Para conservar los decimales de la división hay que hacer un *casting* a `float` o `double`.