

## CAPÍTULO 3

/ /

### » ASIGNACIONES

#### Apilar y Montar

Para la mayoría de las personas comprender los conceptos básicos de la pila y el montón lo hace mucho más fácil de entender temas como el paso de argumentos, el polimorfismo, los hilos, las excepciones y la recolección de basura. En esta sección, nos limitaremos a una visión general, pero ampliaremos estos temas varias veces más a lo largo del libro.

En su mayor parte, las diversas piezas (métodos, variables y objetos) de los programas de Java viven en uno de los lugares en la memoria la pila o el montón.

### Literales, Asignaciones y Variables

Una literal primitiva es meramente una representación de fuente de código de los tipos de datos primitivos, en otras palabras, un entero, un punto flotante, booleano o carácter que escribes en el código.

Lo siguiente son ejemplos de literales primitivos.

'b' //char literal

42 //Integer literal

false //boolean literal

254678.343 //double literal

### Enteros literales

Hay tres formas de representar números enteros en el lenguaje Java: decimal (base 10), octal (base 8) y hexadecimal (base 16)

## Decimales Literales

Los enteros decimales no necesitan explicaciones; se han estado usando desde un grado uno o anterior.

## Literales Octales

Los enteros octogonales solo usan los dígitos del 0 al 7. En Java, se puede representar un entero en forma octal poniendo un cero de lado izquierdo del número.

## Literales Hexadecimales

Hexadecimal (hex para short) son construidos usando 16 distintos símbolos. Porque nunca inventó un sólo dígito simbólico para los números del 10 al 15, se usa caracteres alfábéticos para representar esos dígitos.

Java aceptaría letras mayúsculas o minúsculas para los dígitos adicionales se le permite hasta 16 dígitos en un número hexadecimal, sin incluir el prefijo 0x o la extensión del sufijo opcional L, que se explicará más adelante.

Para el caso de un long en hexadecimal, se pone una 'L' o 'l' dependiendo del caso, si no tiene 0x, se pone la 'L' al final del número, del caso contrario la 'l'

## Punto Flotantes literales

Los números de punto flotante se definen como un número, un símbolo decimal y más números que representan la fracción.

## Literales Booleanas

Un valor booleano solo puede ser definido como verdadero o falso. Aunque en C (y algunos otros lenguajes) esto es común usar números para representar true o false, esto no funcionaría en Java.

## Caracteres Literales

Un carácter literal es representado por un solo carácter en comillas simples.

```
char a = 'd';
```

```
char b = '0';
```

También puede escribir el valor Unicode del carácter, usando la notación Unicode de prefijar el valor lu de la siguiente manera:

```
char letter = '\u004E'; // La letra 'N'
```

Recuerde, los caracteres son sólo enteros sin signo de 16 bits debajo del capó.

## Valores Literales para Strings

Un string es una representación de una fuente de código de valores de objetos String.

## Operadores de Asignación

El operador de asignación en Java (como en otros lenguajes) permite asignar valores a las variables. Si una variable tiene un valor previo y se realiza una asignación, el valor que tenía se reemplazará con el nuevo.

Asignar un valor a una variable parece bastante sencillo, simplemente se asignan las cosas en el lado derecho del "=" a la variable a la izquierda.

### Asignamientos Primitivos

El signo igual (=) se usa para asignar un valor a una variable y se denomina inteligentemente operador de asignación.

En realidad, hay 12 operadores de asignación, pero solo cinco son más comúnmente utilizados.

`int x = 7;`

`int y = x + 2;`

`int z = x - y;`

### Funcióñ Primitiva

Los casts pueden ser implícitos o explicitos. Una conversión implícita significa que no tiene que escribir código para la conversión. La conversión de gran valor en contenedor pequeño se conoce como estrechamiento y requiere una conversión explicita, donde le dice al compilador que está consciente del peligro y acepta la responsabilidad total.

#### Conversión Implícita

`int a = 100;`

`long b = d; // conversión implícita.`

#### Conversión Explícita

`float a = 100.00f;`

`int b = (int)a; // Reporto explícito, el flotador podría  
// perder información`

## Asignación de Números Punto-fLOTantes

los números de punto flotante tienen un comportamiento de asignación ligeramente diferente al de los tipos enteros. Primero, se debe de saber que cada literal de punto flotante es <sup>implicitamente</sup> un double (64 bits), no un flotante.

las siguientes asignaciones compilaría

```
float f = (float) 32.3;
```

```
float g = 32.3f;
```

```
float h = 32.3f;
```

## Asignar una literal que es demasiado grande para la variable

Si intentamos asignar un valor literal que es demasiado grande para la variable, tendríamos un error en el compilador

```
byte a = 128; //byte solo puede valores mínimos de 127
```

El presente código marca un error algo como:

```
TestBytes.java:S: possible loss of precision
```

```
found : int
```

```
required : byte
```

```
byte d = 128;
```

Podemos corregirlo con un casteo

```
byte a = (byte) 128;
```

## Asignación una variable primitiva a otra variable primitiva

La asignación de una variable hacia otra consta en asignar el valor de una hacia la otra como una copia de esta misma.

## Asignación de variables de Referencia

Puede asignar un objeto recién creado a una variable de referencia de objeto de la siguiente manera:

```
Boton b = new Boton();
```

La linea anterior hace tres cosas clave

- \* Hace una variable de referencia llamada b, de tipo botón.

- \* Crea un nuevo objeto botón en el montón

- \* Asigna el objeto botón recién creado a la variable de referencia b.

Puede además asignar null para un objeto de variable de referencia, la cual simplemente significa que la variable está referenciando a algún objeto:

```
Boton c = null;
```

## Variabile de Alcance

El alcance es una propiedad de las variables que hace referencia a su visibilidad (aquella región del programa donde la variable puede utilizarse). Los distintos tipos de variables, tienen distintas reglas de alcance.

Las variables de alcance son aquellas variables que se encuentran en una clase.

## Usando una variable o Elemento Array que es inicializado y sin inicializar

Java nos da la opción de inicializar una variable declarada o dejarla sin inicializar. Cuando intentamos utilizar la variable sin inicializar,

podemos obtener un comportamiento diferente según el tipo de variable o matriz con la que tratamos (primitivos u objetos). El comportamiento también depende del nivel (alcance) en el que estamos declarando nuestra variable.

Una variable de instancia se declara dentro de la clase pero fuera de cualquier método o constructor, mientras que una variable local se declara dentro de un método (o en la lista de argumentos del método).

## Variabiles de Instancia primitivas y de tipo Objeto

Las variables de instancia (también llamadas variables miembro) son variables definidas en el nivel de clase. Eso significa que la declaración

de variable no se realiza dentro de un método, constructor o cualquier otro bloque de inicialización. Las variables de instancia se inicializan a un valor predeterminado cada vez que se crea una nueva instancia, aunque se les puede dar un valor explícito una vez que se haya completado los superconstructores del objeto.

<u>Tipo de Variable</u>	<u>Valor por defecto</u>
Referencia de objeto	Null (no referencia a ningún objeto)
Byte, short, int, long	0
Float, double	0.0
Boolean	false
Char	'\u0000'

### Variabes primitivas de instancia

Las variables primitivas de instancia son las variables definidas dentro de una clase como las características que posee el objeto y en donde se utilizan los tipos de datos primitivos.

### Variabes de instancia de Referencia de Objeto.

Son las variables definidas dentro de una clase como las características que posee el objeto y en donde se utilizan otros objetos.

### Variabes de instancia de arreglos

Una matriz es un objeto; por lo tanto, una variable de instancia de matriz que se declara pero no se inicializa explícitamente tendrá un

valor nulo, al igual que cualquier otra variable de instancia de referencia de objeto.

### Referencia de Objetos Locales

Las referencias de los objetos también se comportan de manera diferente cuando se declaran dentro de un método en lugar de como variables de instancia.

Con la referencia de objeto de variable de instancia puede llegar a salir pero sin dejar una referencia de objeto sin inicializar.

La variable de referencia de instancia siempre puede dar un valor nulo, hasta que este se inicialice. Pero una referencia local no da un valor por defecto, en otras palabras no se inicializan en null.

### Arreglos Locales

Al igual que cualquier otra referencia de objeto, a las referencias de matriz declaradas dentro de un método se les debe asignar un valor antes de usar. Eso sólo significa que debe declarar y construir la matriz. Sin embargo, no es necesario inicializar explícitamente los elementos de una matriz.

## Asignar una variable de referencia a otra

Con las variables primitivas, una asignación de una variable a otra significa que los contenidos (patrón de bits) de una variable se copian de otra.

### » Pasando variables a métodos

#### Pasando variables de objeto de Referencia

Cuando pasa una variable de objeto a un método, debe tener en cuenta que está pasando la referencia del objeto y no el objeto en sí.

Una copia de una variable significa que obtiene una copia de los bits en esa variable, por lo que cuando pasa una variable de referencia, esto pasando una copia de los bits que representan cómo llegar a un objeto específico.

#### Pasando Variables Primitivas

En este caso de paso de variables primitivas, si se realiza una copia de la variable de la cual esté protegiendo la integridad de esta misma pero sin que sea alterado el valor inicial de la que le fue asignado.

Las variables de tipos primitivos almacenan directamente un valor que siempre pertenece al rango de ese tipo. Por ejemplo, una variable int almacena un valor entero como 1, 2, 0, -1, etc.