VARIABLES, EXPRESIONES Y SENTENCIAS

Muchos de los lenguajes de programación utilizan los conceptos definidos en la arquitectura de von Newman para construir programas. En este capítulo conocerá algunos de esos elementos.

Todos en algún momento nos hemos encontrado resolviendo algún problema matemático o aplicando una fórmula para obtener algún valor. ¿Cómo se pueden representar eso valores o variables dentro de un lenguaje de programación?

VALORES Y TIPOS

Como seres humanos estamos acostumbrados a utilizar valores, es decir, números, letras, texto, fechas, horas, etc. Estamos tan cómodos con los valores que a veces pasamos por alto algunos detalles, como por ejemplo que los números pueden ser enteros o reales, positivos o negativos, así como tampoco nos percatamos que el texto no es más que una secuencia de letras que puede incluir signos.

También hemos aprendido a hacer operaciones utilizando esos valores, podemos sumar, restar, multiplicar o dividir, ordenar ascendente o descendente, etc. Es decir, los conceptos de valores y operaciones son algunas de nuestras primeras nociones.

¿Cómo se representan en Java los valores? Para responder a esa pregunta inicie revisando la siguiente tabla.

Tipo	Representación
Entero	-1, 0, 1
Real	-1.3, 0.0, 1.7
Texto	“Loja capital musical del Ecuador”
Letra - caracter	A’, ‘b’, ‘1’

Los valores enteros pueden ser positivos o negativos, al igual que los reales, note que en Java se utiliza el punto (.) como separador decimal. El texto se debe encerrar entre comillas dobles (“”), mientras que los caracteres se encierran entre comillas simples (‘’).

Es momento de trabajar con valores en Java, para ello abra su terminal o ventana de comandos y ejecute el comando *jshell*[1]. Ese comando producirá una salida similar a la

[1] Existe una versión Web de JShell, la puede encontrar en https://tryjshell.org

siguiente:

```
$ jshell
|  Welcome to JShell -- Version 11.0.2
|  For an introduction type: /help intro

jshell>
```

Ha iniciado *jshell* y ahora puede empezar a ingresar algunos valores, como muestra la imagen de la derecha. Ingrese un valor y presione la tecla *Enter*, así como se mencionó anteriormente utilice el punto para separar los decimales, comillas dobles para encerrar el texto y comilla siempre para letras.

```
jshell> 1
$1 ==> 1

jshell> 1.3
$2 ==> 1.3

jshell> "Hola mundo"
$3 ==> "Hola mundo"

jshell> 'A'
$4 ==> 'A'

jshell>
```

```
jshell> 1 + 5
$5 ==> 6

jshell> 3.1 + 4.4
$6 ==> 7.5

jshell> 6.0 / 1.5
$7 ==> 4.0

jshell> 4 / 0
|  Exception java.lang.ArithmeticException: / by zero
|        at (#8:1)
```

Ahora ejecute algunas operaciones básicas con valores numéricos, haga alguna suma entre dos valores, o alguna división, así como muestra la imagen de la izquierda. Ponga especial atención en la última operación. No olvide que no se puede hacer divisiones por 0 ya que eso daría como resultado el infinito, o como muestra la imagen se lanzaría una excepción.

Para salir de JShell es necesario ejecutar el comando */exit*, una vez que se ejecuta ese comando termina la ejecución del *jshell* y su terminal volverá a la normalidad.

Java es un lenguaje de programación que se caracteriza por ser fuertemente "*tipeado*", es decir que cualquier valor debe tener asociado un tipo de dato. Es por ello que en las siguientes líneas se estudiará los tipos de datos en Java.

Java tiene varios tipos de datos, pero empecemos por los siguientes cuatros tipos, cuando sea necesario se describirán el resto de tipos de datos

Tipo de dato	Java
Entero	int
Real	double
Caracter	char

Tipo de dato	Java
Texto	String

Ahora comprobará que dentro de JShell, Java determinó el tipo de dato de cada uno de los valores que ingresa, para ello en una ventana terminal ejecute el comando *jshell -v*. El argumento *-v* señala que retro-alimente cada una de las acciones ejecutadas.

```
$ jshell -v
|  Welcome to JShell -- Version 11.0.2
|  For an introduction type: /help intro

jshell>
```

Vuelva a ingresar valores, enteros, reales, texto y caracteres, como lo hizo anteriormente y observará cómo aparece información adicional que entre otras cosas indica el tipo de dato de cada uno de los valores ingresados, *int*, *double*, *String* o *char*. Además de los tipos de datos de seguro notó el texto que señala que se han creado las variables *$1*, *$2*, *$3* y *$4*.

```
jshell> 1
$1 ==> 1
|  created scratch variable $1 : int

jshell> 1.3
$2 ==> 1.3
|  created scratch variable $2 : double

jshell> "Hola mundo"
$3 ==> "Hola mundo"
|  created scratch variable $3 : String

jshell> 'A'
$4 ==> 'A'
|  created scratch variable $4 : char

jshell>
```

En el siguiente apartado aprenderá todo acerca sobre las variables utilizando el lenguaje de programación Java.

VARIABLES

Las variables son un mecanismo para almacenar los valores que un programa necesita para trabajar. Estos valores se almacenan en la memoria del computador, por lo que una variable es una forma sencilla de acceder a un espacio específico de la memoria.

Las primeras acciones que aprenderá es a declarar y asignar valores a una variable. Iniciemos con la declaración.

Para declarar una variable en el lenguaje de programación Java necesita de dos elementos, el tipo de dato de la variable y el nombre de la variable. La forma estándar se muestra a continuación.

<Tipo de dato> <nombre de la variable>

Es decir que cuando se declara una variable de un tipo de dato, esa variable podrá almacenar valores que correspondan únicamente a ese tipo dato, si trata de almacenar otro valor se lanzará un error. Veamos la declaración de variables en JShell.

```
jshell> int edad
edad ==> 0

jshell> double peso
peso ==> 0.0

jshell> String nombre
nombre ==> null

jshell> char genero
genero ==> ''

jshell>
```

Dentro de JShell al declarar una variable se asigna un valor por defecto según el tipo de dato, así a las variables de tipo de dato entero se asignará 0, a las de tipo de dato real 0.0, al texto *null* y a las de tipo carácter el carácter vacío. Estos valores se pueden modificar, pero para ello necesita conocer la operación de asignación.

Asignar un valor a una variable es una operación que dentro del lenguaje de programación Java utiliza el signo de igual (=). Para ello se necesita el nombre de la variable y el valor que será asignado a la variable. Si la variable tenía un valor anterior este se pierde.

<nombre de la variable>= <valor>

Para asignar un valor a la variable es necesario que esta haya sido declarada con anterioridad, además, el valor que se asignará deberá ser del mismo tipo de dato que se usó en su declaración, si no se cumple con estas requisitos se lanzará un error. Veamos un ejemplo de asignación.

```
jshell> edad = 28
edad ==> 28

jshell> peso = 74.3
peso ==> 74.3

jshell> nombre = "Jorge"
nombre ==> "Jorge"

jshell> genero = 'M'
genero ==> 'M'

jshell> edad = "18"
|  Error:
|  incompatible types: java.lang.String cannot be converted to int
|  edad = "18"
|         ^--^

jshell> isbn = 31123
|  Error:
|  cannot find symbol
|    symbol:   variable isbn
|  isbn = 31123
|  ^--^

jshell>
```

Las últimas dos asignaciones lanzan un error. En el primero se trata de asignar una cadena de texto a una variable declarada como entero. Mientras que, el segundo mensaje de error señala que se trata de usar una variable que no ha sido declarada. Así que si tiene errores como estos ya conoce las causas y cómo resolverlos.

También es posible realizar la declaración y asignación de las variables a esto generalmente se lo conoce como inicialización de una variable. La siguiente imagen muestra esto.

```
jshell> int edad = 28
edad ==> 28

jshell> double peso = 74.3
peso ==> 74.3

jshell> String nombre = "Jorge"
nombre ==> "Jorge"

jshell> char genero = 'M'
genero ==> 'M'
```

Por último es posible que Java infiera el tipo de dato de una variable, para ello es necesario asignar un valor inicial a la variable e identificarlas con la palabra *var*, para comprender mejor esto vea la siguiente porción de código.

```
jshell> var edad = 28
edad ==> 28
|  created variable edad : int

jshell> var peso = 74.3
peso ==> 74.3
|  created variable peso : double

jshell> var nombre = "Jorge"
nombre ==> "Jorge"
|  created variable nombre : String

jshell> var genero = 'M'
genero ==> 'M'
|  created variable genero : char
```

En cada una de las declaraciones no se especifica el tipo de dato de la variable, pero se utiliza el valor inicial para inferir cada uno de los tipos de datos. Esta forma de declaración e inicialización de variables es bastante útil y recuerde que únicamente se puede usar cuando se asigna un valor a una variable.

NOMBRES DE VARIABLES Y PALABRAS RESERVADAS

Java, así como todos los lenguajes de programación, posee estándares para nombrar todos los elementos que se declaran y se usan en un programa y las variables no son la excepción.

Una versión en español del estándar de programación para el lenguaje Java se puede encontrar en: https://www.um.es/docencia/vjimenez/ficheros/practicas/ConvencionesCodigoJava.pdf. En resumen, para las variables se sugiere las siguientes convenciones:

- Una variable se escribe en minúsculas. Java es un lenguaje “case sensitive” es decir que diferencia entre mayúsculas y minúsculas
- Debe iniciar con una letra, aunque podría iniciar con $ o _
- En caso de estar formada para más de una palabra, la primera letra, a partir de la segunda, se escribe en mayúscula.

En palabras de Robert C. Martin, los nombres de variables, además de seguir un estándar de programación, deben ser seleccionados de tal manera que representen la función que desempeña dentro del programa, pero en especial que revelen por qué existen, qué es lo que hacen y cómo usarlas.

Si bien una variable puede tener cualquier nombre, existe un conjunto de nombres que no se puede utilizar ya que tienen un significado dentro del lenguaje de programación. Un listado de las palabras reservadas o Keywords se describen en la siguiente tabla:

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while	true	false
null			

Ninguna de esas palabras se pueden utilizar como nombres de variables ya que son utilizadas por el lenguaje de programación para su funcionamiento.

Finalmente recordar que el uso de estándares de programación hace más legible un programa, de ninguna manera evita o genera errores en los programas.

OPERADORES Y OPERANDOS

Los operadores son símbolos que representan cálculos, mientras que los valores que el operador utiliza se denominan operandos. Así como la mayoría de seres humanos usted debe conocer algunos de los operadores que se utilizan. La siguiente tabla muestra un resumen de los operadores aritméticos de uso común en Java.

	Operador
Suma	+
Resta	-
Multiplicación	*
División	/
Asignación	=

	Operador
Módulo	%

En Java, como en la mayoría de lenguajes de programación, se utiliza un el asterisco (*) para señalar multiplicación. El operador módulo devuelve el residuo de una división entera, es decir sin parte decimal.

Un ejemplo del operador módulo se puede ver en la siguiente imagen. En ese caso se divide 7 para 4 y se devuelve el residuo que es 3.

```
jshell> 7 % 4
$1 ==> 3
```

Otra característica que puede resultar inexplicable es la división de dos número enteros, si se divide 5 para 2, en la aritmética tradicional la respuesta sería 2.5, pero como puede ver más abajo, esto no es así en Java. La razón se encuentra en los tipos de datos de los operadores, es decir, que cuando ambos operadores son del tipo de dato entero, el resultado de la división será un valor entero.

```
jshell> 5 / 2
$4 ==> 2
```

Para obtener un valor con decimales uno de los operadores debe ser del tipo de dato real, puede ser el numerador o del denominador. Analice las siguientes porciones de código

```
jshell> var a = 5.0
a ==> 5.0

jshell> var b = 2
b ==> 2

jshell> a / b
$10 ==> 2.5
```

```
jshell> var a = 5
a ==> 5

jshell> var b = 2.0
b ==> 2.0

jshell> a / b
$7 ==> 2.5
```

```
jshell> var a = 5
a ==> 5

jshell> var b = 2
b ==> 2

jshell> (double)a / b
$13 ==> 2.5
```

```
jshell> var a = 5
a ==> 5

jshell> var b = 2
b ==> 2

jshell> a / (double)b
$16 ==> 2.5
```

En las primeras dos (empezando por la izquierda), se cambia el numerador o el denominador a *double*, mientras que, en las dos últimas se utiliza el *casting* de datos para transformar la variable de entero a real. El *casting* se puede utilizar en cualquiera de los dos operandos y el resultado será el mismo.

Java posee otros operadores, pero por ahora los que revisó son suficientes para realizar un sin número de programas. Cuando sea necesario se comentarán los nuevos operadores.

EVALUACIÓN DE EXPRESIONES

Una expresión es una construcción compuesta por variables y operadores que se forma de acuerdo a la sintaxis del lenguaje y que dan como resultado un único valor.

Se utilizaron varias expresiones para asignar valores a variables o para realizar cálculos, recuerde que el tipo de dato del resultado de la expresión debe ser del mismo tipo de dato de la variable, caso contrario existirá un error. En el caso de la asignación, primero se resuelve la expresión y su resultado se asigna a la variable.

```
jshell> int a = 10
a ==> 10

jshell> a = a - 2
a ==> 8
```

En el código anterior, a la variable *a* se le asigna el valor de 10. En la siguiente sentencia, se resta 2 unidades a esa variable y el resultado de esa expresión se asigna nuevamente a la variable *a*.

En la siguiente porción de código se muestra un ejemplo que calcula cuántos Kilo bytes existen en 16732 bytes.

```
jshell> int totalBytes = 16732
totalBytes ==> 16732

jshell> double kiloBytes = totalBytes / 1024.0
kiloBytes ==> 16.33984375
```

Algunas expresiones pueden ser ambiguas, como por ejemplo `21 + 79 / 100` o `8 / 2 * (2 + 2)` esta última se hizo viral en agosto de 2019 en la red social Twitter. En el siguiente apartado conocerá la respuesta a esas expresiones y sobretodo la razón de los valores que obtuvo.

PRECEDENCIA DE OPERADORES

Cuando más de dos operadores aparecen en una expresión, el orden de evaluación depende de las reglas de precedencia de los operadores. Estas reglas determinan el orden en el que se deben ejecutar las operaciones, para ello asignan un valor de precedencia a cada operador y siguen la siguiente regla: el operador de mayor precedencia se ejecuta en primer lugar.

El orden de los operadores que conoce, en Java, hasta el momento se resume en la siguiente tabla:

Orden	Operador	Descripción	Asociatividad
16	()	Paréntesis	Izquierda a derecha
13	()	Casting	Derecha a izquierda
12	* / %	Multiplicativos	Izquierda a derecha
11	+ -	Adición	Izquierda a derecha
1	=	Asignación	Derecha a izquierda

El resto de la tabla lo puede encontrar en: https://introcs.cs.princeton.edu/java/11precedence/

Siguiendo el orden de precedencia, la expresión: `21 + 79 / 100` da como resultado 21.79 ya que primero se ejecuta la división (orden 12) y luego la suma (orden 11). ¿Y la otra expresión `8 / 2 * (2 + 2)`? Siguiendo la misma regla podríamos decir que primero paréntesis y luego ¿qué resuelve multiplicación o división? Ya que ambas tienen igual precedencia. El siguiente párrafo le ayudará

La asociatividad señala que cuando dos operadores tienen el mismo orden de precedencia, la expresión se evalúa de acuerdo a la asociatividad señalada en esa columna. Volviendo a la expresión `8 / 2 * (2 + 2)` por precedencia queda `8 / 2 * 4`, primero paréntesis, en segundo lugar y por asociatividad (izquierda a derecha) división, `4 * 4` dando como resultado 16. Lo invito a usar JShell para verificar que el resultado es correcto.

Esto puede ser complicado, sobretodo memorizar la asociatividad. Para evitar errores y confusiones es mejor utilizar paréntesis para agrupar los operadores en el orden necesario que demanda el problema, ya que los paréntesis cambian el orden de precedencia de los operaciones que agrupa.

SENTENCIAS

Las sentencias son más o menos equivalentes a las oraciones en existen en los lenguajes naturales. Una sentencia forma una unidad completa de ejecución. Las expresiones pueden convertirse en sentencias si terminan con punto y com (;), así por ejemplo `a = 123;` es una sentencia ya que termina con punto y coma.

A expresiones como las anteriores suelen llamarse como sentencias de expresión. Adicionalmente a las este tipo de sentencias, existen las sentencias de declaración y de control de flujo. Una sentencia de declaración declara una variable, una sentencia de control de flujo, regula el flujo de ejecución de un programa (estas se verán mas adelante).

También existe el concepto de bloque de sentencias, un bloque de sentencias está formado por cero o más sentencias que se encuentran entre llaves balanceadas (llave de apertura { como de cierre }).