CICLO 2

[FORMACIÓN POR CICLOS]

Las Variables y otros Elementos Básicos

DEL LENGUAJE

Ingeni@ Soluciones T.I.C

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Las Variables

A la hora de comprar un equipo de cómputo (un computador, una Tablet, un teléfono móvil o incluso un SmartTV), prestamos atención a diferentes especificaciones. Entre dichas especificaciones encontramos la memoria principal y la memoria de almacenamiento.

La memoria de almacenamiento es aquella en la cual se guardan los archivos que necesita una aplicación para ejecutarse (incluyendo la aplicación misma), imágenes, documentos, archivos multimedia, y en general, todo aquello que queremos tener a salvo una ver el equipo se apaga.

Por otro lado, la memoria principal, o memoria RAM (de *Random Access Memory*), es aquella en la cual se guardan los datos de las aplicaciones en ejecución en un momento dado. La memoria principal tiene la particularidad de que sólo contiene información mientras el equipo este encendido, y se vacía una vez este se apaga.

Es en la memoria principal en la cual se guardan los datos de las aplicaciones que construimos con cualquier lenguaje de programación una vez estas se compilan y mientras están en ejecución. Además, ahí se guardan también los objetos y demás componentes que posibilitan la ejecución del programa. En últimas, los componentes de datos más básicos que se guardan en la memoria principal al ejecutarse una aplicación son las variables.

Una **variable** es un espacio en memoria, especificado a la hora de escribir un programa en cualquier lenguaje de programación, que tiene un nombre, y que puede tener una información conocida o desconocida, denominada valor. En Java, además, una variable tiene un tipo, que determina el tipo de información que puede incluir, el rango de valores que puede adoptar y el tamaño que la variable ocupa en la memoria principal.

En Java, para usar una variable, primero es necesario declararla, al hacer esto, se le asigna un nombre y un tipo, y de paso se reserva el espacio en memoria, así:

```
int sumando;
boolean estaActivado;
double promedio;
```

En el ejemplo anterior, se declararon tres variables, una del tipo **int**, que puede contener números enteros, una del tipo **boolean**, que puede contener valores booleanos (verdadero o falso) y una del tipo doublé, que puede contener valores de coma flotante, es decir, con cifras decimales. Una vez una variable ha sido declarada, a esta se le puede asignar un valor que debe concordar con el tipo de dato de la variable. En caso de no ser así, se genera un error. La asignación de valores se hace, típicamente, con el operador =:

```
sumando = 26;
estaActivado = true;
promedio = 4.5;
```

Al momento de declarar una variable, se le asigna un valor por defecto. Sin embargo, nosotros podemos asignar un valor al momento de declararla de tal manera que desde el principio dicha variable tenga ese valor (que luego podremos cambiar), así:

```
byte dividendo = 47;
char sexo = 'f';
estaEncendido = false;
```

A la hora de nombrar las variables en Java (esto incluye nombres de variables, arreglos y objetos) se debe tener en cuenta lo siguiente:

• Los nombres de variables en Java son *case-sensitive*, es decir, distinguen entre mayúsculas y minúsculas, por lo que **intereses, Intereses e INTERESES** son tres nombres de variable diferentes.
• El nombre de una variable puede ser un identificador alfanumérico que empiece por una letra, el signo $ o guion bajo (_). No se permiten espacios en blanco.
• Se recomiendan palabras auto-descriptivas que no sean una palabra reservada del lenguaje (que se pueden conocer en la referencia de Java).
• Para el nombrado de variables, se recomienda la notación camel-case, es decir: Si el nombre consta de una sola palabra, dicha palabra se escribe toda en minúsculas, si son dos o más palabras, se escribe la primera palabra toda en minúsculas, y las palabras subsecuentes, en minúscula, pero con la primera letra en mayúscula, así: **frecuencia, telefono, direccionResidencia, cambioTraseroActual.**

Tipos de Datos Primitivos

En Java, a la hora de declarar variables, contamos con ocho tipos de datos primitivos, más el tipo **String**, que podemos considerar un tipo de datos especial. Los tipos de datos **byte, short, int** y **long** se usan para almacenar números enteros, la diferencia entre ellos es el tamaño que ocupan en memoria y por tanto el rango de los valores que puede tomar la variable, siendo **byte** el tipo con menor capacidad, y **long** el tipo con mayor capacidad. Los tipos **float** y **double** se usan para almacenar números de coma flotante (con cifras decimales) en cuyo caso, **double** ocupa más espacio que **float** y por tanto es un tipo más preciso. El tipo **boolean** puede tomar solamente uno de dos valores: **true** (verdadero) o **false** (falso). Y finalmente, el tipo de datos **char** puede almacenar un solo carácter. De hecho, una cadena de caracteres en Java, de tipo **String** es, internamente, un arreglo o array de caracteres (**char**), como veremos posteriormente.

En la siguiente tabla se muestra la lista de los tipos de datos primitivos en Java, incluyendo el tamaño que ocupan las variables de cada tipo de memoria, el rango que pueden tomar los valores de cada tipo y un ejemplo de cómo se escribe un valor para su asignación a dicha variable, lo que se denomina literal.

Tipo	Tamaño	Rango	Literal (ejemplo)
byte	8 bit	[-128 , 127]	100
short	16 bit	[-32.768 , 32.767]	10000
int	32 bit	[-231 , -231-1]	100000
long	64 bit	[-263 , 263-1]	100000l ó 100000L
float	32 bit		4.25f ó 4.25F
double	64 bit		42.5 ó 42.5d ó 42.5D ó 4.25e1
boolean	1 bit	true / false	false
char	16 bit	[‘\u0000’ , ‘\uffff’]	‘a’

Operadores Básicos

En Java, además de declarar y asignar valores a variables, podemos cambiar sus valores posteriormente, manipularlas y en general, realizar operaciones sobre ellas. El lenguaje cuenta con un conjunto de operadores básicos, que determinan acciones que podemos hacer con variables de tipo primitivo. Entre los operadores más usado en el lenguaje Java tenemos los aritméticos (que permiten realizar operaciones aritméticas con los números y concatenar cadenas de caracteres), los de comparación (que permiten comparar valores y variables) y los lógicos (que permiten combinar comparaciones lógicas). A continuación, podemos apreciar un ejemplo de cómo se usan algunos de estos operadores[1].

```
package co.edu.udea.udea_ruta2_ciclo2;

public class ClaseOperadores {
   public static void main (String[] args) {
      int resultado = 1 + 2;
      // resultado es ahora 3
      System.out.println("1 + 2 = " + resultado);
      int resultadoOriginal = resultado;

      resultado = resultado - 1;
      // resultado es ahora 2
      System.out.println(resultadoOriginal + " - 1 = " +
resultado);
      resultadoOriginal = resultado;

      resultado = resultado * 2;
      // resultado es ahora 4
      System.out.println(resultadoOriginal + " * 2 = " +
resultado);
      resultadoOriginal = resultado;

      resultado = resultado / 2;
      // resultado es ahora 2
```

[1]También pueden acceder a este código en GitHub: https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/main/java/co/edu/udea/udea_ruta2_ciclo2/ClaseOperadores.java

```
System.out.println(resultadoOriginal + " / 2 = " + resultado);
        resultadoOriginal = resultado;

        resultado = resultado + 8;
        // resultado es ahora 10
        System.out.println(resultadoOriginal + " + 8 = " +
resultado);
        resultadoOriginal = resultado;

        resultado = resultado % 7;
        // resultado es ahora 3
        System.out.println(resultadoOriginal + " % 7 = " +
resultado);

        String primeraCadena = "Esta es";
        String segundaCadena = " una cadena de caracteres
concatenada.";
        String terceraCadena =
primeraCadena+segundaCadena;
        System.out.println(terceraCadena);
        //Imprime la cadena completa

        int i = 3;
        i++;
        // Imprime 4
        System.out.println(i);
        ++i;
        // Imprime 5
        System.out.println(i);
        // Imprime 6
        System.out.println(++i);
        // Imprime 6
        System.out.println(i++);
        // Imprime 7
        System.out.println(i);
```

```
        int valor1 = 1;
        int valor2 = 2;
        if(valor1 == valor2)
            System.out.println("valor1 == valor2");
        if(valor1 != valor2)
            System.out.println("valor1 != valor2");
        if(valor1 > valor2)
            System.out.println("valor1 > valor2");
        if(valor1 < valor2)
            System.out.println("valor1 < valor2");
        if(valor1 <= valor2)
            System.out.println("valor1 <= valor2");

        valor1 = 1;
        valor2 = 2;
        if((valor1 == 1) && (valor2 == 2))
            System.out.println("valor1 es 1 AND valor2 es 2");
        if((valor1 == 1) || (valor2 == 1))
            System.out.println("valor1 es 1 OR valor2 es 1");
    }
}
```

En la siguiente tabla, podemos observar una lista de los operadores básicos en el lenguaje Java y su precedencia. En la tabla los operadores están por orden de precedencia de arriba abajo y de izquierda a derecha, de mayor a menor precedencia. Esto quiere decir que, en una expresión escrita en Java con dos o más operadores, se evaluará primero el de mayor precedencia, luego el que le sigue, y así sucesivamente.

Operadores	Precedencia
Postfix	expr++ expr--
Unario	++expr --expr +expr -expr ~ !
Multiplicativo	* / %
Aditivo	+ -
Shift	<< >> >>>
Relacional (lógico)	< > <= >= instanceof
Igualdad (lógico)	== !=
Bitwise AND	&
Bitwise exclusive OR	^
Bitwise inclusive OR	|
AND lógico	&&
OR lógico	||
Ternario	? :
Asignación	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Expresiones, Sentencias y Bloques

Una vez hemos explorado las variables, literales, llamadas a métodos y operadores, conviene comprender cómo se utilizan estos para construir los demás elementos del lenguaje Java.

En primer lugar, tenemos las **expresiones**. Una expresión se compone de variables, literales (valores), llamadas a métodos y operadores. Se deben construir siguiendo la sintaxis del lenguaje, es decir, las reglas de escritura de Java. Además, las expresiones se evalúan en un solo valor, es decir, el compilador las reduce a un solo valor al momento de ejecutar el programa. Por ejemplo, todo aquello resaltado en negrita en el siguiente código son expresiones:

```
int frecuencia = 0;
unArray[0] = 100;
System.out.println("El primer elemento es: " + unArray[0]);
int resultado = (1 + 2) / 3;
if (valor1 == valor2) {
        System.out.println("valor1 es igual a valor2");
```

Por su parte, una **sentencia** es el equivalente, en Java, a una oración en el lenguaje natural. Define una instrucción que damos usando el lenguaje y debe estar libre de ambigüedades. Al contrario que en otros lenguajes de programación tales como Python, en Java las sentencias terminan en punto y coma (;). Tenemos sentencias de diferentes tipos:

Tipos de sentencias	Ejemplos
Expresiones de asignación	temperatura = 28;
Uso de operadores de incremento	temperatura++;
Invocación de métodos	System.out.println(“Hace calor!!”);
Expresiones de creación de objetos	Avion unAvion = new Avion();
Sentencia de declaración	float nota = 2.9;
Sentencias de control de flujo	Las veremos posteriormente

Finalmente tenemos los bloques. Un **bloque** es un conjunto de cero o más sentencias encerradas entre llaves ({}), y están permitidos en cualquier lugar que se permita una sola sentencia. Los bloques suelen delimitar las sentencias condicionales y de control de flujo (como los ciclos).

```
{
        int c = 1.4 + 1.6;
        System.out.println(“La nota es: “ + c);
}
```

La siguiente figura muestra cuáles elementos pueden estar dentro de cuáles otros elementos de una forma gráfica.

Comentarios

A menudo es necesario explicar dentro del código fuente de una aplicación qué realiza un algoritmo, cómo lo realiza o cuál es el propósito de un conjunto de instrucciones, esto se logra mediante los comentarios. Un **comentario** es una parte del código que puede ser usada con diversos propósitos, generalmente explicativos, y que no tienen ninguna incidencia en la ejecución del código ni son procesados por el compilador. Se pueden definir de dos maneras. La primera la usamos cuando el comentario consta de sólo una línea, usando dos líneas diagonales al principio de la línea:

```
//Esto es un comentario de una sola línea
```

La segunda forma la usamos cuando vamos a definir un comentario de más de una línea. En este caso, usamos una línea diagonal para iniciar el comentario, y viceversa para finalizarlo, así

```
/* Esto es un comentario
de más de
una línea */
```