2.2.Palabras reservadas

Las palabras reservadas son palabras con un significado especial dentro de un lenguaje. En especial, estas palabras son empleadas por el lenguaje Java, y el programador no puede utilizarlas como identificadores. Algunas de estas palabras le resultarán familiares al programador del lenguaje C/C++. Las palabras reservadas señaladas con un asterisco (*) no se utilizan en las últimas versiones de Java.

abstract	boolean	break	byte	byvalue*
case	cast*	catch	char	class
const*	continue	default	do	double
else	extends	false	final	finally
float	for	future*	generic*	goto*
if	implements	import	inner*	instanceof
int	interface	long	native	new
null	operator*	outer*	package	private
protected	public	rest*	return	short
satatic	super	switch	synchronized	this
throw	transient	true	try	var*
void	volatile	while		

Las palabras reservadas se pueden clasificar en las siguientes categorías:

- Tipos de datos: **boolean, float, double, int, char**
- Sentencias condicionales: **if, else, switch**
- Sentencias iterativas: **for, do, while, continue**
- Tratamiento de las excepciones: **try, catch, finally, throw**
- Estructura de datos: **class, interface, implements, extends**
- Modificadores y control de acceso: **public, private, protected, transient**
- Otras: **super**, **null, this**.

2.3.Tipos de datos (primitivos, arreglos, referenciados)

El lenguaje de programación Java cuenta con varios tipos de datos. Estos caen en dos grandes categorías: los tipos clase y los tipos primitivos. Los tipos primitivos son valores simples, no son objetos. Los tipos clase son usados para tipos de datos más complejos, incluyendo todos los tipos que el programador desarrollara.

Datos primitivos

El lenguaje Java define ocho tipos primitivos, los cuales pueden ser considerados en cuatro categorías:

- Lógicos – *boolean*
- Texto – *char*
- Enteros – *byte, short, int y long*
- Punto flotante – *float* y *double*

Tipo	Descripción
boolean	1 byte, valores true y false
char	2 bytes, Unicode, comprende el codigo ascii
byte	1 byte, valor entero entre -128 y 127
short	2 bytes, valor entero entre -32768 y 32767
int	4 bytes, valor entero entre -2,147,483,648 y 2,147,483,647
long	8 bytes, valor entre -9,223,372,036,854,775,808 y 9,223,372,036,854,775,807
float	4 bytes, de -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38
double	1.79769313486232E308

Lógicos

Los valores lógicos son representados usando los tipos boléanos, los cuales toman uno de dos posibles valores literales: true (verdadero) ó false (falso). A continuación se muestra un código de ejemplo que declara e inicializa una variable de tipo boolean.

```
boolean verdad = true;
```

No existe ajuste (cast) entre valores de tipo entero y boléanos, como ocurre en otros lenguajes de programación como C/C++. Solo pueden usarse cualquiera de las dos literales definidas.

Texto

El tipo de datos char de Java se utiliza para representar un carácter, éste tipo emplea 16 bits para el almacenamiento de un carácter y lo efectúa en formato Unicode. Unicode presenta dos ventajas:

1. permite a Java trabajar con los caracteres de todos los idiomas.
2. sus primeros 127 caracteres coinciden con los del código ASCII.

Ahora mostramos un ejemplo de declaración e inicialización de una variable tipo carácter.

```
char ch = 'A';
```

Un literal carácter representa un carácter o una secuencia de escape encerrada entre comillas simples. Por ejemplo, ‘c’ , ‘D‘ . Una cadena de caracteres (string), es un conjunto de cero o más caracteres (incluyendo las secuencias de escape) encerrados entre dobles comillas.

Los caracteres Java utilizan Unicode, que es un esquema universal para codificación de caracteres en 16 bits establecido por el consorcio Unicode para soportar el intercambio, proceso y presentación de los textos escritos en los diferentes idiomas del mundo (véase el sitio Web de Unicode en www . Unicode. org). Unicode toma dos bytes, expresados en cuatro números hexadecimales que corren de ' \u0000' a'\uFFFF'. La mayoría de las computadoras utilizan el código ASCII (Unicode incluye los códigos ASCII de '\u0000' a '\u00FF'). En Java se utiliza el código ASCII, así como Unicode y las secuencias de escape como caracteres especiales.

Precaución:

- Una cadena debe encerrarse entre dobles comillas.
- Un carácter es un Único carácter encerrado entre simples comillas.
- Un carácter Unicode se debe representar mediante el código ' \uXXXX ' .

Enteros

Existen cuatro tipos enteros en Java. Cada tipo es declarado usando una de sus palabras claves, *byte*, *short*, *int* ó *long*. Se pueden crear literales de tipos enteros usando la forma decimal, octal hexadecimal.

2	Forma decimal del numero 2
077	0 indica un número octal
0xBAAC	0x indica un número hexadecimal

Todas las literales enteras son de tipo *int* a no ser que se especifique lo contrario, es decir se puede agregar al final de la literal numérica la letra "L", que indica que el valor será entero largo.

Punto flotante

Los tipos flotantes pueden declararse como *float* o *double*. A continuación veremos algunos ejemplos de número flotantes. Las literales flotantes pueden incluir el punto decimal o el termino exponente (E), además de las letras F para *float* y D para *double*.

3.14	valor simple flotante (double)
6.02E23	valor de punto flotante largo
2.718F	valor de punto flotante simple
123.4E+306D	valor doble flotante

Arreglos unidimensionales

Los arreglos son estructuras de datos que agrupan elementos del mismo tipo (homogéneo). Los arreglos nos permiten referirnos a grupos de objetos por un nombre común. Los arreglos unidimensionales, son aquellos arreglos que cuentan únicamente con una fila y con una serie de columnas que determinan la dimensión del arreglo. En Java es posible crear arreglos de cualquier tipo, ya sea datos primitivos o clases (tipos compuestos).

Declaración de arreglos

La sintaxis es la siguiente:

```
<tipo> [ ] <nombre-arreglo>;
```

Cuando queremos que una variable sea un arreglo únicamente anteponemos los corchetes, esto le indica al compilador que la variable de la derecha será una referencia de arreglo.

En el lenguaje Java un arreglo es un objeto aún cuando el arreglo sea de tipos de datos primitivos, además como el resto de los objetos, la declaración no crea el objeto por sí mismo. En su lugar la declaración de un arreglo crea una referencia que puede ser empleada para un arreglo. El espacio en memoria usado para los elementos del arreglo, es dinámico, por lo que es necesario emplear el operador new para crear el espacio para los elementos del arreglo.

Creación de referencias para arreglos

Se deben crear arreglos como objetos, empleando el operador new, por ejemplo, suponga la creación de un arreglo de caracteres (primitivo).

```
char[] s;
s = new char[26];
for ( int i=0; i<26; i++ ) {
      s[i] = (char) ('A' + i);
}
```

El operador new crea un arreglo con 26 caracteres que serán inicializados cada uno de sus elementos a carácter vació ('\u0000'). De esta forma el arreglo está listo para ser manipulado.

Recuerde que el límite de los arreglos va de 0 a *N-1*.

También es posible crear arreglos de referencias, como arreglos de objetos, por ejemplo, un arreglo de Strings.

```
import java.io.*;
import java.util.Scanner;

public class ArrayString
{
      public static void main( String args[] )
      {
            Scanner s = new Scanner(System.in);
            int dim;
            String [] arreglo;
            System.out.print("tamaño del arreglo ");
            dim = s.nextInt();
            arreglo = new String[dim];
            for (int i=0; i<arreglo.length; i++)
            {
                  System.out.print("ingresa la cadena ");
                  arreglo[i] = s.next();
            }
            s.close();
            System.out.println(" Contenido ");
            for (int i=0; i<arreglo.length; i++)
                  System.out.println( arreglo[i] );
            System.exit(0);
      }
}
```

Inicialización de arreglos

Cuando se crea un arreglo, cada elemento es inicializado. Para el caso del arreglo de caracteres vimos que cada carácter es inicializado por defecto a ('\u0000'). En los arreglos de objetos como el caso del *String*, cada elemento del arreglo es inicializado a null, lo que indica que aún no ha sido ingresado ningún dato dentro de la casilla.

Además es posible crear inicializaciones de manera corta como la siguiente:

```
String [ ] nombres = { "Daniela", "Almudena", "Simón"};
```

O de manera equivalente:

```
String [ ] nombres;
nombres = new String[ 3 ];
nombres[0] = “Daniela”;
nombres[1] = ”Almudena”;
nombres[2] = “Simón”;
```

Arreglos multidimensionales

Java provee un mecanismo muy eficiente y sencillo para la construcción de arreglos multidimensionales, solo es necesario indicar el numero de dimensiones con las que se desea trabajar, es decir, así como con los vectores solo es necesario agregar una dimensión con los corchetes.

Podemos crear arreglos multidimensionales de la siguiente manera:

```
int [ ][ ] dosDim = new int [2][ ];
dosDim[0] = new int [5];
dosDim[1] = new int [5];
```

El objeto que es creado en la primera llamada con *new* es un arreglo que contiene dos elementos.

Cada elemento es una referencia nula (*null*) a un elemento de tipo *array of int*, y debe ser inicializada cada elemento de manera separada, de manera que cada elemento apunta a su arreglo. Debido a esto es posible crear matrices no-rectangulares.

En algunos casos esta inicialización podría parecer tediosa, pues es más común emplear matrices rectangulares, a continuación vemos como sería la declaración e inicialización de una matriz de enteros:

```
int [ ][ ] dosDim = new int [4][5];
```

Veamos un ejemplo de una matriz a al que se le asignan valores de manera aleatoria:

```
import java.io.*;
import java.util.Scanner;
import java.util.Random;
public class Matriz
{
	public static void main( String args[] )
	{
		Scanner s = new Scanner(System.in);
		int f,c;
		System.out.print("numero de filas ");
		f = s.nextInt();


		System.out.print("numero de columnas ");
		c = s.nextInt();
		s.close();
		Random rand = new Random();
		int m[][] = new int [f][c];
		for (int i=0; i < m.length; i++)
		{
			for (int j=0; j < m[i].length; j++)
```

```
                        m[i][j]= rand.nextInt(f*c*10);
                }
                for (int i=0; i < m.length; i++)
                {
                      for (int j=0; j < m[i].length; j++)
                            System.out.print(" "+i+","+j+"-> "+ m[i][j]);
                      System.out.println();
                }
                System.exit(0);
        }
}
```

Como ya se menciono los arreglos en java son objetos, por lo que el número de elementos que pueden ser almacenados forma parte de sus atributos. Un acceso a un límite fuera de su longitud produce un error en tiempo de ejecución. Emplee el atributo length para iterar dentro de un arreglo.

```
public int sumatoria(int [] lista) {
      int total = 0;
      for( int i = 0; i < lista.length; i++ )
            total += lista[i];
      return total;
}
```

Iterar sobre un arreglo es una tarea común, Java en la versión 5.0 incorpora un mecanismo para iterar elementos como una colección de datos, lo que hace un poco más sencilla la declaración.

```
public int sumatoria(int [] lista) {
      int total = 0;
      for( int elemento : list )
            total += elemento;
      return total;
}
```

Esta versión del ciclo *for* puede ser leída como “para cada elemento en la lista haz”.

Redimensión de arreglos

Después de que ha sido creado un arreglo, no puede ser redimensionado su tamaño. Sin embargo puede ser empleada la misma referencia a la variable para declarar una nueva referencia del arreglo por completo.

```
int [] arreglo = new int [6];
arreglo = new int[10];
```

En este caso, el primer arreglo pierde la referencia a todos sus componentes, a no ser que sean almacenados en algún lugar.

Datos referenciados

Existen, además de los datos primitivos ya vistos, datos referenciados (datos tipo clase) que son objetos más complejos o compuestos que permiten realizar más operaciones, debido a que fueron diseñados a partir de clases, entre ellos existe una clase u objetos referenciado de uso muy común llamado String, que es el tipo de dato en el que nos concentraremos en este momento.

La clase string es una de tantas clases incluidas en la librería de clases de Java. La clase string provee la habilidad de almacenar secuencias de caracteres y manipularlas de forma sencilla para almacenar literales. Los objetos strings pueden ser declarados de dos maneras:

```
String nombre = new String("Juan Pérez");
```

ó

```
String nombre = "Juan Pérez";
```

La clase string es la única que permite construir objetos de ese tipo sin emplear el operador *new*, que es el empleado para crear cualquier objeto en Java.

2.4.Variables

2.4.1. Nombres de variables

2.4.2. Inicialización de variables

Variables, declaración y asignación

Las variables son los identificadores construidos para asignar valores que serán manipulados a lo largo del programa. Las declaraciones de variables deben seguir la siguiente sintaxis:

```
<tipo de dato> identificador [ = valor ] ;
```

Las asignaciones siempre deben de corresponder al tipo de datos que se está manejando, existen errores comunes de asignación, como son:

```
int y = 3.1415926;
double w = 175,000;
boolean sw = 1;
float z = 3.14156;
```

Los identificadores comienzan con letra, guión bajo ó signo de dólar, y el resto de los caracteres pueden ser dígitos. Recuerde que los identificadores son case-sensitive.

Debido a que pueden emplearse caracteres no ASCII, tenga presente lo siguiente:

- UNICODE puede soportar diferentes caracteres que lucen igual.
- Los nombres de clase deben ser únicamente caracteres ASCII, debido a que la mayoría de sistemas no soportan UNICODE.
- Un identificador no puede ser una palabra clave, pero puede estar compuesta como parte del nombre.

2.4.3. Visibilidad y vida de las variables

Ámbito. Es importante el lugar donde se efectúa la declaración de las variables, pues éste determina su ámbito. En Java es posible agrupar sentencias simples, encerrándolas entre una pareja de llaves para formar bloques o sentencias compuestas y efectuar declaraciones de variables dentro de dichos bloques, al principio de los métodos o fuera de ellos. Una sentencia compuesta nula es aquella que no contiene ninguna sentencia entre las llaves { }.

Ejemplo:

```
int i=25;
double j=Math.sqrt (20);
i++;
j+=5;
System.out.println(i+" "+j);
//A continuación comienza un bloque
{
       int aux = i;
       i=(int)(j) ;
       j= aux;
} //Fin del bloque
System.out.println(i+" "+j);//aux aquí no está definida
```

Es decir, en Java es posible declarar variables en el punto de utilización de las mismas dentro del programa, pero habrá de tener en cuenta que su ámbito será el bloque en el que han sido declaradas. Como ya se ha indicado, un bloque de sentencias se delimita entre dos llaves y las variables declaradas dentro de un bloque sólo son válidas en dicho bloque. Si un bloque tuviera otros anidados, en los interiores a él. No se pueden declarar variables en bloques interiores con el nombre de otras de ámbito exterior.

Las variables declaradas dentro del cuerpo de un método son *variables locales,* y sólo existirán y se podrá hacer referencia a ellas dentro del cuerpo del método. Las variables que se declaran fuera del cuerpo de un método son variables de instancia y cada instancia de la clase tendrá una copia de dichas variables. Las variables declaradas fuera del cuerpo de los métodos y en cuya declaración se especifique la palabra *static* son variables de clase, esto quiere decir que no se hará una copia de ellas para cada uno de los objetos de la clase y, por tanto, su valor será compartido por todas las instancias de la misma.

2.5.Constantes

Java emplea la palabra clave final para indicarle al compilador de que las variables declaradas bajo esta etiqueta deberán comportarse como si fueran constantes. Esto indica que su valor no puede ser modificado una vez que se haya declarado e inicializado. Las constantes se declaran de la siguiente manera:

```
final double PI = 3.14159;
final char VALOR = ´S´;
final String MENSAJE = “Error”;
```

Si se intenta cambiar el valor de las variables tipo final desde una aplicación, se genera un error de compilación. Es importante recordar, como medida de diseño, que los nombres de las constantes deben escribirse con letras mayúsculas para facilitar su ubicación dentro del código.

También los las funciones miembro (métodos) pueden ser de tipo final, en cuyo caso no podrán ser sobrecargados

2.6.Operadores

2.6.1. Operadores aritméticos, de asignación, unarios, instanceof, incrementales, relacionales, de intercambio (shift), lógicos, concatenación, condicional.

Los operadores de Java son muy parecidos en estilo y funcionamiento a los de C, estos se detallan a continuación:

Operadores Aritméticos: Los habituales que son:

Operador	Operación	Ejemplo
+	suma	suma=n1+n2;
-	Resta	resultado=num1-num2;
*	multiplicación	area=pi*r*r;
/	División	prom=(cal1+cal2+cal3)/3;
%	Residuo	resultado=num%den;

Operadores de Asignación: El principal es '=' pero hay más operadores de asignación con distintas funciones que explicamos brevemente ahora.

Operador	Operación	Ejemplo	Equivalencia
=	Asignación		
+=	Asignación de suma	op1 += op2	op1 = op1 + op2
-=	Asignación de resta	op1 -= op2	op1 = op1 - op2
*=	Asignación de multiplicación	op1 *= op2	op1 = op1 * op2
/=	Asignación de división	op1 /= op2	op1 = op1 / op2
%=	Asignación de residuo	op1 %= op2	op1 = op1 % op2

Operadores Unarios: El mas (+) y el menos (-). Para cambiar el signo del operando.

Operador Instanceof: Nos permite saber si un objeto pertenece a una clase o no.

- NombreObjeto instanceof NombreClase

Operadores Incrementales: Son los operadores que nos permiten incrementar las variables en una unidad. Se pueden usar delante y detrás de la variable dependiendo de lo que queramos, es decir, si queremos que incremente o viceversa antes de utilizar o lo contrario.

++	incremento de 1	a++; o ++a;
--	decremento de 1	numero--; o --numero;

Operadores Relacionales: Permiten comparar variables según relación de igualdad/desigualdad o relación mayor/menor. Devuelven siempre un valor boolean.

Operador	Operación	Ejemplo
>	Mayor que	if(a>b)
<	Menor que	for(i=0;i<10;i++)
==	Iguales	while(salir==false)
!=	Distintos	…
>=	Mayor o igual que	…
<=	Menor o igual que	…

Operadores Lógicos: Nos permiten construir expresiones lógicas.

Operador	Operación	Ejemplo
&&	Y lógico, opcionalmente compara la segunda expresión	expresión1 && expresión2
||	O lógico, opcionalmente compara la segunda expresión	expresión1 || expresión2
!	Negación	!operación 1
&	Y lógico, siempre compara ambas expresiones	expresión1 & expresión2
|	O lógico, siempre compara ambas expresiones	expresión1 | expresión2
?	versión reducida del if-else	expresión ? : operación 1 : operación 2

Operador de concatenación con cadena de caracteres '+':

- Por Ejemplo: System.out.println("El total es"+ result +"unidades");

Operadores que actúan a nivel de bits: Son mucho menos utilizados por eso los explicamos mas por encima.

Operadores de desplazamiento

Operador	Operación	Ejemplo
<<	Desplaza los bits de n a la izquierda k veces	n<<k
>>	Desplaza los bits de n a la derecha k veces	n>>k
>>>	Desplaza los bits de n a la derecha k veces (sin signo)	n>>>k

Operadores de lógica de bits

Operador	Operación	Ejemplo
&	Operador and a nivel de bit	n1 & n2
|	Operador or a nivel de bit	n1 | n2
^	Operador de or exclusivo	n1 ^ n2
~	Operador de complemento	~n1

2.6.2. Precedencia de operadores

Cuando en una sentencia aparecen varios operadores el compilador deberá de elegir en qué orden aplica los operadores. A esto se le llama *precedencia*. Los operadores con mayor precedencia son evaluados antes que los operadores con una precedencia relativa menor.

Cuando en una sentencia aparecen operadores con la misma precedencia:

- Los operadores de asignación son evaluados de derecha a izquierda.
- Los operadores binarios, (menos los de asignación) son evaluados de izquierda a derecha.

Se puede indicar explícitamente al compilador de Java cómo se desea que se evalúe la expresión con paréntesis balanceados ( ). Para hacer que el código sea más fácil de leer y mantener, es preferible ser explícito e indicar con paréntesis que operadores deben ser evaluados primero.

La siguiente tabla muestra la precedencia asignada a los operadores de Java. Los operadores de la tabla están listados en orden de precedencia: cuanto más arriba aparezca un operador, mayor es su precedencia. Los operadores en la misma línea tienen la misma precedencia:

Tipo de operadores	**Operadores de este tipo**
Operadores posfijos	[ ] . (parámetros) expr++ expr--
Operadores unarios	++expr --expr +expr -expr ~ !
Creación o conversión	new (tipo) expr
Multiplicación	* / %
Suma	+ -
Desplazamiento	<<
Comparación	< <= = instanceof
Igualdad	== !=
AND a nivel de bit	&
OR a nivel de bit	^
XOR a nivel de bit	|
AND lógico	&&
OR lógico	||
Condicional	? :
Asignación	= += -= *= /= %= &= ^= |= <<= = =

2.6.3. Promoción y casting

Ajuste de datos primitivos

El ajuste primitivo o *cast*, significa asignar el valor de un tipo de variable a una de otro tipo. Si dos tipos son compatibles, Java realiza el ajuste automáticamente. Por ejemplo un valor tipo *int* puede ser asignado sin ningún problema a una variable tipo *long*.

En los casos donde la información se pueda perder en la asignación, el compilador requiere que se le confirme la asignación a través del operador *cast*. Por ejemplo:

```
long big = 99l;
int medium = big // asignación invalida
int medium = (int) big // cast aplicado
```

Es necesario que el tipo de dato destino sea encerrado entre paréntesis y usado de manera prefija indicando que la expresión debe ser modificada.

Las variables pueden ser *promocionadas* automáticamente a tipos más grandes (como de *int* a *long*)

```
double x = 12.414F; // 12.414F es flotante, correcto
float y = 12.414; // 12.414 es doble, incorrecto
```

Cuando se realizan operaciones como la suma y los dos operandos son de tipo de dato primitivo, el tipo de dato resultante es determinado por el operando de tipo más grande, por lo que en algunos casos podría ser un problema de sobre flujo, por ejemplo:

```
short a, c;
int b;
a = 1; b = 2;
c = a + b;
```

Causa un error debido a los tipos short, a no ser que la variable c sea declarada como int, para subsanar este problema se sugiere realizar un *cast* en la operación que permita el ajuste.

```
c = (short) (a + b);
```

Etapas para crear un programa

La creación de un programa debe comenzar con la escritura del código fuente correspondiente a la aplicación. Cada programa Java debe tener al menos una clase.

Un ejemplo de una aplicación Java sencilla que sirva de modelo es el popular <<Hola mundo>>.

```
//Esta aplicación visualiza: ¡Hola! Bienvenido a Java
public class Bienvenido
{
      public static void main (String[] argc)
      {
            System.out.println("¡Hola! Bienvenido a Java");
      }
}
```

Las etapas que preparan un programa para su ejecución son:

1. *Crear una carpeta de proyecto* en la que se recojan todos los archivos significativos, incluyendo clases que se desean incluir.
2. Utilizar un programa *editor* que introduzca cada línea del programa fuente en memoria y lo guarde en la carpeta proyecto como un archivo fuente.
3. Utilizar el programa *compilador* para traducir el programa fuente en *bytecode* (código en bytes). Si existen *errores de sintaxis* (un error gramatical de una línea en un programa Java), el compilador visualiza esos errores en una ventana.
4. Utilizar el programa editor para corregir esos errores, modificando y volviendo a guardar el programa fuente. Cuando el programa fuente está libre de errores, el compilador guarda su traducción en *bytecode* como un archivo.
5. El intérprete Java (JVM) traduce y ejecuta cada instrucción en *bytecode.*
6. Si el código no funciona correctamente se puede utilizar el depurador para ejecutar el programa paso a paso y examinar el efecto de las instrucciones individuales.