Programación Orientada a Objetos

Tema 1-2:
Sintaxis básica del lenguaje Java

Contenidos

Tema 1-2: Sintaxis básica de Java

COMENTARIOS


```
// comentarios para una sola línea

/* comentarios de una o
   más líneas
*/

/** comentario de documentación,
    de una o más líneas
*/
```


IDENTIFICADORES

Normas obligatorias:

- Los identificadores diferencian entre mayúsculas y minúsculas. Por ejemplo los identificadores "NombreDeVariable" y "nombredevariable" no son iguales.
- Deben comenzar con una letra, subrayado (_) o símbolo de dolar ($).
- No puede usarse como identificador una palabra reservada.
- Los caracteres posteriores al primero pueden ser números.
- No existe longitud máxima.

Normas recomendadas:

- ✓ Los nombres de clases empiezan con mayúsculas.
- ✓ La primera letra de las variables y métodos con minúscula.
- ✓ Los nombres compuestos se unen con una letra mayúscula en el comienzo de cada palabra.
- ✓ Por ejemplo:

  saldoCuenta

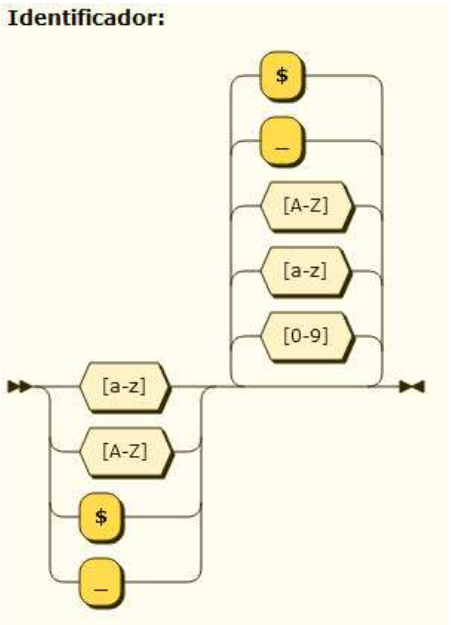
  calcularInteresCuenta()

- ✓ Se usan nombres largos y significativos. Por ejemplo para una variable donde guardar un nombre de usuario, una buena idea sería denominarla "`nombreUsuario`". Sería un error denominarla "`u`".

IDENTIFICADORES

Serían identificadores válidos:

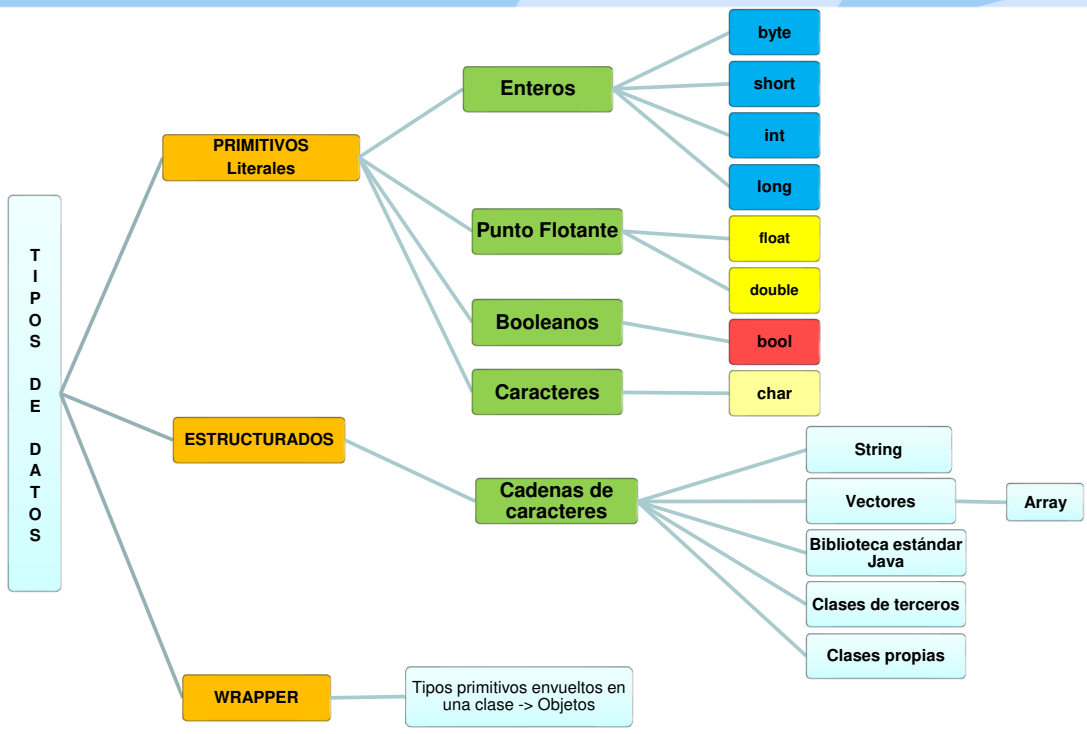
```
identificador1
nombreUsuario
_variable_del_sistema
$total
```



Y su uso sería, por ejemplo:

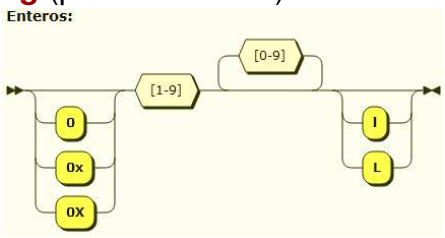
```
int contadorPrincipal;
char letraInicial;
float $cantidadEnEuros;
```


TIPOS DE DATOS

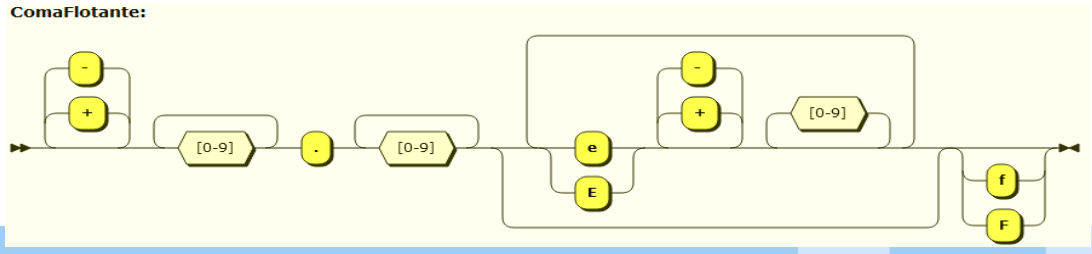


Primitivos o Literales:

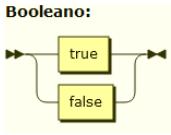
- Enteros: Representan valores de tipo **int** o **long** (por defecto **int**).
  - Hay tres tipos decimal, octal y hexadecimal.
  - Ejemplos:
    int (32 bits): 21 077 0xDC00
    long (64 bits) (número + l o L) : 1234567890L



- Reales en coma flotante: Representan valores de tipo **float** o **double** (por defecto **double**).
  - Representan números decimales con partes fraccionarias
  - Ejemplo: 634.3346 //estándar 6.343346e2 //científica
  - Por defecto double, para float: número + f o F. 634.3346F //float



Primitivos o Literales:



- Booleanos:
  - true y false
- Caracteres: Representan valores de tipo **char** (caracteres Unicode).
  - Tabla de caracteres UNICODE (65536 caracteres).
  - Ejemplo: `char variableCaracter='A';`
  - Caracteres no imprimibles:

\\	Barra invertida
\f	Salto de página
\b	Retroceso
\r	Retorno de carro
\f	Alimentación de formularios
\t	Tabulador horizontal
\n	Línea nueva
\'	Comillas simples
\"	Comillas dobles
\u????	Unicode hexadecimal
\ddd	Unicode octal

Estructurados:

- Cadenas: Representan objetos de tipo String. Caracteres entre comillas dobles. Se pueden utilizar caracteres de escape \.

```
String cadena="Esto es una cadena literal";
```

Ejemplos: `""`, `"Hola Mundo"`, `"Esto es una\n\"cadena\""`

Java es un lenguaje fuertemente "tipado".

- Ejemplos:

```
//declaramos una variable de tipo int (32 bits con signo)
int contador;
//declaramos una variable de tipo carácter.
char carac;
//la variable temperatura es de tipo double (64 bits con signo)
double temperatura;
```

  - Opcionalmente se puede inicializar la variable declarada con algún valor:

```
int contador=0;
char carac='P';
short enteroCorto=-432;
double temperatura=23.65;
double decimal1=345.23e12;
float decimal2=0.545; //Mal. Correcto: float decimal2=0.545f;
```

  - También se pueden realizar declaraciones múltiples:

```
char carac1, carac2, carac3, carac4;
double primerDecimal, segundoDecimal;
int a,b,c,d,e=6; //solo se inicializa a 6 la variable e
```


- **Conversiones entre diferentes tipos de datos:**

```
int littleInt;
long bigInt = 123;
littleInt = bigInt; // ERROR!! 32 bits ⇒ 64 bits
```

  - Conversión ***cast***:

```
int littleInt;
long bigInt = 123;
littleInt = (int) bigInt; // CORRECTO!!
```

De\A	byte	short	char	int	long	float	double
byte	No	No	No	No	No	No	No
short	Si	No	No	No	No	No	No
char	Si	Si	No	No	No	No	No
int	Si	Si	Si	No	No	No	No
long	Si	Si	Si	Si	No	No	No
float	Si	Si	Si	Si	Si	No	No
double	Si	Si	Si	Si	Si	Si	No

Operadores:

Precedencia + / -

Posfijos	[] . (params) expr++ expr--
Prefijos (D)	++expr --expr +expr -expr ~ !
Creación o conversión	new (tipo) expr
Multiplicación, División, Resto División	* / %
Suma	+ -
Desplazamiento de bits	<< >> >>>
Relacionales	< > <= >= instanceof
Igualdad	== !=
Y bits	&
O Exclusivo bits	^
O Inclusivo bits	|
Y lógica	&&
O lógica	||
Condicional	?:
Asignación (D)	= += -= *= /= %= >>=
	<<= >>>= &= ^= |=

- Orden de evaluación de izquierda a derecha.
- Excepto en &&, || y ?: se evalúan todos los operandos antes de llevar a cabo la operación.
- Para cambiar la precedencia, utilizamos paréntesis.
- Para las cadenas, se pueden utilizar los operadores + y += para la concatenación.

- Ejemplos:

```
int cont1 = 1, cont2 = 2;
cont1++;          //cont1=2
cont2--;          //cont2=1
cont1 += cont2;   //equivale a: cont1 = cont1 + cont2;
cont1 == cont2;   //false
cont1 != cont2;   //true
int mayor = (cont1 < cont2) ? cont2 : cont1;
String nombre = "Nombre " + "Apellido";
```

SEPARADORES

- () - paréntesis. Listas de parámetros en la definición y llamada a métodos. Definir precedencia en expresiones, contener expresiones para control de flujo y rodear las conversiones de tipo.
- {} - llaves. Se utiliza para definir un bloque de código. También para contener los valores iniciales de un array.
- [] - corchetes. Para declarar arrays o matrices. También se utiliza cuando se referencian valores de matriz.
- ; - punto y coma. Separa sentencias.
- , - coma. Separa identificadores consecutivos en una declaración de variables. También se utiliza para encadenar sentencias dentro de una sentencia for.
- . - punto. Para separar nombres de paquete de subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

ARRAYS

- Un *array* es una estructura de programación que almacena un determinado número de elementos del mismo tipo. Para poder disponer y usar un array en Java hay que realizar dos pasos:
  - Declarar el array.
  - Reservar espacio para un número determinado de elementos mediante la instrucción `new`.
- Ejemplo:

```
int numeros[];     int[] numeros;
numeros = new int[20];
```

O también: `int[] numeros = new int[20];`

- Más ejemplos:

```
char[] vocales={'A','E','I','O','U'};
String nombres[] = {"Juan","Pepe","Pedro","Maria"}; //equivale a:
String nombres[];
nombres = new String[4];
nombres[0] = new String( "Juan" );
nombres[1] = new String( "Pepe" );
nombres[2] = new String( "Pedro" );
nombres[3] = new String( "María" );
```

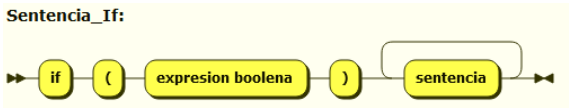
```
int tabla[][] = new int[10][3];
tabla.length;          /* 10 */
tabla[0].length;       /*  3 */
```

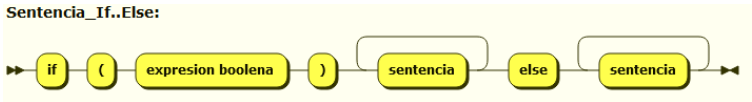

- **Sentencias Condicionales (if):**

```
if (expresión booleana)
        sentencia
```

o bien

```
if (expresión booleana)
{
        sentencias
}
else
{
        sentencias2
}
```





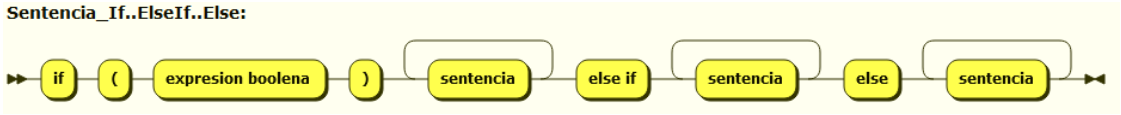
Se evalúa la expresión booleana y si el resultado es **true**, se ejecuta la sentencia asociada. Si el resultado es **false** y existe parte **else**, se ejecuta la sentencia asociada a ésta.
Como *sentencias2* podríamos tener otra sentencia **if** (sentencias **if** anidadas).
Un bloque **else** siempre se asocia al if más cercano que no tenga **else** asociado.

- **Sentencias Condicionales (if):**

Ejemplos:

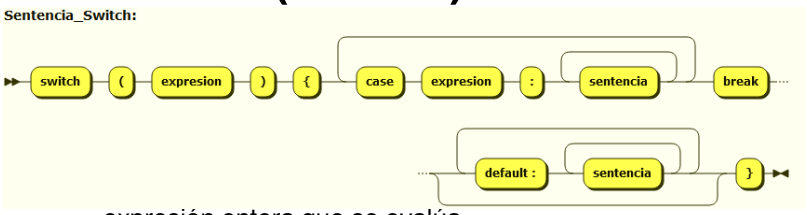
```
if (x > 10)
      y--;
else if (x > 5)
      y++;
else
      y = 0;
```

```
int a;
if(a==0)
  System.out.println("a = 0");
else if(a==1)
  System.out.println("a = 1");
else if(a==2)
  System.out.println("a = 2");
else
  System.out.println("a es mayor
que 2 o menor que 0");
```



- **Sentencias Condicionales (switch):**

Ejemplo:




```
String mensaje;
int valorEntero;
. . .
switch(valorEntero)
{
   case 1: mensaje = "Uno";
                   break;
   case 2: mensaje = "Dos";
                   break;
   case 3: mensaje = "Tres";
                   break;
   default: mensaje = "desconocido";
}
```

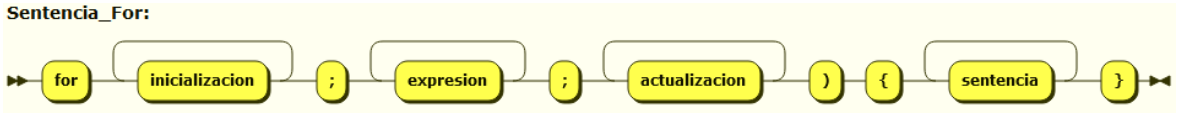
expresión entera que se evalúa

Se ejecuta la primera sentencia después de la etiqueta **case** con el mismo valor.

Finaliza el bloque

Si el valor de la expresión entera no coincide con ninguno de los listados en las etiquetas **case**, se ejecuta la primera sentencia asociada a la parte **default**. Esta parte es opcional.

- **Sentencias de Bucle (for):**




```
for( expr1 inicio; expr2 test; expr3 incremento )
{
  sentencias;
}
```

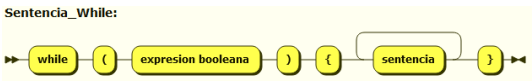
Ejemplos:

```
for(int a=0,b=0; a < 7; a++,b+=2 )
```

```
int contador;
for( contador=1; contador <= 12; contador++ )
{
          System.out.println("Iteración "+contador);
}
```


- **Sentencias de Bucle (while):**

1)
```
while (expresión booleana)
{
      sentencias
}
```



2)
```
do
{
      sentencias
}
while (expresión booleana)
```

Si al evaluar *expresión booleana* el resultado es true, se ejecutan las *sentencias.*
En 2), las *sentencias* se ejecutan al menos una vez.

Ejemplo:

```
int contador=1;
while( contador <= 12 )
{
          System.out.println("Iteración "+contador);
          contador++;
}
```

- **Control General del Flujo:**

Etiquetas Podemos etiquetar sentencias:
etiqueta:sentencia

break Permite salir de cualquier bloque. Se suele utilizar para interrumpir un bucle **while**, **do** o **for**, o bien una sentencia **switch**.
Por defecto, sale del bucle más interno, pero con etiquetas, podemos hacer que salga de cualquier bucle anidado.

break [*etiqueta*];

continue Salta al final del cuerpo de un bucle while, do o for, evaluándose la expresión booleana de control. Suele utilizarse para saltar un elemento del rango del bucle. Puede saltarse al final de cualquier bucle anidado utilizando etiquetas.

continue [*etiqueta*];

Universidad de Alcalá
CONTROL DEL FLUJO
Departamento Ciencias de la Computación

- **Control General del Flujo:**

Ejemplo:

```
uno: for( )
          {
           dos: for( )
              {
                /* código */
                continue;          // seguiría en el bucle interno
                /* código */
                continue uno;      // seguiría en el bucle principal
                /* código */
                break uno;         // se saldría del bucle principal
                /* código */
              }
          }
```

- **Control General del Flujo:**

- Return:
  Termina la ejecución de un método y devuelve un valor del tipo de retorno. `return [expresión];`

Ejemplos:

```
void f() {
        ...
        return;
}
```

```
double op(double x, double y) {
        ...
        return x * y / (x + y);
}
```

```
int func()
{
    if( a == 0 ) return 1;
    return 0;    // es imprescindible porque se devuelve un entero
}
```