SINTAXIS

PROGRAMACIÓN

[Escuela]
[Título del curso]

ENTORNO INTEGRADO DE DESARROLLO

Un IDE o Entorno Integrado de Desarrollo es una herramienta con el cual poder desarrollar y probar proyectos en un lenguaje determinado.

Una buena opción para empezar a programar en Java es instalar Sublime. Sublime es un IDE muy liviano y muy intuitivo y su instalación es sumamente sencilla.

Se puede descargar de este sitio Web: http://www.geany.org/Download/Releases

Una vez instalado el programa, hay que configurar la variable PATH en Windows (Panel de control sistema Opciones Avanzadas variables de entorno)

Modificar la variable PATH añadiendo el siguiente texto dependiendo del JDK instalado previamente en tu ordenador:

C:\Archivos de programa\Java\jdk1.6.0_19\bin; C:\Archivos de programa\Java\jdk1.6.0_19\lib; %PATH%

FUNDAMENTOS

Introducción

Java es un lenguaje orientado a objetos, que se deriva en alto grado de C++, de tal forma que puede ser considerado como un C++ nuevo y modernizado o bien como un C++ al que se le han amputado elementos heredados del lenguaje estructurado C.

Tokens

Un *token* es el elemento más pequeño de un programa que es significativo para el compilador. Estos *tokens* definen la estructura de Java.

Cuando se compila un programa Java, el compilador analiza el texto, reconoce y elimina los espacios en blanco y comentarios y extrae *tokens* individuales. Los *tokens* resultantes se compilan, traduciéndolos a código de byte Java, que es independiente del sistema e interpretable dentro de un entorno Java.

Los códigos de byte se ajustan al sistema de máquina virtual Java, que abstrae las diferencias entre procesadores a un procesador virtual único.
Los *tokens* Java pueden subdividirse en cinco categorías: Identificadores, palabras clave, constantes, operadores y separadores.

a.) Identificadores

Los identificadores son *tokens* que representan nombres asignables a variables, métodos y clases para identificarlos de forma única ante el compilador y darles nombres con sentido para el programador.

Todos los identificadores de Java diferencian entre mayúsculas y minúsculas (Java es *Case Sensitive* o *Sensible a mayúsculas*) y deben comenzar con una letra, un subrayado(_) o símbolo de dólar($). Los caracteres posteriores del identificador pueden incluir las cifras del 0 al 9. Como nombres de identificadores no se pueden usar palabras claves de Java.

Además de las restricciones mencionadas existen propuestas de estilo. Es una práctica estándar de Java denominar:

Las clases: ***Clase*** **o** ***MiClase*****.**
Las interfaces: ***Interfaz*** **o** ***MiInterfaz*****.**
Los métodos: ***metodo()*** **o** ***metodoLargo().***
Las variables: ***altura*** **o** ***alturaMedia*****.**
Las constantes: ***CONSTATE*** **o** ***CONSTANTE_LARGA*****.**
Los paquetes: ***java.paquete.subpaquete*****.**

Sin entrar en más detalle en la siguiente línea de código se puede apreciar la declaración de una variable entera (*int*) con su correspondiente identificador:

Ej: ***int alturaMedia;***

b.) Palabras clave

Las palabras claves son aquellos identificadores reservados por Java para un objetivo determinado y se usan sólo de la forma limitada y específica. Java tiene un conjunto de palabras clave más rico que C o que C++, por lo que sí está aprendiendo Java con conocimientos de C o C++, asegúrese de que presta atención a las palabras clave de Java.

Las siguientes palabras son palabras reservadas de Java:

abstact	***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***
static	***super***	***switch***	***syncroniced***	***this***
throw	***throws***	***transient***	***true***	***try***
var	***void***	***volatile***	***while***	

Las palabras subrayadas son palabras reservadas pero no se utilizan. La definición de estas palabras clave no se ha revelado, ni se tiene un calendario respecto a cuándo estará alguna de ellas en la especificación o en alguna de las implementaciones de Java.

c.) Literales y constantes

Los literales son sintaxis para asignar valores a las variables. Cada variable es de un tipo de datos concreto, y dichos tipos de datos tienen sus propios literales.

Mediante determinados modificadores (*static* y *final*) podremos crear variables *constantes*, que no modifican su valor durante la ejecución de un programa. Las constantes pueden ser numéricas, booleanas, caracteres (Unicode) o cadenas (*String*). Las cadenas, que contienen múltiples caracteres, aún se consideran constantes, aunque están implementadas en Java como objetos.

Veamos un ejemplo de constante declarada por el usuario:

```
final static int ALTURA_MAXIMA = 200;
```

Se puede observar que utilizamos *final static*, para que la variable sea total y absolutamente invariable.

d.) Operadores

Conocidos también como operandos, indican una evaluación o computación para ser realizada en objetos o datos, y en definitiva sobre identificadores o constantes. Los operadores admitidos por Java son:

+	^	<=	++	%=
>>>=	-	~	>=	-
&=	.	*	&&	<<
==	<<=	[	/	||
>>	+=	^=	]	%
!	>>>	=	!=	(
&	<	*=	)	|
>	?!!	/=	>>	

Así por ejemplo el siguiente fragmento de código incrementa el valor de una variable en dos unidades, mediante la utilización del operador aritmético *+* que se utiliza para la suma:

```
int miNumero=0;
miNumero = miNumero + 2;
```

e.) Separadores

Se usan para informar al compilador de Java de cómo están agrupadas las cosas en el código.

Los separadores admitidos por Java son: *{ } , : ;*

f.) Comentarios y espacios en blanco

El compilador de Java reconoce y elimina los espacios en blanco, tabuladores, retornos de carro y comentarios durante el análisis del código fuente.

Los comentarios se pueden presentar en tres formatos distintos:

Formato	Uso
/*comentario*/	Se ignoran todos los caracteres entre /* */. Proviene del C
//comentario	Se ignoran todos los caracteres detrás de // hasta el fin de línea. Proviene del C++
/**comentario*/	Lo mismo que /* */ pero se podrán utilizar para documentación automática.

Por ejemplo, la siguiente línea de código presenta un comentario:

```
int alturaMinima = 150; // No menos de 150 centímetros
```

Expresiones

Los operadores, variables y las llamadas a métodos pueden ser combinadas en secuencias conocidas como expresiones. El comportamiento real de un programa Java se logra a través de expresiones, que se agrupan para crear *sentencias*.

Una expresión es una serie de variables, operadores y llamadas a métodos (construida conforme a la sintaxis del lenguaje) que se evalúa a un único valor.

Entre otras cosas, las expresiones son utilizadas para realizar cálculos, para asignar valores a variables, y para ayudar a controlar la ejecución del flujo del programa. La tarea de una expresión se compone de dos partes: realiza el cálculo indicado por los elementos de la expresión y devuelve el valor obtenido como resultado del cálculo.

Los operadores devuelven un valor, por lo que el uso de un operador es una expresión. Por ejemplo, la siguiente sentencia es una expresión:

```
int contador=1; contador++;
```

La expresión *contador++* en este caso particular se evalúa al valor *1*, que era el valor de la variable *contador* antes de que la operación ocurra, pero la variable *contador* adquiere un valor de *2*.

El tipo de datos del valor devuelto por una expresión depende de los elementos utilizados en la expresión. La expresión *contador++* devuelve un entero porque *++* devuelve un valor del mismo tipo que su operando y *contador* es un entero. Otras expresiones devuelven valores booleanos, cadenas...

Una expresión de llamada a un método se evalúa al valor de retorno del método; así el tipo de dato de la expresión de llamada a un método es el mismo que el tipo de dato del valor de retorno de ese método.

Otra sentencia interesante sería:

```
in.read( ) != -1 // in es un flujo de entrada
```

Esta sentencia se compone de dos expresiones:

La primera expresión es una llamada al método *in.read().* El método *in.read()* ha sido declarado para devolver un entero, por lo que la expresión *in.read()* se evalúa a un entero.

La segunda expresión contenida en la sentencia utiliza el operador *!=*, que comprueba si dos operandos son distintos. En la sentencia en cuestión, los operandos son *in.read()* y -1. El operando *in.read()* es válido para el operador *!=* porque i*n.read()* es una expresión que se evalúa a un entero, así que la expresión *in.read()!=-1* compara dos enteros. El valor devuelto por esta expresión será verdadero o falso dependiendo del resultado de la lectura del fichero *in*.

Como se puede observar, Java permite construir sentencias (expresiones compuestas) a partir de varias expresiones más pequeñas con tal que los tipos de datos requeridos por una parte de la expresión concuerden con los tipos de datos de la otra.

Bloques y ámbito

En Java el código fuente está dividido en partes separadas por llaves, denominas *bloques*. Cada bloque existe independiente de lo que está fuera de él, agrupando en su interior sentencias (expresiones) relacionadas.

Desde un bloque externo parece que todo lo que está dentro de llaves se ejecuta como una sentencia. Pero, ¿qué es un bloque externo?. Esto tiene explicación si entendemos que existe una *jerarquía de bloques*, y que un bloque puede contener uno o más subbloques anidados.

El concepto de ámbito está estrechamente relacionado con el concepto de bloque y es muy importante cuando se trabaja con variables en Java. El ámbito se refiere a cómo las secciones de un programa (bloques) afectan el tiempo de vida de las variables.

Toda variable tiene un ámbito, en el que es usada, que viene determinado por los bloques. Una variable definida en un bloque interno no es visible por el bloque externo. Las llaves de separación son importantes no sólo en un sentido lógico, ya que son la forma de que el compilador diferencie dónde acaba una sección de código y dónde comienza otra, sino que tienen una connotación estética que facilita la lectura de los

programas al ser humano.

Así mismo, para identificar los diferentes bloques se utilizan sangrías. Las sangrías se utilizan para el programador, no para el compilador. La sangría (también denominada *indentación*) más adecuada para la estética de un programa Java son dos espacios:

```
{
    //Bloque externo
    int x = 1;
    {
        //Bloque interno invisible al exterior
        int y = 2;
    }
    x = y; // Da error: Y fuera de ámbito
}
```

TIPOS DE DATOS

Tipos de datos simples

Es uno de los conceptos fundamentales de cualquier lenguaje de programación. Estos definen los métodos de almacenamiento disponibles para representar información, junto con la manera en que dicha información ha de ser interpretada.

Para crear una variable (de un tipo simple) en memoria debe declararse indicando su tipo de variable y su identificador que la identificará de forma única. La sintaxis de declaración de variables es la siguiente:

```
TipoSimple Identificador1, Identificador2;
Int num1, num2, nota, media=0;
```

Esta sentencia indica al compilador que reserve memoria para dos variables del tipo simple *TipoSimple* con nombres *Identificador1* e *Identificador2.*

Los tipos de datos en Java pueden dividirse en dos categorías: simples y compuestos. Los simples son tipos nucleares que no se derivan de otros tipos, como los enteros, de coma flotante, booleanos y de carácter. Los tipos compuestos se basan en los tipos simples, e incluyen las cadenas, las matrices y tanto las clases como las interfaces, en general.

Cada tipo de datos simple soporta un conjunto de literales que le pueden ser asignados, para darles valor. En este apartado se explican los tipos de datos simples (o primitivos) que presenta Java, así como los literales que soporta (sintaxis de los valores que se les puede asignar).

a.) Tipos de datos enteros

Se usan para representar números enteros con signo. Hay cuatro tipos: *byte, short, int y long.*

Tipo	Tamaño
byte	1Byte (8 bits)
short	2 Bytes (16 bits)
int	4 Bytes (32 bits)
long	8 Bytes (64 bits)

Tabla 5:
Tipos de datos enteros

Literales enteros
Son básicos en la programación en Java y presentan tres formatos:

Decimal: Los literales decimales aparecen como números ordinarios sin ninguna notación especial.
Hexadecimal: Los hexadecimales (base 16) aparecen con un *0x* ó *0X* inicial, notación similar a la utilizada en C y C++.
Octal: Los octales aparecen con un 0 inicial delante de los dígitos.

Por ejemplo, un literal entero para el número decimal 12 se representa en Java como 12 en decimal, como 0xC en hexadecimal, y como 014 en octal.

Los literales enteros se almacenan por defecto en el tipo *int*, (4 bytes con signo), o si se trabaja con números muy grandes, con el tipo *long*, (8 bytes con signo), añadiendo una L ó l al final del número.

La declaración de variables enteras es muy sencilla. Un ejemplo de ello sería:

```
long numeroLargo = 0xC; // Por defecto vale 12
```

b.) Tipos de datos en coma flotante

Se usan para representar números con partes fraccionarias. Hay dos tipos de coma flotante: *float* y *double*. El primero reserva almacenamiento para un número de precisión simple de 4 bytes y el segundo lo hace para un numero de precisión doble de 8 bytes.

Tipo	Tamaño
float	4 Byte (32 bits)
double	8 Bytes (64 bits)

Tabla 6: Tipos de datos numéricos en coma flotante

Literales en coma flotante

Representan números decimales con partes fraccionarias. Pueden representarse con notación estándar (563,84) o científica (5.6384e2).

De forma predeterminada son del tipo *double* (8 bytes). Existe la opción de usar un tipo más corto (el tipo *float* de 4 bytes), especificándolo con una F ó f al final del número.

La declaración de variables de coma flotante es muy similar a la de las variables enteras. Por ejemplo:

```
double miPi = 314.16e-2 ; // Aproximadamente

float temperatura = (float)36.6; // Paciente sin fiebre
```

Se realiza un moldeado a *temperatura*, porque todos los literales con decimales por defecto se consideran *double*.

c.) Tipo de datos boolean

Se usa para almacenar variables que presenten dos estados, que serán representados por los valores *true* y *false*. Representan valores bi-estado, provenientes del denominado *álgebra de Boole*.

Literales Booleanos

Java utiliza dos palabras clave para los estados: *true* (para verdadero) y *false* (para falso). Este tipo de literales es nuevo respecto a C/C++, lenguajes en los que el valor de falso se representaba por un 0 numérico, y verdadero cualquier número que no fuese el 0.

Para declarar un dato del tipo booleano se utiliza la palabra reservada *boolean:*

```
boolean reciboPagado = false; // ¡¿Aun no nos han pagado?!
```

d.) Tipo de datos carácter

Se usa para almacenar caracteres *Unicode* simples. Debido a que el conjunto de caracteres *Unicode* se compone de valores de 16 bits, el tipo de datos *char* se almacena en un entero sin signo de 16 bits.

Java a diferencia de C/C++ distingue entre matrices de caracteres y

cadenas. Literales carácter

Representan un único carácter (de la tabla de caracteres Unicode 1.1) y aparecen dentro de un par de comillas simples. De forma similar que en C/C++. Los caracteres especiales (de control y no imprimibles) se representan con una barra invertida ('\') seguida del código carácter.

Descripción	Representación	Valor Unicode
Caracter Unicode	*udddd*	
Numero octal	*ddd*	
Barra invertida	\\	*u005C*
Continuación	\	\
Retroceso	*b*	*u0008*
Retorno de carro	*r*	*u000D*
Alimentación de formularios	*f*	*u000C*
Tabulación horizontal	*t*	*u0009*
Línea nueva	*n*	*u000A*
Comillas simples	\’	*u0027*

Comillas dobles	\"	*\u0022*
Números arábigos ASCII	*0-9*	*\u0030 a \u0039*
Alfabeto ASCII en mayúsculas	*A.-Z*	*\u0041 a \u005A*
Alfabeto ASCII en minúsculas	*a.-z*	*\u0061 a \u007A*

Las variables de tipo *char* se declaran de la siguiente forma:

```
char letraMayuscula = 'A'; //Observe la necesidad de las ''

char letraV = '\u0056'; //Letra 'V'
```

e.) Conversión de tipos de datos

En Java es posible transformar el tipo de una variable u objeto en otro diferente al original con el que fue declarado. Este proceso se denomina "conversión", "moldeado" o "tipado". La conversión se lleva a cabo colocando el tipo destino entre paréntesis, a la izquierda del valor que queremos convertir de la forma siguiente:

```
char c = (char)System.in.read();
```

La función *read* devuelve un valor *int*, que se convierte en un *char* debido a la conversión *(char)*, y el valor resultante se almacena en la variable de tipo carácter *c*.

El tamaño de los tipos que queremos convertir es muy importante. No todos los tipos se convertirán de forma segura. Por ejemplo, al convertir un *long* en un *int*, el compilador corta los 32 bits superiores del *long* (de 64 bits), de forma que encajen en los 32 bits del *int*, con lo que si contienen información útil, esta se perderá.

Por ello se establece la norma de que "en las conversiones el tipo destino siempre debe ser igual o mayor que el tipo fuente":

Tipo Origen	Tipo Destino
byte	*double, float, long, int, char, short*
short	*double, float, long, int*
char	*double, float, long, int*
int	*double, float, long*
long	*double, float*
float	*double*

A. Vectores y Matrices

Una matriz es una construcción que proporciona almacenaje a una lista de elementos del mismo tipo, ya sea simple o compuesto. Si la matriz tiene solo una dimensión, se la denomina *vector*.

En Java los vectores se declaran utilizando corchetes (*[y])*, tras la declaración del tipo de datos que contendrá el vector. Por ejemplo, esta sería la declaración de un vector de números enteros (*int*):

```
int vectorNumeros[ ]; // Vector de números
```

Se observa la ausencia de un número que indique cuántos elementos componen el vector, debido a que Java no deja indicar el tamaño de un vector vacío cuando le declara. La asignación de memoria al vector se realiza de forma explícita en algún momento del programa.

Para ello o se utiliza el operador *new*:

```
int vectorNumeros = new int[ 5 ]; // Vector de 5 números
```

O se asigna una lista de elementos al vector:

```
int vectorIni = { 2, 5, 8}; // == int vectorIni[3]=new
int[3];
```

Se puede observar que los corchetes son opcionales en este tipo de declaración de vector, tanto después del tipo de variable como después del identificador.

Si se utiliza la forma de *new* se establecerá el valor *0* a cada uno de los elementos del vector.

B. Cadenas

En Java se tratan como una clase especial llamada *String*. Las cadenas se gestionan internamente por medio de una instancia de la clase *String*. Una instancia de la clase *String* es un objeto que ha sido creado siguiendo la descripción de la clase.

Cadenas constantes

Representan múltiples caracteres y aparecen dentro de un par de comillas dobles. Se implementan en Java con la clase *String*. Esta representación es muy diferente de la de C/C++ de cadenas como una matriz de caracteres. Cuando Java encuentra una constante de cadena, crea un caso de la clase *String* y define su estado, con los caracteres que aparecen dentro de las comillas dobles.

Vemos un ejemplo de cadena declarada con la clase *String* de

Java: `String capitalUSA = "Washington D.C.";`

```
String nombreBonito = "Amelia";
```

Ejercicios a realizar y entregar:

Todos los ejercicios se pueden realizar con el IDE Sublime y guardar con el nombre indicado y extensión .java ⌨ Ejemplo: *ejemplo1.java*

Ejercicio de ejemplo:

```
// EJEMPLO1
// SUMAR DOS VALORES ENTEROS INDICADOS EN LAS VARIABLES VALOR1 Y
VALOR2
// Y MOSTRAR EL RESULTADO POR PANTALLA


public class ejemplo1 {

        public static void main (String args[]) {
                //VARIABLES ENTERAS A UTILIZAR
                int     valor1;
                int     valor2;
                int
                resultado;
                //INDICAMOS LOS VALORES A SUMAR
                valor1 = 5;
                valor2 = 7;
                //REALIZAMOS LA OPERACION SUMA
                resultado = valor1 + valor2;
                //MOSTRAMOS EL RESULTADO POR PANTALLA
                System.out.println("El resultado es: " + resultado);
```

Ejercicio1:

Crear dos variables de tipo de *double* para mostrar por pantalla en otra variable el resultado de la suma, resta, multiplicación y división. Guardar como *ejercicio1.java*

Ejercicio 2:

Mostrar por pantalla la tabla de multiplicar del 0 al 10 del valor indicado en una variable de tipo *integer*. Por ejemplo la tabla del 8. Guardar como *ejercicio2.java*

Ejercicio 3:

Mostrar por pantalla la media aritmética de tres valores guardados en variables. Indica que tipo de datos seria el apropiado para este ejercicio. Guardar como *ejercicio3.java*