

TECNOLOGICO NACIONAL DE MEXICO Campus Ocotlán



INGENIERIA EN SISTEMAS COMPUTACIONALES

Lenguajes y automatas

Caso de estudio

Nombre del Alumno: ***Sergio Mateus Guzmán Zúñiga***

Número de Control: ***19630255***

Maestro: Villalvazo Mateos Raul

Caracteres ASCII de control			Caracteres ASCII imprimibles					
00	NULL	(carácter nulo)	32	espacio	64	@	96	`
01	SOH	(inicio encabezado)	33	!	65	A	97	a
02	STX	(inicio texto)	34	"	66	B	98	b
03	ETX	(fin de texto)	35	#	67	C	99	c
04	EOT	(fin transmisión)	36	\$	68	D	100	d
05	ENQ	(consulta)	37	%	69	E	101	e
06	ACK	(reconocimiento)	38	&	70	F	102	f
07	BEL	(timbre)	39	'	71	G	103	g
08	BS	(retroceso)	40	(72	H	104	h
09	HT	(tab horizontal)	41)	73	I	105	i
10	LF	(nueva línea)	42	*	74	J	106	j
11	VT	(tab vertical)	43	+	75	K	107	k
12	FF	(nueva página)	44	,	76	L	108	l
13	CR	(retorno de carro)	45	-	77	M	109	m
14	SO	(desplaza afuera)	46	.	78	N	110	n
15	SI	(desplaza adentro)	47	/	79	O	111	o
16	DLE	(esc.vínculo datos)	48	0	80	P	112	p
17	DC1	(control disp. 1)	49	1	81	Q	113	q
18	DC2	(control disp. 2)	50	2	82	R	114	r
19	DC3	(control disp. 3)	51	3	83	S	115	s
20	DC4	(control disp. 4)	52	4	84	T	116	t
21	NAK	(conf. negativa)	53	5	85	U	117	u
22	SYN	(inactividad sínc)	54	6	86	V	118	v
23	ETB	(fin bloque trans)	55	7	87	W	119	w
24	CAN	(cancelar)	56	8	88	X	120	x
25	EM	(fin del medio)	57	9	89	Y	121	y
26	SUB	(sustitución)	58	:	90	Z	122	z
27	ESC	(escape)	59	;	91	[123	{
28	FS	(sep. archivos)	60	<	92	\	124	
29	GS	(sep. grupos)	61	=	93]	125	}
30	RS	(sep. registros)	62	>	94	^	126	~
31	US	(sep. unidades)	63	?	95	_		
127	DEL	(suprimir)						

ASCII extendido
(Página de código 437)

128	Ç	160	á	192	Ł	224	Ó
129	û	161	í	193	ł	225	õ
130	é	162	ó	194	┐	226	ô
131	â	163	ú	195	└	227	ò
132	ä	164	ñ	196	—	228	ö
133	à	165	Ñ	197	+	229	Õ
134	á	166	ª	198	ä	230	μ
135	ç	167	º	199	Ä	231	þ
136	ê	168	¿	200	Ł	232	Þ
137	ë	169	®	201	┌	233	Ú
138	è	170	¬	202	└	234	Ù
139	ï	171	½	203	┐	235	Ý
140	î	172	¼	204	└	236	ý
141	ì	173	¡	205	=	237	Ÿ
142	Ä	174	«	206	≠	238	—
143	Á	175	»	207	□	239	·
144	É	176	⋯	208	ø	240	≡
145	æ	177	⋮	209	Ð	241	±
146	Æ	178	⋮	210	È	242	—
147	ô	179	└	211	È	243	¾
148	ö	180	└	212	È	244	¶
149	ò	181	À	213	¡	245	§
150	ù	182	Á	214	í	246	÷
151	ú	183	Â	215	î	247	°
152	ÿ	184	©	216	ï	248	°
153	Ö	185	└	217	└	249	…
154	Ü	186	└	218	└	250	·
155	ø	187	└	219	■	251	¹
156	£	188	└	220	■	252	ª
157	Ø	189	¢	221	└	253	²
158	×	190	¥	222	└	254	■
159	f	191	└	223	■	255	nbsp

de uso frecuente
(idioma español)

ñ	alt + 164
Ñ	alt + 165
@	alt + 64
¿	alt + 168
?	alt + 63
¡	alt + 173
!	alt + 33
:	alt + 58
/	alt + 47
\	alt + 92

vocales con acento
(acento agudo español)

á	alt + 160
é	alt + 130
í	alt + 161
ó	alt + 162
ú	alt + 163
Á	alt + 181
É	alt + 144
Í	alt + 214
Ó	alt + 224
Ú	alt + 233

vocales
con diéresis

ä	alt + 132
ë	alt + 137
ï	alt + 139
ö	alt + 148
ü	alt + 129
Ä	alt + 142
Ë	alt + 211
Ï	alt + 216
Ö	alt + 153
Ü	alt + 154

símbolos matemáticos		símbolos comerciales		comillas, llaves paréntesis	
$\frac{1}{2}$	alt + 171	\$	alt + 36	"	alt + 34
$\frac{1}{4}$	alt + 172	£	alt + 156	'	alt + 39
$\frac{3}{4}$	alt + 243	¥	alt + 190	(alt + 40
¹	alt + 251	¢	alt + 189)	alt + 41
³	alt + 252	¤	alt + 207	[alt + 91
²	alt + 253	®	alt + 169]	alt + 93
<i>f</i>	alt + 159	©	alt + 184	{	alt + 123
±	alt + 241	®	alt + 166	}	alt + 125
×	alt + 158	°	alt + 167	«	alt + 174
÷	alt + 246	°	alt + 248	»	alt + 175

Objetivo del lenguaje: El proposito principal del lenguaje sera de multiproposito

Sintaxis:

Entradas

Scanner → Lector

int → Entero

String → Cadena o Texto

float → Flotante

double → Doble

boolean → Booleano

Salida

Print → Imprimir o mostrar

Operaciones

Suma → +

Resta → -

Multiplicación → *

División → /

Módulo → %

Operaciones Aritméticas

+ → Suma

- → Resta

* → Multiplicación

/ → División

% → Módulo

Operaciones Racionales

Igual a $\rightarrow ==$

No igual a $\rightarrow !=$

Mayor que $\rightarrow >$

Menor que $\rightarrow <$

Mayor o igual que $\rightarrow >=$

Menor o igual que $\rightarrow <=$

Operadores lógicos

Y lógico $\rightarrow \&\&$

O lógico $\rightarrow ||$

No lógico $\rightarrow !$

Operador de asignación

Asignación $\rightarrow =$

Condiciones

If \rightarrow Si

If - else \rightarrow Si – No

for \rightarrow Repetir

While \rightarrow Mientras

Do while \rightarrow Hacer mientras

`abstract` → Abstracto → Se utiliza para definir clases y métodos abstractos.

`assert` → Afirmar → Sirve para afirmar que una condición es cierta.

`boolean` → Booleano → Tipo de dato primitivo booleano (lógico), que puede ser `true` o `false`.

`break` → Romper → Instrucción de salto que interrumpe (rompe) la ejecución de un bucle o de una instrucción de control alternativa múltiple (`switch`).

`byte` → Entero_8 → Tipo de dato primitivo número entero (integer) de 8 bits.

`case` → Caso → Caso de una instrucción de control alternativa múltiple (`switch`).

`catch` → Atrapar → Cláusula de un bloque `try` donde se especifica una excepción.

`char` → Entero_16 → Tipo de dato primitivo carácter (valor Unicode) de 16 bits.

`class` → Clase → Sirve para definir una clase.

`const *` → Constante* → No se utiliza.

`continue` → Continuar → Instrucción de salto que interrumpe (rompe) la ejecución de la iteración de un bucle. Pero, permitiendo continuar al bucle seguir realizando otras iteraciones.

`default` → Defecto → Caso por defecto de una instrucción de control alternativa múltiple (`switch`).

`do` → Hacer → Se usa en la sintaxis de un bucle hacer mientras (`do while`).

`double` → Doble → Tipo de dato primitivo número real en coma flotante con precisión doble (`double-precision floating-point`) de 64 bits.

`else` → Sino → Si no, en una instrucción de control alternativa doble (`if else`).

`enum` → Enumerar → Sirve para definir tipos de datos enumerados.

`extends` → Extensión → Cláusula que permite indicar la clase padre de una clase.

`final` → Fin → Permite indicar que una variable no se puede modificar, un método no se puede redefinir o de una clase no se puede heredar.

`finally` → Finalizar → Clausula que permite especificar un bloque de código que siempre se ejecutará, se produzca o no una excepción en un bloque `try`.

`float` → Flotante → Tipo de dato primitivo número real en coma flotante con precisión simple (single-precision floating-point) de 32 bits.

`for` → Repetir → Instrucción de control repetitiva para.

`goto *` → Salto → Instrucción de salto (ir a). No se usa.

`if` → Si → Se emplea para escribir instrucciones de control alternativas simples (`if`) o dobles (`if else`).

`implements` → Implementar → Sirve para definir la o las interfaces de una clase.

`import` → Importar → Permite importar un paquete (`package`).

`instanceof` → Instancia-F → Operador que permite saber si un objeto es una instancia de una clase concreta.

`int` → Entero → Tipo de dato primitivo número entero (`integer`) de 32 bits.

`interface` → Interfaz → Se utiliza para declarar una interfaz.

`long` → Longitud → Tipo de dato primitivo número entero (`integer`) de 64 bits.

`native` → Nativo → Modificador que se utiliza para indicar que un método está implementado en un lenguaje de programación (distinto a Java) dependiente de la plataforma.

`new` → Nuevo → Operador que se utiliza para crear un objeto nuevo de una clase.

`package` → Paquete → Agrupa a un conjunto de clases.

private → Privado → Modificador de acceso para indicar que un elemento es accesible únicamente desde la clase donde se ha definido.

protected → Protección → Modificador de acceso para indicar que un elemento es accesible desde la clase donde se ha definido, subclases de ella y otras clases del mismo paquete (package).

public → Público → Modificador de acceso para indicar que un elemento es accesible desde cualquier clase.

return → Retorno → Se usa para indicar el valor de retorno de un método.

short → Corto → Tipo de dato primitivo número entero (integer) de 16 bits.

static → Estático → Permite especificar que un elemento es único en una clase, no pudiendo existir instancias de esa clase que contengan a dicho elemento.

strictfp → Estático-F → Se usa para indicar que se tienen que utilizar cálculos en coma flotante estricto (strict floating point).

super → Super → Permite invocar a un método o constructor de la superclase.

switch → Cambiar → Instrucción de control alternativa múltiple.

synchronized → Sincronizar → Modificador que se utiliza para indicar que un método o bloque de código es atómico.

this → Este → Se utiliza para referenciar al objeto actual, así como para invocar a un constructor de la clase a la que pertenece dicho objeto.

throw → Lanza → Permite lanzar una excepción

throws → Lanzar → Sirve para indicar las excepciones que un método puede lanzar.

transient → Transito → Sirve para especificar que un atributo no sea persistente.

try → Tratar → Permite especificar un bloque de código donde se quieren atrapar excepciones.

void → Vacío → Tipo de dato vacío (sin valor).

volatile → Volatil → Modificador que se usa para indicar que el valor de un atributo que está siendo utilizado por varios hilos (threads) esté sincronizado.

while → Mientras → Se usa para escribir bucles mientras (while) y bucles hacer mientras (do while).

Funcionamiento de los ciclos:

Los ciclos en mi lenguaje,, son estructuras de control que permiten ejecutar un bloque de código repetidamente mientras se cumple una condición específica. Ofrecere tres tipos principales de ciclos: for, while, y do-while.

Gramatica :

Mi lenguaje solo aceptara letras minusculas para una mayor sencillez (recomendacion del maestro)

- Nombres de identificadores: Los nombres de variables, métodos y clases deben seguir ciertas reglas:
- Deben comenzar con una letra (a-z) o un guion bajo (_).

Pueden contener letras, dígitos (0-9) y guiones bajos.

No pueden ser palabras clave reservadas de Java (como public, class, if, etc.).

Puntuación: Java utiliza una serie de símbolos de puntuación y operadores para definir estructuras de control y expresiones. Estos incluyen paréntesis, llaves, corchetes, comas, punto y coma, operadores aritméticos, operadores lógicos, etc. Estos símbolos deben usarse correctamente según las reglas sintácticas.

Comentarios: Puedes agregar comentarios en tu código Java para hacerlo más legible. Los comentarios de una sola línea comienzan con //, y los comentarios de varias líneas se encierran entre /* y */.

- Bloques de código: Las estructuras de control como if, for, while, do-while y las definiciones de clases y métodos utilizan bloques de código delimitados por llaves {} para agrupar declaraciones relacionadas.
- · Declaraciones terminadas con punto y coma: Cada declaración en Java generalmente debe terminar con un punto y coma ;. Esto indica el final de la declaración.
- · Uso correcto de paréntesis: Los paréntesis () se utilizan para agrupar expresiones y para definir argumentos en llamadas a métodos y declaraciones de métodos.
- · Heredar reglas de formato: Aunque no son reglas gramaticales estrictas, seguir convenciones de formato de código (como sangrías, nombres de variables significativos) es importante para que el código sea legible y mantenible.
- · Palabras clave reservadas: Java tiene un conjunto de palabras clave reservadas que no pueden ser utilizadas como identificadores (nombres de variables, clases, métodos, etc.) en tu código. Algunas palabras clave incluyen class, public, static, void, if, while, for, entre otras.

Estructuras y tipos de datos:

Tipos de Datos Primitivos: Estos son tipos de datos básicos incorporados en el lenguaje Java y no son objetos. Los tipos de datos primitivos incluyen:

int: Números enteros.

double: Números en punto flotante de doble precisión.

boolean: Valores true o false, (long y short también están incluidos).

Arreglos (Arrays): Los arreglos son colecciones de elementos del mismo tipo. Pueden ser unidimensionales o multidimensionales y se utilizan para almacenar múltiples valores del mismo tipo.

Clases y Objetos Personalizados: Java permite a los programadores crear sus propias clases y objetos personalizados para modelar datos y comportamientos específicos de la aplicación.

Analizador lexico y sintactico:

Analizador Léxico (Scanner):

El analizador léxico, también conocido como scanner o tokenizer, es la primera etapa del proceso de compilación.

Su función principal es dividir el código fuente en tokens, que son unidades léxicas básicas, como palabras clave, identificadores, operadores y literales (como números y cadenas de caracteres).

El analizador léxico también puede eliminar comentarios y espacios en blanco del código fuente, ya que estos no son relevantes para la estructura del programa.

Los tokens generados se pasan al analizador sintáctico para su posterior procesamiento.

Analizador Sintáctico (Parser):

El analizador sintáctico es la segunda etapa del proceso de compilación, que sigue al análisis léxico.

Su objetivo es verificar si la secuencia de tokens generada por el analizador léxico cumple con las reglas sintácticas del lenguaje Java, es decir, si el código fuente está escrito de acuerdo con la gramática del lenguaje.

El analizador sintáctico crea un árbol de sintaxis abstracta (AST) a partir de los tokens, que representa la estructura jerárquica del programa.

Si el código fuente contiene errores sintácticos, el analizador sintáctico los detecta y puede generar mensajes de error que indican la ubicación y la naturaleza de los errores.

Si el análisis sintáctico es exitoso y no se encuentran errores, se pasa el AST resultante a las etapas posteriores del proceso de compilación, como el análisis semántico y la generación de código intermedio.

