	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	38/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Guía práctica de estudio 03: Fundamentos y sintaxis del lenguaje

---





---

### *Elaborado por:*

M.C. M. Angélica Nakayama C.  
Ing. Jorge A. Solano Gálvez

### *Autorizado por:*

M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	39/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Guía práctica de estudio 03: Fundamentos y sintaxis del lenguaje

### Objetivo:

Implementar programas utilizando:

- Diversos tipos de datos.
- Expresiones (operadores, declaraciones, etc.)


### Actividades:

- Utilizar diferentes tipos de datos en un lenguaje de programación.
- Implementar expresiones entre diferentes tipos de datos.

### Introducción

Los lenguajes de programación tienen elementos básicos que se utilizan como bloques constructivos, así como reglas para que esos elementos se combinen. Estas reglas se denominan **sintaxis del lenguaje**. Solamente las instrucciones sintácticamente correctas pueden ser interpretadas por la computadora. Los programas que contengan errores de sintaxis son rechazados por la máquina.

La **sintaxis** de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	40/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Los elementos básicos constructivos de un programa son:

- Palabras reservadas (propias de cada lenguaje).
- Identificadores (nombres de variables, nombres de funciones, nombre del programa, etc.)
- Caracteres especiales (alfabeto, símbolos de operadores, delimitadores, comentarios, etc.)
- Expresiones.
- Instrucciones.

**NOTA:** En esta guía se tomará como caso de estudio el lenguaje de programación JAVA, sin embargo, queda a criterio del profesor el uso de éste u otro lenguaje orientado a objetos.


## Sintaxis básica

Una de las primeras cosas que hay que tener en cuenta es que Java es un lenguaje **sensitivo** a mayúsculas/minúsculas. El compilador Java hace distinción entre mayúsculas y minúsculas, esta distinción no solo se aplica a palabras reservadas del lenguaje sino también a nombres de variables y métodos.

La sintaxis de Java es muy parecida a la de C y C++, por ejemplo, las sentencias finalizan con ';', los bloques de instrucciones se delimitan con llaves { y }, etc. A continuación, se explican los puntos más relevantes de la sintaxis de Java.

## Comentarios

Los comentarios son muy útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. Además, permite que cualquier persona distinta al programador original pueda comprender el código escrito de una forma más rápida. Se recomienda acostumbrarse a comentar el código desarrollado. De esta forma se simplifica también la tarea de estudio y revisión posteriores.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	41/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

En Java existen tres tipos de comentarios:

- Comentarios de una sola línea.

*// Esta es una línea comentada.*

- Comentarios de bloques.

*/\* Aquí el comentario por bloque*

*y puede abarcar diferentes renglones \*/*


- Comentarios de documentación (**JavaDoc**).

*/\*\* Los comentarios de documentación se realizan de este modo \*/*

## Identificadores

En Java los identificadores comienzan por una letra del alfabeto inglés, un guión bajo \_ o el símbolo de dólar \$, los siguientes caracteres del identificador pueden ser letras o dígitos (0-9). No se debe iniciar con un dígito. No hay un límite en lo concerniente al número de caracteres que pueden tener los identificadores.

Las reglas del lenguaje respecto a los nombres de variables son muy amplias y permiten mucha libertad al programador, pero es habitual seguir ciertas normas que facilitan la lectura y el mantenimiento de los programas.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	42/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


En Java es habitual utilizar nombres con minúsculas, con las excepciones que se indican en los puntos siguientes:

- Los nombres de **clases** e **interfaces** comienzan siempre por mayúscula. Ejemplos: *Geometria*, *Rectangulo*, *Dibujable*, *Graphics*, *Iterator*.
- Cuando un nombre consta de varias palabras se declaran una a continuación de otra, poniendo con mayúscula la primera letra de la palabra que sigue (**CammelCase**). Ejemplos: *elMayor()*, *VentanaCerrable*, *RectanguloGrafico*, *addWindowListener()*.
- Los nombres de **objetos**, **métodos**, **variables miembro** y **variables locales** de los métodos, comienzan siempre por minúscula. Ejemplos: *main()*, *dibujar()*, *numRectangulos*, *x*, *y*, *r*.
- Los nombres de las **variables finales**, es decir de las **constantes**, se definen siempre con mayúsculas. Ejemplo: *PI*

Sin embargo, éstas no son las únicas normas para la nomenclatura en Java, también existen las **convenciones de código**, las cuales son importantes para los programadores dado que mejoran la lectura del programa, permitiendo entender código nuevo mucho más rápidamente y más a fondo.

Para que funcionen, todos los programadores deben seguir la convención, por esta razón, es muy importante generar el hábito de aplicar convenciones y estándares de programación ya que de esta manera no solo se enfoca en las funcionalidades, sino que también se aporta a la calidad de los desarrollos.

Las convenciones de código o **Java Code Conventions**, pueden ser consultadas en el siguiente enlace: <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	43/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

### *Palabras reservadas*

Las siguientes son palabras reservadas utilizadas por Java y no pueden ser usadas como identificadores.

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert	enum				


También existen las literales reservadas *null*, *true* y *false*, las cuales tampoco pueden ser usadas como identificadores.

## **Tipos de datos**

En Java existen dos grupos de tipos de datos: tipos primitivos y tipos referencia.

### *Tipos de dato primitivos*

Se llaman **tipos primitivos** a aquellos datos sencillos que contienen los tipos de información más habituales: valores booleanos, caracteres y valores numéricos enteros o de punto flotante.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	44/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Java dispone de ocho tipos primitivos:


Tipo	Definición
boolean	<code>true</code> o <code>false</code>
char	Carácter Unicode de 16 bits
byte	Entero en complemento a dos con signo de 8 bits
short	Entero en complemento a dos con signo de 16 bits
int	Entero en complemento a dos con signo de 32 bits
long	Entero en complemento a dos con signo de 64 bits
float	Real en punto flotante según la norma IEEE 754 de 32 bits
double	Real en punto flotante según la norma IEEE 754 de 64 bits

En Java al contrario que en C o C++ el tamaño de los tipos primitivos no depende del sistema operativo o de la arquitectura ya que en todas las arquitecturas y bajo todos los sistemas operativos el tamaño en memoria es el mismo.

Es posible recubrir los tipos primitivos para tratarlos como cualquier otro objeto en Java. Así, por ejemplo, existe una clase envoltura del tipo primitivo **int** llamado **Integer**. La utilidad de estas clases se explicará en otro momento.

### *Tipos de dato referencia*

Los **tipos de dato referencia** representan datos compuestos o estructuras, es decir, referencias a objetos. Estos tipos de dato almacenan las direcciones de memoria y no el valor en sí (similares a los apuntadores en C). Una referencia a un objeto es la dirección de un área en memoria destinada a representar ese objeto.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	45/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Entrada y salida de datos por consola

Una de las operaciones más habituales que tiene que realizar un programa es intercambiar datos con el exterior. Para ello la **API** de java incluye una serie de clases que permiten gestionar la entrada y salida de datos en un programa, independientemente de los dispositivos utilizados para el envío/recepción de datos.

Para el envío de datos al exterior se utiliza un flujo de datos de impresión o **print stream**. Esto se logra usando la siguiente expresión:

```
System.out.println("Mi mensaje");
```

De manera análoga, para la recepción o lectura de datos desde el exterior se utiliza un flujo de datos de entrada o **input stream**. Para lectura de datos se utiliza la siguiente sintaxis:

```
Scanner sc = new Scanner(System.in);
```

```
String s = sc.next(); //Para cadenas
```

```
int x = sc.nextInt(); //Para enteros
```


Para usar la clase Scanner se debe incluir al inicio del archivo la siguiente línea:

```
import java.util.Scanner;
```

Al finalizar su uso se debe cerrar el flujo utilizando el método **close**. Para este ejemplo:

```
sc.close();
```



	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	46/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Operadores

Java es un lenguaje rico en operadores, los cuales son casi idénticos a los de C/C++.

### Operadores aritméticos:

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (\*), división (/) y resto de la división o módulo (%).

### Operadores de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación por excelencia es el operador igual (=). La forma general de las sentencias de asignación con este operador es:


*variable = expression;*

Java dispone de otros operadores de asignación. Se trata de versiones abreviadas del operador (=) que realizan operaciones “acumulativas” sobre una variable.

Operador	Utilización	Expresión equivalente
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

### Operadores unarios

Los operadores más (+) y menos (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica. Su uso en Java es el estándar de estos operadores.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	47/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

### Operador instanceof

El operador **instanceof** permite saber si un objeto **pertenece o no** a una determinada clase. Es un operador binario cuya forma general es:

*objectName instanceof ClassName*

Este operador devuelve **true** si el objeto pertenece a la clase o **false** en caso contrario.

### Operador condicional

El operador condicional **?**, tomado de C/C++, permite realizar bifurcaciones condicionales sencillas. Su forma general es la siguiente:

*booleanExpression ? res1 : res2*


Donde se evalúa **booleanExpression** y se devuelve **res1** si el resultado es **true** y **res2** si el resultado es **false**. Es el único operador ternario (tres argumentos) de Java. Como todo operador que devuelve un valor puede ser utilizado en una expresión.

### Operadores incrementales

Java dispone del operador incremento (**++**) y decremento (**--**). El operador (**++**) incrementa en una unidad la variable a la que se aplica, mientras que (**--**) la reduce en una unidad. Estos operadores se pueden utilizar de dos formas:

- Precediendo a la variable (por ejemplo: **++i**). En este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.
- Siguiendo a la variable (por ejemplo: **i++**). En este caso primero se utiliza la variable en la expresión (con el valor anterior) y luego se incrementa.

La actualización de contadores en ciclos **for** es una de las aplicaciones más frecuentes de estos operadores.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	48/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

### Operadores relacionales

Los operadores relacionales sirven para realizar **comparaciones** de igualdad, desigualdad y relación de menor o mayor. El resultado de estos operadores es siempre un valor **boolean** (**true** o **false**) según se cumpla o no la relación considerada.

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes


Estos operadores se utilizan con mucha frecuencia en las estructuras de control.

### Operadores lógicos

Los operadores lógicos se utilizan para construir **expresiones lógicas**, combinando valores lógicos (**true** y/o **false**) o los resultados de los operadores relacionales.

Operador	Nombre	Utilización	Resultado
&&	AND	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
	OR	op1    op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	negación	! op	true si op es false y false si op es true
&	AND	op1 & op2	true si op1 y op2 son true. Siempre se evalúa op2
	OR	op1   op2	true si op1 u op2 son true. Siempre se evalúa op2

Debe notarse que en ciertos casos el segundo operando no se evalúa porque ya no es necesario (si ambos tienen que ser true y el primero es false, ya se sabe que la condición de que ambos sean true no se va a cumplir). Esto puede traer resultados no deseados y por eso se han añadido los operadores (&) y (|) que garantizan que los dos operandos se evalúan siempre.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	49/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

### Operador de concatenación de cadenas de caracteres

El operador más (+) se utiliza también para concatenar cadenas de caracteres. Por ejemplo, para escribir una cantidad con un rótulo y valores puede utilizarse la sentencia:

```
System.out.println("El total asciende a " + result + " unidades");
```


Donde el operador de concatenación se utiliza dos veces para construir la cadena de caracteres que se desea imprimir por medio del método println( ). La variable numérica result es convertida automáticamente por Java en cadena de caracteres para poderla concatenar. En otras ocasiones se deberá llamar explícitamente a un método para que realice esta conversión.

### Operadores a nivel de bits

Java dispone también de un conjunto de operadores que actúan a nivel de bits. Las operaciones de bits se utilizan con frecuencia para definir señales o flags, esto es, variables de tipo entero en las que cada uno de sus bits indica si una opción está activada o no.

Operador	Utilización	Resultado
>>	op1 >> op2	Desplaza los bits de op1 a la derecha una distancia op2
<<	op1 << op2	Desplaza los bits de op1 a la izquierda una distancia op2
>>>	op1 >>> op2	Desplaza los bits de op1 a la derecha una distancia op2 (positiva)
&	op1 & op2	Operador AND a nivel de bits
	op1   op2	Operador OR a nivel de bits
^	op1 ^ op2	Operador XOR a nivel de bits (1 si sólo uno de los operandos es 1)
~	~op2	Operador complemento (invierte el valor de cada bit)

Operador	Utilización	Equivalente a
&=	op1 &= op2	op1 = op1 & op2
=	op1  = op2	op1 = op1   op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2


	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	50/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

### *Precedencia de operadores*

El **orden** en que se realizan las operaciones es fundamental para determinar el resultado de una expresión. La siguiente lista muestra el orden en que se ejecutan los distintos operadores en una sentencia, de mayor a menor precedencia:

<b>postfix operators</b>	[ ] . (params) expr++ expr--
<b>unary operators</b>	++expr --expr +expr -expr ~ !
<b>creation or cast</b>	new (type)expr
<b>multiplicative</b>	* / %
<b>additive</b>	+ -
<b>shift</b>	<< >> >>>
<b>relational</b>	< > <= >= instanceof
<b>equality</b>	== !=
<b>bitwise AND</b>	&
<b>bitwise exclusive OR</b>	^
<b>bitwise inclusive OR</b>	
<b>logical AND</b>	&&
<b>logical OR</b>	
<b>conditional</b>	? :
<b>assignment</b>	= += -= *= /= %= &= ^=  = <<= >>= >>>=

En Java, todos los operadores binarios (excepto los operadores de asignación) se evalúan de **izquierda a derecha**. Los operadores de asignación se evalúan de derecha a izquierda, lo que significa que el valor de la derecha se copia sobre la variable de la izquierda.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	51/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía

*Martín, Antonio*

***Programador Certificado Java 2.***

*Segunda Edición.*

*México*

*Alfaomega Grupo Editor, 2008*

*Sierra Katy, Bates Bert*

***SCJP Sun Certified Programmer for Java 6 Study Guide***

*Mc Graw Hill*

*Dean John, Dean Raymond.*

***Introducción a la programación con Java***

*Primera Edición.*

*México*

*Mc Graw Hill, 2009*