

**MASTER EN MODELIZACIÓN
MATEMÁTICA, ESTADÍSTICA Y
COMPUTACIÓN
2011-2012**

Curso: Bases de datos y programación
orientada a objetos
Parte POO

Programación sin Orientación a Objetos.

Programación sin Orientación a Objetos

abstract	boolean	break	byte	case	catch
char	class	const*	continue	default	do
double	else	enum	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

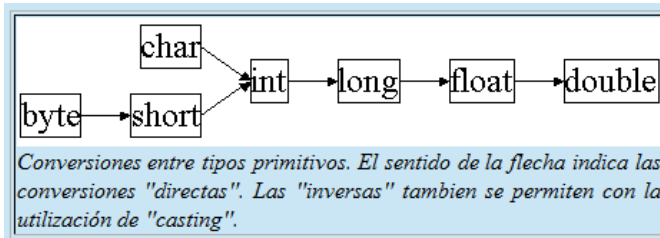
* Palabra clave no usada en la actualidad

** añadida para Java 2

Los términos **null**, **true** y **false** están predefinidos pero no son palabras reservadas.

Tipos PRIMITIVOS (no son objetos. Java es Híbrido)
Son SIEMPRE IGUALES (no cambian con las plataformas)

Tipos primitivos		
Enteros [complemento a dos con signo]	byte	8 bits
	short	16 bits
	int	32 bits
	long	64 bits
Reales [IEEE 754]	float	32 bits
	double	64 bits
Caracteres [Unicode]	char	16 bits
Booleanos [dpte. implementación]	boolean	
no-tipo		
void		



```
float f;
double g=3.14159;
f=(float)g;

long l=32; //la constante 32 es int y se convierte automáticamente a long
char c=(char)l;
```

Palabras reservadas en Java					
abstract	assert	boolean	break	byte	case
catch	char	class	const*	continue	default
do	double	else	enum	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				

Tienen sus equivalentes como objetos.
 (Hay otros tipos sólo como objetos, p.ej. Binario, precisión infinita, etc.)

```
public class MaxVariablesDemo {
    public static void main(String args[]) {

        // enteros
        byte maximoByte = Byte.MAX_VALUE;
        short maximoShort = Short.MAX_VALUE;
        int maximoInteger = Integer.MAX_VALUE;
        long maximoLong = Long.MAX_VALUE;

        // reales
        float maximoFloat = Float.MAX_VALUE;
        double maximoDouble = Double.MAX_VALUE;

        // otros tipos primitivos
        char unChar = 'S'; //existe Character
        boolean unBoolean = Boolean.TRUE; //también válido: boolean
        unBoolean=true

        // (aquí continuará la definición de la clase)
    }
}
```

IDENTIFICADORES`(letra|_|$) (letra|digito|_|$)*`**LITERALES**

ENTERO:

`{[+]-} [0[x]] constante_entera [{l|L}]`

“0” indica constante expresada en octal.

“0x” indica constante expresada en hexadecimal.

“L” y “l” indican tipo long.

REAL:

`{[+]-} parte_entera,parte_fraccionaria [{e|E} {[+]-} exponente]`

“parte_entera”, “parte_fraccionaria” y “exponente” son const. enteras.

BOOLEANO:

`{true|false}`

CARACTERES:

(ejemplos) `'x'`, `\n`, `\u001B`

CADENAS:

(ejemplos) `“hola”`, `“\”Hola\”`OBJETO NULO: `null`*Identificadores y literales en Java.*

Java sensible a la capitalización, y no pone límites a la longitud de los identificadores.

Sobre estas características se “acuerdan” numerosas convenciones (no las exige el JDK ni los IDEs, pero las siguen los desarrolladores), p.ej. “esto” es un objeto, “Esto” es una clase, “setElement” es una rutina que tiene por función dar valor a un objeto o variable “element”, etc

Operador de asignación

Op.	Uso	Operación
=	op1 = op2	Asigna Op2 a op1

Operadores Aritméticos

Op.	Uso	Descripción
+	op1 + op2	Suma op1 y op2 (*también usado como concatenación de cadenas)
-	op1 - op2	Resta op2 de op1
*	op1 * op2	Multiplica op1 por op2
/	op1 / op2	Divide op1 por op2
%	op1 % op2	Obtiene el resto de la división de op1 entre op2
++	op++	Incrementa op en 1; evalúa op antes del incremento
++	++op	Incrementa op en 1; evalúa op después del incremento
--	op--	Decrementa op en 1; evalúa op antes del incremento
--	--op	Decrementa op en 1; evalúa op después del incremento
+	+op	“Promueve” op a int si es byte, short, o char
-	-op	Niega op aritméticamente

++op op++ --op op--

Si x=1 e y=1

Entonces (z= ++x + y) resulta x=2, y=1, z=3

Equivale a (x=x+1; z=x+y)

Si x=1 e y=1

Entonces (z= x++ + y) resulta x=2, y=1, z=2

Equivale a (z=x+y ; x=x+1)

Lo mismo sucede con -op y op--

Operadores Relacionales y Condicionales

Op.	Uso	"true" si...
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 es mayor o igual que op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 y op2 son iguales
!=	op1 != op2	op1 y op2 son distintos
&&	op1 && op2	op1 y op2 son ambos "true", evalúa op2 condicionalmente
	op1 op2	bien op1 o bien op2 es "true", evalúa op2 condicionalmente
!	! op	op es falso
&	op1 & op2	op1 y op2 son ambos ciertos, siempre evalúa op1 y op2
	op1 op2	bien op1 o bien op2 es "true", siempre evalúa op1 y op2
^	op1 ^ op2	si op1 y op2 son uno cierto y otro falso

Operadores de desplazamiento y lógicos

Op.	Uso	Operación
>>	op1 >> op2	Desplaza los bits de op1 a la derecha en op2 posiciones
<<	op1 << op2	Desplaza los bits de op1 a la izquierda en op2 posiciones
>>>	op1 >>> op2	Desplaza los bits de op1 a la derecha en op2 posiciones (sin signo)
&	op1 & op2	"Y" lógico bit a bit
	op1 op2	"O" lógico bit a bit
^	op1 ^ op2	"O exclusivo" lógico bit a bit
~	~op2	Complemento bit a bit

Otros operadores

Op.	Use	Description
?:	op1 ? op2 : op3	Si op1 es "true", devuelve op2, si no, devuelve op3.
[]	type []	Declara un array de longitud indeterminada de elementos tipo.
[]	type[op1]	Crea un array con op1 elementos. Usado con new.
[]	op1[op2]	Accede al elemento de índice op2 en el array op1. Los índices comienzan en cero y van hasta la longitud menos uno..
.	op1.op2	Es una referencia al miembro op2 de op1.
()	op1(params)	Declara o llama al método llamado op1 con los parámetros especificados. Los parámetros en la lista se separan por comas, y ésta puede estar vacía.
(type)	(type) op1	Convierte (cast) op1 al tipo "type". Se arroja una excepción si el tipo de op1 no es compatible con "type".
new	new op1	Crea un nuevo objeto o array. op1 es una llamada a un constructor o una especificación de array.
instanceof	op1 instanceof op2	Devuelve el valor "true" si op1 es una instancia de op2

op1 ? op2 : op3

Si b=true, x=1 e y=2
Entonces (z = b?x:y) resulta z=1

Si b=false, x=1 e y=2
Entonces (z = b?x:y) resulta z=2

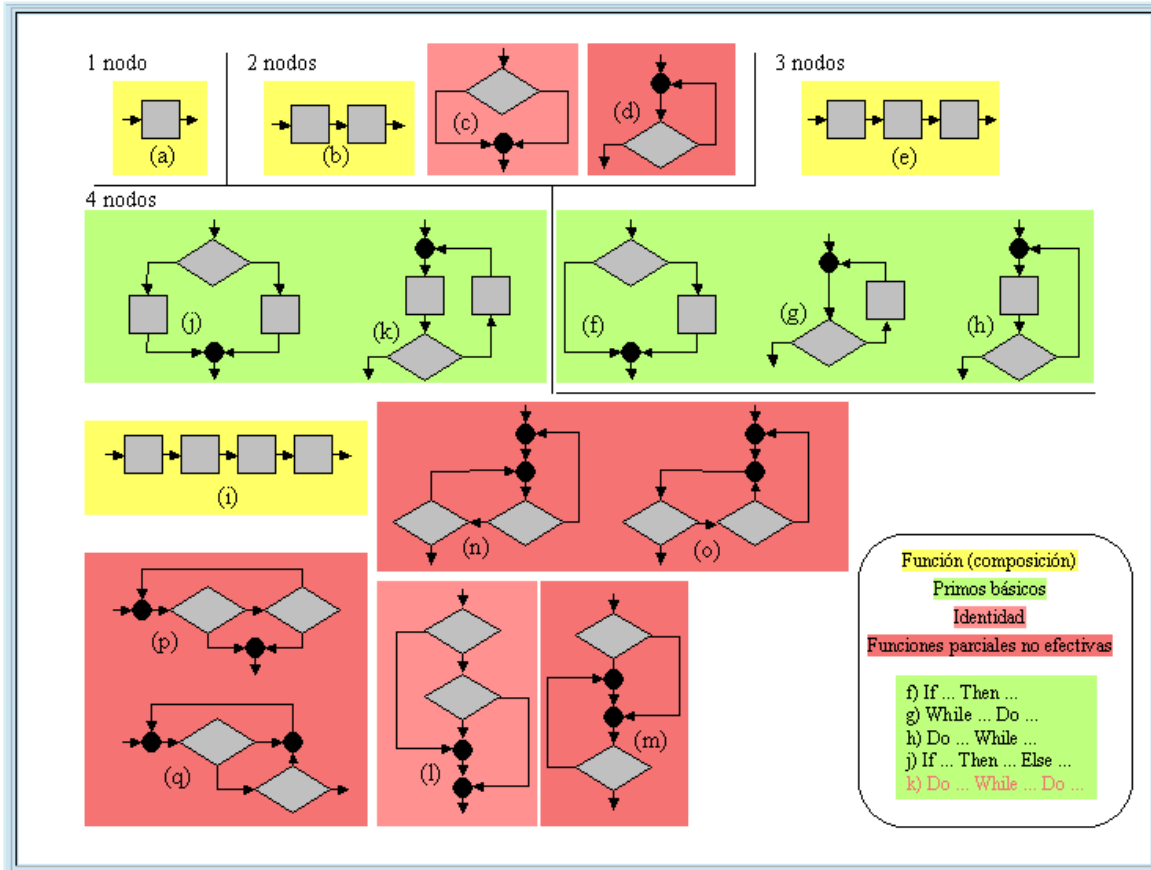
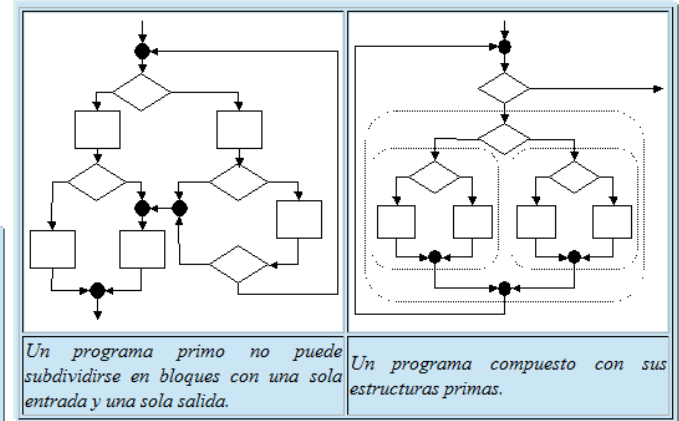
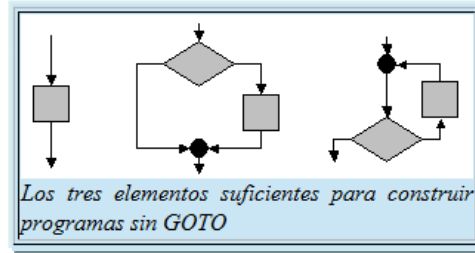
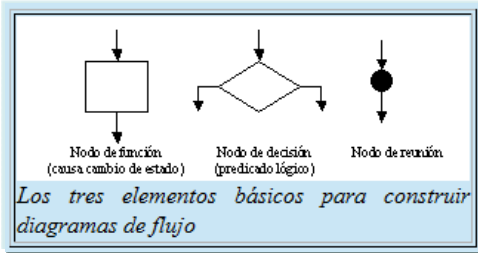
b, x e y pueden ser expresiones

(véase ejemplo en sentencia if-then-else)

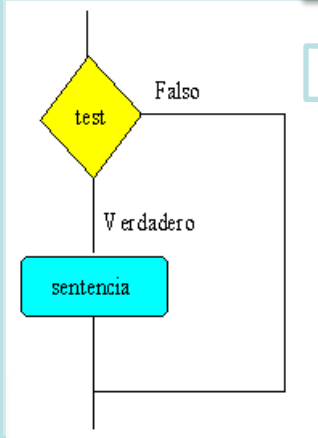
Operadores de asignación combinados

Op.	Uso	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
=	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

Asociatividad	Operadores
→	() [.
←	- ~ ! ++ --
←	new (tipo) expresión
→	* / %
→	+ -
→	<< >> >>>
→	< <= > >= instanceof
→	== !=
→	&
→	^
→	
→	&&
→	
←	?:
←	= *= /= %= += -= <<= >>= >>>= &= = ^=



Sentencia ::= sentencia_simple | { sentencia_simple ; * }



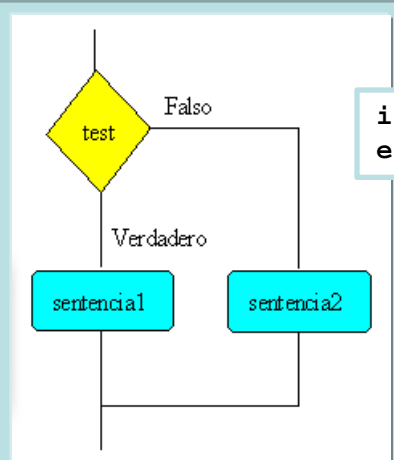
`if (expresion) sentencia;`

If (then)

```

if (numeroBoleto==numeroSorteo)
    System.out.println("has obtenido un premio");
  
```

abstract	boolean	break	byte	case	catch
char	class	const*	continue	default	do
double	else	enum	extends	final	finally
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



`if (expresion) sentencia1;
else sentencia2;`

If (then) else

```

if (numeroBoleto==numeroSorteo)
    premio=1000;
else
    premio=0;
  
```

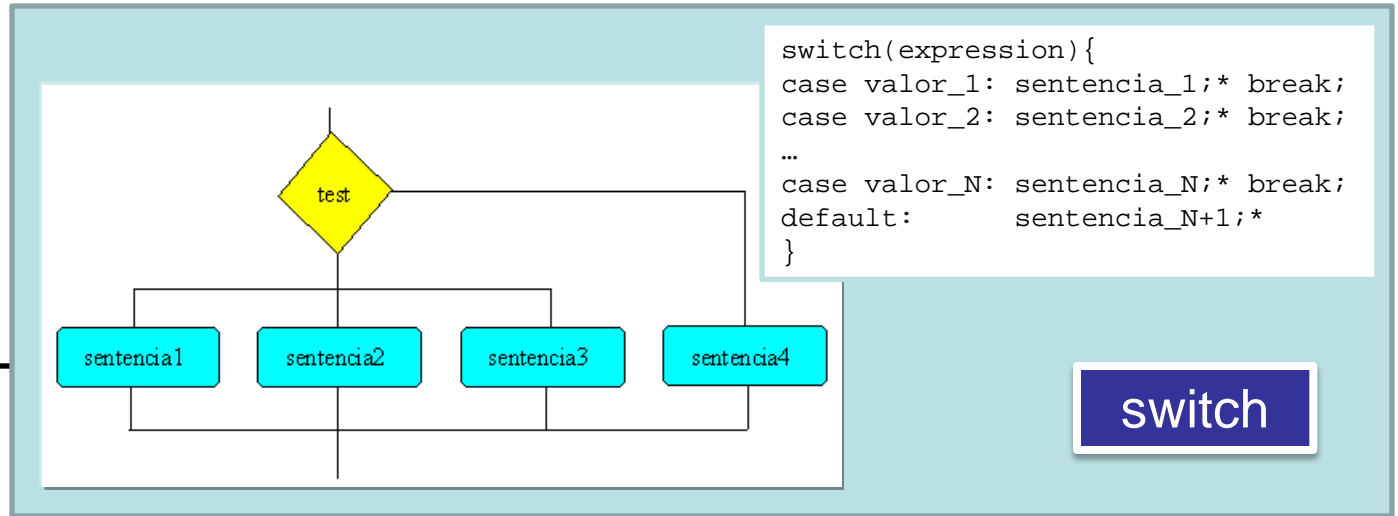
Utilizando el operador ternario:
`premio= (numeroBoleto==numeroSorteo)? 1000 : 0;`

Otro ejemplo más significativo:
`int signo=(exponente%2==0)?1:-1;`

Generalización:

```

if (expresion==valor_1) sentencia_1
else if (expresion==valor_2) sentencia_2
else ...
if (expresion==valor_N) sentencia_N
else sentencia_N+1;
    
```



```

switch (mes) {
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12: numDias = 31; break;
  case 4:
  case 6:
  case 9:
  case 11: numDias = 30; break;
  case 2: if ( ((año % 4 == 0) && !(año % 100 == 0)) || (año % 400 == 0) )
            numDias = 29;
            else numDias = 28; break;
  default: numdias=0;
}
    
```

for (inicialización; condición de mantenimiento; incremento) sentencia

```

for (int i = 0; i < 10; i++) System.out.println(i);
for (int i=20; i >= 2; i -= 2) System.out.println(i);
    
```

for

Hay otra versión del "for" ligada a colecciones

while (expresion) sentencia

```

int i=0;
while (i<10) {
    System.out.println(i);
    i++;
}
    
```

while

do sentencia **while** (expresion)

```

int i=0;
do {
    System.out.println(i);
    i++;
} while (i < 10);
    
```

do while

break, continue y etiquetas

```
for (int i = 0; i < 10; i++) {
    //...otras sentencias
    if (condicionFinal) break;
    //...otras sentencias
}
```

```
while (true) {
    //...otras sentencias
    if (condicionFinal) break;
    //...otras sentencias
}
```

```
nivelX:
for (int i=0; i<20; i++) {
    //... }
    while (j<70) {
        //... }
        if (i*j==500) break nivelX;
        //... }
    //...
}
```

```
for (int i = 0; i < 10; i++) {
    //...otras sentencias
    if (condicionFinal) continue;
    //...otras sentencias
}
```

```
while (true) {
    //...otras sentencias
    if (condicionFinal) continue;
    //...otras sentencias (en algún punto un break)
}
```

```
nivelX:
for (int i=0; i<20; i++) {
    //... }
    while (j<70) {
        //... }
        if (i*j==500) continue nivelX;
        //... }
    //...
}
```

return

Return ;
Return expresión;

Hay otra sentencia (try-catch-finalize) ligada a objetos

break con etiqueta

```
public class Prueba {
    public static void main(String[] args) {
        nivelX:
        for (int i=0; i<10; i++) {
            System.out.print("\nfor "+i+": ");
            int j=0;
            while (j<10) {
                if (i*j==32) break nivelX;
                System.out.print("(" +i+ "." +j+" ");
                j++; }
            System.out.println("for end"); } } }
```

```
C:\>java Prueba
```

```
for 0: (0.0) (0.1) (0.2) (0.3) (0.4) (0.5) (0.6) (0.7) (0.8) (0.9) for end
```

```
for 1: (1.0) (1.1) (1.2) (1.3) (1.4) (1.5) (1.6) (1.7) (1.8) (1.9) for end
```

```
for 2: (2.0) (2.1) (2.2) (2.3) (2.4) (2.5) (2.6) (2.7) (2.8) (2.9) for end
```

```
for 3: (3.0) (3.1) (3.2) (3.3) (3.4) (3.5) (3.6) (3.7) (3.8) (3.9) for end
```

```
for 4: (4.0) (4.1) (4.2) (4.3) (4.4) (4.5) (4.6) (4.7)
```

```
C:\>
```

continue con etiqueta

```
public class Prueba {
    public static void main(String[] args) {
        nivelX:
        for (int i=0; i<10; i++) {
            System.out.print("\nfor "+i+": ");
            int j=0;
            while (j<10) {
                if (i*j==32) continue nivelX;
                System.out.print("(" +i+"."+j+" ");
                j++; }
            System.out.println("for end"); } } }
```

```
C:\>java Prueba

for 0: (0.0) (0.1) (0.2) (0.3) (0.4) (0.5) (0.6) (0.7) (0.8) (0.9) for end
for 1: (1.0) (1.1) (1.2) (1.3) (1.4) (1.5) (1.6) (1.7) (1.8) (1.9) for end
for 2: (2.0) (2.1) (2.2) (2.3) (2.4) (2.5) (2.6) (2.7) (2.8) (2.9) for end
for 3: (3.0) (3.1) (3.2) (3.3) (3.4) (3.5) (3.6) (3.7) (3.8) (3.9) for end
for 4: (4.0) (4.1) (4.2) (4.3) (4.4) (4.5) (4.6) (4.7)
for 5: (5.0) (5.1) (5.2) (5.3) (5.4) (5.5) (5.6) (5.7) (5.8) (5.9) for end
for 6: (6.0) (6.1) (6.2) (6.3) (6.4) (6.5) (6.6) (6.7) (6.8) (6.9) for end
for 7: (7.0) (7.1) (7.2) (7.3) (7.4) (7.5) (7.6) (7.7) (7.8) (7.9) for end
for 8: (8.0) (8.1) (8.2) (8.3)
for 9: (9.0) (9.1) (9.2) (9.3) (9.4) (9.5) (9.6) (9.7) (9.8) (9.9) for end

C:\>
```