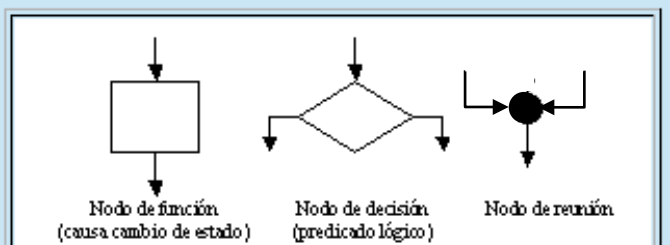
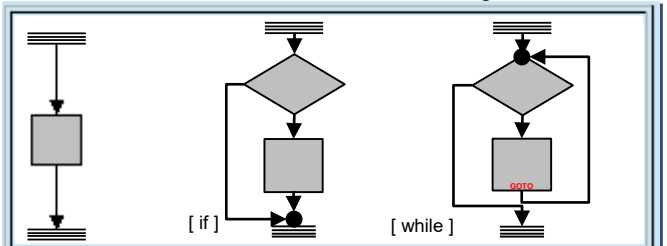
Programación estructurada.

"Paradigma 0": GOTO

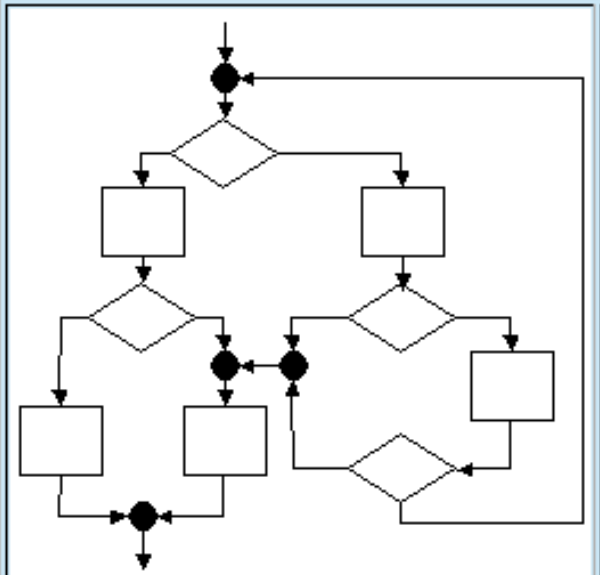


Los tres elementos básicos para construir diagramas de flujo

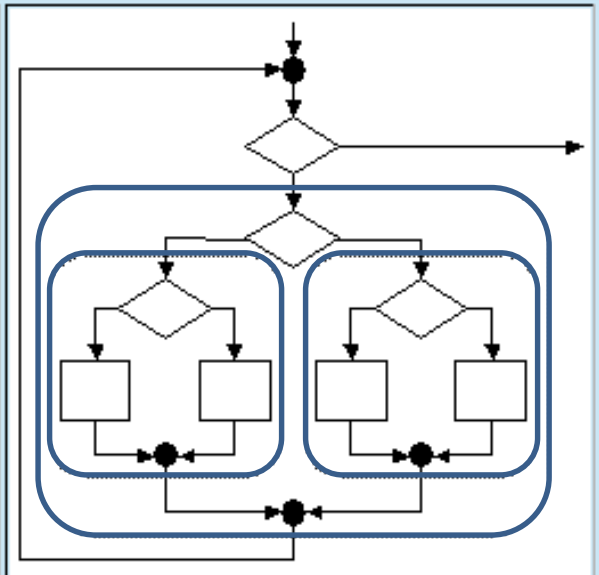


Los tres elementos suficientes para construir programas sin GOTO

"Paradigma 1": programación estructurada (el GOTO es pernicioso)

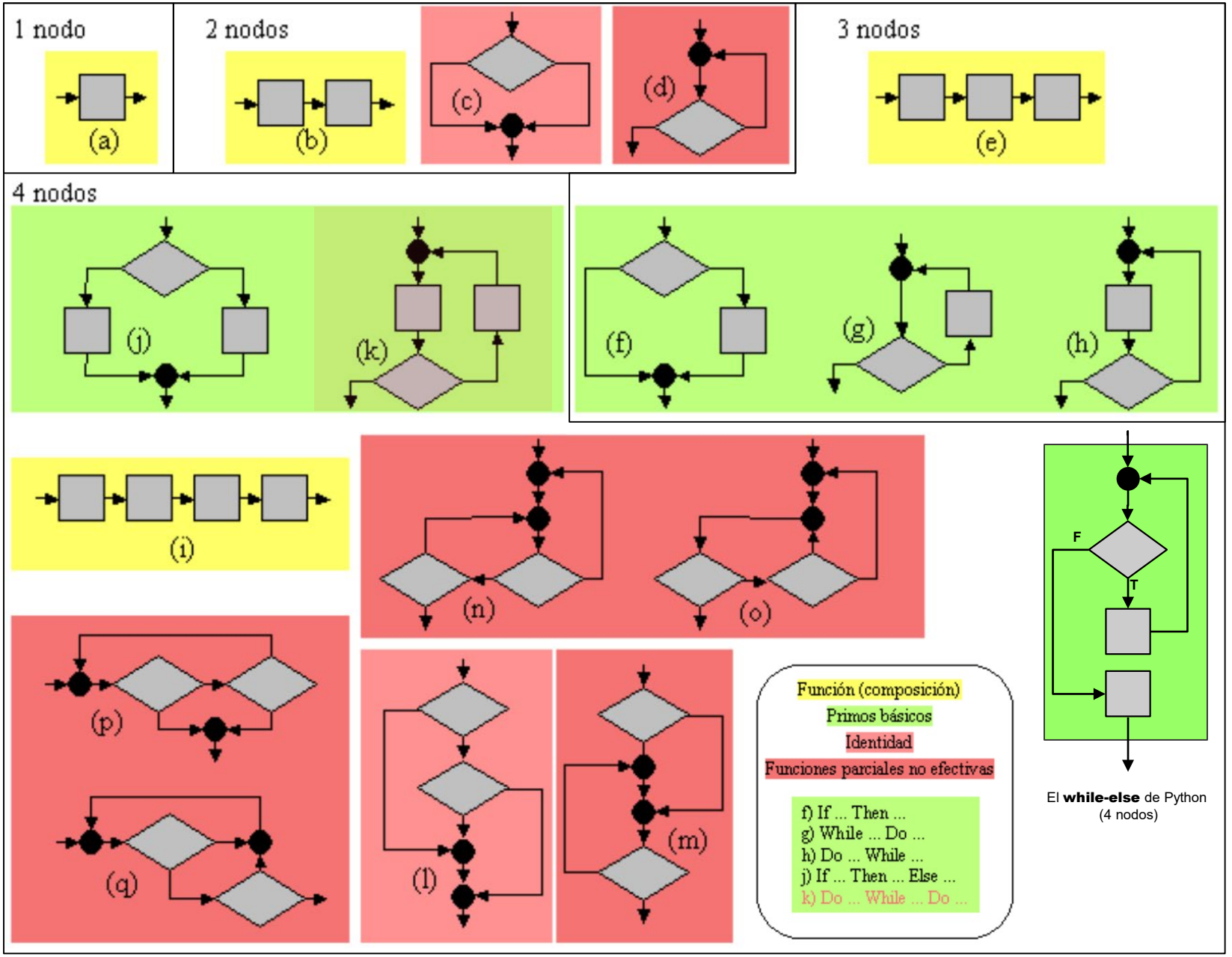


Un programa primo no puede subdividirse en bloques con una sola entrada y una sola salida.



Un programa compuesto con sus estructuras primas.

Puede explorarse exhaustivamente la combinatoria de estruturas...



El **while-else** de Python
(4 nodos)

Break:
posibilidad de abortar estructuras. Particularmente ciclos.

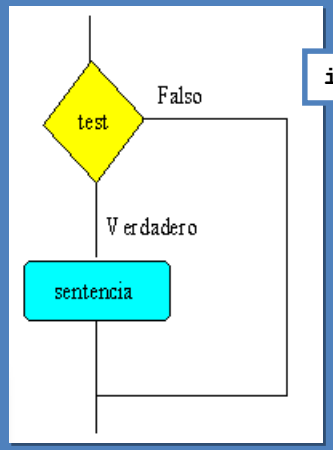
En realidad no supone una desestructuración sino la inclusión de determinadas estructuras primas más complejas (con más de 4 nodos)

Return:
posibilidad de abortar rutinas.

En cierto modo es lo mismo, aunque no equivale a aceptar una estructura prima más.

Mecanismo de “excepciones”:
Una generalización de lo anterior que permite “abortos parametrizados”.
No necesariamente ligado a la orientación a objetos, pero típicamente presente en ese paradigma.

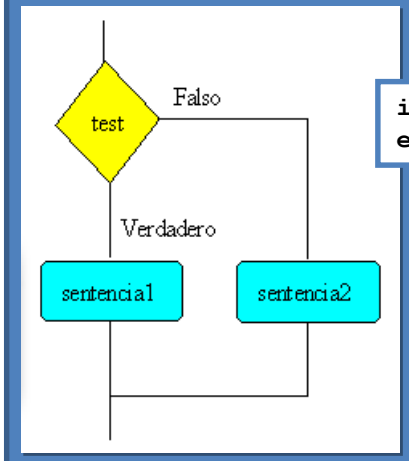
Sentencia::= sentencia_simple | { sentencia_simple;*}




```
if (expresion) sentencia;
```

If (then)

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




```
if (expresion) sentencia1;
else sentencia2;
```

If (then) else

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

Ejemplo tomado de Angel Franco: http://www.sc.ehu.es/sbweb/fisica/cursoJava/Intro.htm

Rupturas de secuencia I

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

Utilizando el operador ternario:

```
premio= (numeroBoleto==numeroSorteo)? 1000 :0;
```

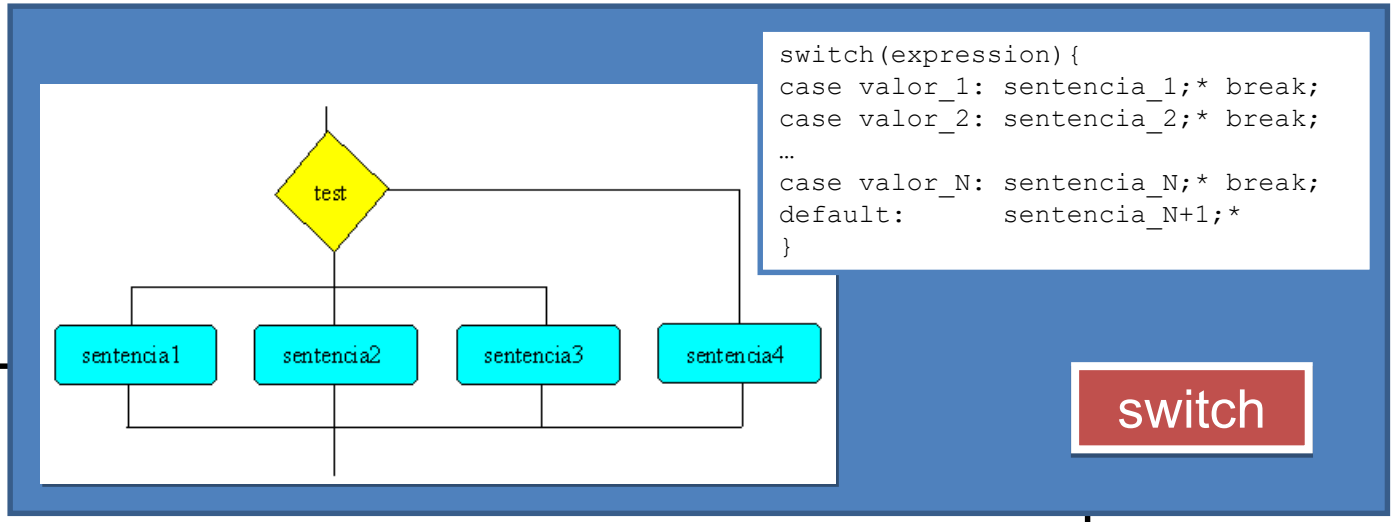
Otro ejemplo. Para calcular un término de la serie :

$$f(x) = \sum_{i=0}^{N} (-1)^i g(x)$$

```
terminoIesimo=(i%2==0?1:-1) * g(x);
```

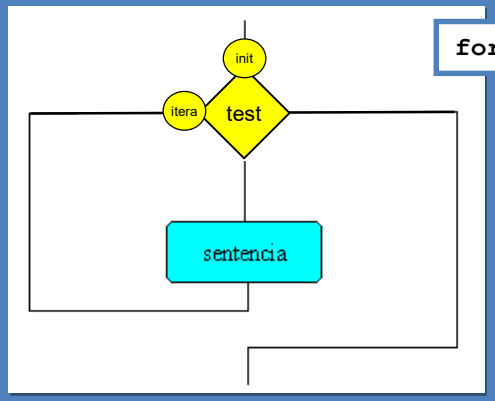
Generalización:
If (expresion==valor_1) sentencia_1
else if (expresion==valor_2) sentencia_2
else …
if (expresion==valor_N) sentencia_N
else sentencia_N+1;

Rupturas de secuencia II




```
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;
}
```

Ciclos




```
for (inicialización; condición de mantenimiento; iteración) sentencia
```

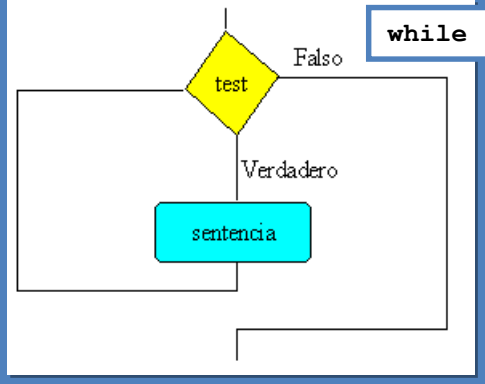
```
for (int i = 0; i < 10; i++) System.out.println(i);

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

Hay otra versión del "for" ligada a colecciones

```
int[] indices={2,3,5,7,11};
for (int i: indices ) …
```

También usable con colecciones de objetos (se verá más adelante)

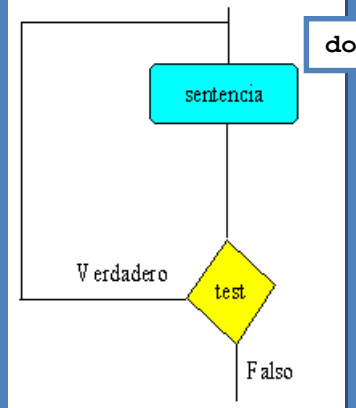
for




```
while (expresión) sentencia
```

while

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




```
do sentencia while (expresion)
```

do while

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

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;
```

(métodos)

```
atributos retorno nombre(parámetros) {
      // sentencias
}
```

```
Parámetros es una lista separada por comas de pares
          tipo/clase identificador
```

```
Ejemplo:
public static int suma(int a, int b) {
      return a+b;
}
```

Hay otras 2 sentencias:
try *y* ***try-with-resources***
ligadas a objetos…
…por lo que se verán en el siguiente tema

Y una más:
assert
no sólo ligada a objetos sino al modelo de gestión de errores…
…por lo que se verá aún más adelante

```
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"); }
    }
}
```

break con etiqueta

```
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:\>
```

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

continue con etiqueta

```
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:\>
```

El **While-else** de Python en Java

El while-else de Python no tiene nada de especial a no ser que se produzca un break dentro del ciclo, en cuyo caso no se ejecuta la sentencia afectada por el else. Veámos cómo hacer esto con Java

Versión clásica:

```
boolean abortado=false;
while( <condición> ) {
      // sentencias...
      if (<se_da_condición_para_abortar>) {
            abortado=true;
            break;
            }
      // sentencias…
      }
if (not abortado) // acción tras recorrer todos los elementos;
```

Una versión algo más interesante:

```
whileAndThen:{
while( hay_más_elementos_a_comprobar ){
            // sentencias…
            if (<se_da_condición_para_abortar>) break whileAndThen;
            // sentencias …
            }
// acción correspondiente al else de Python;
}
```

* He llamado **whileAndThen** a la etiqueta, y no **whileElse**, porque la palabra “else” de Phyton no es muy afortunada (tiene su lógica “interna”, pero no es nada clara)