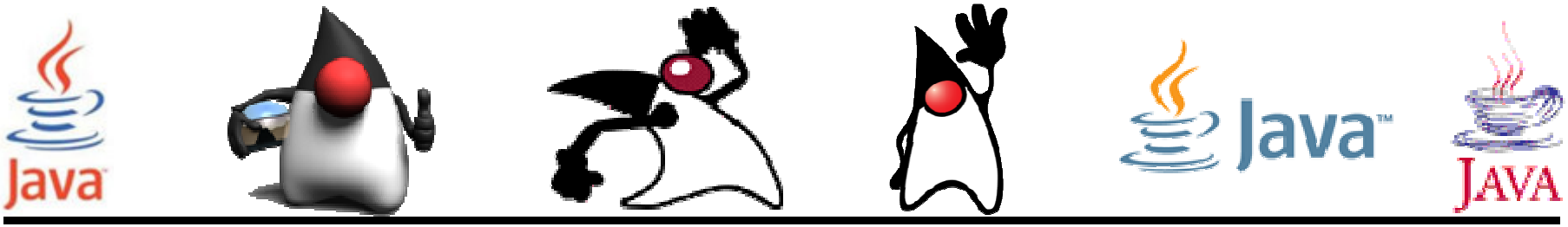


Instrucciones de Control

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Usar las instrucciones de control de decisión (if, else, switch) que permiten la selección de secciones específicas de código a ejecutarse.
- Usar las instrucciones de control de repetición (while, do-while, for) que permiten ejecutar secciones específicas de código un número de veces.
- Usar las instrucciones de ramificación (break, continue, return) que permiten la redirección del flujo del programa.

Índice

- Estructuras de control
- Instrucciones if, if – else, if – else – else – if, switch
- Instrucciones while, do – while, for
- Instrucciones break, continue, return

Estructuras de Control

- Estructuras de Control: permiten cambiar el orden de ejecución de las instrucciones en los programas.
- Dos tipos de estructuras de control
  - Estructuras de control de decisión
    - Permiten seleccionar las secciones específicas de código a ejecutar.
  - Estructuras de control de repetición
    - Permiten ejecutar un múmero de veces una sección específica de código.

Estructuras de Control de Decisión

- Son instrucciones que permiten seleccionar y ejecutar bloques específicos de código y saltar otras secciones.
- Tipos:
  - Instrucción if
  - Instrucción if - else
  - Instrucción if - else - if

Instrucción if

- Especifica la sentencia (o bloque de código) que será ejecutada si y solo si una expresion booleana es verdadera.
- Sintaxis de la instrucción if

```
if (expresion_booleana) {
    instruccion(es);
}
```

donde expresion_booleana es una expresión booleana o variable booleana.

Instrucción if

Diagrama de flujo

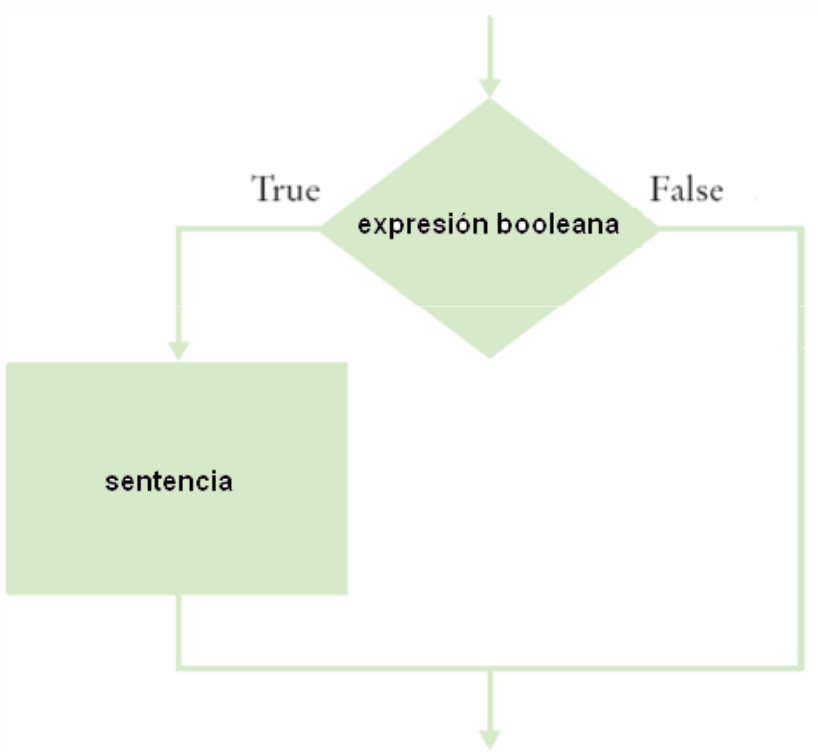
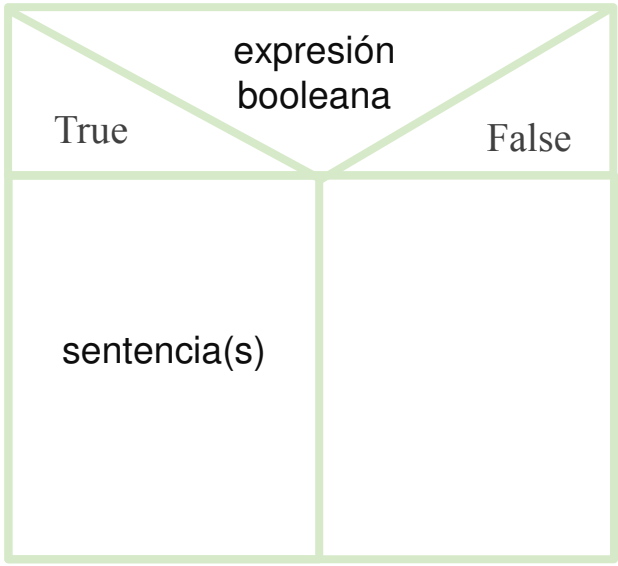


Diagrama de cajas



Ejemplo de if

```
import java.util.Scanner;

public class MensajeNota {
  public static void main( String[] args ){
    Scanner in = new Scanner( System.in );
    System.out.print("Nota: ");
    int nota = in.nextInt();
    if (nota > 7 ){
      System.out.println("Felicitaciones");
      System.out.println("Has aprobado");
    }
  }
}
```

Recomendaciones de codificación

- La parte de la instrucción expresion_booleana debe evaluarse a un valor booleano. Esto significa que la ejecución de la condición debe resultar en un valor true o false.
- Indentar las instrucciones dentro del bloque if. Ej.:

```
if (expresion_booleana) {
    instruccion1;
    instruccion2;
    instruccion3;
}
```

Instrucción if - else

- Se usa cuando se desea ejecutar una determinada sentencia si una condición es true y una sentencia diferente si la condición es false.
- Sintaxis de la instrucción if - else

```
if (expresion_booleana) {
    instruccion1;
    . . .
}
else {
    instruccion2;
    . . .
}
```

Instrucción if - else

Diagrama de flujo

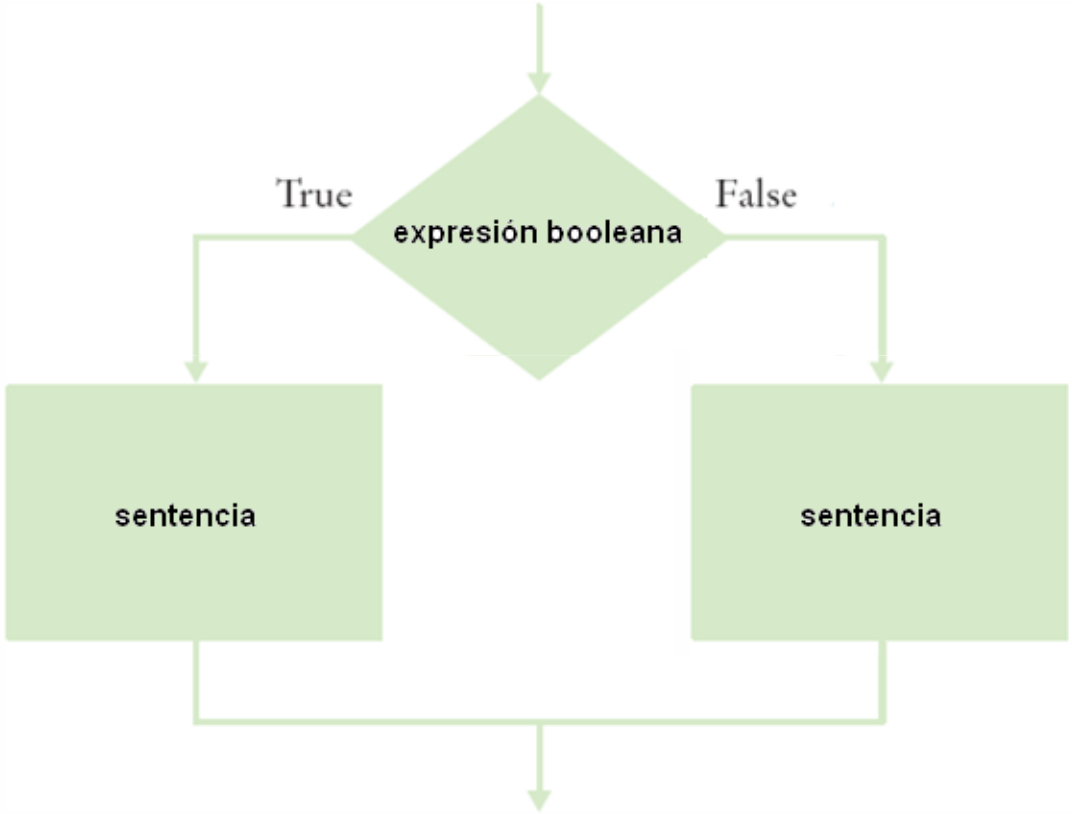
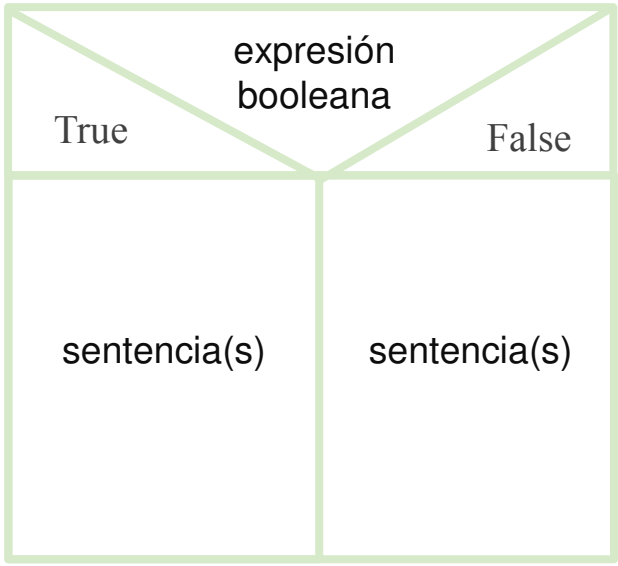


Diagrama de cajas



Ejemplo de if - else

```
import java.util.Scanner;

public class MensajeNota1 {
  public static void main( String[] args ){
    Scanner in = new Scanner( System.in );
    System.out.print("Nota (0 – 10): ");
    double nota = in.nextDouble();
    if (nota > 7 )
      System.out.println("Has aprobado");
    else
      System.out.println("Suspendido");
  }
}
```

Ejemplo de if - else

```
import java.util.Scanner;

public class MensajeNota2 {
  public static void main( String[] args ){
    Scanner in = new Scanner( System.in );
    System.out.print("Nota (0 – 10): ");
    double nota = in.nextDouble();
    if (nota > 7 ) {
      System.out.println("Felicitaciones");
      System.out.println("Has aprobado");
    }
    else {
      System.out.println("Suspendido");
    }
  }
}
```

Recomendaciones de codificación

- Para evitar confusiones, siempre colocar la instrucción o instrucciones de un bloque if – else dentro de llaves.
- Se puede tener bloques if – else anidados. Ej.:

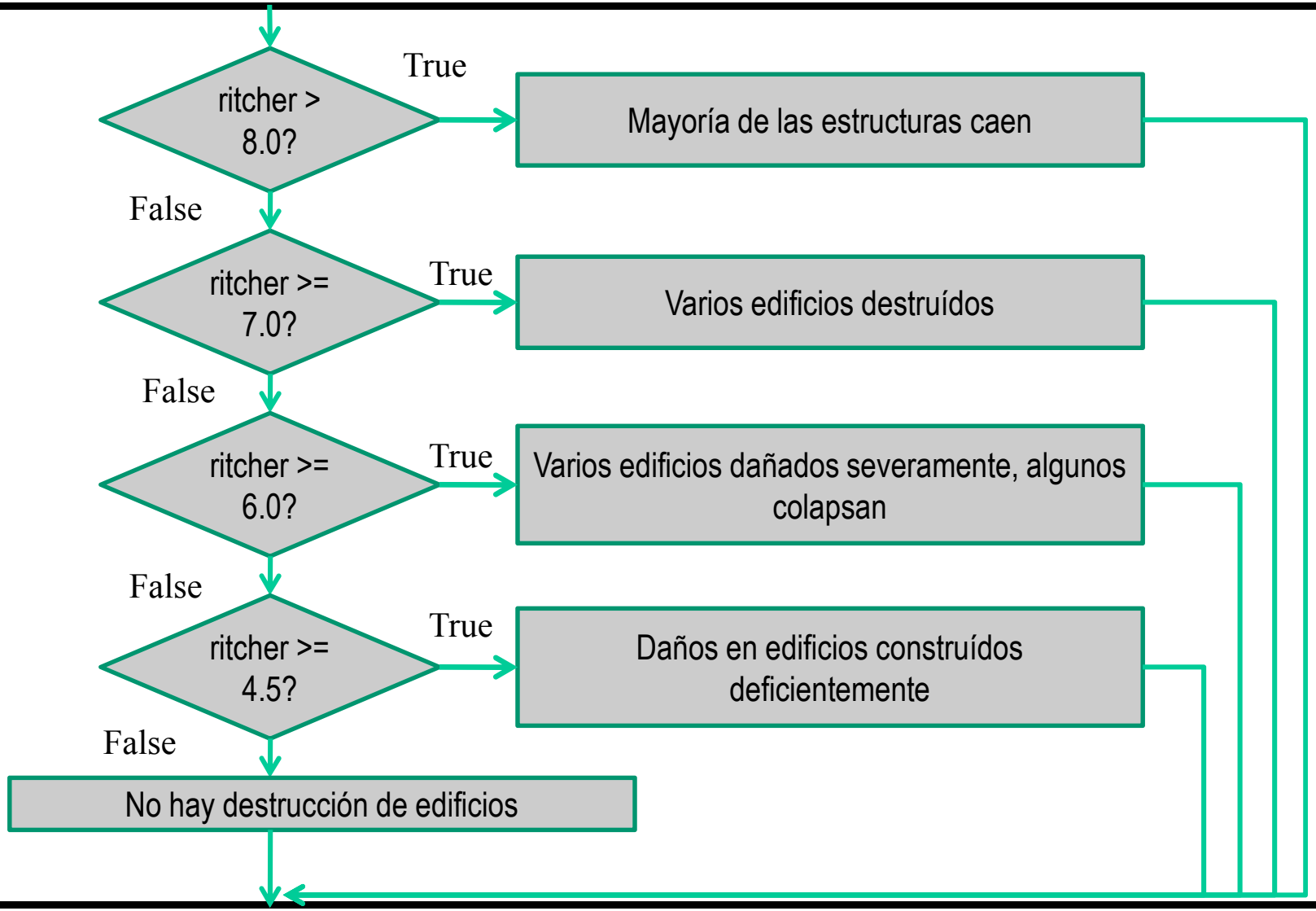
```
if (expresion_booleana) {
    if (expresion_booleana) {
        instrucciones;
    }
else {
    instrucciones;
}
```

Instrucción if – else – else if

- La sentencia en la parte else de un bloque if – else puede ser otra estructura if – else.
- La cascada de estructuras permite construir selecciones más complejas.
- Sintaxis de la instrucción if – else – else if

```
if (expresion_booleana1)
   instruccion1;
else if (expresion_booleana2)
   instruccion2;
else
   instruccion3;
```

Instrucción if – else – else if



Ejemplo de if – else – else if

```
import java.util.Scanner;

public class MensajeNota3 {
 public static void main( String[] args ){
   Scanner in = new Scanner( System.in );
   System.out.print("Nota (0-10): ");
   double nota = in.nextDouble();
   if (nota == 10 ) {
     System.out.println("Matricula de Honor");}
   else if ((nota >= 9) && (nota < 10)) {
     System.out.println("Sobresaliente");     }
   else if (nota >= 7 ) {
     System.out.println("Notable");          }
   else if (nota >= 5 ) {
     System.out.println("Aprobado");         }
   else {
     System.out.println("Suspenso");         }
  }
}
```

Errores comunes

- Añadir un punto y coma al final de la condición del if.

```
if (planta > 13) ;
  planta--;
```

- La condición del if no produce un valor boolean.

```
int number = 0;
if (number) {
   //instrucciones
}
```

- Escribir `elseif` en lugar de `else if`
- Usar = en lugar de == en las comparaciones.

Errores comunes

- Cuando se comparan números reales los errores de redondeo pueden conducir a resultados inesperados.

```
double r = Math.sqrt(2.0);
if (r * r == 2.0)
{
  System.out.println(“Math.sqrt(2.0) al cuadrado es 2.0");
}
else
{
  System.out.println("Math.sqrt(2.0) al cuadrado no es 2.0
   sino " + r * r);
}
```

```
Output:
Math.sqrt(2.0) al cuadrado no es 2.0 sino
   2.0000000000000044
```

El uso de EPSILON

- Usar un valor muy pequeño para comparar la diferencia de valores reales si son muy próximos.
  - La magnitud de la diferencia debe ser menor a un umbral.
  - Matemáticamente se puede escribir que x e y son próximos si: $|x - y| < \varepsilon$

```
final double EPSILON = 1E-14;
double r = Math.sqrt(2.0);
if (Math.abs(r * r - 2.0) < EPSILON)
{
  System.out.println("Math.sqrt(2.0) al cuadrado es aprox.
   2.0");
}
```

Comparación de cadenas de caracteres

- Para comparar el contenido de dos Strings se usa el método equals:

```
if ((string1.equals(string2)) ...
```

- Si se quiere comparar la posición de memoria de dos Strings se usa el operador ==.

```
if (string1 == string2) ...
```

- En caso de que una cadena coincida exactamente con una constante se puede usar ==

```
String nickname = "Rob";
. . .
if (nickname == "Rob") // Resultado true
```

Orden lexicográfico de Strings

- Para comparar Strings en el orden del diccionario se usa el método compareTo
  - string1 antes de string2

```
if (string1.compareTo(string2) < 0)
```

  - string1 después de string2

```
if (string1.compareTo(string2) > 0)
```

  - string1 igual a string2

```
if (string1.compareTo(string2) == 0)
```

  Las letras MAYÚSCULAS están antes de las minúsculas.

  Los espacios en blanco están antes de los c.imprimibles.

  Los dígitos están antes de las letras

Ramificación alternativa: switch

- La instrucción switch permite seleccionar un case (rama) basado en un *valor entero*.
- Sintaxis de la instrucción switch

```
switch (expresion_switch) {
    case case_selector1:
        instruccion1; break;
    case case_selector2:
        instruccion2; break;
    ...
    default:
        instruccion;
}
```

Ramificación alternativa: switch

- En la sintaxis de switch

  `expresion_switch` es una expresión entera o carácter

  `case_selector1, case_selector2, ...` son constantes enteras o carácter únicas

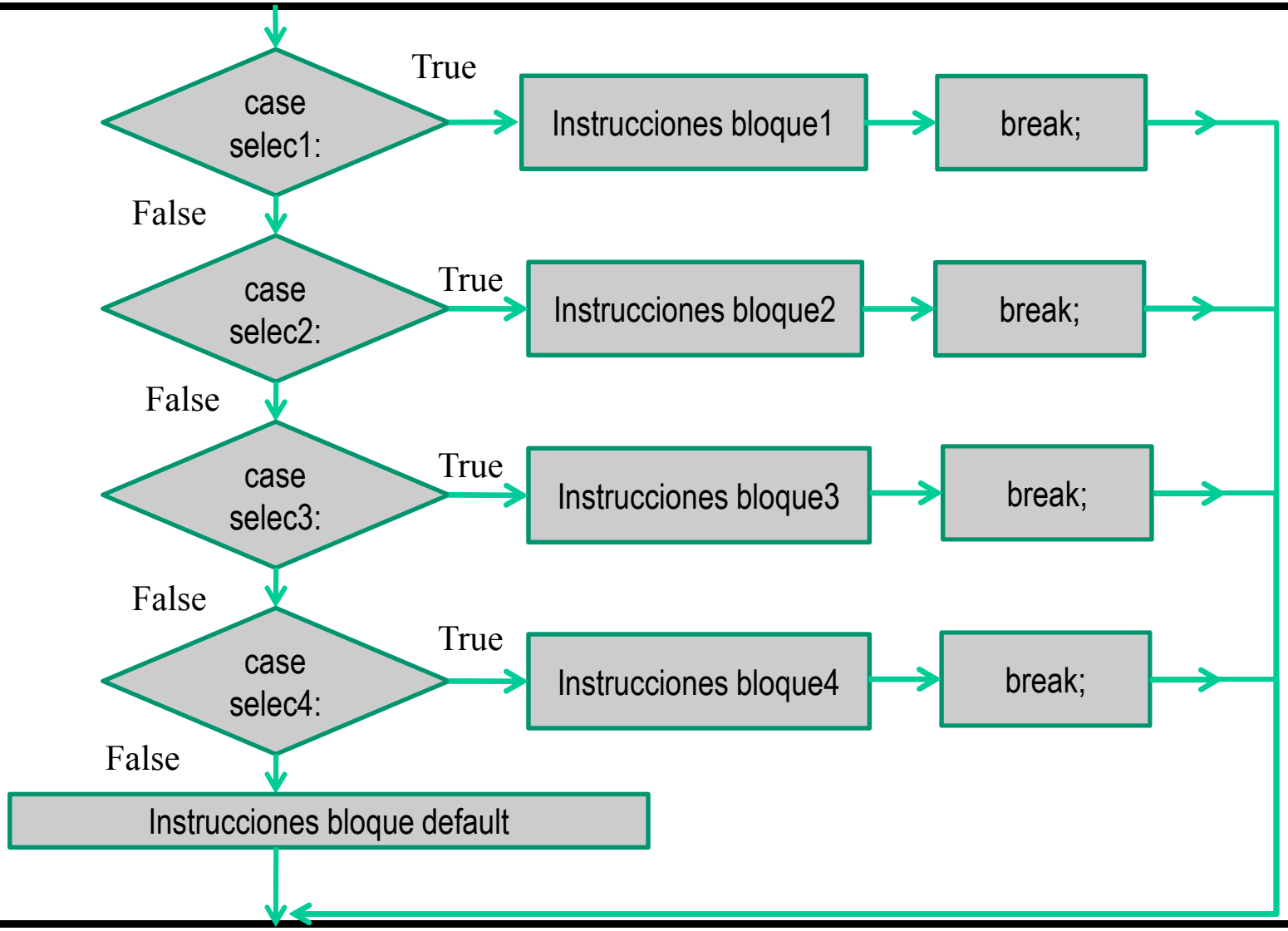
  `break` finaliza cada case

  `default` captura cualquier otro valor

Funcionamiento de switch

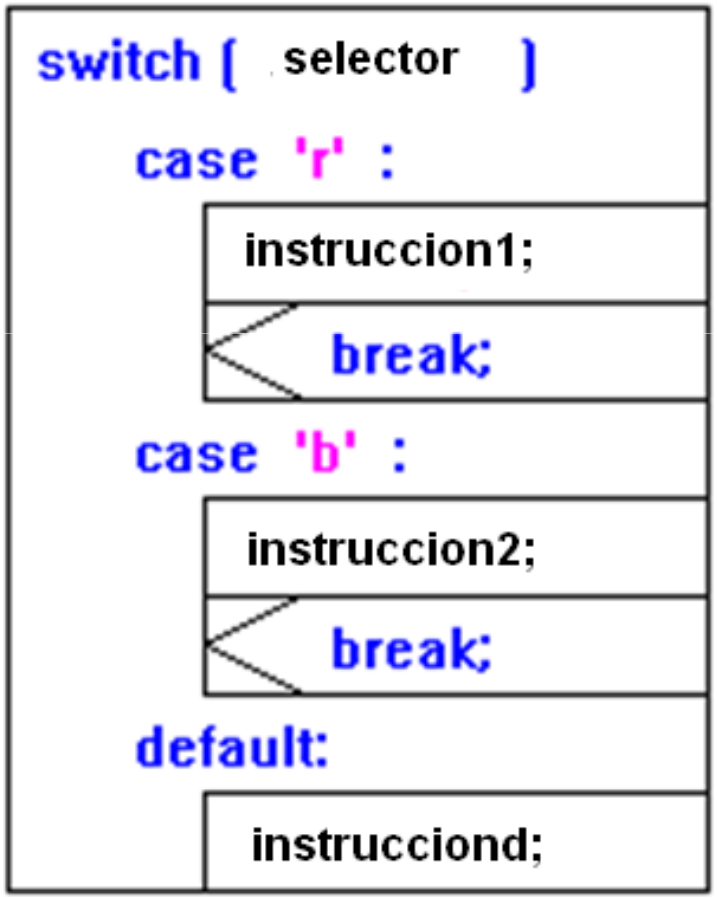
- Cuando se encuentra una instrucción switch:
  - Primero se evalúa la expresion_switch y salta al case cuyo selector coincide con el valor de la expresión.
  - Se ejecutan las instrucciones que siguen al case seleccionado hasta que se encuentra un break que produce un salto a la siguiente instrucción que sigue a switch.
  - Si ninguno de estos casos se cumple, el bloque default se ejecuta si existe. Notar que la parte default es opcional.

Instrucción switch

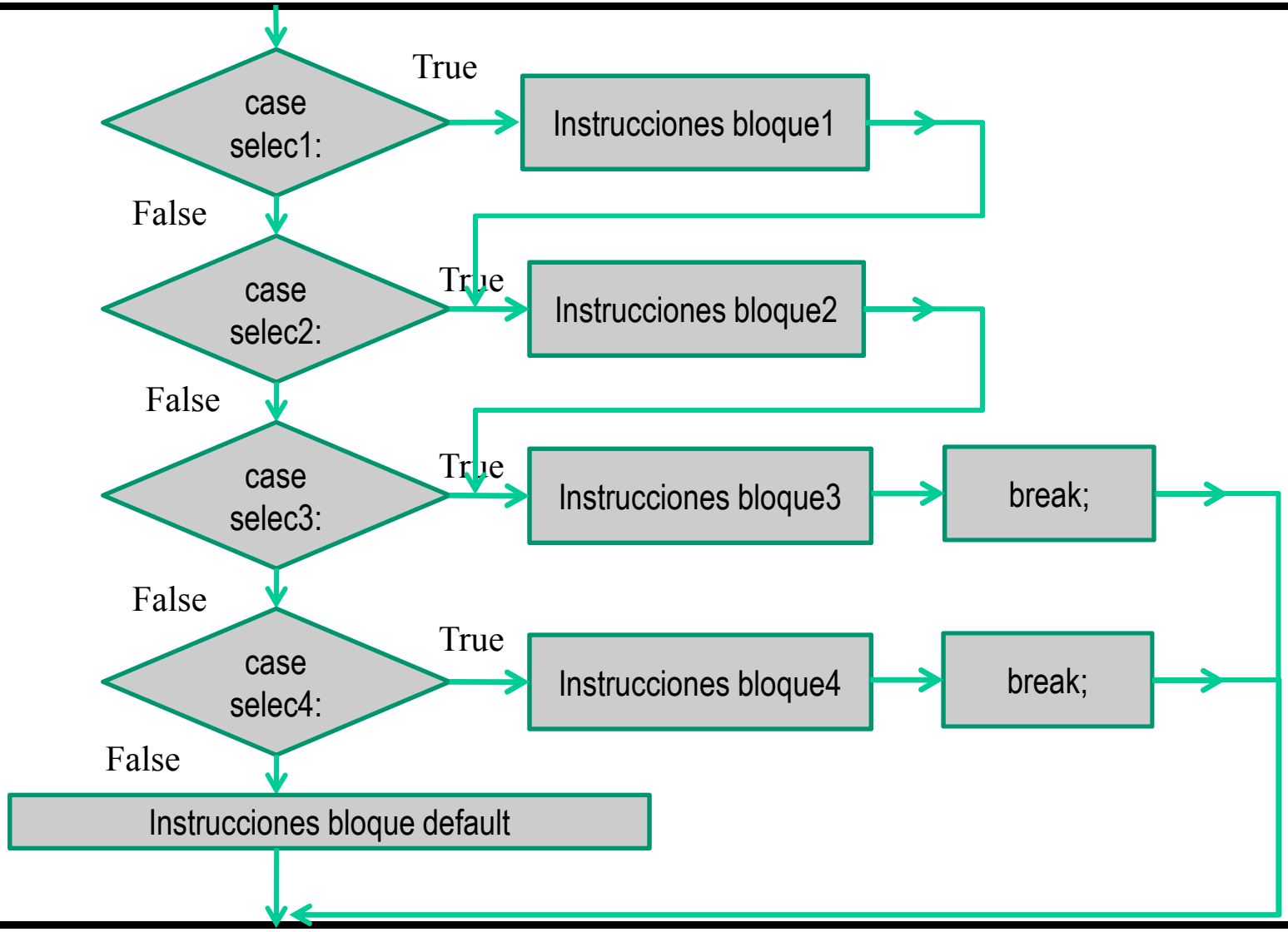


Instrucción switch

Diagrama de cajas



Instrucción switch



Ejemplo de switch

```
import java.util.Scanner;

public class MensajeNota4 {
 public static void main( String[] args ){
   Scanner in = new Scanner( System.in );
   System.out.print("Nota (0-10): ");int nota = in.nextInt();
   switch (nota) {
     case 10: System.out.println("Matricula de Honor"); break;
     case 9:  System.out.println("Sobresaliente"); break;
     case 8:
     case 7:  System.out.println("Notable"); break;
     case 6:
     case 5:  System.out.println("Aprobado"); break;
     case 4: case 3: case 2: case 1: case 0:
              System.out.println("Suspenso"); break;
     default: System.out.println("Nota mal ingresada");
   }
 }
}
```

Ejemplo de switch (1)

```
import java.util.Scanner;

public class SelSwitch {
  public static void main(String[] args) {
    Scanner in = new Scanner( System.in );
    System.out.printf("Pulsa una tecla: ");
    String s = in.nextLine();
    char c = s.charAt(0);

    switch (c) {
      case '0':  case '1':  case '2':  case '3':  case '4':
      case '5':  case '6':  case '7':  case '8':  case '9':
        System.out.printf("Tecla numerica "); break;
      case ' ':  case '\n':  case '\t':
        System.out.printf("blanco\n"); break;
      default: System.out.printf("Otra tecla\n"); break;
    }
  }
}
```

Instrucciones de repetición - Ciclos

- Permiten ejecutar de forma repetida un bloque específico de instrucciones.
- Tipos:
  - ciclo while
  - ciclo do – while
  - ciclo for
- Cada ciclo requiere de los siguientes pasos:
  - Inicialización (preparar el inicio del ciclado)
  - Condición (comprobar si se debe ejecutar el ciclo)
  - Actualización (cambiar algo cada vez al final del ciclo)

Ciclo while

- Permite repetir una instrucción o bloque de instrucciones mientras se cumpla una condición.
- Sintaxis:

```
while (expresion_boolean) {
  instruccion1;
  instruccion2;
  ...
}
```

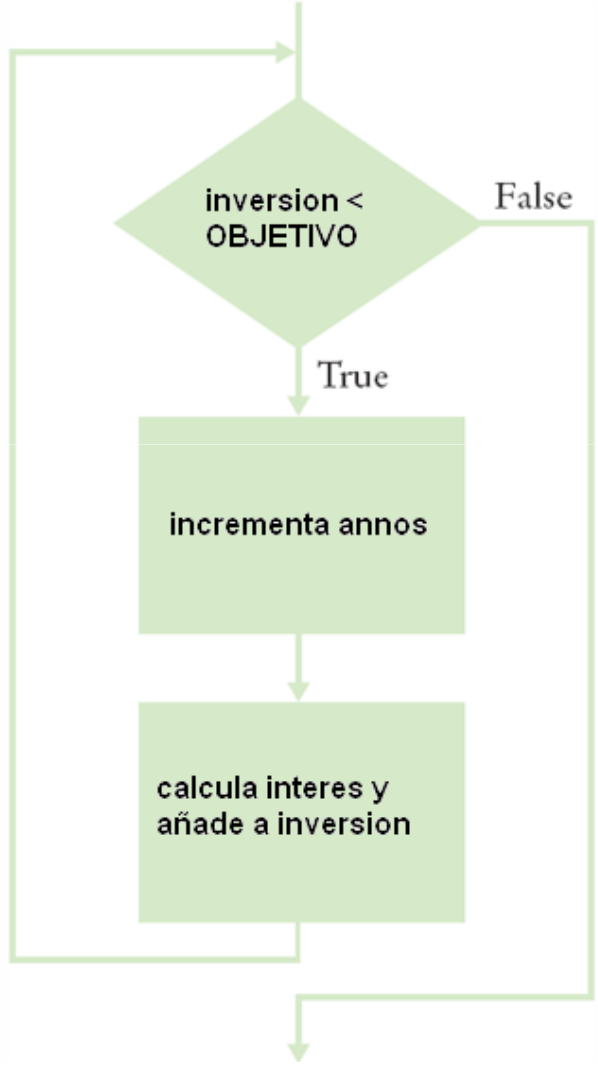
las instrucciones dentro del ciclo while se ejecutan mientras expresion_boolean sea true.

Ejemplo de ciclo while

```
public class DoblarInversion {
  public static void main( String[] args ){
    final double TASA = 5;
    final double INVERSION_INICIAL = 10000;
    final double OBJETIVO = 2*INVERSION_INICIAL;

    double inversion = INVERSION_INICIAL;
    int annos = 0;
    while (inversion < OBJETIVO){
      annos++;
      double interes = inversion*TASA/100;
      inversion += interes;
    }
    System.out.println("A\0x00a4os: " + annos);
  }
}
```

Ciclo while




```
while (inversion < OBJETIVO)
{
  annos++;
  double interes = inversion*TASA/100;
  inversion = inversion + interes;
}
```

Ciclo while

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

// ciclo infinito
while (true){
   System.out.println("hola");
}

// no hay ciclo
while (false){
   System.out.println("hola");
}
```

Ejemplo de ciclo while (1)

```
// Imprime tabla de conversion Fahrenheit-Celsius 0-100
public class ConvFCw {
  public static void main(String[] args) {
    final double RANGO_INICIAL = 0;  // limite inf. tabla
    final double RANGO_FINAL = 100;  // limite sup. tabla
    final double PASO = 10 ;         // tamanno paso
    int fahrenheit;
    double celsius;

    fahrenheit = (int) RANGO_INICIAL;
    System.out.printf("Fahrenheit \t Celsius \n");
    while (fahrenheit <= RANGO_FINAL )
    {
      celsius = 5.*(fahrenheit - 32)/9;
      System.out.printf("%7d \t %.3f \n", fahrenheit, celsius);
      fahrenheit += PASO;
    }
  }
}
```

Ciclo while – Sumas y promedios

```
double total = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   total = total + input;
}
```

- Suma de Valores
  - Inicializar total a 0
  - Usar ciclo `while` con centinela

- Promedio de Valores
  - Usar total de Valores
  - Inicializar count a 0
    - Incrementar total por input
  - Comprobar si count > 0
    - antes de dividir!

```
double total = 0;
int count = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   total = total + input;
   count++;
}
double average = 0;
if (count > 0)
   { average = total / count; }
```

Ciclo while – Máximo y Mínimo

```
double largest = in.nextDouble();
while (in.hasNextDouble())
{
   double input = in.nextD
   if (input > largest)
   {
      largest = input;
   }
}
```

```
double smallest = in.nextDouble();
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   if (input > smallest)
   {
      smallest = input;
   }
}
```

- Leer el primer valor
  - Este el largest (o smallest) que se tiene hasta este momento
- Iterar mientras se tenga un número valido (no centinela)
  - Leer otro valor de entrada
  - Comparar nueva entrada con largest (o smallest)
  - Actualizar largest (o smallest) si es necesario

Patrones de programación

- Los patrones de programación del sumatorio y productorio (usuales en algoritmos numéricos) son:

$$\sum_{i=0}^{N} termino_i$$

```
suma=0; i = 0;
while (i <= N)
{
     suma = suma + termino_i;
     i = i + 1;
}
```

$$\prod_{i=0}^{N} termino_i$$

```
prod=1; i = 0;
while (i <= N)
{
     prod = prod * termino_i;
     i = i + 1;
}
```

Ejemplo de ciclo while (2)

- Calcular el valor de *seno(x)* en los puntos resultantes de dividir el intervalo [0,π] en N subintervalos, según su formulación en serie :

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \equiv sen(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \quad x \in \Re$$

El valor del seno tendrá un error menor que un valor dado ε especificado por el usuario, siendo el error cometido menor que el valor absoluto del último término de la serie que se toma.

Ejemplo de ciclo while (2) - Observaciones

- Serie para el cálculo de **seno(x)**:

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \qquad error = |t_i| < \varepsilon$$

- Se observa que los términos de la serie son recurrentes:

$$t_i = -t_{i-1} \cdot \frac{x \cdot x}{2 \cdot i \cdot (2 \cdot i + 1)} \qquad t_0 = x$$

- Datos: N_interv número de subintervalos en [0,π]
  eps error en el cálculo del seno

Ejemplo de ciclo while (2) - Pseudocódigo

```
leer N_interv, eps
dx = pi / N_interv
x = 0
while (x <= pi)
        t = x
        i = 1
        seno = x
        while | t | > eps
             t = − t * (x*x)/(2*i*(2*i + 1)
             seno = seno + t
             i = i + 1
        end_while
        print x, seno
        x = x + dx
 end_while
```

Ejemplo de ciclo while (1)

```
/** ************************************************
*  Prog. que imprime el seno para valores en subintervalos   *
* entre 0 y PI precison especificadas por el usuario         *
* calculada con una serie de Taylor y la funcion de Math.sin *
* @author: Pedro Corcuera                                    *
\************************************************/
import java.util.Scanner;

public class CalculoSeno {
  public static void main( String[] args ){
    double seno, x, term, dx, eps;
    int i,ninterv;

    Scanner in = new Scanner( System.in );
    System.out.printf("Ingresa numero de intervalos ");
    ninterv = in.nextInt();
    System.out.printf("Ingresa precision ");
    eps = in.nextDouble();
    dx = Math.PI/ninterv;
```

Ejemplo de ciclo while (1)

```
    x = 0;
    while (x <= Math.PI)
    {
      term=x;
      seno=x;
      i=1;
      while (Math.abs(term)> eps)
      {
        term = -term*x*x/(2*i*(2*i+1));
        seno += term;
        i += 1;
      }
      System.out.printf("seno ( %f ) = %f \t sin = %f\n",
                        x, seno, Math.sin(x));
      x += dx;
    }
  }
}
```

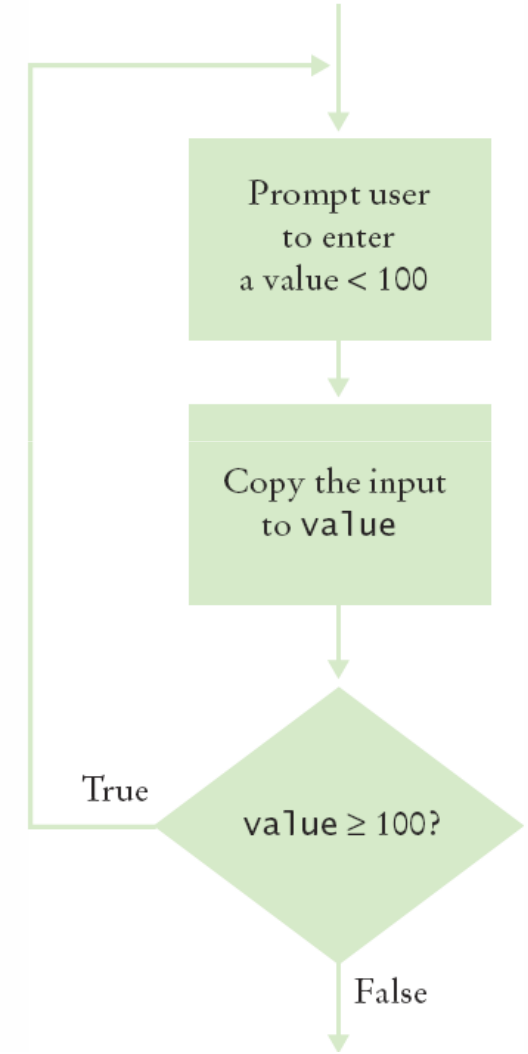
Ciclo do - while

- Similar a while: las instrucciones dentro del ciclo do-while se ejecutan si se cumple una condición que se comprueba al final.
- Sintaxis:

```
do {
  instruccion1;
  instruccion2;
  ...
} while (expresion_boolean);
```

las instrucciones dentro del ciclo se ejecutan por lo menos una vez.

Ciclo do - while




```
int value;
do
{
   System.out.println("Escribe un entero <
    100: ");
   value = in.nextInt();
}
while (value >= 100);  // test
```

Ejemplo de do - while

```
import java.util.Scanner;

public class ValidaInput {
  public static void main( String[] args ){
    int valor;

    Scanner in = new Scanner( System.in );
    do {
      System.out.println("Escribe un entero < 100");
      valor = in.nextInt();
    } while (valor >= 100);
    System.out.println("Valor: " + valor);
  }
}
```

Ejemplo de do – while (1)

```
import java.util.Scanner; //Calcula promedio lista enteros

public class Promedio {
  public static void main( String[] args ){
    int n, cont = 1; float x, promedio, suma = 0;

    Scanner in = new Scanner( System.in );
    System.out.printf("Promedio de numeros enteros\n");
    System.out.printf("Cuantos numeros? ");
    n = in.nextInt();
    do {
      System.out.printf("Ingresa numero %2d = ",cont);
      x = in.nextFloat(); suma += x; ++ cont;
    } while ( cont <= n);
    promedio = suma / n;
    System.out.printf("El promedio es: %.3f\n", promedio);
  }
}
```

Ciclo for

- Permite la ejecución de las instrucciones dentro del ciclo for un determinado número de veces.
- Sintaxis:

```
for (exp_ini; condicion_ciclo ; exp_paso)
{
 instruccion1;
 instruccion2;
 ...
}
```

exp_ini inicializa la variable del ciclo

condicion_ciclo compara la variable del ciclo con un valor límite

exp_paso actualiza la variable del ciclo

Ciclo for

- La instrucción for es una forma compacta de while.

version ciclo `while`

```
int i = 5;          // inicializacion
while (i <= 10)     // comprobacion
{
   sum = sum + i;
   i++;             // actualizacion
}
```

version ciclo `for`

```
for (int i = 5; i <= 10; i++)
{
   sum = sum + i;
}
```

Ejemplo de for

```
import java.util.Scanner;

public class TablaInversion {
  public static void main( String[] args ){
    final double TASA = 5;
    final double INVERSION_INICIAL = 10000;

    double inversion = INVERSION_INICIAL;
    System.out.print("Ingresa numero de a\u00a4os: ");
    Scanner in = new Scanner(System.in);
    int nannos = in.nextInt();
    for (int anno=1; anno <= nannos; anno++) {
      double interes = inversion*TASA/100;
      inversion += interes;
      System.out.printf("%4d %10.2f\n",anno, inversion);
    }
  }
}
```

Ejemplo de for (1)

```
public class ConvFCfor {
  public static void main(String[] args) {
    final double RANGO_INICIAL = 0;  // limite inf. tabla
    final double RANGO_FINAL = 100;  // limite sup. tabla
    final double PASO = 10 ;         // tamanno paso
    int fahrenheit; double celsius;

    System.out.printf("Fahrenheit \t Celsius \n");
    for (fahrenheit = (int) RANGO_INICIAL;
         fahrenheit <= RANGO_FINAL ;
         fahrenheit += PASO) {
      celsius = 5.*(fahrenheit - 32)/9;
      System.out.printf("%7d \t %.3f \n", fahrenheit,
                        celsius);
    }
  }
}
```

Ciclo for – contando coincidencias

- Contando coincidencias
  - Inicializar contador a 0
  - Usar ciclo `for`
  - Añadir contador en cada coincidencia

```
int upperCaseLetters = 0;
Scanner in = new Scanner( System.in );
String str = in.next();
for (int i = 0; i < str.length(); i++)
{
  char ch = str.charAt(i);
  if (Character.isUpperCase(ch))
  {
    upperCaseLetters++;
  }
}
```

Ejemplo de for (2) – Ciclos anidados

```
public class TablaPotencias {
  public static void main( String[] args ) {
    final int NMAX = 4;
    final double XMAX = 10;

// Impresión cabecero

// Impresion valores tabla
    for (double x = 1; x <= XMAX; x++)
    {
      for (int n = 1; n <= NMAX; n++)
      {
        System.out.printf("%10.0f", Math.pow(x,n));
      }
      System.out.println();
    }
  }
}
```

1	2	3	4
x	x	x	x
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561
10	100	1000	10000

Ejemplo de for (3) – Simulación

```
import java.util.Scanner;

public class LanzaDados {
  public static void main( String[] args ) {
    int NLANZA;

    Scanner in = new Scanner( System.in );
    System.out.printf("Numero de lanzamientos : ");
    NLANZA = in.nextInt();
    for (int n = 1; n <= NLANZA; n++)
    {
      int d1 = (int)(Math.random()*6) + 1; //random 1-6
      int d2 = (int)(Math.random()*6) + 1;
      System.out.println(d1 + "  " + d2);
    }
    System.out.println();
  }
}
```

Instrucciones de salto o ramificación

- Permiten la redirección del flujo de ejecución de un programa.
- Java ofrece tres instrucciones de ramificación:
  - break
  - continue
  - return

Etiquetas de sentencias

- Cada sentencia de un programa puede tener una etiqueta opcional, que es un identificador simple.
- Sintaxis:

```
[EtiquetaSentencia:] Sentencia
```

- Ejemplo:

```
Ciclo1: while (i-- > 0) {
   Ciclo2: while (j++ < 100) {
      // ...
   }
}
```

Instrucción break

- Hay dos versiones:
  - break sin etiquetar.
  - break etiquetado.
- El break sin etiquetar:
  - si está dentro de un case de switch, termina la instrucción switch y transfiere el control del flujo a la siguiente instrucción que sigue al switch.
  - también se puede usar para terminar un ciclo for, while o do – while.

Instrucción break

```
public static void main( String[] args )
{
   int i;
   for(i = 0; i < 100; i++)
   {
      if(i % 17 == 0) /* Es multiplo de 17? */
            break;      /*Sale del bucle*/
      printf("%d",i);
   }
}
```

Instrucción break etiquetado

- El break etiquetado:
  - el control del flujo se transfiere a la instrucción que sigue a la sentencia etiquetada.
  - Sintaxis general:

```
break [EtiquetaSentencia:];
```

Instrucción continue

- De forma similar a break hay dos versiones: sin etiquetar y etiquetado.
- continue sin etiquetar:
  - salta al final del bloque del ciclo más interior y evalúa la expresión booleana que controla el ciclo, saltando básicamente el resto de la iteración del ciclo.
- continue etiquetado:
  - salta el ciclo en curso hasta la sentencia marcada por la etiqueta.

Instrucción continue

```
public static void main( String[] args )
{
   int i;
   for(i = 0; i < 100 ; i++)
   {
       if(i % 2 == 0)    /* par? */
             continue;/*Comienzo la iteración*/
       printf("%d ",i);
   }
}
```

Ejemplo de break y continue

```
public class EjBreakContinue {
  public static void main( String[] args ) {
    System.out.printf("Ejemplo break\n"); int i = 0;
    while (i <= 20) {
      i++;
      if (i % 5 == 0) break;
      System.out.println(i);
    }

    System.out.printf("Ejemplo continue\n"); i = 0;
    while (i <= 20) {
      i++;
      if (i % 5 == 0) continue;
      System.out.println(i);
    }
  }
}
```

Instrucción return

- Se usa para terminar la ejecución de un método y retornar el control en la sentencia que hizo la invocación del método.
- Sintaxis:

```
return [expresion];
```

  - el tipo de dato de la expresión devuelta debe coincidir con el tipo de dato del valor devuelto en la declaración del método.
  - cuando un método es declarado void no se incluye ninguna expresión.