Universidad
Carlos III de Madrid

Tema 4

Control de flujo

Programación
2015-2016

Tema 4. Control de flujo

- **Introducción.**
- Estructuras condicionales.
- Estructuras de repetición.

¿Qué es el flujo de un programa?

- Es el orden en que se ejecutan las instrucciones de un programa
- El orden normal es instrucción por instrucción, es decir en ***secuencia***.
- El ***bloque*** permite esta ejecución secuencial
- Un bloque contiene las instrucciones entre las *llaves*:

```
{
   sentencia 1;
   sentencia 2;
   sentencia 3;
   …
}
```

Modificando el flujo de un programa

- En Java, se puede modificar el flujo secuencial mediante las estructuras de control:
  - Estructuras **condicionales**: un bloque sólo se ejecuta bajo ciertas condiciones
  - Estructuras de **repetición**: un mismo bloque se ejecuta repetidamente

- Ejemplos:
  - Escribir un mensaje de saludo si encuentras el nombre de una persona
  - Escribir “¡Programación!” 10 o 100 o 1000 veces

Interacción con el usuario

Salida: Escribir en la pantalla

```
System.out.println("Hola");
```

Entrada: Leer del teclado

```
import java.util.Scanner
  ...
Scanner teclado = new Scanner(System.in);
...
String nombre = teclado.nextLine();
int horas = teclado.nextInt();
double precio = teclado.nextDouble();
```

Tema 4. Control de flujo

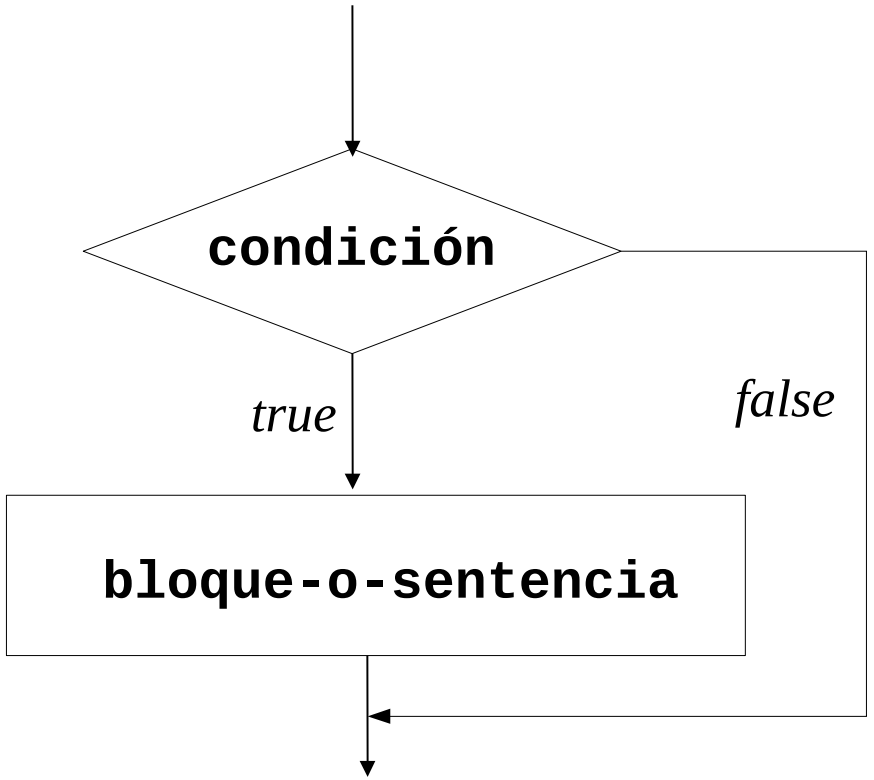
- Introducción.
- **Estructuras condicionales.**
- Estructuras de repetición.

Instrucciones condicionales

- **`if`**
- **`if-else`**
- **`if-else-if`**
- **`switch`**
- El operador **`?:`**

Instrucción if

Diagrama de flujo



Sintaxis:

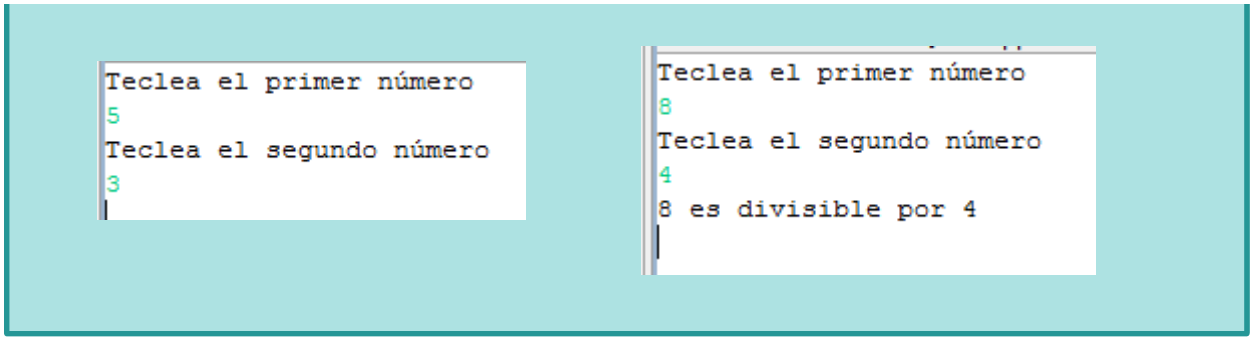
```
if (<condición>){
   <bloque-o-sentencia>
      . . . . . .
}
```

Ejercicios

- Determinar si un número es divisible por otro (el resultado de la división es un entero)
- Dados dos números si el primer número es divisible por el segundo, mostrar el cociente en la pantalla

Determinar si un número es divisible por otro (el resultado de la división es un entero)

```java
import java.util.Scanner;

public class EsDivisible {
//Determinar si un número es divisible por otro
    public static void main(String[] args) {
        int num1, num2;
        Scanner lectura=new Scanner(System.in);
        //lectura del primer número
        System.out.println("Teclea el primer número");
        num1=lectura.nextInt();
        //lectura del segundo número
        System.out.println("Teclea el segundo número");
        num2=lectura.nextInt();
        if (num1%num2==0)
            System.out.println(num1+" es divisible por "+num2);
    }

}
```



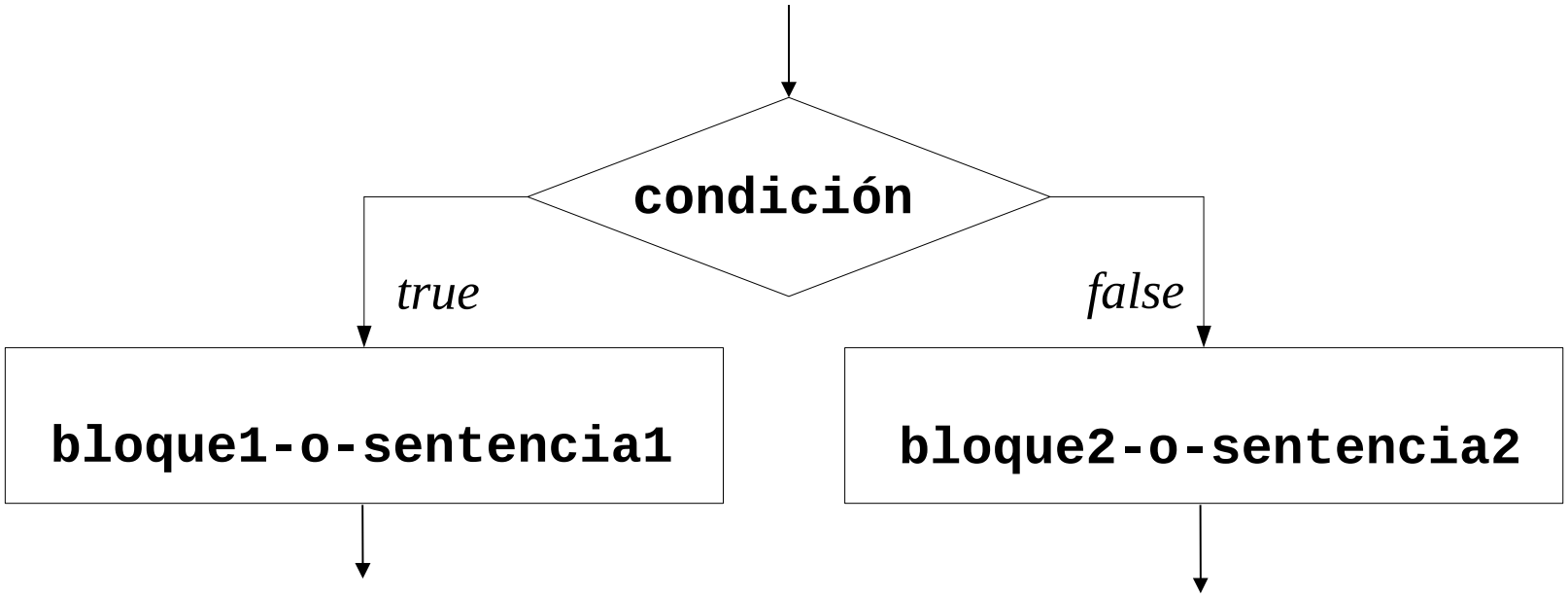
Dados dos números si el primer número es divisible por el segundo, mostrar el cociente en la pantalla

```java
import java.util.Scanner;

public class EsDivisible {
//Determinar si un número es divisible por otro
    public static void main(String[] args) {
        int num1, num2;
        Scanner lectura=new Scanner(System.in);
        //lectura del primer número
        System.out.println("Teclea el primer número");
        num1=lectura.nextInt();
        //lectura del segundo número
        System.out.println("Teclea el segundo número");
        num2=lectura.nextInt();
        if (num1%num2==0){
            System.out.println(num1+" es divisible por "+num2);
            System.out.println("el cociente es "+num1/num2);
        }
    }

}
```

```
Teclea el primer número
8
Teclea el segundo número
4
8 es divisible por 4
el cociente es 2
```

```
Teclea el primer número
5
Teclea el segundo número
7
```

Instrucción if- else



Sintaxis:

```
if (<condición>){
    <bloque1-o-sentencia1>
     }
else{
    <bloque2-o-sentencia2>
     }
```

Ejercicio

- Comprobar si un número entero está dentro del intervalo [-128, ..., 127], y:
  - Si es cierto, escribir que el número es un byte
  - En caso contrario, mostrar mensaje de error

Comprobar si un número entero está dentro del intervalo [-128, ..., 127], y: Si es cierto, escribir que el número es un byte. En caso contrario, mostrar mensaje de error

```
import java.util.Scanner;

public class EsByte {

    public static void main(String[] args) {
        int num;
        Scanner reader =new Scanner(System.in);
        System.out.println("Teclea número");
        num=reader.nextInt();
        if (num>=-128&&num<=127)
            System.out.println("el número "+num+" es byte");
        else
            System.out.println("error");


    }

}
```

```
Teclea número
45
el número 45 es byte
```

```
Teclea número
130
error
```

Instrucción if- else - if

Sintaxis:

```
if (<condición_1>)
        <bloque-o-sentencia_1>
else if (<condición_2>)
    <bloque-o-sentencia_2>
    ...
else if (<condición_I>)
    <bloque-o-sentencia_I>
else
    <bloque-o-sentencia_N>
```

Instrucción if- else - if

- Si no hay llaves, cada **`else`** siempre está asociado con el **`if`** más cercano

- Ejemplo: comprobar si los números **x** e **y** son mayores que 0

Instrucción if- else - if

```
if (x > 0)
   if (y > 0)
       System.out.println(“Ambos son mayores que 0");
else
   System.out.println(“Alguno no es mayor que 0");
```

```
if (x > 0) {
   if (y > 0){
       System.out.println(“Ambos son mayores que 0");
      }
   else {
       System.out.println(“Alguno no es mayor que 0");
   }
}
```

Instrucción if- else - if

```
if (x > 0)  {
   if (y > 0)
       System.out.println(“Ambos son mayores que 0");
}
else
   System.out.println(“Alguno no es mayor que 0");
```

```
if (x > 0)  {
   if (y > 0) {
       System.out.println(“Ambos son mayores que 0");
    }
}
else {
   System.out.println(“Alguno no es mayor que 0");
}
```

Ejercicios

- Determinar si un número es positivo, negativo o cero

- Comprobar si un carácter es un dígito, una letra mayúscula o una letra minúscula

  Código ASCII de:
  - dígito: 48 – 57
  - letra mayúscula: 65 – 90
  - letra minúscula: 97 – 122

Determinar si un número es positivo, negativo o cero

```
import java.util.Scanner;

public class PositivoNegativoCero {

    public static void main(String[] args) {
        int num;
        Scanner reader =new Scanner(System.in);
        System.out.println("Teclea número");
        num=reader.nextInt();
        if (num>0)
            System.out.println("el numero "+num+" es positivo");
        else
            if (num<0)
                System.out.println("el numero "+num+" es negativo");
            else
                System.out.println("el numero "+num+" es cero");
    }

}
```

Comprobar si un carácter es un dígito, una letra mayúscula o una letra minúscula

```
import java.util.Scanner;

public class DigitoMayusculaMinuscula {

    public static void main(String[] args) {
        Scanner reader =new Scanner(System.in);
        System.out.println("Teclea carácter");
        char a;
        a=reader.next().charAt(0);
        if (a>48&&a<57)
            System.out.println(a+" es digito");
        else {
            if (a>65 && a<90)
                System.out.println(a+" es mayúscula");
            else
                if (a>97 && a<122)
                    System.out.println(a+" es minúscula");
        }
    }
}
```

Instrucción switch

- A menudo la condición de un **`if-else-if`** depende de una sola variable de tipo primitivo
- Tipos primitivos = palabras reservadas; por ejemplo **`byte`**, **`int`**, **`double`** pero no **`String`**
- En este caso se puede utilizar otra instrucción llamada **`switch`**
- Es más compacto que un **`if-else-if`**

Instrucción switch

Sintaxis:

```
switch (<selector>) {
      case <etiqueta_1>:
                  <sentencias1>;
                              break;
      case <etiqueta_2>:
                  <sentencias2>;
                              break;
      ...
      case <etiqueta_n>:
                  <sentenciasn>;
                              break;
      default:
                  <sentenciasD>;  // opcional
}
```

Ejercicio

- Leer un número entero entre 1 y 10. Después mostrar ese número escrito en números romanos

Leer un número del 1 al 10 y mostrarlo en números romanos

```
import java.util.Scanner;
public class NumerosRomanos {
        public static void main(String[] args) {
                int num;
                Scanner reader =new Scanner(System.in);
                System.out.println("Teclea número");
                num=reader.nextInt();
                switch (num){
                        case 1:
                                System.out.println("I");
                                break;
                        case 2:
                                System.out.println("II");
                                break;
                        case 3:
                                System.out.println("III");
                                break;
                        case 4:
                                System.out.println("IV");
                                break;
                        case 5:
                                System.out.println("V");
                                break;
                        case 6:
                                System.out.println("VI");
                                break;
                        case 7:
                                System.out.println("VII");
                                break;
                        case 8:
                                System.out.println("VIII");
                                break;
                        case 9:
                                System.out.println("IX");
                                break;
                        case 10:
                                System.out.println("X");
                                break;
                }
        }
}
```

¿Qué pasa si excluimos break?

- ¡El programa pasa a la siguiente instrucción dentro del **`switch`**!
- La función de **`break`** es cambiar el control de flujo

El operador ?:

- Es el único operador ternario en Java (requiere tres operandos)
- Sintaxis:
  <operando1> ? <operando2> : <operando3>
- **operando1** tiene que ser una condición
- **operando2** y **operando3** pueden ser expresiones de cualquier tipo siempre que sean los dos del mismo tipo
- El resultado de la expresión es
  operando2 si **operando1** es **true**
  operando3 si **operando1** es **false**

Palabras reservadas

`class`	inicio de programa
`public, static, void`	método ‘main’
`byte, short, int, long`	enteros
`float, double`	reales
`char`	caracteres
`boolean`	tipo Booleano
`true, false`	literales Booleanos
`if, else, switch`	instrucción condicional
`case, break, default`	partes de **`switch`**

Ejercicio

S3-Clase: Condicionales

Tema 4. Control de flujo

- Introducción.
- Estructuras condicionales.
- **Estructuras de repetición.**

Repetición

- A veces hay que ejecutar la misma operación más de una vez

- Ejemplos:
  - Escribir ¡Bienvenidos! en la pantalla 10 o 100 o 1000 veces
  - Escribir todas las letras del abecedario en la pantalla
  - Sumar los gastos anuales de una empresa

- En cada caso, se pueden escribir las mismas instrucciones secuencialmente (esto es muy ineficaz)

- En cambio podemos usar bucles (estructuras de repetición)

Estructuras de repetición

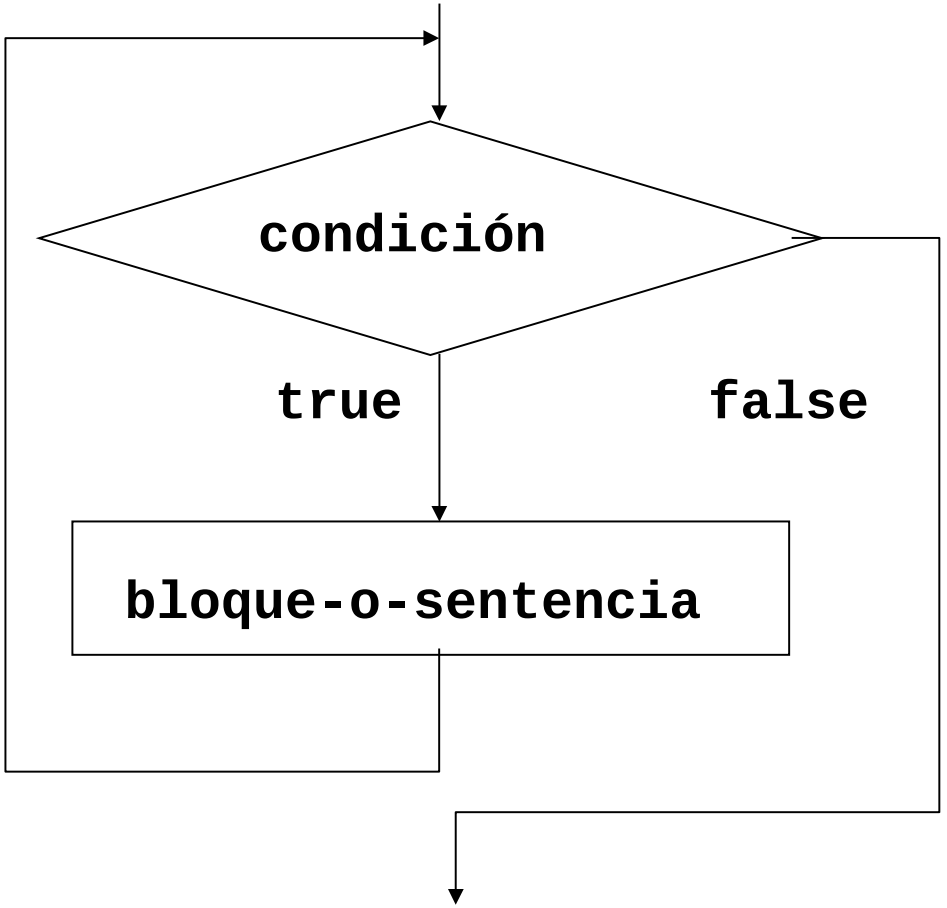
While

do-while

for

Bucle while

Diagrama de flujo



Sintaxis java:

```
while (<condición>){

    <bloque-o-sentencia>

}
```

Condición de while

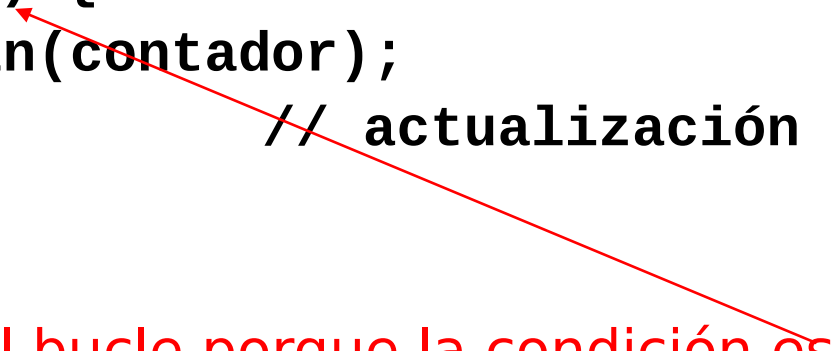
- La condición de un **`while`** sirve para determinar cuando debe terminar el bucle
- ¡Si la condición siempre se cumple (**`true`**), el bucle **`while`** nunca termina! (bucle infinito)
- En consecuencia, algo en la condición tiene que cambiar
- Cada bucle necesita una variable que aparezca en la condición y cuyo valor cambie en cada pasada
- Hay que inicializar el valor de la variable de bucle y actualizar su valor dentro del bucle

Ejemplo 1

```
int contador = 1;     // inicialización
while (contador < 6) {  // condición
  System.out.println(contador);
  contador++;                  // actualización
}
```

Ejemplo 2

```
int contador = 1;     // inicialización
while (contador < 0) {  // condición
  System.out.println(contador);
  contador--;                  // actualización
}
```



Nunca se entra en el bucle porque la condición es **`false`** desde el principio

Ejemplo 3

```
int contador = 1;    // inicialización
while (contador != 10) {// condición
  System.out.println(contador);
  contador += 2;     // actualización
}
```

El bucle nunca termina porque la condición nunca llega a ser **`false`**

Ejemplo 4

```
int contador = 1;     // inicialización
while (contador < 6)     // condición
  System.out.println(contador);
  contador++;                  // actualización
```

El bucle nunca termina porque el valor de la variable de bucle no se actualiza dentro del bucle

Ejercicio

- Escribir ¡Bienvenidos! en la pantalla 10 veces

Escribir ¡Bienvenidos! en la pantalla 10 veces

```
public class Bienvenidos {

   public static void main(String[] args) {
      int contador = 0;
      while (contador < 10)   {
         System.out.println("Bienvenidos");
         contador++;
      }
   }

}
```

Ejercicio

- ¿Cuál es el valor de la variable x después de ejecutar el siguiente bucle?

```
int x = 1;
int i = 1;
while (i <= 4) {
  x = x + 2*i;
  i++;
}
```

valor de la variable x después de ejecutar el bucle

```
public class Bucle {

   public static void main(String[] args) {
      int x = 1;
      int i = 1;
      while (i <= 4) {
         x = x + 2*i;
         i++;
         System.out.println("i:"+i+"  x:"+x);
      }
   System.out.println("x final:"+x);
   }

}
```

```
i:2  x:3
i:3  x:7
i:4  x:13
i:5  x:21
x final:21
```

Ejercicio

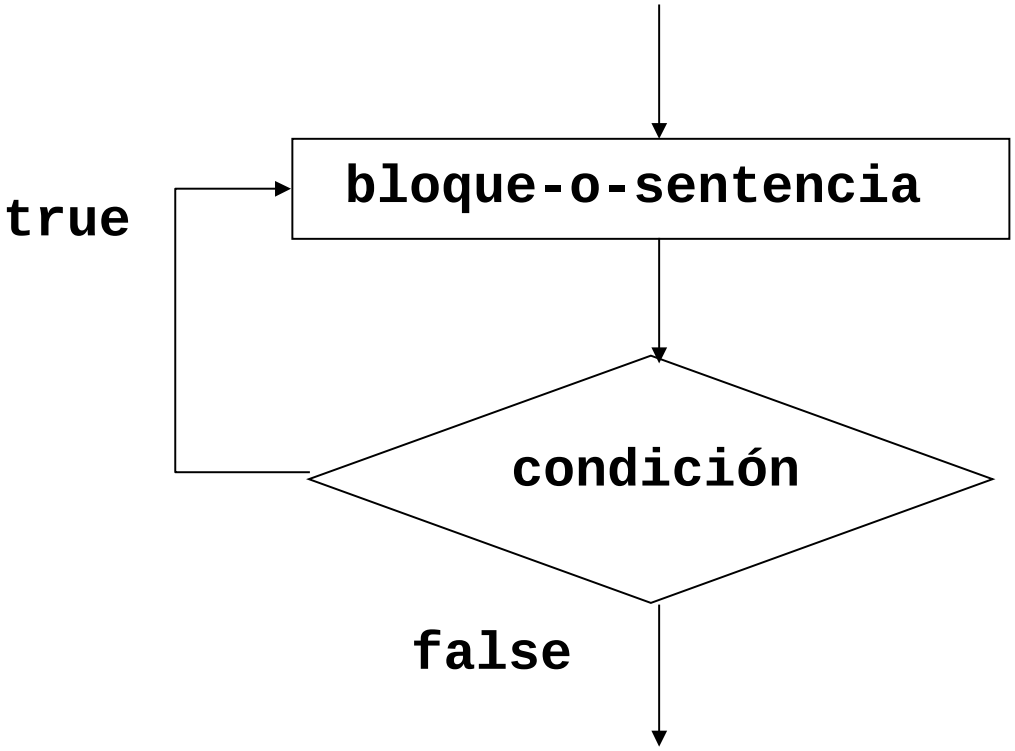
- El factorial de un número entero n, que se escribe n!, es el producto de n por todos los enteros que le preceden.
  - Por ejemplo, 4! = 1*2*3*4 = 24.

Calcular el valor de 30!

```
public class Factorial {

   public static void main(String[] args) {

      int contador = 30;
      int factorial = 1;
         while (contador > 0)   {
            factorial *= contador;
            contador--;
         }
      System.out.println("30! = " + factorial);
   }

}
```

Bucle do - while

Diagrama de Flujo



Sintaxis java:

```
do {
   <bloque-o-sentencia>

}while (<condición>);
```

Bucle do - while

- La diferencia entre un bucle **`while`** y un bucle **`do-while`** es que el bloque del **`do-while`** siempre se ejecuta por lo menos una vez
- La condición se prueba al final del bucle
- Siempre se puede convertir en un **`while`**, pero hay casos en que el **`do-while`** es más compacto

Ejercicio

- Introducir un número entero entre 0 y 10

  Mientras que el número esté fuera del rango indicado, introducir un número de nuevo

Introducir un número entero entre 0 y 10

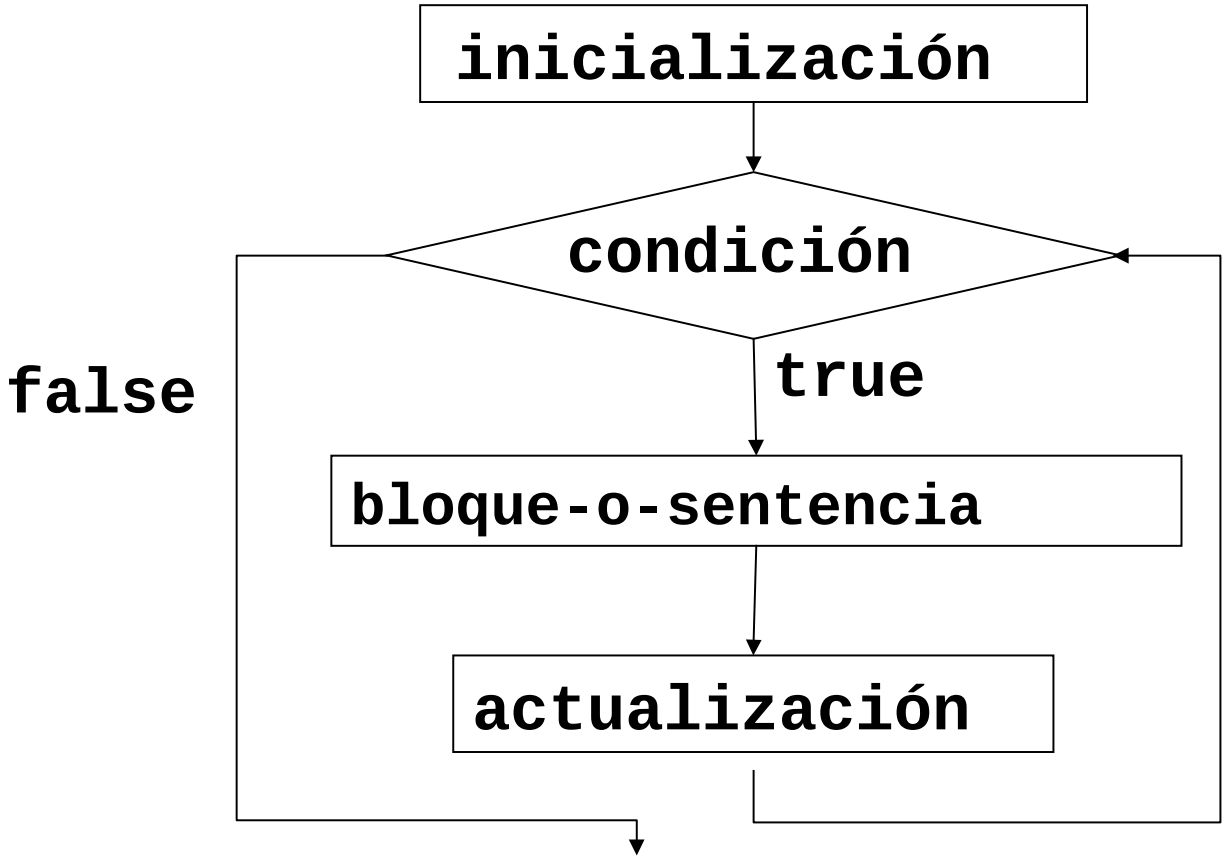
```java
import java.util.Scanner;

public class Rango {

    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int numero;

        do{
            System.out.println("Introduce numero (0-10):");
            numero = teclado.nextInt();
        }while((numero>=0)&&(numero>=10));

        System.out.println("Correcto");
        teclado.close();
    }

}
```

```
Introduce numero (0-10):
21888
Introduce numero (0-10):
3
Correcto
```

Bucle for

Sintaxis java:

```
for (<inicialización>;<condición>;<actualización>)
    <bloque-o-sentencia>
```



Bucle for

Sintaxis java:

```
for (<inicialización>;<condición>;<actualización>)
    <bloque-o-sentencia>
```

Modo de funcionamiento:

a) Crea e inicializa las variables definidas en **`inicialización`**.

b) Comprueba si se cumple la **`condición`**, si no se cumple no lo hace ninguna vez (0 a n).

c) Ejecuta el cuerpo del bucle (**`bloque-o-sentencia`**) una vez.

d) Ejecuta la **`actualización`** (modifica las variables)

e) Comprueba la condición otra vez [vuelve a b)] y así indefinidamente hasta que deje de cumplirse la condición.

Ejemplo para imprimir de 0 a 9:

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

Bucle for

Sintaxis java:

```
for (<inicialización>;<condición>;<actualización>)
    <bloque-o-sentencia>
```

En **inicialización** y **actualización** se pueden poner varias variables separadas por comas

inicialización y **actualización** no tienen porqué compartir variables, aunque es lo habitual (lo útil)

Se considera mala práctica modificar los valores de las variables de control dentro del bucle (para eso está la actualización).

Cualquiera de las tres expresiones puede faltar, pero mantenemos los puntos y coma.

Si **condición** no existe: bucle infinito.

Bucle for

Tiempo de vida de variables usadas en el for:

Si declaramos dentro, la variable sólo existe ahí.

```
for (int a=0; a<10; a++){
    … ...
    System.out.println(a);
}
```

Si declaramos fuera, cualquier cambio dentro afecta a la variable.

```
int a;
for (a=0; a<10; a++){
    … ...
}
System.out.println(a);
```

Ejemplos de uso de **inicialización**:

```
for (int a=0;...)
```

```
int a;
 for (a=0;...)
```

```
int a=0;
for ( a;...)
```

Bucle for

Si en **inicialización** declaramos varias variables, **TODAS** tienen que ser del mismo tipo (no vale `for(int a, double b;…)`)

```
for(int i=0, j=0, k=0; i+j<10; i++, j++)
   k=i+j;
```

Pero si están declaradas fuera, **sí** se puede dar valor a variables de distinto tipo.

```
int i;
float j, k;
for(i=0, j=0; i+j<10; i++, j++)
   k=i+j;
```

Bucle for

- Es equivalente a un bucle **`while`**
- Como en el caso de **`do-while`**, muchas veces un bucle **`for`** es más compacto que un **`while`**

Comparación entre while y for

```
int i = 0;
while (i < 10) {
  System.out.println(“¡Bienvenido!”);
  i++;
}

for (int i = 0; i < 10; i++)
  System.out.println (“¡Bienvenido!”);
```

Bucles anidados

Ejemplo: escribir las letras del abecedario 10 veces

```
for (int i = 0; i < 10; i++) {
  for (char c = 97; c <= 122; c++){
      System.out.print(c);
  }
  System.out.println();
}
```

break y continue

- Son dos instrucciones que cambian el control de flujo dentro de un bucle
- **`break`**: origina la salida del bucle
- **`continue`**: continua dentro del bucle sin ejecutar las demás instrucciones
- Se deben usar con cuidado

Ejemplo

```
int a = 5;
for (int i = 1; i <= 10; i++) {
      if (i % 2 == 0){
            continue;
      }
      a = a + i;
      if (a > 20){
            break;
      }
}
```

Métodos length() y charAt() de String

A una variable de tipo String se pueden aplicar varios métodos:

```
String s = “¡Hola Mundo!”;
int longitud = s.length(); // longitud 12
char c = s.charAt(1);      // carácter H
```

La longitud se puede usar para recórrer los carácteres de la cadena

Uso de bucles: Recorrer un String

```
String s = “¡Hola Mundo!”;
int i = 0;
while (i < s.length()) {
  System.out.println(s.charAt(i));
  i++;
}
```

¿Qué pasa si inicializamos **`i = 1`** y cambiamos la condición del bucle por **`i <= s.length()`**?

Ejercicio

S5-Clase: Bucles