UD02.1.Estructuras de Control. Condicionales Apuntes

DAM1-Programación 2023-24

Expresiones lógicas

- Estructuras de control en Java

Introducción

Un programa no tiene por qué ejecutar siempre la misma secuencia de instrucciones. Puede darse el caso de que, dependiendo del valor de alguna expresión o de alguna condición, interese ejecutar o evitar un conjunto de sentencias. Esta funcionalidad la brindan las instrucciones `if`, `if-else` y `switch`.

2.1. Expresiones lógicas

En primer lugar, hablaremos un poco sobre los condicionales. Una condición no es más que una expresión relacional o lógica. El valor de una condición siempre es de tipo booleano: verdadero o falso. En Java existen dos literales que representan este resultado: `true` y `false`.

La diferencia entre un operador relacional y uno lógico es que, mientras el primero utiliza como operandos expresiones numéricas, el segundo emplea expresiones booleanas. Pero ambos generan valores booleanos.

2.1.1. Operadores relacionales

Los operadores relacionales son aquellos que comparan expresiones numéricas para generar valores booleanos. Recordemos que solo existen dos posibles valores: verdadero o falso. Los operadores relacionales disponibles se recogen en la Tabla 2.1.

Tabla 2.1. Operadores relacionales

Símbolo	Descripción
==	Igual que
!=	Distinto que
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Veamos algunos ejemplos de expresiones relacionales.

a + *b* <= *18* será cierto si el valor de *a* más el valor de *b* es menor o igual que 18. Supongamos dos posibles casos:

a) Con $a = 10$ y $b = 2 \rightarrow 10 + 2 = 12$ y resulta que $12 \leq 18$ es `true`.

b) Con $a = 20$ y $b = 1 \rightarrow 20 + 1 = 21$ pero $21 \nleq 18$, que es `false`.

$2 \times 5 == 10$ es `true`, ya que 2 por 5 son 10.

1 != 2 es `true` también, ya que 1 es distinto de 2.

Actividad propuesta 2.1

Realiza un programa que almacene la evaluación de distintas expresiones relacionales en variables booleanas, y muestra el valor de dichas variables.

Actividad propuesta 2.2

Solicita por teclado un número de tipo `int` al usuario y escribe un programa que muestre `true` o `false`, dependiendo de si el número es positivo.

2.1.2. Operadores lógicos

Los operadores lógicos permiten construir condiciones más complejas, ya que estos operan y generan valores booleanos. Sus operadores se definen en la Tabla 2.2.

Tabla 2.2. Operadores lógicos

Símbolo	Descripción
&&	Operador Y
||	Operador O
!	Operador negación

Operador `&&`: será cierto si ambos operandos son ciertos (Tabla 2.3).

Tabla 2.3. Tabla de verdad del operador &&

a	b	a && b
falso	falso	falso
cierto	falso	falso
falso	cierto	falso
cierto	cierto	cierto

Operador `||`: es cierto si cualquiera (uno o ambos) de los operandos es cierto, como muestra la Tabla 2.4.

Tabla 2.4. Tabla de verdad del operador `||`

a	b	a || b
falso	falso	falso
cierto	falso	cierto
falso	cierto	cierto
cierto	cierto	cierto

Operador `!`: niega —cambia— el valor al que se aplica, convirtiendo `true` en `false` y viceversa, como se aprecia en la Tabla 2.5.

Tabla 2.5. Tabla de verdad del operador `!`

a	!a
falso	cierto
cierto	falso

Veamos algunos ejemplos. Supongamos que *a* vale 3 y *b* vale 5; en tal caso,

a + *b* <= 18 → es cierto y

a == 4 → es falso

Entonces:

!(*a* + *b* <= 18) resulta falso.

(*a* + *b* <= 18) && (*a* == *4*) resulta falso, ya que el operador && devuelve falso si cualquiera de los operandos también lo hace.

!(*a* + *b* <= 18) || (*a* == 4) también es falso, ya que ambos operandos los son.

(*a* + *b* <= 18) || *(cualquier condición)* resulta cierto, independientemente del valor de la segunda condición, debido a que el operador ||, para devolver cierto, solo requiere que uno de los operandos sea cierto. En cambio,

(*a* == 4) && *(cualquier condición)* resulta falso independientemente del valor del segundo operando, ya que si uno de los operandos es falso, la expresión completa es falsa.

Actividad propuesta 2.3

Escribe una aplicación que pida al usuario dos números enteros y muestre: `true`, si ambos números son distintos entre sí o alguno de ellos es cero; y `false` en caso contrario.

Actividad propuesta 2.4

Realiza un programa que informe al usuario (mostrando `true`) si un primer número es múltiplo de otro número. Ambos números se piden por teclado.

Condicional simple: if

2.2. Condicional simple: if

La sentencia `if` proporciona un control sobre un conjunto de instrucciones que pueden ejecutarse o no, dependiendo de la evaluación de una condición. Los dos posibles valores (`true` o `false`) de esta determinan si el bloque de instrucciones de `if` se ejecuta (cuando la condición es `true`) o no (condición `false`). La sintaxis es la siguiente:

```
if (condición) {
   bloque de instrucciones
   ...
}
```

La Figura 2.1 nos muestra el flujo de control de un condicional simple, que es:

1. Se ejecutan las instrucciones anteriores a `if`.
2. Cuando le llega el turno a `if`, se evalúa la condición. El resultado será: `true` o `false`.
3. En caso de que la evaluación de la condición resulte `false`, no se ejecuta el bloque de instrucciones y se salta a la siguiente instrucción después de la estructura `if`.
4. Si, por el contrario, la evaluación de la condición es `true`, se ejecutará el bloque de instrucciones que contiene `if`. Este bloque de instrucciones puede albergar cualquier tipo de sentencia, incluido otro `if`.

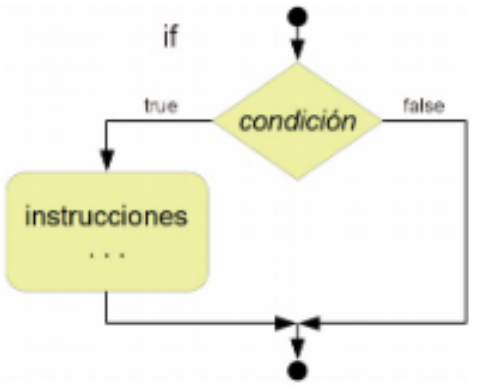

Figura 2.1. *Funcionamiento de la instrucción* `if`.

Un bloque de instrucciones es un conjunto de sentencias delimitadas mediante llaves (`{}`). Dentro de un bloque de instrucciones es posible utilizar cualquier número de sentencias, e incluso definir otros bloques de instrucciones. También existe la posibilidad de declarar variables, que solo podrán ser usadas dentro del bloque donde se declaran. El lugar o bloque donde es posible utilizar una variable se denomina *ámbito de la variable*. Hasta ahora hemos visto dos ámbitos de variables: las variables declaradas en `main`, que se denominan *variables locales*, y las variables declaradas en un bloque de instrucción, denominadas *variables de bloque*. El funcionamiento de ambas variables es idéntico; la única distinción entre variables locales y de bloque reside en su ámbito.

Veamos un ejemplo de una sentencia `if`:

```
a = 3;
if (a + 1 < 10) {
   a = 0;
   System.out.println("Hola");
}
System.out.println("El valor de a es: " + a);
```

Al evaluar la condición `(a + 1 < 10)` resulta: `3 + 1 < 10`, que da verdadero (`true`). Al ser cierta la evaluación de la condición, se ejecutará el bloque de instrucciones de `if`:

- `a = 0;` se asigna a la variable `a` un valor cero.
- Se muestra en pantalla el mensaje «Hola».

Una vez terminada la ejecución del bloque `if`, se continua con la siguiente instrucción, que muestra el mensaje «El valor de a es 0».

Veamos otro ejemplo similar con distinto resultado:

```
a = 9;
if (a + 1 < 10) {
   a = 0;
   System.out.println("Hola");
}
System.out.println ("El valor de a es: " + a);
```

En este caso, la condición resulta ser falsa, ya que diez no es menor que diez ($10 \not< 10$); en todo caso, son iguales. El bloque de instrucciones de `if` se ignora, ejecutándose directamente la siguiente instrucción, que mostraría el mensaje «El valor de a es 9».

E0201. Diseña una aplicación que solicite un número al usuario e indique si es par o impar.

Condicional doble: if-else

2.3. Condicional doble: if-else

Existe otra versión de la sentencia `if`, denominada `if-else`, donde se especifican dos bloques de instrucciones. El primero (bloque `true`) se ejecutará cuando la condición resulte verdadera y el segundo (bloque `false`), cuando la condición resulte falsa. Ambos bloques son mutuamente excluyentes, es decir, en cada ejecución de la instrucción `if-else` solo se ejecutará uno de ellos. La sintaxis es:

```
if (condición) {
   bloque true //se ejecuta cuando la condición es cierta
} else {
   bloque false //se ejecuta cuando la condición es falsa
}
```

La Figura 2.2 describe los posibles flujos de control de un programa que utilice un condicional doble.

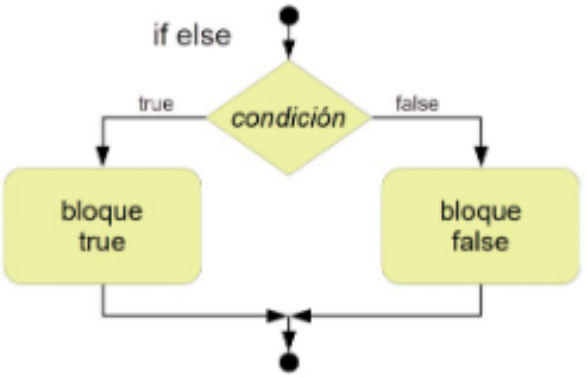

Figura 2.2. *Funcionamiento de la instrucción* `if-else`.

Veamos un ejemplo:

```
if (a > 0) {
   System.out.println("Valor positivo");
} else {
   System.out.println("Valor negativo o cero");
}
```

Supongamos que al evaluar la condición resulta cierta, lo que significa que la variable `a` tiene un valor mayor que 0. Entonces se ejecuta el primer bloque de instrucciones mostrando el mensaje «Valor positivo». Si, por el contrario, al evaluar la condición resulta falsa, lo que significa que `a` tiene un valor menor o igual que cero, se ignorará el primer bloque y se ejecutará el segundo, mostrando «Valor negativo o cero».

E0202. Pedir dos números enteros y decir si son iguales o no.

E0203. Solicitar dos números distintos y mostrar cuál es el mayor.

E0204. Implementar un programa que pida por teclado un número decimal e indique si es un número casi-cero, que son aquellos, positivos o negativos, que se acercan a 0 por menos de 1 unidad, aunque curiosamente el 0 no se considera un número casi-cero. Ejemplos de números casi-cero son el 0,3, el -0,99 o el 0,123; algunos números que no se consideran casi-ceros son: el 12,3, el 0 o el -1.

Tabla de verdad del ejercicio

`(numero == 0)`	`(numero > -1)`	`(numero < 1)`	`casiCero`
false	false	false	false
false	false	true	false
false	true	false	false
false	**true**	**true**	**true**
true	false	false	false
true	false	true	false
true	true	false	false
true	true	true	false

Año Bisiesto (Bisiesto.java). Haz un programa que pida por teclado un número de año y que muestre un mensaje indicando si el año es bisiesto o no.
Investiga el agoritmo para calcular si un año es bisiesto o no.

“Año bisiesto es el divisible entre 4, salvo que sea año secular es decir divisible por 100, en cuyo caso también ha de ser divisible entre 400.”

Tabla de verdad del ejercicio:

`(anho % 4 == 0)`	`(anho % 100 == 0)`	`(anho % 400 == 0)`	`bisiesto`
false	false	false	false
false	false	true	false
false	true	false	false
false	true	true	false
true	**false**	**false**	**true**
true	**false**	**true**	**true**
true	true	false	false
true	**true**	**true**	**true**

Son bisiestos: 4, 40, 120, 144, 400, 800, 2024. No son bisiestos: 3, 100, 2100

Factura (Factura.java). Escribe un programa para emitir la factura de compra de un producto, introduciendo el precio del producto y el número de unidades compradas. La factura deberá añadir al total un IVA (Impuesto del Valor Añadido) del 21%. Si el precio final con IVA es superior a 100 euros se aplicará un descuento del 5%.

Ejemplos para pruebas:

- Precio = 10 €, Unidades = 5 => Precio Final = 60,50 €
- Precio = 20 €, Unidades = 7 => Precio Final = 160,93 €

2.3.1. Operador ternario

El operador ternario permite seleccionar un valor de entre dos posibles, dependiendo de la evaluación de una condición. La sentencia

```
variable = condición ? valor1 : valor2
```

es equivalente a utilizar un condicional doble de la forma

```
if (condición) {
   variable = valor1;
} else {
   variable = valor2;
}
```

Es recomendable utilizar el operador ternario por economía y legibilidad del código, en lugar de un `if-else`, cuando sea posible.

Un ejemplo para calcular el máximo de dos números introducidos por teclado es:

```
Scanner sc = new Scanner(System.in);
int a = sc.nextInt();
int b = sc.nextInt();
int maximo = a > b ? a : b;
System.out.println("El máximo es: " + maximo);
```

E0205. Pedir dos números y mostrarlos ordenados de forma decreciente.

2.3.2. Anidación de condicionales

Cuando debamos realizar múltiples comprobaciones, podemos anidar tantos `if` o `if-else` como necesitemos, unos dentro de otros. La anidación de condicionales hace que las comprobaciones sean excluyentes, y resulta un código más eficiente. Veamos un ejemplo:

```
if (a - 2 == 1) {
   System.out.println("Hola ");
} else {
   if (a - 2 == 5) {
      System.out.println("Me ");
   } else {
      if (a - 2 == 8) {
         System.out.println("Alegro ");
      } else {
         if (a - 2 == 9) {
            System.out.println("De ");
         } else {
            if (a - 2 == 11) {
               System.out.println("Conocerte.");
            } else {
               System.out.println("Sin coincidencia");
            }
         }
      }
   }
}
```

Podemos aprovechar una cualidad que poseen tanto `if` como `else`, y es que cuando el bloque de instrucciones del condicional está formado por una única sentencia, no es necesario utilizar llaves (`{ }`), aunque son recomendables. De todas formas, cuando hay muchos casos alternativos es habitual eliminar las llaves de los bloques `else`, consiguiendo un código más compacto. De esta manera, el ejemplo anterior podría reescribirse

```
if (a - 2 == 1) {
   System.out.println("Hola ");
} else if (a - 2 == 5) {
   System.out.println("Me ");

} else if (a - 2 == 8) {
   System.out.println("Alegro ");
} else if (a - 2 == 9) {
   System.out.println("De ");
} else if (a - 2 == 11) {
   System.out.println("Conocerte.");
} else {
   System.out.println("Sin coincidencia");
}
```

Debugger. A medida que los programas se van complicando puede ser útil utilizar las funciones de depuración paso a paso de entornos de desarrollo como Netbeans. Aprende a detener la ejecución de tus programas en los puntos del código (*breakpoint*) cuyo funcionamiento quieras analizar con más detalle.

- Run and Debug Java in Visual Studio Code

E0206. Realizar de nuevo la actividad 3 considerando el caso de que los números introducidos sean iguales.

E0207. Pedir tres números y mostrarlos ordenados de mayor a menor.

Posibles entradas:
1 2 3
1 3 2
2 3 1
2 1 3
3 1 2
3 2 1

Resultado menor a mayor: 1 2 3
Resultado mayor a menor: 3 2 1

E0208. Pedir los coeficientes de una ecuación de segundo grado ($ax^2 + bx + c = 0$) y mostrar sus soluciones reales. Si no existen, habrá que indicarlo. Las soluciones de una ecuación de segundo grado son:

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

E0209. Escribir una aplicación que indique cuántas cifras tiene un número entero introducido por teclado, que estará comprendido entre 0 y 99999.

Realiza una versión del programa anterior para que indique cuantes cifras tiene un número comprendido entre -99999 y 99999.

Condicional múltiple: switch

2.4. Condicional múltiple: switch

En ocasiones, el hecho de utilizar muchos `if` o `if-else` anidados suele producir un código poco legible y difícil de mantener. Para estos casos Java dispone de la sentencia `switch`, cuya sintaxis es:

```
switch (expresión) {
   case valor1:
      conjunto instrucción 1
   case valor2:
      conjunto instrucción 2
   ...

   case valorN:
      conjunto instrucción N
   default:
      conjunto instrucción default
}
```

La evaluación de expresión debe dar un resultado entero, convertible en entero o un valor de tipo `String`. La cláusula `default` es opcional. Disponemos de la Figura 2.3 para describir el comportamiento de un condicional múltiple.

Argot técnico

La clase `String` es una herramienta disponible en Java que permite crear y manipular texto. Se considera texto cualquier palabra, frase o conjunto de caracteres, como por ejemplo: «Hola», «Mi nombre es Pepe» o «abcd1234».

En la Unidad 6 veremos a fondo las cadenas de texto y la clase `String`.

La dinámica del `switch` es la siguiente:

1. Evalúa `expresión` y obtiene su valor.
2. Compara, uno a uno, el valor obtenido con cada valor de las cláusulas `case`. Se puede utilizar en un mismo `case` varios valores separados por coma.
3. En el momento en que coincide con alguno de ellos, ejecuta el conjunto de instrucciones de esa cláusula `case` y de todas las siguientes.
4. Si no existe coincidencia alguna, se ejecuta el conjunto de instrucciones de la cláusula `default`, siempre y cuando esta esté presente. No olvidemos que es opcional.

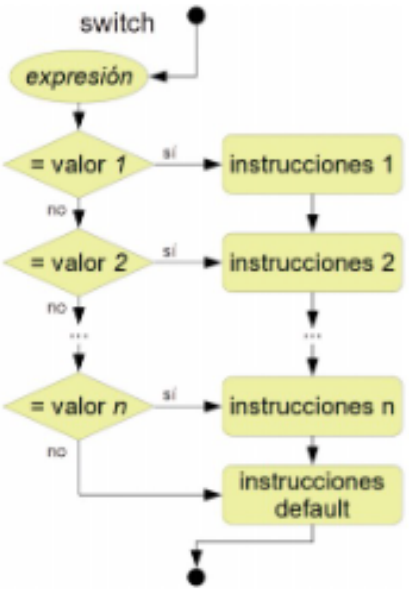

Figura 2.3. *Funcionamiento de la instrucción* `switch`.

A continuación se muestra un ejemplo sencillo:

```
a = 10;
switch (a-2) {
   case 1:
      System.out.print("Hola ");
   case 5:
      System.out.print("Me ");
   case 8:
      System.out.print("Alegro ");
   case 9:
      System.out.print("De ");
   case 11:
      System.out.print("Conocerte. ");
   default:
      System.out.print("Sin coincidencia");
}
```

Veamos cómo se ejecuta `switch`:

1. Evaluamos la expresión, en este caso `a`, que vale 10, menos 2 resulta 8.
2. Se comprueba uno a uno el valor de cada `case`. En el ejemplo, el tercer `case` lleva asociado el valor 8, que coincide con el resultado de la evaluación de la expresión.
3. Se ejecuta el conjunto de instrucciones desde el tercer `case` hasta el último, incluyendo la cláusula `default`.

El resultado final es:

```
Alegro De Conocerte. Sin coincidencia
```

Como puede verse, dependiendo del orden en el que coloquemos las cláusulas `case` se ejecutará un conjunto de instrucciones u otro. Java dispone de la sentencia `break` que, colocada al final de cada bloque de sentencias, impide que la ejecución continúe en el bloque siguiente. Esto nos permite hacer que solo se ejecute un bloque. De esta forma, es más sencillo escribir el `switch` sin tener que preocuparse del orden en el que se colocan los `case`.

La Figura 2.4 describe los flujos de control si utilizamos `break`.

Por otra parte, se puede emplear un sistema mixto, donde algunos bloques de instrucciones acaben en `break` y otros no, según nuestra conveniencia.

A continuación se muestra un ejemplo de código que usa `break` en todos los bloques:

```
a = 1;
switch (a*2) {
   case 1:
      System.out.println("Hola");
      break;
   case 2:
      System.out.println("Paco");
      break;
```

```
    case 3:
        System.out.println("Adiós");
        break;
    default:
        System.out.println("Sin coincidencia");
        break;
}
```

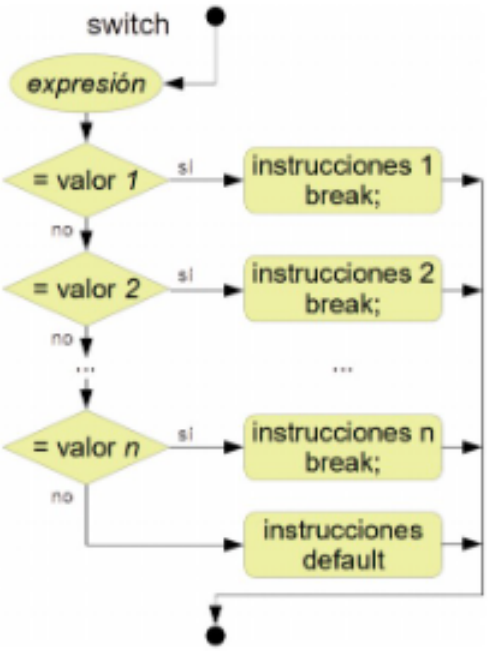

Figura 2.4. *Funcionamiento de la instrucción* `switch` *con* `break`.

Nótese que el último `break` no es necesario, pero, aparte de homogeneizar el código, facilita una futura modificación del programa por si decidimos añadir otros `case` al final.

En este ejemplo, el valor de la expresión principal del `switch` coincide con el segundo `case`. Se ejecuta el bloque de instrucciones asociado al segundo `case`, pero `break` impide que se ejecuten los siguientes. La salida por pantalla sería «Paco».

Veamos otro ejemplo:

```
a = 1;
switch (a*2) {
    case 1:
        System.out.println("Hola");
        break;
    case 2:
        System.out.println("Paco");
    case 3:
        System.out.println("Adiós");
        break;
    default:
        System.out.println("Sin coincidencia");
}
```

Este ejemplo es similar al anterior, pero hemos eliminado algunos `break`. Se ejecutará el bloque del segundo `case`, junto a los bloques de los siguientes `case`. La ejecución continuará hasta encontrar el `break` del tercer `case`. La salida que se obtiene es:

```
Paco
Adiós
```

Sintaxis alternativa de switch

- [Switch expression en Java 17 – Adam Gamboa G]

En las últimas versiones de Java se ha añadido una nueva forma de utilizar la instrucción `switch` en la que no es necesario usar `break`. Si el valor de la expresión coincide con algún `case`, solamente se ejecutará el bloque de instrucciones asociado a dicho `case`. La forma de distinguir entre la versión clásica de `switch` y la nueva es que esta, en lugar de utilizar dos puntos (:) después de cada `case`, emplea un guion seguido de un mayor que (`->`).

En la nueva versión de `switch`, cada bloque de instrucciones de un `case` debe ir entre llaves, excepto si el bloque de instrucciones está formado por una única instrucción. En este caso no es necesario encerrar la instrucción entre llaves.

Veamos cómo mostrar una nota numérica (del 1 al 10) en una calificación textual:

```
switch (nota) {
   case 0,1,2,3,4 -> { //bloque formado por dos instrucciones: entre llaves
      System.out.println("Suspenso.");
      System.out.println("Ánimo...");
   }
   case 5 -> //bloque de una única instrucción: podemos obviar las llaves
      System.out.println("Suficiente.");
   case 6 ->
      System.out.println("Bien.");
   case 7, 8 ->
      System.out.println("Notable");
   case 9, 10 -> {
      System.out.println("Sobresaliente.");
      System.out.println("Enhorabuena");
   }
   default ->
      System.out.println("Nota incorrecta");
}
```

switch como expresión con yield.

- [What does the new keyword "yield" mean in Java 13? - Stack Overflow]

Además, el nuevo `switch` también permite que se use como una expresión, es decir, toda la sentencia `switch` se sustituirá por un valor, con el que podremos operar o simplemente asignarlo a una variable. Para que `switch` se use como una expresión, dentro de cada bloque de instrucciones `case` debe especificarse el valor que sustituirá a la instrucción `switch`. Para ello usaremos la palabra reservada `yield`. El uso de `yield` implica que todos los bloques de instrucciones deberán estar delimitados por llaves (indistintamente de si están formados por una o más instrucciones).

Veamos ahora cómo asignar a la variable `dias` el número de días que tiene un mes (descrito con un número del 1 al 12):

```
System.out.println("Escriba un mes (1 al 12):");
int mes = new Scanner(System.in).nextInt();
int dias = switch (mes) {

   case 1, 3, 5, 7, 8, 10, 12 -> {
      yield 31; //estos meses tienen 31 días
   }
   case 2 -> {
      yield 28; //febrero tiene 28 días
   }
   case 4, 6, 9, 11 -> {
      yield 30; //el resto de meses tiene 30 días
   }
   default -> {
      System.out.println("Error: el mes es incorrecto");
      yield -1; //con -1 indicamos que hay un error
   }
};
System.out.println("Días: " + dias);
```

En este ejemplo hay que señalar que la asignación (marcada con el operador `=` en rojo) no finaliza hasta que se escribe toda la instrucción `switch`, y que, como toda instrucción, necesita su punto y coma final (también en rojo). Toda la sentencia `switch` se sustituye por el valor que, en cada caso, indique `yield`.

E0210. Pedir una nota entera de 0 a 10 y mostrarla de la siguiente forma: insuficiente (de 0 a 4), suficiente (5), bien (6), notable (7 y 8) y sobresaliente (9 y 10).

E0211. Idear un programa que solicite al usuario un número comprendido entre 1 y 7, correspondiente a un día de la semana. Se debe mostrar el nombre del día de la semana al que corresponde. Por ejemplo, el número 1 corresponde a “lunes” y el 6 a “sábado”.

E0212. Pedir el día, mes y año de una fecha e indicar si la fecha es correcta. Hay que tener en cuenta que existen meses con 28, 30 y 31 días (no se considerarán los años bisiestos).

E0213. Escribir un programa que pida una hora de la siguiente forma: hora, minutos y segundos. El programa debe mostrar qué hora será un segundo más tarde. Por ejemplo: hora actual (10:41:59) => hora actual + 1 segundo (10:42:00)

E0214. Crear una aplicación que solicite al usuario una fecha (día, mes, año) y muestre la fecha correspondiente al día siguiente.

Actividad propuesta 2.5

Escribir un programa que calcule el dinero recaudado en un concierto. La aplicación solicitará el aforo máximo del local, el precio por entrada (suponemos que todas las entradas tienen el mismo precio) y el número de entradas vendidas. Hay que tener en cuenta que si el número de entradas vendidas no supera el 20% del aforo del local, se cancela el concierto. Si el número de entradas vendidas no supera el 50% del aforo del local, se realiza una rebaja del 25% del precio de la entrada.