



Condicionales

Objetivos

- Escribir y probar código que haga uso de estructuras de selección.
- Clasificar, reconocer y utilizar los operadores del lenguaje en expresiones.
- Reconocer la estructura de un programa informático, identificando y relacionando los elementos propios de Java.
- Diseñar aplicaciones cuya ejecución no es siempre lineal, permitiendo, a partir de la entrada de datos, discriminar entre distintos escenarios.

Contenidos

- 2.1. Expresiones lógicas
- 2.2. Condicional simple: if
- 2.3. Condicional doble: if-else
- 2.4. Condicional múltiple: switch

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
<code>==</code>	Igual que
<code>!=</code>	Distinto que
<code><</code>	Menor que
<code><=</code>	Menor o igual que
<code>></code>	Mayor que
<code>>=</code>	Mayor o igual que

Veamos algunos ejemplos de expresiones relacionales.

$a + b \leq 18$ será cierto si el valor de a más el valor de b es menor o igual que 18. Supongamos dos posibles casos:

- Con $a = 10$ y $b = 2 \rightarrow 10 + 2 = 12$ y resulta que $12 \leq 18$ es `true`.
- Con $a = 20$ y $b = 1 \rightarrow 20 + 1 = 21$ pero $21 \not\leq 18$, 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
<code>&&</code>	Operador Y
<code> </code>	Operador O
<code>!</code>	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 `!`

<code>a</code>	<code>!a</code>
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.

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`.

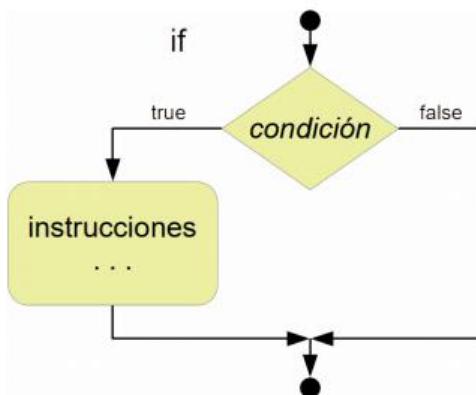


Figura 2.1. Funcionamiento de la instrucción `if`.

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».

Actividad resuelta 2.1

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

Solución

```
import java.util.Scanner;
/* Vamos a introducir por teclado un número entero. Para saber si es par
 * o impar comprobamos el resto de dividir por 2. */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num; //número introducido por el usuario
        System.out.print("Introduzca un número: ");
        num = sc.nextInt();
        if (num % 2 == 0) { //si num es par
            System.out.println("Es par");
        } else { // es impar
            System.out.println("Es impar");
        }
    }
}
```

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.

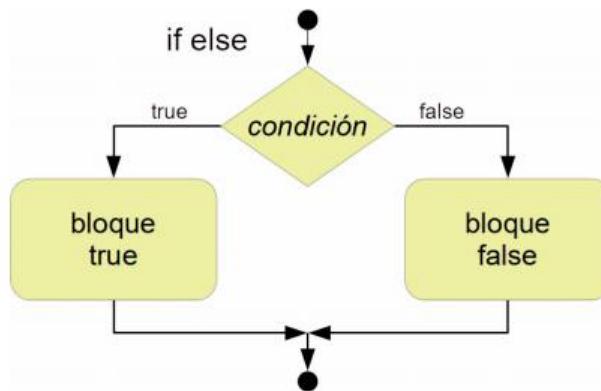


Figura 2.2. Funcionamiento de la instrucción `if-else`.

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».

Actividad resuelta 2.2

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

Solución

```
import java.util.Scanner;

/* Leemos dos números enteros, que tendremos que comparar para decidir si son
 * iguales o distintos */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número: ");
        int n1 = sc.nextInt(); //primer número
```

```

        System.out.print("Introduzca otro número: ");
        int n2 = sc.nextInt();
        if (n1 == n2) { //si n1 es igual que n2
            System.out.println("Ambos números son iguales");
        } else {
            System.out.println("Los números son distintos");
        }
    }
}

```

Actividad resuelta 2.3

Solicitar dos números distintos y mostrar cuál es el mayor.

Solución

```

import java.util.Scanner;
// Leemos dos números (en este caso enteros), que compararemos con el operador >
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número: ");
        int n1 = sc.nextInt();
        System.out.print("Introduzca otro número: ");
        int n2 = sc.nextInt();
        // el caso donde ambos números son iguales no se contempla e imprimiría
        // en pantalla que n2 es mayor que n1
        if (n1 > n2) {
            System.out.println(n1 + " es mayor que " + n2);
        } else {
            System.out.println(n2 + " es mayor que " + n1);
        }
    }
}

```

Actividad resuelta 2.4

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.

Solución

```

import java.util.*;
/* Un número casi-cero es el que se encuentra en el intervalo (-1, 1), donde
 * se excluye el -1, el 0 y el 1. Para comprobar si un número es casi-cero
 * tendremos que utilizar una condición con una expresión lógica. */
public class Main {
    public static void main(String[] args) {

```

```

Scanner sc = new Scanner(System.in);
sc.useLocale(Locale.US); //para utilizar punto (.) con los decimales
System.out.print("Introduzca un número real positivo o negativo: ");
double num = sc.nextDouble();
//un casi-cero es mayor que -1, menor que 1 y no es 0
if (-1 < num && num < 1 && num != 0) {
    System.out.println("Es un número casi-cero");
} else {
    System.out.println("No es un casi-cero");
}
}
}

```

■■■ 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);

```

Actividad resuelta 2.5

Pedir dos números y mostrarlos ordenados de forma decreciente.

Solución

```

import java.util.Scanner;
/* Para ordenar dos números hay que compararlos. Es posible hacer el programa
 * usando if-else, pero en este caso vamos a hacerlo con el operador ternario.
 */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

```

```

int n1, n2; //números introducidos por el usuario
int mayor, menor; //contendrán el mayor y el menor de n1 y n2
System.out.print("Introduzca un número: ");
n1 = sc.nextInt();
System.out.print("Introduzca otro: ");
n2 = sc.nextInt();
mayor = n1>n2 ? n1 : n2; //si n1 es el mayor, entonces mayor=n1, si no = n2
menor = n1<n2 ? n1 : n2; //si n1 es menor que n2, entonces menor=n1, si no = n1
System.out.println(mayor + ", " + menor);
}
}

```

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");
}

```

Actividad resuelta 2.6

Realizar de nuevo la Actividad resuelta 2.3 considerando el caso de que los números introducidos sean iguales.

Solución

```

import java.util.Scanner;
// En esta versión contemplamos la posibilidad que ambos números sean iguales.
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número: ");
        int n1 = sc.nextInt();
        System.out.print("Introduzca otro número: ");
        int n2 = sc.nextInt();
        if (n1 == n2) {
            System.out.println("Son iguales");
        } else {
            //si no son iguales podemos decidir cuál es el mayor
            if (n1 > n2) {
                System.out.println(n1 + " es mayor que " + n2);
            } else {
                System.out.println(n2 + " es mayor que " + n1);
            }
        }
    }
}

```

Actividad resuelta 2.7

Pedir tres números y mostrarlos ordenados de mayor a menor.

Solución

```

import java.util.Scanner;
/* Supondremos que todos los números introducidos por teclado son distintos.
 * Para el caso de números iguales solo hay que utilizar el operador >=
 * Vamos a plantear tantos condicionales como casos existen con tres números. */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a, b, c; //números a ordenar

```

```
System.out.print("Introduzca primer número: ");
a = sc.nextInt();
System.out.print("Introduzca segundo número: ");
b = sc.nextInt();
System.out.print("Introduzca tercer número: ");
c = sc.nextInt();
if (a > b && b > c) {
    System.out.println(a + ", " + b + ", " + c);
} else if (a > c && c > b) {
    System.out.println(a + ", " + c + ", " + b);
} else if (b > a && a > c) {
    System.out.println(b + ", " + a + ", " + c);
} else if (b > c && c > a) {
    System.out.println(b + ", " + c + ", " + a);
} else if (c > a && a > b) {
    System.out.println(c + ", " + a + ", " + b);
} else if (c > b && b > a) {
    System.out.println(c + ", " + b + ", " + a);
}
}
```

Actividad resuelta 2.8

Pedir los coeficientes de una ecuación de segundo grado y mostrar sus soluciones reales. Si no existen, habrá que indicarlo. Hay que tener en cuenta que las soluciones de una ecuación de segundo grado, $ax^2 + bx + c = 0$, son:

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

Solución

```

import java.util.*;
/* Para calcular las soluciones de una ecuación de segundo grado hay que aplicar
 * una sencilla fórmula. El único inconveniente es comprobar que no existan
 * divisiones por 0 o que no calculemos la raíz cuadrada de un número negativo.
 * Estos errores producen una parada de la ejecución del programa. */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        sc.useLocale(Locale.US);
        double a, b, c; // coeficientes  $ax^2 + bx + c = 0$ 
        double x1, x2, d; // soluciones y discriminante.
        System.out.print("Introduzca primer coeficiente (a): ");
        a = sc.nextDouble();
        System.out.print("Introduzca segundo coeficiente: (b): ");
        b = sc.nextDouble();
        System.out.print("Introduzca tercer coeficiente: (c): ");
        c = sc.nextDouble();
        // calculamos el discriminante
        d = (b * b - 4 * a * c);
        if (d < 0) { // como hay que calcular la raíz cuadrada de d, este no
            // puede ser negativo
    }
}

```

```
        System.out.println("No existen soluciones reales");
    } else {
        // si a=0 se produce una división por cero. Y en este caso, ni siquiera
        //sería una ecuación de 2º grado
        if (a == 0) {//si a es igual a 0
            System.out.println("No es una ecuación de segundo grado");
        } else {
            x1 = (-b + Math.sqrt(d))/(2 * a); //sqrt() calcula la raíz cuadrada
            x2 = (-b - Math.sqrt(d))/(2 * a);
            System.out.println("Solución 1: " + x1);
            System.out.println("Solución 2: " + x2);
        }
    }
}
```

Actividad resuelta 2.9

Escribir una aplicación que indique cuántas cifras tiene un número entero introducido por teclado, que estará comprendido entre 0 y 99 999.

Solución

```
import java.util.Scanner;
/* Los números comprendidos entre 0 y 9, inclusives, tienen una cifra.
 * Los números comprendidos entre 10 y 99, inclusives, tienen 2 cifras.
 * Los números comprendidos entre 100 y 999, inclusives, tienen 3 cifras.
 * Etc. */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número entre 0 y 99.999: ");
        int num = sc.nextInt();
        if (num < 10) {
            System.out.println("Tiene 1 cifra");
        } else if (num < 100) {
            System.out.println("Tiene 2 cifras");
        } else if (num < 1000) {
            System.out.println("Tiene 3 cifras");
        } else if (num < 10000) {
            System.out.println("Tiene 4 cifras");
        } else if (num < 100000) {
            System.out.println("Tiene 5 cifras");
        }
    }
}
```

El algoritmo propuesto en la Actividad resuelta 2.9 está pensado para trabajar con números positivos. Si deseamos saber cuántas cifras tiene un número negativo, habrá que modificarlo, ya que, por ejemplo, el número -23 , al ser menor que 10 , lo consideraría de una única cifra, cuando en realidad tiene dos.

Sería un buen ejercicio modificar la actividad para que indique cuántas cifras tiene un número comprendido entre -99 999 y 99 999.

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.

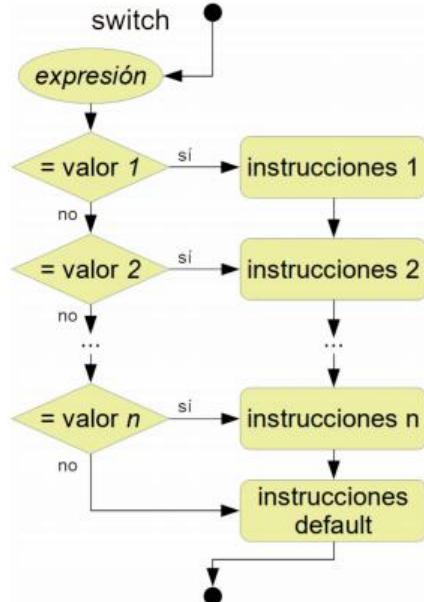


Figura 2.3. Funcionamiento de la instrucción `switch`.

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;
}

```

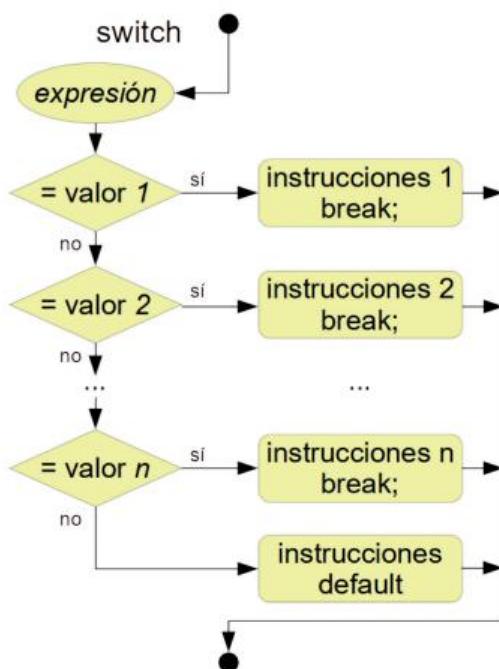


Figura 2.4. Funcionamiento de la instrucción `switch` con `break`.

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
```

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("Enhорabuena");
    }
    default ->
        System.out.println("Nota incorrecta");
}
```

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 `días` 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 días = 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`.

Nota técnica



Si NetBeans no es capaz de ejecutar un programa donde hemos usado la instrucción `switch` con `->`, es posible que estemos usando alguna versión antigua de Java. Se recomienda actualizar la versión de Java e incluso la de NetBeans.

Actividad resuelta 2.10

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).

Solución a)

```

import java.util.Scanner;
/* Hay alumnos que no están muy de acuerdo con esta clasificación de las notas,
 * y toda calificación mayor que 3 debe ser Notable. :-) */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca una nota: ");
        int nota = sc.nextInt();
        if (0 <= nota && nota < 5) { //se podría utilizar 0<=nota && nota <=4
            System.out.println("Insuficiente");
        } else if (nota == 5) {
            System.out.println("Suficiente");
        } else if (nota == 6) {
            System.out.println("Bien");
        } else if (nota == 7 || nota == 8) { //si nota es 7 u 8
            System.out.println("Notable");
        } else if (nota == 9 || nota == 10) {
            System.out.println("Sobresaliente");
        }
    }
}

```

```
        } else if (nota == 9 || nota == 10) { //si nota es 9 o 10
            System.out.println("Sobresaliente");
        } else {
            System.out.println("Error: nota no válida");
        }
    }
}
```

Solución b)

```
import java.util.Scanner;
/*Vamos a resolver el ejercicio utilizando una estructura switch en lugar de if's
 *anidados.*/
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca una nota: ");
        int nota = sc.nextInt();
        switch (nota) {
            case 0, 1, 2, 3, 4 ->
                System.out.println("Insuficiente");
            case 5 ->
                System.out.println("Suficiente");
            case 6 ->
                System.out.println("Bien");
            case 7, 8 ->
                System.out.println("Notable");
            case 9, 10 ->
                System.out.println("Sobresaliente");
            default ->
                System.out.println("Error: nota no válida");
        }
    }
}
```

Actividad resuelta 2.11

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».

Solución

```
import java.util.Scanner;
// Utilizaremos switch para distinguir las distintas alternativas.
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número de 1 a 7: ");
        int dia = sc.nextInt();
        switch (dia) {
            case 1 -> System.out.println("lunes");
            case 2 -> System.out.println("martes");
            case 3 -> System.out.println("miércoles");
            case 4 -> System.out.println("jueves");
            case 5 -> System.out.println("viernes");
            case 6 -> System.out.println("sábado");
            case 7 -> System.out.println("domingo");
        }
    }
}
```

```
        case 3 -> System.out.println("miércoles");
        case 4 -> System.out.println("jueves");
        case 5 -> System.out.println("viernes");
        case 6 -> System.out.println("sábado");
        case 7 -> System.out.println("domingo");
    }
}
```

Actividad resuelta 2.12

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 considerará los años bisiestos).

Solución

```
import java.util.Scanner;
// Hay que considerar que no todos los meses tienen el mismo número de días.
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int dia, mes, año;
        boolean fechaCorrecta; //variable que indica si la fecha es correcta.
        System.out.print("Introduzca día: ");
        dia = sc.nextInt();
        System.out.print("Introduzca mes: ");
        mes = sc.nextInt();
        System.out.print("Introduzca año: ");
        año = sc.nextInt();
        if (año == 0) { // el único año que no existe es el 0
            fechaCorrecta = false;
        } else {
            // primero comprobaremos febrero (mes = 2)
            if (mes == 2 && (1 <= dia && dia <= 28)) {
                fechaCorrecta = true;
            } else // veremos si es un mes de 30 días
            if ((mes == 4 || mes == 6 || mes == 9 || mes == 11)
                && (1 <= dia && dia <= 30)) {
                fechaCorrecta = true;
            } else { // comprobaremos si es un mes de 31 días
                if ((mes == 1 || mes == 3 || mes == 5 || mes == 7 || mes == 8 ||
                    mes == 10 || mes == 12)
                    && (1 <= dia && dia <= 31)) {
                    fechaCorrecta = true;
                } else { //en cualquier otro caso
                    fechaCorrecta = false;
                }
            }
            if (fechaCorrecta) {
                System.out.println("Fecha OK: " + dia + "/" + mes + "/" + año);
            } else {
                System.out.println("Fecha incorrecta");
            }
        }
    }
}
```

Actividad resuelta 2.13

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]

Solución

```
import java.util.Scanner;
/* Suponemos que la hora introducida por el usuario es correcta. El algoritmo
 * incrementa los segundos en 1. Esto puede hacer que salgan del rango 0..59, en
 * este caso, pondremos los segundos a 0 e incrementaremos los minutos.
 * Igualmente tenemos que comprobar que los minutos no se salgan de rango. E igual
 * para las horas. */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int h, m, s; // horas , minutos y segundos
        System.out.print("Introduzca hora: ");
        h = sc.nextInt();
        System.out.print("Introduzca minutos: ");
        m = sc.nextInt();
        System.out.print("Introduzca segundos: ");
        s = sc.nextInt();
        s++; // incrementamos los segundos
        if (s > 59) { //si los segundos superan 59
            s = 0; //los reiniciamos a 0
            m++; //e incrementamos los minutos
            if (m > 59) { //si los minutos superan 59,
                m = 0; //los reiniciamos
                h++; //e incrementamos la hora
                if (h > 23) { //si la hora supera 23
                    h = 0; //reiniciamos la hora a 0
                }
            }
        }
        System.out.println("Hora + 1 segundo: " + h + ":" + m + ":" + s);
    }
}
```

Actividad resuelta 2.14

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

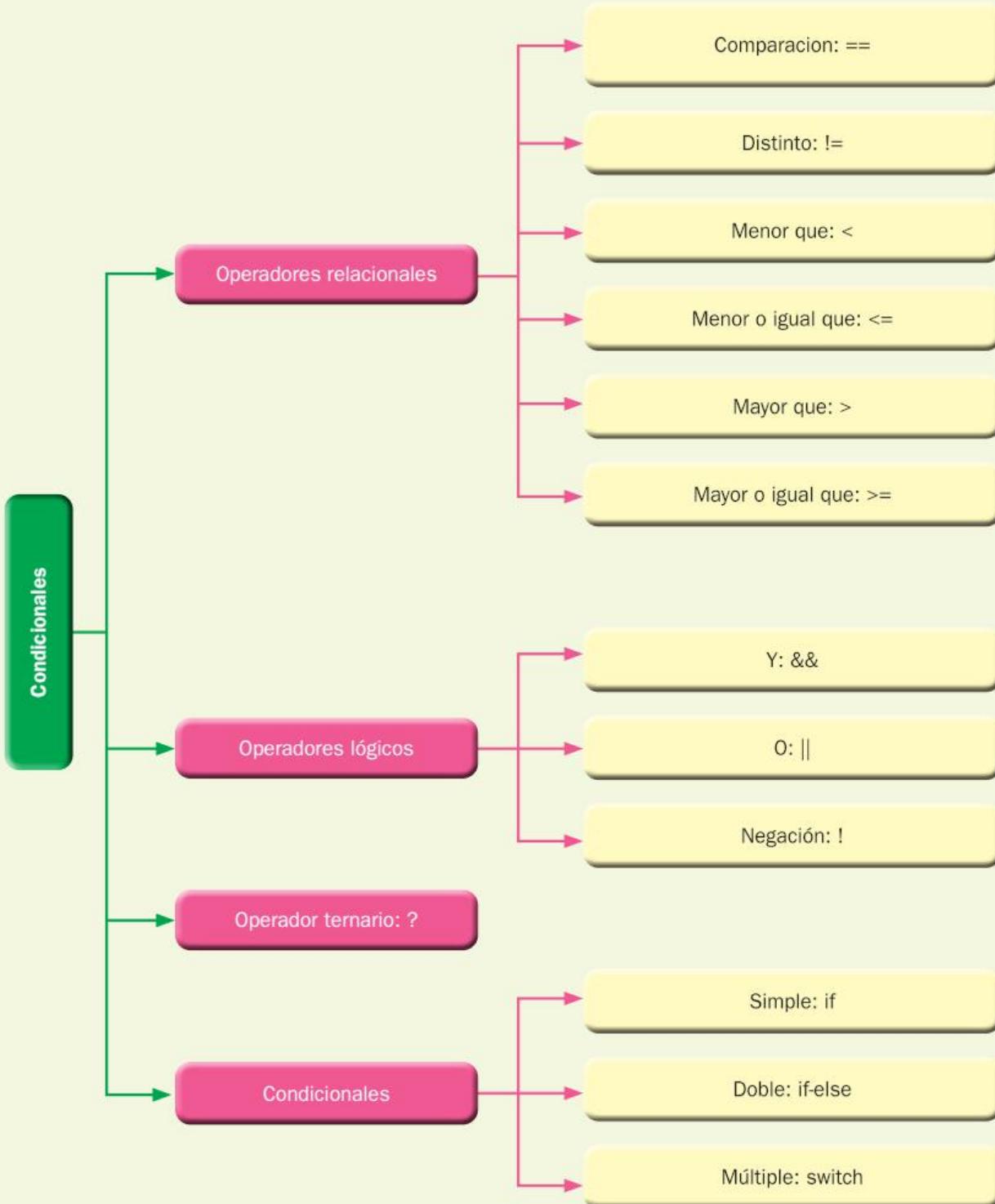
Solución

```
import java.util.Scanner;
/* Similar al anterior, en el que incrementábamos la hora. En este caso
 * la dificultad es que no todos los meses tienen el mismo número de días. Por eso,
 * lo primero que haremos es ver cuántos días tiene el mes de la fecha.
 * No tendremos en cuenta los años bisiestos y suponemos correcta la
 * fecha introducida. */
public class Main {
```

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int diasDelMes=0; // Aquí guardaremos el número de días que tiene el mes  
    System.out.print("Introduzca día: ");  
    int dia = sc.nextInt();  
    System.out.print("Introduzca mes: ");  
    int mes = sc.nextInt();  
    System.out.print("Introduzca año: ");  
    int año = sc.nextInt();  
    // suponemos que la fecha introducida es correcta  
    diasDelMes = switch(mes) {  
        case 2 -> 28; //si hay una sola instrucción se puede obviar yield y {}  
        case 4, 6, 9, 11 -> 30;  
        default -> 31;  
    };  
    dia++; // incrementamos el día  
    if (dia > diasDelMes) { //si dia supera el número de días del mes  
        dia = 1; //reiniciamos día a 1  
        mes++; //e incrementamos el mes  
        if (mes > 12) { // si mes supera 12  
            mes = 1; //lo reiniciamos a 1  
            año++; //e incrementamos el año  
        }  
    }  
    //El año -1 pasó al año +1. El año 0 nunca existió.  
    // Para evitar que el año pase del -1 al 0  
    if (año == 0) {  
        año = 1;  
    }  
    System.out.println(dia + "/" + mes + "/" + año);  
}
```

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.



Actividades de comprobación

- 2.1.** Los operadores lógicos operan con valores booleanos, resultando:
- a) Valores enteros.
 - b) Valores enteros y booleanos.
 - c) Otros tipos de valores.
 - d) Solo valores booleanos.
- 2.2.** La evaluación de una expresión relacional puede generar un valor de tipo:
- a) Entero.
 - b) Real.
 - c) Booleano.
 - d) Todos los anteriores.
- 2.3.** La expresión `3==3 && 2<3 && 1!=2` resulta:
- a) Cierto.
 - b) Falso.
 - c) No se puede evaluar.
 - d) No genera un booleano, ya que la expresión es aritmética.
- 2.4.** La siguiente expresión, donde interviene la variable booleana `a: 3!=3 || a || 1<2`, resulta:
- a) Dependerá del valor `a`.
 - b) Cierto.
 - c) Falso.
 - d) No se puede evaluar.
- 2.5.** Elige los valores de las variables enteras (`a, b` y `c`) que permiten que la evaluación de la siguiente expresión sea cierta: `a<b && b!=c && b<=c`:
- a) $a = 1, b = 1, c = 2$.
 - b) $a = 2, b = 1, c = 2$.
 - c) $a = 1, b = 2, c = 2$.
 - d) $a = 1, b = 2, c = 3$.
- 2.6.** El bloque de instrucciones de una sentencia `if` se ejecutará:
- a) Siempre.
 - b) Nunca.
 - c) Dependerá de la evaluación de la expresión utilizada.
 - d) Todas las respuestas anteriores son correctas.
- 2.7.** En una sentencia `if-else` los bloques de instrucciones (bloque `true` y bloque `false`) pueden ejecutarse:
- a) Simultáneamente.
 - b) Es posible, dependiendo de la condición utilizada, que no se ejecute ninguno.
 - c) Siempre se ejecutarán al menos uno y son excluyentes.
 - d) Todas las anteriores son incorrectas.

2.8. ¿Qué valor toma la variable `a` en la siguiente expresión: `a = 1<2 ? 3:4;`

- a) 1.
- b) 2.
- c) 3.
- d) 4.

2.9. La cláusula `default` en la sentencia `switch` es:

- a) Obligatoria y tiene que ser la última que aparezca.
- b) Obligatoria, pero puede aparecer en cualquier lugar.
- c) Opcional y tiene que ser la última que aparezca.
- d) Opcional y puede usarse en cualquier lugar.

2.10. Realiza una traza del siguiente fragmento de código y selecciona el valor que toma finalmente la variable `a`:

```
a = 0;
switch a+1 {
    case 0:
        a = 2;
    case 1:
        a = 3;
    case 2:
        a++;
        break;
    case 3:
        a--;
        break;
}
```

- a) 1
- b) 2.
- c) 3.
- d) 4.

Actividades de aplicación

2.11. Escribe una aplicación que solicite al usuario un número comprendido entre 0 y 9999. La aplicación tendrá que indicar si el número introducido es capicúa.

2.12. El DNI consta de un entero de 8 dígitos seguido de una letra que se obtiene a partir del número de la siguiente forma:

$$\text{letra} = \text{número DNI} \bmod 22$$

Basándote en esta información, elige la letra a partir de la numeración de la siguiente tabla:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

y diseña una aplicación en la que, dado un número de DNI, calcule la letra que le corresponde. Observa que un número de 8 dígitos está dentro del rango del tipo `int`.

- 2.13.** En una granja se compra diariamente una cantidad (`comidaDiaria`) de comida para los animales. El número de animales que alimentar (todos de la misma especie) es `numAnimales`, y sabemos que cada animal come una media de `kilosPorAnimal`.

Diseña un programa que solicite al usuario los valores anteriores y determine si disponemos de alimento suficiente para cada animal. En caso negativo, ha de calcular cuál es la ración que corresponde a cada uno de los animales.

Nota: Evitar que la aplicación realice divisiones por cero.

- 2.14.** Escribe un programa que solicite al usuario un número comprendido entre 1 y 99. El programa debe mostrarlo con letras, por ejemplo, para 56, se verá: «cincuenta y seis».

- 2.15.** Escribe una aplicación que solicite por consola dos números reales que corresponden a la base y la altura de un triángulo. Deberá mostrarse su área, comprobando que los números introducidos por el usuario no son negativos, algo que no tendría sentido.

- 2.16.** Utiliza el operador ternario para calcular el valor absoluto de un número que se solicita al usuario por teclado.

- 2.17.** Realiza el «juego de la suma», que consiste en que aparezcan dos números aleatorios (comprendidos entre 1 y 99) que el usuario tiene que sumar. La aplicación debe indicar si el resultado de la operación es correcto o incorrecto.

- 2.18.** Modifica la Actividad de aplicación 2.17 para que, además de los dos números aleatorios, también aparezca la operación que debe realizar el jugador: suma, resta o multiplicación.

- 2.19.** Crea una aplicación que solicite al usuario cuántos grados tiene un ángulo y muestre el equivalente en radianes. Si el ángulo introducido por el usuario no se encuentra en el rango de 0° a 360° , hay que transformarlo a dicho rango.

Nota: El operador módulo puede ayudarnos a convertir un ángulo a su equivalente en el rango comprendido de 0° a 360° .

Actividades de ampliación

- 2.20.** Busca en internet información sobre los operadores bit a bit y de desplazamiento que implementa Java. ¿Para qué serían útiles? Justifica tu respuesta.

- 2.21.** Realiza una investigación sobre George Boole, el padre del álgebra de Boole, en la que se basa todo el funcionamiento de los ordenadores.

- 2.22.** Además de las estructuras condicionales vistas en esta unidad, existen otras utilizadas en otros lenguajes de programación. Haz una búsqueda de cuáles son dichas estructuras condicionales.

- 2.23.** Como se ha visto, la comparación de valores de tipo double puede acarrear problemas debido a los pequeños errores de precisión que produce un ordenador al efectuar operaciones con decimales. Investiga cuáles son los métodos habituales para poder realizar comparaciones de números reales en un lenguaje de programación.