Actividad resuelta 3.1

Diseñar un programa que muestre, para cada número introducido por teclado, si es par, si es positivo y su cuadrado. El proceso se repetirá hasta que el número introducido sea 0.

Actividad resuelta 3.2

Implementar una aplicación para calcular datos estadísticos de las edades de los alumnos de un centro educativo. Se introducirán datos hasta que uno de ellos sea negativo, y se mostrará: la suma de todas las edades introducidas, la media, el número de alumnos y cuántos son mayores de edad.

Actividad resuelta 3.3

Codificar el juego «el número secreto», que consiste en acertar un número entre 1 y 100 (generado aleatoriamente). Para ello se introduce por teclado una serie de números, para los que se indica: «mayor» o «menor», según sea mayor o menor con respecto al número secreto. El proceso termina cuando el usuario acierta o cuando se rinde (introduciendo un –1).

Actividad resuelta 3.4

Un centro de investigación de la flora urbana necesita una aplicación que muestre cuál es el árbol más alto. Para ello se introducirá por teclado la altura (en centímetros) de cada árbol (terminando la introducción de datos cuando se utilice –1 como altura). Los árboles se identifican mediante etiquetas con números únicos correlativos, comenzando en 0. Diseñar una aplicación que resuelva el problema planteado.

Actividad resuelta 3.5

Desarrollar un juego que ayude a mejorar el cálculo mental de la suma. El jugador tendrá que introducir la solución de la suma de dos números aleatorios comprendidos entre 1 y 100. Mientras la solución introducida sea correcta, el juego continuará. En caso contrario, el programa terminará y mostrará el número de operaciones realizadas correctamente.

Actividad resuelta 3.6

Escribir una aplicación para aprender a contar, que pedirá un número *n* y mostrará todos los números del 1 a *n*.

Solución

Actividad resuelta 3.7

Escribir todos los múltiplos de 7 menores que 100.

Solución

```
/* Vamos a utilizar un bucle for, inicializando la i a 0, e iterando hasta que el valor
```

Actividad resuelta 3.8

Pedir diez números enteros por teclado y mostrar la media.

Solución

```
import java.util.Scanner;
/* Como tenemos claro que vamos a solicitar 10 números al usuario, utilizaremos
```

Actividad resuelta 3.9

Implementar una aplicación que pida al usuario un número comprendido entre 1 y 10. Hay que mostrar la tabla de multiplicar de dicho número, asegurándose de que el número introducido se encuentra en el rango establecido.

Solución

Actividad resuelta 3.10

Diseñar un programa que muestre la suma de los 10 primeros números impares.

Actividad resuelta 3.11

Pedir un número y calcular su factorial. Por ejemplo, el factorial de 5 se denota 5! y es igual a $5 \times 4 \times 3 \times 2 \times 1 = 120$.

Solución

Actividad resuelta 3.12

Pedir 5 calificaciones de alumnos y decir al final si hay algún suspenso.

Solución

- `break`: interrumpe completamente la ejecución del bucle.
- `continue`: detiene la iteración actual y continúa con la siguiente.

Cualquier programa puede escribirse sin utilizar `break` ni `continue`; se recomienda evitar los, ya que rompen la secuencia natural de las instrucciones. Veamos un ejemplo:

```
i = 1;
```

Actividad resuelta 3.13

Dadas 6 notas, escribir la cantidad de alumnos aprobados, condicionados (nota igual a cuatro) y suspensos.

Solución

Sin embargo, el uso descuidado de bucles anidados puede convertir un algoritmo en algo ineficiente, disparando el número de instrucciones ejecutadas.

Actividad resuelta 3.14

Diseñar una aplicación que muestre las tablas de multiplicar del 1 al 10.

Actividad resuelta 3.15

Pedir por consola un número *n* y dibujar un triángulo rectángulo de *n* elementos de lado, utilizando para ello asteriscos (*). Por ejemplo, para *n* = 4:

```
* * * *
* * *
* *
*
```

3.1. Un bucle `do-while` se ejecutará, como mínimo:

a) Cero veces.
b) Una vez.
c) Infinitas veces.
d) Ninguna de las opciones anteriores es correcta.

3.2. El uso de llaves para encerrar el bloque de instrucciones de un bucle:

a) Es siempre opcional.
b) Es opcional si el bloque está formado por una única instrucción.
c) En cualquier caso, su uso es obligatorio.
d) El programador decide su uso.

3.3. La instrucción que permite detener completamente las iteraciones de un bucle e

a) `stop`.
b) `break`.
c) `continue`.
d) `finish`.

3.4. La instrucción que permite detener la iteración actual de un bucle, continuando c la siguiente, si procede, es:

a) `stop`.
b) `break`.
c) `continue`.
d) `finish`.

3.5. De un bucle `do-while`, cuya condición depende de una serie de variables que en bloque de instrucciones no se modifican, se puede afirmar:

a) Que su número de iteraciones será siempre una.
b) Que el número de iteraciones será siempre par.
c) Que las variables cambiarán automáticamente en cualquier momento.
d) Ninguna de las opciones anteriores es correcta.

3.6. **¿Cuántas veces se ejecutará el bloque de instrucciones del bucle más interno en el siguiente fragmento de código?**

```
for(i=1; i<=10; i++) {
   for(i=1; i<=5; i++) {
      System.out.println("Hola");
   }
}
```

a) 10 veces.
b) 5 veces.
c) 50 veces.
d) Infinitas veces.

3.7. **Analiza el siguiente código y busca qué valores de `a` y `b` implican un menor número de iteraciones:**

```
for (int i=a; i<=a+b; i++) {
   for(int j=a+b; j>=0; j--) {
      ...
   }
}
```

a) a=1 y b=3.
b) a=3 y b=1.
c) a=1 y b=1.
d) a=3 y b=3.

3.8. **En cada iteración, el incremento de un bucle `for` se ejecuta:**

a) En primer lugar.
b) Después de la inicialización.
c) Después de evaluar la condición.
d) Justo al finalizar cada iteración.

3.9. **Una variable que se declara dentro de su bloque de instrucciones solo se podrá utilizar:**

a) En cualquier parte del programa.
b) En todos los bucles.
c) Dentro del bloque de instrucciones donde se ha declarado.
d) Todas las opciones anteriores son correctas.

3.10. **En un bucle `for`, la inicialización, condición e incremento son:**

a) Todos obligatorios.
b) Todos opcionales.
c) La inicialización siempre es obligatoria.
d) La condición siempre es obligatoria.

3.11. Realiza un programa que convierta un número decimal en su representación binaria. Hay que tener en cuenta que desconocemos cuántas cifras tiene el número que introduce el usuario.

Por simplicidad, iremos mostrando el número binario con un dígito por línea.

3.12. Modifica la Actividad de aplicación 3.11 para que el usuario pueda introducir un número en binario y el programa muestre su conversión a decimal.

3.13. Escribe un programa que incremente la hora de un reloj. Se pedirán por teclado la hora, minutos y segundos, así como cuántos segundos se desea incrementar la hora introducida. La aplicación mostrará la nueva hora. Por ejemplo, si las 13:59:51 se incrementan en 10 segundos, resultan las 14:00:01.

3.14. Realiza un programa que nos pida un número *n*, y nos diga cuántos números hay entre 1 y *n* que sean primos. Un número primo es aquel que solo es divisible por 1 y por él mismo. Veamos un ejemplo para $n = 8$:

Comprobamos todos los números del 1 al 8

- 1 → primo
- 2 → primo
- 3 → primo
- 4 → no primo
- 5 → primo
- 6 → no primo
- 7 → primo
- 8 → no primo

Resultan un total de 5 números primos.

3.15. Diseña una aplicación que dibuje el triángulo de Pascal, para *n* filas. Numerando las filas y elementos desde 0, la fórmula para obtener el *m*-ésimo elemento de la *n*-ésima fila es:

$$E(n, m) = \frac{n!}{m!(n-m)!}$$

Donde *n*! es el factorial de *n*.
Un ejemplo de triángulo de Pascal con 5 filas ($n = 4$) es:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

3.16. Solicita al usuario un número *n* y dibuja un triángulo de base y altura *n*, de la forma (para *n* igual a 4):

```
   *
  * *
 * * *
* * * *
```

3.17. Para dos números dados, *a* y *b*, es posible buscar el máximo común divisor (el número más grande que divide a ambos) mediante un algoritmo ineficiente pero sencillo: desde el menor de *a* y *b*, ir buscando, de forma decreciente, el primer número que divide a ambos simultáneamente. Realiza un programa que calcule el máximo común divisor de dos números.

3.18. De forma similar a la Actividad de aplicación 3.17, implementa un algoritmo que calcule el mínimo común múltiplo de dos números dados.

3.19. Calcula la raíz cuadrada de un número natural mediante aproximaciones. En el caso de que no sea exacta, muestra el resto. Por ejemplo, para calcular la raíz cuadrada de 23, probamos $1^2 = 1$, $2^2 = 4$, $3^2 = 9$, $4^2 = 16$, $5^2 = 25$ (nos pasamos), resultando 4 la raíz cuadrada de 23 con un resto (23 – 16) de 7.

© Ediciones Paraninfo

3.20. Escribe un programa que solicite al usuario las distintas cantidades de dinero de las que dispone. Por ejemplo: la cantidad de dinero que tiene en el banco, en una hucha, en un cajón y en los bolsillos. La aplicación mostrará la suma total de dinero de la que dispone el usuario.

La manera de especificar que no se desea introducir más cantidades es mediante el cero.