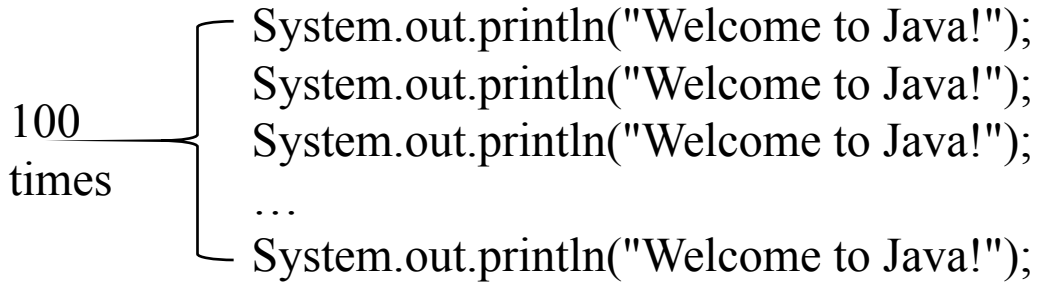
Capítulo 4

Bucles

Motivación

Supongamos que necesitamos imprimir una cadena (por ejemplo, "Bienvenido a Java!")
Cientos de veces. Sería tedioso tener que escribir la siguiente declaración cien veces:

```
System.out.println("Bienvenido a Java!")
```



Entonces, ¿cómo resolver este problema?

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java");
  count++;
}
```

Objetivos

- Escribir programas para ejecutar sentencias repetidamente utilizando un bucle while.
- Seguir la estrategia de diseño de bucle para desarrollar bucles.
- Controlar un bucle con un valor centinela.
- Obtener una entrada grande de un archivo mediante la redirección de entrada en lugar de escribir desde el teclado.
- Escribir bucles usando declaraciones do-while.
- Escribir bucles usando sentencias for.
- Descubrir las similitudes y diferencias de tres tipos de enunciados de bucle.
- Escribir bucles anidados.
- Aprender las técnicas para minimizar errores numéricos.
- Aprender bucles de una variedad de ejemplos .
- Implementar el control de programa con interrupción y continuar.

Bucle while

- Nos permite **repetir** la ejecución de un bloque de sentencias **un número indeterminado de iteraciones**.

- No necesitamos saber a priori cuántas veces se va a repetir el bucle. Se utiliza una expresión booleana que, mientras sea cierta, nos mantiene en el bucle.

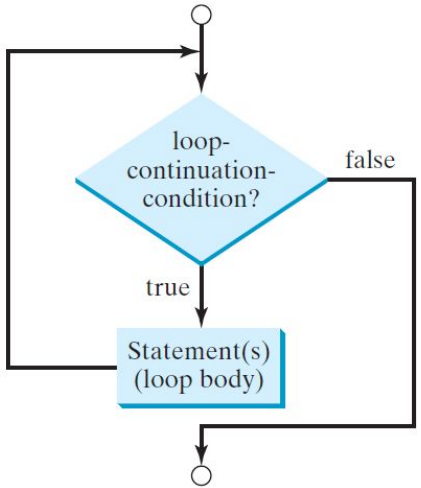
- Una de las sentencias del cuerpo del bucle debe modificar algo para que la expresión booleana, en algún momento, se evalúe como false. Las variables de la expresión de control se deben declarar e inicializar antes del bucle.

- Sintaxis

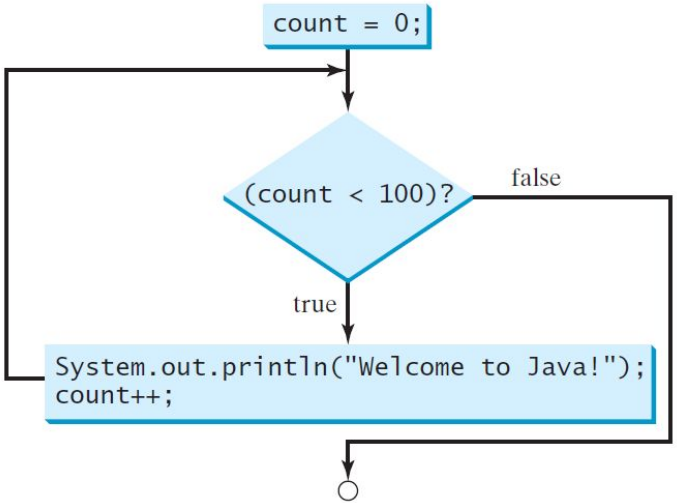
```
while(condición) {
// acciones a ejecutar
}
```

Diagrama de flujo del bucle `while`

```
while (condición del bucle) {
  // cuerpo de bucle;
  Declaración(es);
}
```




```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
```



Seguimiento de while

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Inicializar contador

Seguimiento de while

(count < 2) es true

```
int count = 0;

while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Seguimiento de while

Muestra Welcome to Java

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Seguimiento de while

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Incrementa count en 1
count ahora es 1

Seguimiento de while

```
int count = 0;

while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) sigue siendo cierto
count es 1

Seguimiento de while

Muestra Welcome to Java

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Seguimiento de while

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Incrementa count en 1
count ahora es 2

Seguimiento de while

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) es falso ya que count es 2 ahora

Seguimiento de while

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

El bucle termina. Ejecuta la siguiente instrucción después del bucle.

Problemas: Repetir hasta condición correcta

EJEMPLO1: Realizar un programa que pida al usuario que escriba una respuesta para una pregunta en la suma de dos dígitos individuales. El programa debe permitir que el usuario ingrese una nueva respuesta hasta que sea correcta.

EJEMPLO2: Escriba un programa que genere aleatoriamente un número entero entre 0 y 10, inclusive. El programa le pide al usuario que introduzca un número continuamente hasta que el número coincida con el número generado aleatoriamente. Para cada entrada de usuario, el programa le dice al usuario si la entrada es demasiado baja o demasiado alta, para que el usuario pueda elegir la siguiente entrada de forma inteligente.

EJEMPLO3: Realizar un programa de aprendizaje de sustracción matemática que genere cinco preguntas para cada ejecución e informa del número de respuestas correctas después de que un estudiante responda a las cinco preguntas.

Tipo de la condición de finalización

No utilice valores de coma flotante para verificar la igualdad en un control de bucle. Dado que los valores de punto flotante son aproximaciones para algunos valores, usarlos podría resultar en valores de contador imprecisos y resultados inexactos.

Considere el siguiente código para calcular 1 + 0.9 + 0.8 + ... + 0.1:

```
double item = 1; double sum = 0;
while (item != 0) { // No garantiza que item será 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```

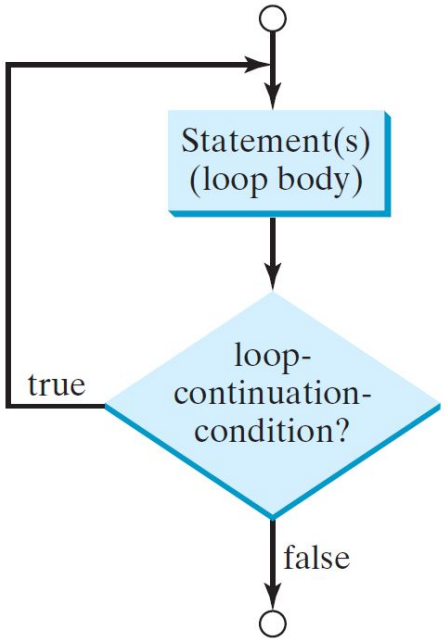
Bucle `do-while`

- Nos permite **repetir** la ejecución de un bloque de sentencias **un número indeterminado de iteraciones**.
- **Variante con respecto al bucle while: se garantiza que al menos se ejecuta una iteración**.
- Se utiliza una expresión booleana que, mientras sea cierta, nos mantiene en el bucle.
- Una de las sentencias del cuerpo del bucle debe modificar algo para que la expresión booleana, en algún momento, se evalúe como false.
- Las variables de la expresión de control se deben declarar e inicializar antes del bucle.

Bucle `do-while`

Sintaxis

```
do {
      // Loop body;
      // Acciones a ejecutar
} while (condición de continuidad);
```



Ejemplo uso del Bucle `do-while`

- Una aplicación muy usual de los bucles do-while es la de implementar menús o solicitar información al usuario para validarla.

- Mientras no introduzca la información correcta, la volvemos a solicitar.

- Ejemplo: menú con varias opciones. Salimos del programa si pulsamos un 0.

Bucle `for`

- Nos permite **repetir** la ejecución de un bloque de sentencias **un número determinado de iteraciones**, que se conoce **a priori**.

- **Sintaxis**

```
for (declaración/inicialización; condición;
incremento/decremento) {
    // acciones a ejecutar
}
```

- **declaración/inicialización**:
  - Se ejecuta una única vez, al inicio del bucle.
  - Sirve para declarar y/o inicializar una (o más) variable que nos servirá para gestionar el bucle.
  - Si se declara la variable aquí, no tiene vida más allá del bucle.

Bucle `for`

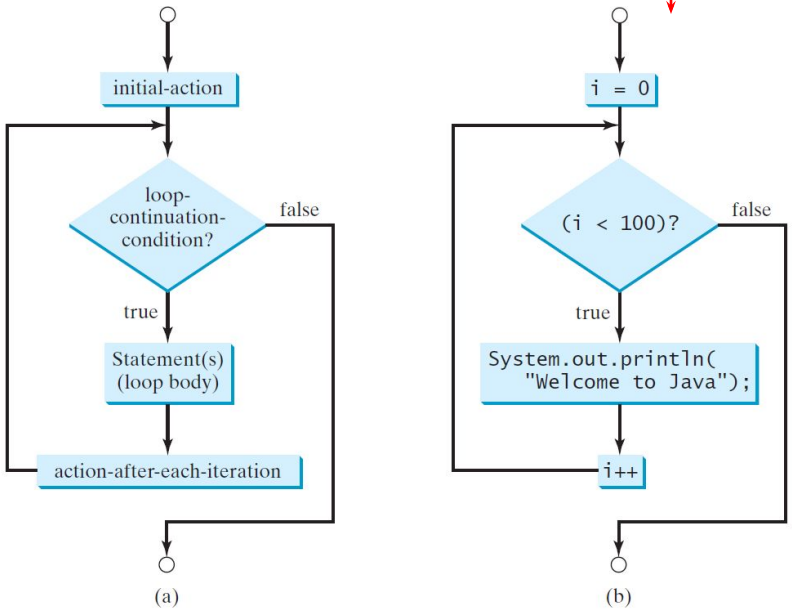
- **condición**:
  - Es una expresión booleana.
  - Sirve para indicar si debemos continuar en el bucle o salir de él.
  - Condición de continuación: si es true, se vuelve a realizar una iteración más del bucle.

- **incremento/decremento**:
  - Se ejecuta para cada iteración, después de ejecutar las acciones del cuerpo del bucle.
  - Sirve para incrementar o decrementar las variables de control (declaradas o inicializadas en el propio bucle).
  - Se puede incrementar o decrementar en más de una unidad.

Bucles `for`

```
for (Inicializa-accion; condición continuidad
      del bucle; iteración) {
      // cuerpo de bucle;
      Declaracion(es);
}
```

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println("Welcome to Java!");
}
```



Seguimiento del bucle for

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

Declara i

Seguimiento del bucle for

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

Ejecutar inicializador
i ahora vale 0

Seguimiento del bucle for

(i < 2) es true
Ya que i es 0

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println( "Welcome to Java!");
}
```

Seguimiento del bucle for

Muestra Welcome to Java

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Seguimiento del bucle for

Ejecuta la sentencia de ajuste
i ahora es1

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Seguimiento del bucle for

(i < 2) sigue siendo true
ya que i es 1

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Seguimiento del bucle for

Muestra Welcome to Java

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Seguimiento del bucle for

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Ejecuta la sentencia de ajuste
i ahora es 2

Seguimiento del bucle for

(i < 2) es false
ya que i es 2

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Seguimiento del bucle for

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Termina el bucle. Ejecutar la siguiente instrucción después del bucle

Nota

La acción inicial en un bucle for puede ser una lista de cero o más expresiones separadas por comas. La acción después de cada iteración en un bucle for puede ser una lista de cero o más sentencias separadas por comas. Por lo tanto, los dos bucles siguientes son correctos. Sin embargo, rara vez se utilizan en la práctica.

```
for (int i = 1; i < 100; System.out.println(i++));
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
 // Do something
}
```

Si se omite la condición de continuidad de bucle en un bucle for, es implícitamente verdadera. Por lo tanto, la afirmación dada en (a), que es un bucle infinito, es correcta. Sin embargo, es mejor usar el bucle equivalente en (b):

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

Errores comunes

Agregar un punto y coma al final de la cláusula for antes del cuerpo de bucle es un error común, como se muestra a continuación:

```
for (int i=0; i<10; i++);
{
  System.out.println("i es " + i);
}
```

Correcto

De manera similar, el siguiente bucle también es incorrecto:

Error Logico

```
int i=0;
while (i < 10);
{
  System.out.println("i es " + i);
  i++;
}
```

En el caso del bucle do, el siguiente punto y coma es necesario para finalizar el bucle.

```
int i=0;
do {
  System.out.println("i is " + i);
  i++;
} while (i<10);
```

Correcto

¿Qué bucle utilizar?

Las tres formas de las declaraciones de bucle, while, do-while, y for, son equivalentes; es decir, puede escribir un bucle en cualquiera de estas tres formas. Por ejemplo, en la siguiente figura el bucle while en (a) siempre se puede convertir en el siguiente bucle for en (b):

```
while (condición de continuidad del bucle)
  // cuerpo del bucle
}
```

(a)

Equivalente

```
for ( ; condición de continuidad del bucle; )
  // cuerpo del bucle
}
```

(b)

Un bucle for en (a) en la siguiente figura generalmente se puede convertir en el siguiente bucle while en (b) excepto en ciertos casos especiales.

```
for (acción inicial;
     condición de continuidad del bucle;
     action-after-each-iteration) {
  // cuerpo del bucle;
}
```

(a)

Equivalente

```
acción inicial;
while (condición de continuidad del bucle)
  // cuerpo del bucle;
  acción-después-cada-iteración;
}
```

(b)

Recomendaciones

Utilice el que sea más intuitivo y cómodo para usted.

En general, se puede utilizar un bucle for si se conoce el número de repeticiones, como, por ejemplo, cuando se necesita imprimir un mensaje 100 veces.

Se puede utilizar un bucle while si no se conoce el número de repeticiones, como en el caso de leer los números hasta que la entrada sea 0.

Un bucle do-while se puede usar para reemplazar un bucle while si el cuerpo del bucle tiene que ser ejecutado antes de probar la condición de continuación.

Minimizar Errores Numéricos

Los errores numéricos que involucran números de coma flotante son inevitables. En esta sección se explica cómo minimizar tales errores mediante un ejemplo.

He aquí un ejemplo que suma una serie que comienza con 0,01 y termina con 1,0. Los números de la serie se incrementarán en 0.01, de la siguiente manera: 0.01 + 0.02 + 0.03 y así sucesivamente.

Uso de break y continue

Dos instrucciones que nos sirven para salir del bucle o saltar una iteración.
Su uso no es muy recomendable: Difícil de depurar errores.
Es preferible articular el bucle o su organización de otra forma.

BREAK

- Nos permite salir del bucle “en cualquier momento”.
- Suele ir asociado a un bloque if.
- Ejemplo: identificar si un número es primo, buscando sus divisores.

CONTINUE

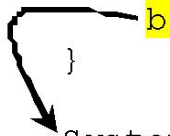
- Nos permite terminar una iteración sin finalizar todas sus sentencias, y continuar con la siguiente iteración.

Ejemplo uso de break

```java
package ejemplo;
import java.util.Scanner;

public class EjemploBreak {
    public static void main(String[] args) {
        var scanner = new Scanner(System.in);
        System.out.print("Introduce un número entero positivo: ");
        int numero = scanner.nextInt();
        boolean esPrimo = true;

        // El número 1 es primo, ya que tiene un único divisor
        if (numero < 2) {
            esPrimo = false;
        }

        for (int i = 2; i < numero; i++) {
            if (numero % i == 0) {
                esPrimo = false;
                break;
            }
        }

        if (esPrimo) {
            System.out.println(numero + " es un número primo.");
        } else {
            System.out.println(numero + " no es un número primo.");
        }
        scanner.close();
    }
}
```

Ejemplo uso de break

```
public class TestBreak {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      sum += number;
      if (sum >= 100)
        break;
    }

    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
  }
}
```



Ejemplo uso de continue

Programa que sume los números del 1 al 10, menos el 7.

```
package ejemplo;

public class EjemploContinue {
    public static void main(String[] args) {
        int suma = 0;
        for (int i = 1; i <= 10; i++) {
            if (i == 7) {
                continue;
            }
            suma += i;
        }
        System.out.println("La suma de los números del 1 al 10, excepto el 7, es: " + suma);
    }
}
```

Ejemplo uso de continue

```
public class TestContinue {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }

    System.out.println("The sum is " + sum);
  }
}
```

Problemas

Problema1: Encontrar el mayor divisor común.
Escriba un programa que pida al usuario que introduzca dos enteros positivos y encuentre su máximo divisor común.

Solución: Supongamos que introduce dos números enteros 4 y 2, cuyo máximo divisor común es 2. Supongamos que introduce dos números enteros 16 y 24, su máximo divisor común es 8. Entonces, ¿cómo encontrar el máximo divisor común? Sea los dos enteros de entrada n1 y n2. Usted sabe que el número 1 es un divisor común, pero puede no ser el divisor común más grande. Así que puede comprobar si k(for k = 2, 3, 4, y así sucesivamente) es un divisor común para n1 y n2, hasta que k sea mayor que n1 o n2.

Problemas

Problema2: Supongamos que la matrícula para una universidad es de 1,000€ este año y la matrícula aumenta 7% cada año. ¿En cuántos años se duplicará la matrícula??

Fundamentación de la matrícula Futura:

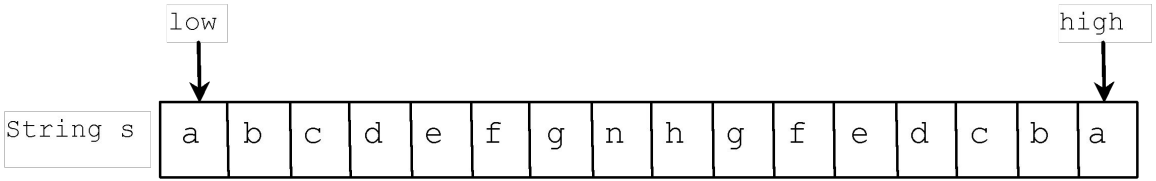
```
double matricula = 1000;   int año= 0     // Año 0
matricula = matricula * 1.07; año++;      // Año 1
matricula = matricula * 1.07; año++;      // Año 2
matricula = matricula * 1.07; año++;      // Año 3
...
```

Problemas

PROBLEMA3 Comprobación del palíndromo

Una cadena es un palíndromo si se lee igual hacia adelante y hacia atrás. Las palabras "radar", "oso" y “reconocer", por ejemplo, son todos palíndromos.

El problema es escribir un programa que solicite al usuario que ingrese una cadena e informe si la cadena es un palíndromo. Una solución es verificar si el primer carácter de la cadena es el mismo que el último. Si es así, compruebe si el segundo carácter es el mismo que el penúltimo carácter. Este proceso continúa hasta que se encuentra una discordancia o se verifican todos los caracteres de la cadena, excepto el carácter medio si la cadena tiene un número impar de caracteres.



Problemas

PROBLEMA4 Mostrar números primos

Escriba un programa que muestre los 50 primeros números primos en cinco líneas, cada una de los cuales contiene 10 números. Un entero mayor que 1 es primo si su único divisor positivo es 1 o él mismo. Por ejemplo, 2, 3, 5 y 7 son números primos, pero 4, 6, 8 y 9 no lo son.

Solución: el problema se puede dividir en las siguientes tareas:

- Para number = 2, 3, 4, 5, 6, ..., probar si number es primo.
- Determinar si un número determinado es primo.
- Contar los números primos.
- Imprimir cada número primo e imprimir 10 números por línea.