Unidad 3

Uso de estructuras de control

Contenidos

- Parte 1: Estructuras de Repetición 1 3
- Parte 2: Estructuras de Repetición 2 9
- Parte 3: Estructuras de Salto 15
- Conclusión del Bloque: Uso de estructuras de control 22

Parte 1: Estructuras de Repetición 1

Las estructuras de control son fundamentales en cualquier lenguaje de programación, ya que permiten alterar el flujo de ejecución de un programa. En Java, estas estructuras se clasifican en **selección**, **repetición** y **salto**.

Este bloque abordará en profundidad cada tipo de estructura, proporcionando ejemplos detallados, ejercicios prácticos, y un test para evaluar el aprendizaje.

1. Estructuras de Selección

Las estructuras de selección se utilizan para ejecutar diferentes bloques de código dependiendo de ciertas condiciones. En Java, estas incluyen:

1. `if`
2. `if-else`
3. `else if`
4. `switch`

1.1. Estructura `if`

La estructura `if` evalúa una condición booleana y ejecuta un bloque de código si la condición es verdadera.

Sintaxis

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
}
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          int numero = 10;
          if (numero > 0) {
                System.out.println("El número es positivo.");
          }
    }
}
```

Salida:

```
El número es positivo.
```

2. Estructura `if-else`

El `if-else` se utiliza para definir una alternativa si la condición del `if` no se cumple.

Sintaxis

```
if (condición) {
    // Código si la condición es verdadera
} else {
    // Código si la condición es falsa
}
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          int numero = -5;
          if (numero > 0) {
                System.out.println("El número es positivo.");
          } else {
                System.out.println("El número es negativo o cero.");
          }
    }
}
```

Salida:

```
El número es negativo o cero.
```

3. Estructura `else if`

La estructura `else if` permite evaluar múltiples condiciones en secuencia.

Sintaxis

```
if (condición1) {
    // Código si condición1 es verdadera
} else if (condición2) {
    // Código si condición2 es verdadera
} else {
    // Código si ninguna condición es verdadera
}
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          int numero = 0;
          if (numero > 0) {
                System.out.println(“El número es positivo.”);
          } else if (numero < 0) {
                System.out.println(“El número es negativo.”);
          } else {
                System.out.println(“El número es cero.”);
          }
    }
}
```

Salida:

```
El número es cero.
```

4. Estructura `switch`

El `switch` evalúa una expresión y ejecuta el bloque de código correspondiente a su valor.

Sintaxis

```
switch (expresión) {
    case valor1:
          // Código si expresión == valor1
          break;
    case valor2:
          // Código si expresión == valor2
          break;
    default:
          // Código si ninguno de los casos se cumple
          break;
}
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          int dia = 3;
          switch (dia) {
                case 1:
                      System.out.println(“Lunes”);
                      break;
                case 2:
                      System.out.println(“Martes”);
                      break;
                case 3:
                      System.out.println(“Miércoles”);
                      break;
                default:
                      System.out.println(“Día no válido”);
          }
    }
}
```

Salida:

```
Miércoles
```

5. Ejercicios Prácticos

Ejercicio 1: Comparar dos números

Escribe un programa que:

1. Lea dos números enteros del usuario.
2. Use un `if-else` para determinar cuál es mayor, menor o si son iguales.

Ejercicio 2: Clasificación de edades

Crea un programa que:

1. Lea la edad de una persona.
2. Use `else if` para clasificarla en las categorías:

- Niño (< 12 años).
- Adolescente (12-17 años).
- Adulto (18-64 años).
- Mayor (65+ años).

Ejercicio 3: Calculadora con `switch`

Crea una calculadora que:

1. Lea dos números y una operación (`+`, , , `/`).
2. Use `switch` para realizar la operación y mostrar el resultado.

6. Test de Evaluación

1. ¿Qué estructura de selección se usa para evaluar múltiples condiciones?

a) `if`
b) `else if`
c) `switch`
d) Ninguna de las anteriores

2. ¿Qué palabra clave detiene la ejecución en un bloque `switch`?

a) `stop`
b) `halt`
c) `break`
d) `exit`

3. ¿Qué ocurre si no se incluye un `default` en un `switch`?

a) Se lanza un error de compilación.

b) No ocurre nada si no hay coincidencias.

c) Siempre se ejecuta el último caso.

d) El programa finaliza.

Parte 2: Estructuras de Repetición 2

Las estructuras de repetición permiten ejecutar un bloque de código varias veces, ya sea un número fijo de iteraciones o mientras una condición específica se cumpla. En Java, las estructuras de repetición principales son:

1. `for`: **Ideal para un número fijo de iteraciones.**
2. `while`: **Ejecuta el bloque mientras la condición sea verdadera.**
3. `do-while`: **Similar a `while`, pero garantiza al menos una ejecución del bloque.**

1. Estructura `for`

El bucle `for` se utiliza cuando el número de iteraciones es conocido de antemano.

Sintaxis

```
for (inicialización; condición; actualización) {
    // Código a ejecutar en cada iteración
}
```

1. **Inicialización: Se ejecuta una vez antes de que comience el bucle.**
2. **Condición: Se evalúa antes de cada iteración; si es `false`, el bucle termina.**
3. **Actualización: Se ejecuta al final de cada iteración.**

Ejemplo Básico

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.println("Iteración: " + i);
        }
    }
}
```

Salida:

```
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
```

1.1. Bucle `for` con Arrays

Puedes recorrer un array utilizando un bucle `for`.

Ejemplo

```
  public class Main {
      public static void main(String[] args) {
            int[] numeros = {10, 20, 30, 40, 50};
            for (int i = 0; i < numeros.length; i++) {
                  System.out.println("Elemento en índice " + i + ": " + numer-
os[i]);
            }
      }
  }
```

Salida:

```
  Elemento en índice 0: 10
  Elemento en índice 1: 20
  Elemento en índice 2: 30
  Elemento en índice 3: 40
  Elemento en índice 4: 50
```

1.2. Bucle `for-each`

El bucle `for-each` se utiliza para recorrer colecciones o arrays de manera más simplificada.

Sintaxis

```
  for (tipo elemento : colección) {
      // Código a ejecutar para cada elemento
  }
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
        int[] numeros = {10, 20, 30, 40, 50};
        for (int numero : numeros) {
            System.out.println("Número: " + numero);
        }
    }
}
```

Salida:

```
Número: 10
Número: 20
Número: 30
Número: 40
Número: 50
```

2. Estructura `while`

El bucle `while` se utiliza cuando no se conoce de antemano cuántas iteraciones se necesitarán. Ejecuta el bloque mientras la condición sea verdadera.

Sintaxis

```
while (condición) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

Ejemplo Básico

```
public class Main {
    public static void main(String[] args) {
        int contador = 0;
        while (contador < 5) {
            System.out.println("Contador: " + contador);
            contador++;
        }
    }
}
```

Salida:

```
Contador: 0
Contador: 1
Contador: 2
Contador: 3
Contador: 4
```

3. Estructura `do-while`

El bucle `do-while` es similar a `while`, pero garantiza que el bloque de código se ejecutará al menos una vez, ya que la condición se evalúa al final.

Sintaxis

```
do {
    // Código a ejecutar al menos una vez
} while (condición);
```

Ejemplo Básico

```
public class Main {
    public static void main(String[] args) {
          int contador = 0;
          do {
                System.out.println("Contador: " + contador);
                contador++;
          } while (contador < 5);
    }
}
```

Salida:

```
Contador: 0
Contador: 1
Contador: 2
Contador: 3
Contador: 4
```

4. Comparación de Estructuras de Repetición

Estructura	Cuándo Usarla	Características Clave
`for`	Cuando el número de iteraciones es conocido.	Inicialización, condición, y actualización en una línea.
`while`	Cuando las iteraciones dependen de una condición dinámica.	La condición se evalúa antes de cada iteración.
`do-while`	Cuando necesitas garantizar al menos una ejecución.	La condición se evalúa después del bloque.

5. Ejercicios Prácticos

Ejercicio 1: Suma de Números con `for`

1. Solicita al usuario un número `n`.
2. Usa un bucle `for` para calcular la suma de los números del 1 al `n`.

Ejercicio 2: Validar Entrada con `while`

1. Pide al usuario que introduzca un número positivo.
2. Usa un bucle `while` para repetir la solicitud hasta que se ingrese un número válido.

Ejercicio 3: Tablas de Multiplicar con `do-while`

1. Genera la tabla de multiplicar de un número ingresado por el usuario.
2. Usa un bucle `do-while` para garantizar al menos una ejecución.

6. Test de Evaluación

1. ¿Cuál de estas estructuras garantiza al menos una ejecución del bloque de código?

a) `for`
b) `while`
c) `do-while`
d) Ninguna

2. ¿Qué bucle usarías si conoces de antemano el número de iteraciones necesarias?

a) `for`
b) `while`
c) `do-while`
d) Todos son válidos

3. ¿Qué ocurre si la condición de un `while` nunca es falsa?

a) El programa lanza un error.
b) El bucle se ejecuta infinitamente.
c) El programa finaliza automáticamente.
d) Ninguna de las anteriores.

4. ¿Cuál es la diferencia entre un bucle for y `for-each`?

a) `for-each` requiere índices explícitos.
b) `for-each` solo funciona con arrays y colecciones.
c) `for` es más eficiente que `for-each`.
d) `for-each` no tiene diferencias significativas respecto al `for`.

Parte 3: Estructuras de Salto

Las **estructuras de salto** alteran directamente el flujo de ejecución en un programa, permitiendo salir de un bucle, continuar con la siguiente iteración o transferir el control a otra parte del código. En Java, las principales estructuras de salto son:

1. `break`: **Sale de un bucle o una estructura** `switch`.
2. `continue`: **Salta a la siguiente iteración de un bucle.**
3. `return`: **Finaliza la ejecución de un método y devuelve un valor opcional.**
4. `throw`: **Lanza una excepción y transfiere el control al bloque** `catch` **correspondiente.**

1. Estructura `break`

La instrucción `break` se utiliza para salir inmediatamente de un bucle o una estructura `switch`. Una vez ejecutado, el control del programa continúa con la instrucción que sigue al bucle o `switch`.

1.1. Uso en Bucles

Sintaxis

```
for (inicialización; condición; actualización) {
    if (condición_de_salida) {
        break; // Salir del bucle
    }
}
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i == 5) {
                break; // Termina el bucle al llegar a 5
            }
            System.out.println("Número: " + i);
        }
    }
}
```

Salida:

```
Número: 1
Número: 2
Número: 3
Número: 4
```

1.2. Uso en `switch`

El `break` se usa en estructuras `switch` para salir de un caso específico. Sin él, el programa ejecutará todos los casos siguientes, incluso si no coinciden.

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          int dia = 3;
          switch (dia) {
                case 1:
                      System.out.println("Lunes");
                      break;
                case 2:
                      System.out.println("Martes");
                      break;
                case 3:
                      System.out.println("Miércoles");
                      break;
                default:
                      System.out.println("Día no válido");
          }
    }
}
```

Salida:

```
Miércoles
```

2. Estructura `continue`

La instrucción `continue` permite saltar a la siguiente iteración de un bucle, omitiendo las instrucciones restantes en la iteración actual.

2.1. Uso en Bucles

Sintaxis

```
for (inicialización; condición; actualización) {
    if (condición_de_omisión) {
        continue; // Salta a la siguiente iteración
    }
    // Código que se omite si se ejecuta 'continue'
}
```

Ejemplo

```
public class Main {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i % 2 == 0) {
                continue; // Omite los números pares
            }
            System.out.println("Número impar: " + i);
        }
    }
}
```

Salida:

```
Número impar: 1
Número impar: 3
Número impar: 5
Número impar: 7
Número impar: 9
```

3. Estructura `return`

La instrucción `return` finaliza inmediatamente la ejecución de un método y puede devolver un valor opcional al método que lo llamó.

3.1. Uso en Métodos

Sintaxis

```
public int sumar(int a, int b) {
    return a + b; // Devuelve la suma de a y b
}
```

Ejemplo

```
java
public class Main {
    public static void main(String[] args) {
        int resultado = sumar(5, 10);
        System.out.println("Resultado: " + resultado);
    }
    public static int sumar(int a, int b) {
        return a + b; // Finaliza y devuelve el resultado
    }
}
```

Salida:

```
Resultado: 15
```

3.2. Uso para Salir de Métodos sin Retorno

En métodos `void`, el `return` se utiliza para terminar la ejecución del método.

Ejemplo

```
public class Main {
    public static void main(String[] args) {
          imprimirNumero(5);
          imprimirNumero(-3);
    }
    public static void imprimirNumero(int numero) {
          if (numero < 0) {
                System.out.println("Número negativo, no se procesa.");
                return; // Termina el método
          }
          System.out.println("Número: " + numero);
    }
}
```

Salida:

```
Número: 5
Número negativo, no se procesa.
```

4. Estructura `throw`

La instrucción `throw` se utiliza para lanzar excepciones, lo que transfiere el control al bloque `catch` correspondiente. Aunque no es una estructura de salto estándar, su funcionalidad es relevante para el flujo de control.

Ejemplo

```
  public class Main {
      public static void main(String[] args) {
            try {
                  dividir(10, 0);
            } catch (ArithmeticException e) {
                  System.out.println("Error: " + e.getMessage());
            }
      }
      public static void dividir(int a, int b) {
            if (b == 0) {
                  throw new ArithmeticException("No se puede dividir entre
cero");
            }
            System.out.println("Resultado: " + (a / b));
      }
  }
```

Salida:

```
  Error: No se puede dividir entre cero
```

5. Ejercicios Prácticos

Ejercicio 1: Salir de un bucle con `break`

Crea un programa que lea números del usuario y se detenga cuando el usuario introduzca `-1`.

Ejercicio 2: Saltar iteraciones con `continue`

Escribe un programa que imprima los números del 1 al 20, pero omita los múltiplos de 3.

Ejercicio 3: Uso de `return`

Crea un método que reciba un número y devuelva `true` si es par, o `false` en caso contrario.

Ejercicio 4: Lanzar Excepciones con `throw`

Diseña un programa que simule un cajero automático. Si el usuario intenta retirar más dinero del saldo disponible, lanza una excepción personalizada `SaldoInsuficienteException`.

6. Test de Evaluación

1. ¿Qué hace la instrucción `break`?

a) Salta a la siguiente iteración del bucle.
b) Finaliza el programa.
c) Sale de un bucle o estructura `switch`.
d) Lanza una excepción.

2. ¿Cuál es la principal diferencia entre `break` y `continue`?

a) `break` finaliza el bucle; `continue` omite el resto de la iteración actual.
b) `break` solo funciona en `switch`; `continue` en bucles.
c) `break` lanza una excepción; `continue` no.
d) No hay diferencias.

3. ¿Qué ocurre si se usa `return` en un método `void`?

a) Lanza un error de compilación.
b) Termina el método inmediatamente.
c) Devuelve un valor predeterminado.
d) Nada.

4. ¿Qué palabra clave se usa para lanzar una excepción?

a) `throw`
b) `catch`
c) `break`
d) `return`

Conclusión del Bloque: Uso de estructuras de control

Las **estructuras de control** son fundamentales para cualquier lenguaje de programación, ya que permiten dirigir el flujo de ejecución del código. Este bloque ha cubierto en detalle las **estructuras de selección**, **estructuras de repetición**, y **estructuras de salto**, con ejemplos prácticos, ejercicios y una guía completa para entender sus usos y aplicaciones.

1. Puntos Clave del Bloque

1. Estructuras de Selección

- **Permiten tomar decisiones en el programa**:
  - El `if` evalúa una condición para ejecutar un bloque de código si es verdadera.
  - El `if-else` ofrece una alternativa en caso de que la condición sea falsa.
  - El `else if` permite evaluar múltiples condiciones de forma secuencial.
  - El `switch` simplifica la selección de múltiples casos en base al valor de una expresión.

Estas estructuras son ideales para manejar escenarios donde el programa necesita tomar decisiones lógicas en tiempo de ejecución.

2. Estructuras de Repetición

- **Facilitan la ejecución de bloques de código múltiples veces**:
  - El `for` es ideal para iteraciones conocidas de antemano, como recorrer arrays o listas.
  - El `while` se utiliza cuando el número de iteraciones depende de una condición dinámica.
  - El `do-while` garantiza al menos una ejecución del bloque de código, incluso si la condición es falsa desde el inicio.

Estas estructuras optimizan el manejo de datos iterativos y permiten realizar tareas repetitivas de manera eficiente.

3. Estructuras de Salto

- **Alteran el flujo de ejecución del programa**:
  - El `break` permite salir inmediatamente de un bucle o un `switch`.
  - El `continue` salta a la siguiente iteración de un bucle, omitiendo el resto del bloque actual.
  - El `return` finaliza un método y devuelve un valor opcional.
  - El `throw` permite manejar errores y excepciones lanzando mensajes controlados.

Estas estructuras proporcionan flexibilidad y control adicional, mejorando la precisión en la ejecución del programa.

2. Importancia de las Estructuras de Control

El dominio de las estructuras de control es esencial para cualquier desarrollador. Permiten:

1. **Controlar el flujo del programa:**
   - Decidir qué acciones tomar, cuántas veces ejecutarlas, y cuándo detenerlas.
2. **Optimizar la lógica:**
   - Reducir código redundante mediante el uso de bucles y condiciones.
3. **Mejorar la legibilidad:**
   - Facilitar la comprensión y mantenimiento del código al estructurar lógicamente las decisiones y repeticiones.
4. **Manejar errores y excepciones:**
   - Prevenir fallos en tiempo de ejecución y proporcionar mensajes claros al usuario.

3. Ejercicio Final del Bloque

Objetivo

Diseñar un programa completo que utilice todas las estructuras de control vistas en este bloque.

Problema

Crea un programa para un sistema de gestión de calificaciones que permita:

1. **Registrar calificaciones de estudiantes.**
2. **Determinar el promedio de las calificaciones.**
3. **Mostrar un mensaje basado en el promedio:**
   - Excelente (90+)
   - Bueno (70-89)
   - Regular (50-69)
   - Insuficiente (<50)
4. **Repetir el proceso hasta que el usuario decida salir.**

Solución

Clase Principal

```
import java.util.Scanner;
public class SistemaCalificaciones {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean continuar = true;
        while (continuar) {
            System.out.println("Introduce el número de estudiantes:");
            int numEstudiantes = scanner.nextInt();
            int[] calificaciones = new int[numEstudiantes];
            // Registrar calificaciones
            for (int i = 0; i < numEstudiantes; i++) {
                System.out.println("Introduce la calificación del estudiante " + (i + 1) + ":");
                calificaciones[i] = scanner.nextInt();
            }
            // Calcular promedio
            int suma = 0;
            for (int calificacion : calificaciones) {
                suma += calificacion;
            }
            double promedio = (double) suma / numEstudiantes;
            // Determinar nivel
            String nivel;
            if (promedio >= 90) {
                nivel = "Excelente";
            } else if (promedio >= 70) {
                nivel = "Bueno";
            } else if (promedio >= 50) {
                nivel = "Regular";
            } else {
                nivel = "Insuficiente";
            }
            System.out.println("Promedio del grupo: " + promedio);
            System.out.println("Nivel: " + nivel);
            // Preguntar si desea continuar
            System.out.println("¿Deseas registrar otro grupo? (sí/no):");
            scanner.nextLine(); // Limpiar buffer
            String respuesta = scanner.nextLine();
```

```
                continuar = respuesta.equalsIgnoreCase("sí");
            }
            System.out.println("¡Gracias por usar el sistema de califica-
ciones!");
            scanner.close();
        }
    }
```

Salida Esperada

```
Introduce el número de estudiantes:
3
Introduce la calificación del estudiante 1:
85
Introduce la calificación del estudiante 2:
90
Introduce la calificación del estudiante 3:
78
Promedio del grupo: 84.33
Nivel: Bueno
¿Deseas registrar otro grupo? (sí/no):
no
¡Gracias por usar el sistema de calificaciones!
```

4. Test de Evaluación

1. ¿Qué estructura de control utilizarías para iterar sobre un array?

a) `if`
b) `switch`
c) `for`
d) `break`

2. ¿Cuál de las siguientes instrucciones detiene inmediatamente un bucle?

a) `continue`
b) `break`
c) `return`
d) `switch`

3. ¿Qué estructura garantiza al menos una ejecución del bloque de código?

a) `for`
b) `while`
c) `do-while`
d) Ninguna de las anteriores

4. ¿Qué palabra clave se utiliza para finalizar un método y devolver un valor?

a) `break`
b) `continue`
c) `return`
d) `throw`

5. ¿Qué ocurre si un `switch` no tiene un caso coincidente y no incluye `default`?

a) Lanza un error en tiempo de ejecución.
b) Se ejecuta el último caso del `switch`.
c) No ocurre nada; se omite el `switch`.
d) El programa finaliza.