CLASE 5. Estructuras Iterativas

Estructuras Algorítmicas Básicas

En la programación, las estructuras algorítmicas permiten definir el flujo de ejecución de un programa, es decir, el orden y la forma en que las instrucciones se ejecutan. Una estructura algorítmica básica es el conjunto de instrucciones que permite a un programa tomar decisiones y repetir bloques de código según ciertos criterios. Las estructuras de control son esenciales para la creación de programas dinámicos y flexibles.

¿Qué son las estructuras de control?

Las estructuras de control son herramientas fundamentales que permiten modificar el flujo de ejecución de un programa. Sin ellas, un programa seguiría un flujo lineal, ejecutando las instrucciones en el orden en que aparecen. Las estructuras de control permiten tomar decisiones (condicionales) o repetir un conjunto de instrucciones (bucles), lo que incrementa la versatilidad de los programas.

Tipos de Estructuras de Control

Las estructuras de control se pueden clasificar en tres grandes grupos: secuenciales, selectivas o condicionales, e iterativas. Cada una tiene una funcionalidad específica dentro de un programa y se utilizan en diferentes escenarios según la lógica requerida.

1. Estructuras Secuenciales

Una estructura secuencial es la forma más básica de control de flujo. En este tipo de estructura, las instrucciones se ejecutan una tras otra, de manera ordenada, en el mismo orden en que están escritas. Es ideal para tareas que no requieren decisiones ni repeticiones.

2. Estructuras Selectivas o Condicionales

Las estructuras selectivas permiten al programa tomar decisiones basadas en condiciones. Según el resultado de una evaluación lógica (verdadero o falso), el programa puede ejecutar uno u otro bloque de código.

Tipos de Estructuras Selectivas:

- **Alternativa Simple (Si-Entonces / If-Then):**
  La estructura más sencilla, donde una condición se evalúa y, si es verdadera, se ejecuta un bloque de código. Si es falsa, no se ejecuta nada.

```
if (condición) {
    // Bloque de código si la condición es
verdadera
}
```

- **Alternativa Doble (Si-Entonces-Sino / If-Then-Else):**
  A diferencia de la alternativa simple, esta permite definir un segundo bloque de código que se ejecutará si la condición es falsa.

```
if (condición) {
    // Bloque de código si la condición es
verdadera
} else {
```

```
    // Bloque de código si la condición es falsa
}
```

- **Estructuras de Decisión Anidadas (En Escalera):**
  En este tipo de estructuras, múltiples condiciones se evalúan en secuencia. Si una condición es verdadera, se ejecuta el bloque correspondiente y se omiten las siguientes evaluaciones.

```
if (condición1) {
    // Bloque de código para la condición1
} else if (condición2) {
    // Bloque de código para la condición2
} else {
    // Bloque de código si ninguna condición es
verdadera
}
```

- **Alternativa Múltiple (Segun-Sea, Caso de / Switch-Case):**
  Esta estructura se usa cuando se desea comparar el valor de una variable con múltiples casos predefinidos. Es útil cuando se tiene una lista fija de condiciones posibles.

```
switch (expresión) {
    case valor1:
        // Bloque de código para el caso valor1
        break;
    case valor2:
        // Bloque de código para el caso valor2
        break;
    default:
        // Bloque de código si ningún caso
coincide
}
```

3. Estructuras Iterativas o Repetitivas

Las estructuras iterativas permiten que un bloque de código se ejecute repetidamente mientras una condición sea verdadera o hasta que se cumpla una determinada condición. Son esenciales para tareas repetitivas como recorrer elementos de una lista o realizar cálculos iterativos.

Concepto de Bucle e Iteración:

Un bucle (o iteración) es una estructura que permite repetir un conjunto de instrucciones varias veces, según una condición o un contador.

Tipos de Estructuras Iterativas

1. Estructura `while`

La estructura `while` ejecuta un bloque de código mientras una condición sea verdadera. La condición se evalúa antes de cada iteración, lo que significa que si la condición es falsa desde el principio, el bloque de código no se ejecutará ni una sola vez.

Sintaxis en Java:

```
while (condición) {
    // Bloque de código que se ejecuta mientras la
condición sea verdadera
}
```

Ejemplo:

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

2. Estructura `do-while`

La estructura `do-while` es similar a la `while`, con la diferencia de que primero se ejecuta el bloque de código y luego se evalúa la condición. Esto garantiza que el bloque de código se ejecute al menos una vez, incluso si la condición es falsa desde el principio.

Sintaxis en Java:

```
do {
    // Bloque de código
} while (condición);
```

Ejemplo:

```
int i = 0;
do {
    System.out.println(i);
    i++;
} while (i < 5);
```

3. Estructura `for`

La estructura `for` es útil cuando se conoce de antemano cuántas veces debe repetirse un bloque de código. Se define un contador, una condición y una actualización en una única línea.

Sintaxis en Java:

```
for (inicialización; condición; actualización) {
    // Bloque de código que se repite mientras la
condición sea verdadera
}
```

Ejemplo:

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

Bucle `for` Anidado

Un bucle `for` anidado es un bucle dentro de otro bucle. Son útiles para recorrer matrices bidimensionales u otros conjuntos de datos complejos.

Ejemplo:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        System.out.println("i = " + i + ", j = " + j);
    }
}
```

Bucles Infinitos

Un bucle infinito es aquel que nunca termina de ejecutarse porque la condición siempre es verdadera o porque se omite la actualización del contador. Los bucles infinitos pueden causar problemas de rendimiento y deben evitarse a menos que se implementen intencionalmente (por ejemplo, en servidores que esperan conexiones continuamente).

Ejemplo de Bucle Infinito:

```
while (true) {
    System.out.println("Este bucle nunca termina.");
}
```

Las estructuras iterativas y selectivas son herramientas fundamentales en la programación. Dominar su uso es esencial para crear algoritmos eficientes y efectivos. Ya sea que estemos repitiendo tareas con bucles o tomando decisiones con estructuras condicionales, estas herramientas permiten que los programas respondan dinámicamente a las necesidades del usuario y a las condiciones específicas del entorno de ejecución.