Flow Control in Java

Definition

> The **sequential order** in which Java statements are executed at runtime by the **JVM** is known as **Flow Control**.

- **Controlling Statements** manipulate the execution path of a program.

Categories of Flow Control Statements

1. **Selection Statements**
    - `if-else`
    - `else-if` (also known as `if-else-if-else` ladder)
    - `switch`
2. **Iterative Statements**
    - `for` loop
    - `while` loop
    - `do-while` loop
3. **Transfer Statements**
    - `break`
    - `continue`

Selection Statements

Selection statements execute **one out of multiple possible blocks** based on conditions.

`if-else`

- Executes one of two blocks based on a **Boolean condition**.

Syntax:

```java
class IfElse {
    public static void main(String[] args) {
        int x = 56;
        System.out.print(x + " is ");
        if (x % 2 == 0) {
            System.out.println("Even.");
        } else {
            System.out.println("Odd.");
        }
    }
}
```

Notes:

- The `if` block is **mandatory**; the `else` block is **optional**.
- An `else` block **cannot exist independently**.
- No code is allowed between `if` and `else`.
- Braces `{}` can be omitted **only if** the block contains exactly one non-declarative statement.

✅ `int x = 10; if (x == 10) x = 20;`
❌ `if (true) int x = 10;`

`else-if` Ladder

- Allows for **multiple condition checks**, where **only one** matching block is executed.

Syntax:

```java
class IfElseIfElseLadder {
    public static void main(String[] args) {
        int a = 5, b = 8, c = 7;
        if (a > b && a > c)
            System.out.print(a);
        else if (b > a && b > c)
            System.out.print(b);
        else if (c > a && c > b)
            System.out.print(c);

        System.out.println(" is the greatest of " + a + ", " + b + ", " + c + ".");
    }
}
```

`switch` Statement

Syntax:

```java
class SwitchCase {
    public static void main(String[] args) {
        int x = 4;
        switch (x) {
            case 1:
                System.out.println("case1");
                x = 10;
                break;
            case 2:
                System.out.println("case2");
                x = 20;
                break;
            case 3:
                System.out.println("case3");
                x = 30;
                break;
            case 4:
                System.out.println("case4"); // case4
                x = 40;
                break;
            default:
                System.out.println("defaultcase");
                x = 99;
```

```java
                    break;
            }
            System.out.println(x); // 40
        }
}
```

Rules for Case Labels:

1. **Duplicates** are not allowed.
2. Must be **constants**.
3. Must be within the **range** of the switch variable.
4. The `default` case:
    - Can appear **anywhere** inside the switch.
    - Must be followed by a `break` to avoid fall-through.
    - Is **optional**.
    - Cannot be duplicated.

Fall-Through Example:

```java
class FallThrough {
    public static void main(String[] args) {
        int n = 2;
        switch (n) {
            case 1: System.out.println("1"); break;
            case 2: System.out.println("2");
            case 3: System.out.println("3");
            case 4: System.out.println("4"); break;
            case 5: System.out.println("5"); break;
            default: System.out.println("d");
        }
    }
}
// Output:
// 2
// 3
// 4
```

Iterative Statements

`for` Loop

- Best used when the **exact number of iterations is known**.

Syntax:

```java
for (initialization; condition; update) {
    // loop body
}
```

Example:

```java
class ForLoop {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 1; i < 11; i++) {
            sum += i;
        }
        System.out.println("Sum = " + sum); // 55
    }
}
```

`while` Loop

- Used when the **number of iterations is unknown**.

Syntax:

```java
while (condition) {
    // loop body
}
```

Example:

```java
class WhileLoop {
    public static void main(String[] args) {
        int sum = 0, i = 1;
        while (i <= 10) {
            sum += i;
            i++;
        }
        System.out.println("Sum = " + sum); // 55
    }
}
```

`do-while` Loop

- Guarantees that the loop **executes at least once**.

Syntax:

```java
do {
    // loop body
} while (condition);
```

Example 1:

```java
class DoWhileLoop {
    public static void main(String[] args) {
        int sum = 0, i = 1;
        do {
            sum += i;
            i++;
        } while (i <= 10);
        System.out.println("Sum = " + sum); // 55
    }
}
```

Example 2:

```java
class DoWhileLoopFalse {
    public static void main(String[] args) {
        do {
            System.out.println("Hi");
        } while (false);
        // Output: Hi (executed once despite false condition)
    }
}
```

Transfer Statements

These control the **flow within loops or switch blocks** by either **exiting** or **skipping** iterations.

`break`

- Terminates the **nearest loop** or **switch block**.

`continue`

- Skips the **current iteration** and proceeds to the **next**.

Example:

```java
class BreakContinue {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i == 4)
                break;
            System.out.println(i);
        }
        System.out.println("4 - 10 not executed.");

        for (int i = 1; i <= 10; i++) {
            if (i == 4) {
                System.out.println("4 not executed.");
                continue;
            }
            System.out.println(i);
        }
     }
}
```