Chapter 5: Control Statements

Control statements in Java determine the flow of program execution. They can be categorized into selection statements, iteration statements, and jump statements.

---

1. Java's Selection Statements

Selection statements allow a program to choose between different paths of execution.

1.1 If Statement

- **Syntax**:
```
if (condition) {
    // Statements to execute if the condition is true
}
```
- **How it works**:
  The `if` statement evaluates a condition. If true, the code block inside the `if` is executed.

1.2 Nested Ifs

- **Definition**:
  An `if` statement inside another `if`.

- **Syntax**:

```
if (condition1) {
    if (condition2) {
        // Code if both conditions are true
    }
}
```
- **Use Case**: Useful for decision trees.

1.3 The If-Else-If Ladder

- **Definition**:
  A chain of `if-else` statements to handle multiple conditions.

- **Syntax**:

```
if (condition1) {
    // Code for condition1
} else if (condition2) {
    // Code for condition2
} else {
```

```
    // Code if no conditions are true
}
```

- **Purpose**: To test multiple conditions sequentially.

1.4 Switch Statement

- **Definition**:
  Used for selecting one of many blocks of code to execute.

- **Syntax**:

```
switch (expression) {
    case value1:
        // Code for value1
        break;
    case value2:
        // Code for value2
        break;
    default:
        // Default code
}
```

- **Key Points**:
  - `expression` must evaluate to `int`, `char`, `String`, or `enum`.
  - The `break` statement prevents fall-through.

1.5 Nested Switch Statements

- **Definition**:
  A `switch` inside another `switch`.

- **Syntax**:

```
switch (expression1) {
    case value1:
        switch (expression2) {
            case value2:
                // Nested switch code
                break;
        }
        break;
}
```

- **Use Case**: Useful for complex decision-making.

---

2. Iteration Statements

Iteration statements execute a block of code repeatedly.

2.1 While Loop

- **Definition**:
  Executes a block of code as long as a condition is true.

- **Syntax**:

```
while (condition) {
    // Code to execute
}
```

2.2 Do-While Loop

- **Definition**:
  Similar to `while`, but ensures the block is executed at least once.

- **Syntax**:

```
do {
    // Code to execute
} while (condition);
```

2.3 For Loop

- **Definition**:
  A compact way to iterate, typically when the number of iterations is known.

- **Syntax**:

```
for (initialization; condition; iteration) {
    // Code to execute
}
```

2.4 Declaring Loop Control Variables Inside the For Loop

- **Example**:

```
for (int i = 0; i < 10; i++) {
    // Code
}
```

- The variable `i` is only accessible within the loop.

2.5 Using the Comma

- **Definition**:
  Allows multiple variables to be initialized or updated in a loop.

- **Example**:

```
for (int i = 0, j = 10; i < j; i++, j--) {
    // Code
}
```

2.6 Some For Loop Variations

- **Infinite for loop**:
```
for (;;) {
    // Infinite loop
}
```
- **Loop with no initialization**:
```
int i = 0;
for (; i < 5; i++) {
    // Code
}
```

2.7 The For-Each Version of the For Loop

- **Definition**:
  Simplified loop for iterating over arrays or collections.

- **Syntax**:

```
for (type var : array) {
    // Code
}
```

2.8 Iterating Over Multidimensional Arrays

- **Example**:
```
int[][] arr = {{1, 2}, {3, 4}};
for (int[] row : arr) {
    for (int val : row) {
        // Code
    }
}
```

2.9 Applying the Enhanced For

- **Definition**:
  Simplifies iteration for collections like `ArrayList`.

2.10 Nested Loops

- **Definition**:
  A loop inside another loop.

- **Example**:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        // Nested loop code
    }
}
```

---

3. Jump Statements

Jump statements control the flow by transferring control to another part of the program.

3.1 Using Break

- **Definition**:
  Terminates the loop or `switch`.
- **Example**:

```
for (int i = 0; i < 10; i++) {
    if (i == 5) break;
}
```

3.2 Using Break to Exit a Loop

- **Purpose**: To exit from a loop immediately.

3.3 Using Break as a Form of Goto

- **Syntax**:

```
outer: for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (j == 2) break outer;
    }
}
```

3.4 Using Continue

- **Definition**:
  Skips the current iteration and continues with the next iteration.
- **Example**:

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) continue;
    // Code
}
```

3.5 Return Statement

- **Definition**:
  Exits from the current method and optionally returns a value.
- **Syntax**:

```
return; // for void methods
return value; // for methods with return type
```

Conclusion

Control statements are essential for determining the flow of a Java program. Mastering them helps in writing efficient and logical code for real-world applications.