Java Control Flow Statements: A Comprehensive Guide

1. Conditional Statements

1.1 if Statement

Theory

- Used to execute code block based on a boolean condition
- Allows decision-making in program flow
- Simplest form of conditional execution

Example

```
public class IfStatementDemo {
    public static void main(String[] args) {
        int age = 20;

        // Basic if statement
        if (age >= 18) {
            System.out.println("You are an adult");
        }

        // If-else statement
        if (age >= 18) {
            System.out.println("You can vote");
        } else {
            System.out.println("You cannot vote yet");
        }

        // Nested if statement
        if (age >= 18) {
            if (age >= 65) {
                System.out.println("Senior citizen");
            } else {
                System.out.println("Adult");
            }
        }
    }
}
```

1.2 switch Statement

Theory

- Allows testing a variable against multiple values
- Provides an alternative to multiple if-else blocks

- Supports different data types (byte, short, char, int, String, enum)

Example

```
public class SwitchStatementDemo {
    public static void main(String[] args) {
        // Traditional switch (Java 7 and earlier)
        int day = 3;
        switch (day) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            default:
                System.out.println("Another day");
        }

        // Enhanced switch (Java 12+)
        String season = "SUMMER";
        switch (season) {
            case "WINTER" -> System.out.println("It's cold");
            case "SUMMER" -> System.out.println("It's hot");
            case "SPRING", "AUTUMN" -> System.out.println("Mild weather");
            default -> System.out.println("Invalid season");
        }

        // Switch with yield (Java 14+)
        int numLetters = switch (season) {
            case "WINTER" -> 6;
            case "SUMMER" -> 6;
            case "SPRING" -> 6;
            case "AUTUMN" -> 6;
            default -> 0;
        };
    }
}
```

2. Looping Statements

2.1 for Loop

Theory

- Used for iterating a block of code a known number of times
- Consists of initialization, condition, and increment/decrement

Example

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

        // Enhanced for loop (for-each)
        int[] numbers = {1, 2, 3, 4, 5};
        for (int num : numbers) {
            System.out.println("Number: " + num);
        }

        // Nested for loop
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.println("i: " + i + ", j: " + j);
            }
        }
    }
}
```

2.2 while Loop

Theory

- Executes a block of code while a condition is true
- Condition is checked before each iteration
- Used when number of iterations is not known beforehand

Example

```
public class WhileLoopDemo {
    public static void main(String[] args) {
        // Standard while loop
        int count = 0;
        while (count < 5) {
            System.out.println("Count: " + count);
            count++;
        }

        // Infinite loop with break
        while (true) {
            System.out.println("Enter a number (0 to exit):");
            int input = new java.util.Scanner(System.in).nextInt();

            if (input == 0) {
```

```
                break;  // Exit the loop
            }

            System.out.println("You entered: " + input);
        }
    }
}
```

2.3 do-while Loop

Theory

- Similar to while loop, but guarantees at least one execution
- Condition is checked after the block of code is executed
- Useful when you want the code to run at least once

Example

```
public class DoWhileLoopDemo {
    public static void main(String[] args) {
        int x = 10;

        // do-while loop
        do {
            System.out.println("Value of x: " + x);
            x++;
        } while (x < 5);  // This will run once, even though condition is false

        // Practical example with user input
        java.util.Scanner scanner = new java.util.Scanner(System.in);
        int userInput;

        do {
            System.out.println("Enter a number between 1 and 10:");
            userInput = scanner.nextInt();
        } while (userInput < 1 || userInput > 10);

        System.out.println("Valid input received: " + userInput);
    }
}
```

3. Control Transfer Statements

3.1 break Statement

Theory

- Exits the current loop or switch statement
- Transfers control to the statement immediately following the loop or switch

Example

```
public class BreakStatementDemo {
    public static void main(String[] args) {
        // Breaking out of a loop
        for (int i = 0; i < 10; i++) {
            if (i == 5) {
                break;  // Exit loop when i is 5
            }
            System.out.println(i);
        }

        // Breaking out of nested loops
        outer: for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (i == 1 && j == 1) {
                    break outer;  // Breaks out of both loops
                }
                System.out.println("i: " + i + ", j: " + j);
            }
        }
    }
}
```

3.2 continue Statement

Theory

- Skips the current iteration of a loop
- Continues with the next iteration

Example

```
public class ContinueStatementDemo {
    public static void main(String[] args) {
        // Skipping even numbers
        for (int i = 0; i < 10; i++) {
            if (i % 2 == 0) {
                continue;  // Skip even numbers
            }
            System.out.println(i);
        }

        // Complex continue usage
        for (int i = 0; i < 10; i++) {
            if (i == 3 || i == 7) {
                continue;  // Skip specific iterations
            }
            System.out.println(i);
```

```
        }
    }
}
```

4. Advanced Control Flow

4.1 Ternary Operator

Theory

- Shorthand for if-else statement
- Provides a concise way to make decisions

Example

```
public class TernaryOperatorDemo {
    public static void main(String[] args) {
        int age = 20;

        // Ternary operator
        String status = (age >= 18) ? "Adult" : "Minor";
        System.out.println(status);

        // Nested ternary operator
        int x = 10, y = 20, z = 30;
        int max = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z);
        System.out.println("Maximum value: " + max);
    }
}
```

5. Best Practices

Control Flow Guidelines

1. Keep conditional logic simple and readable
2. Avoid deep nesting of conditional statements
3. Use switch statements for multiple conditions on a single variable
4. Prefer enhanced for-each loops when possible
5. Use break and continue judiciously
6. Consider readability over clever code

Common Pitfalls

- Infinite loops
- Unintended fall-through in switch statements
- Complex nested conditions
- Forgetting break in switch cases

Conclusion

Control flow statements are fundamental to creating logic in Java programs. They allow you to make decisions, repeat actions, and control the execution path of your code. Practice and understand each type of control flow statement to write more efficient and readable code.

Practice Exercises

1. Create a program that checks if a number is prime
2. Implement a simple menu-driven calculator
3. Write a program to find the factorial of a number using different loop types
4. Create a game that generates a random number and asks user to guess it

Recommended Learning Path

1. Master basic if-else and switch statements
2. Practice different loop types
3. Understand break and continue
4. Learn about advanced control flow techniques
5. Solve coding challenges to reinforce understanding