Control Flow- Conditionals

Introduction

Conditionals are fundamental constructs in Java that allow you to make decisions and control the flow of your program based on specified conditions. They enable you to execute different code blocks depending on whether certain conditions are met.

1. `if` statement

The if statement is used when you want to execute a block of code only if a specific condition evaluates to true. If the condition is false, the code inside the block is skipped.

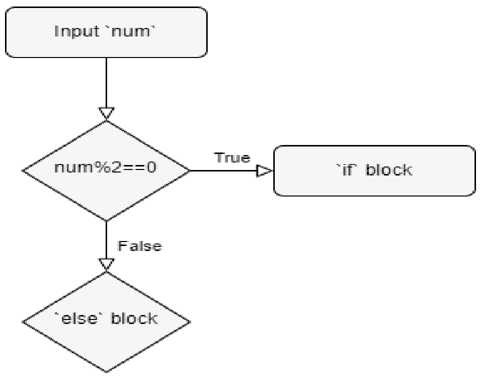
```
if (condition) {
    // Code to execute if the condition is true
}
```

2. `if-else` statement

The if-else statement allows you to provide an alternative code block executed when the condition is false. It provides a way to handle both cases.

```
 if (num%2==0) {
    System.out.println("Even Number");
}else {
    System.out.println("Odd Number");}
```

The above code can be visualised as below:

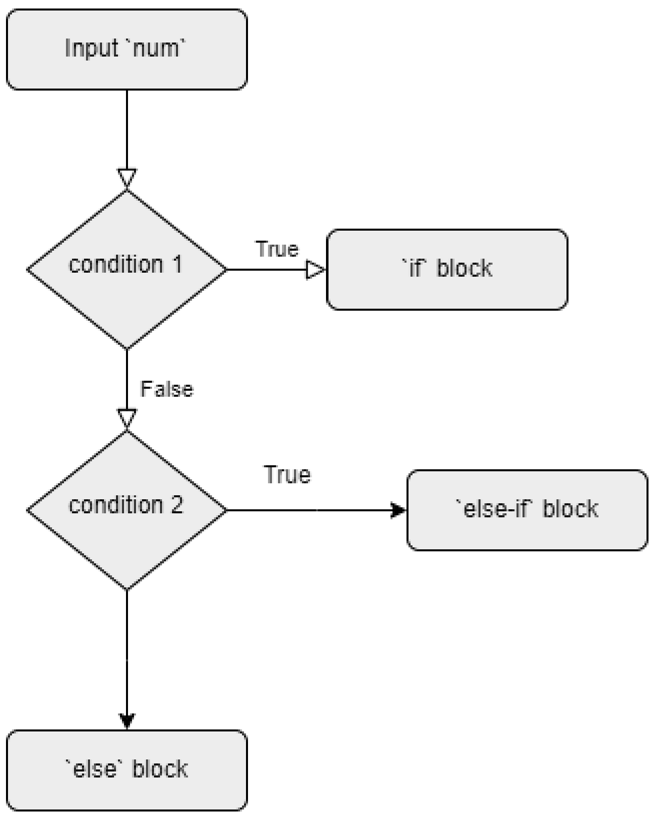


3. `if-else-if` statement

The *if-else-if statement* is used when you have multiple conditions to check sequentially. It allows you to specify different code blocks for each condition.

```
if (condition1) {
    // Code to execute if condition1 is true
} else if (condition2) {
    // Code to execute if condition2 is true
} else {
    // Code to execute if none of the conditions are true
}
```

The above code can be visualised as:



Key points to remember:

- Use meaningful variable and method names to enhance code readability.
- Always enclose code blocks within curly braces `{}` even for single statements to avoid ambiguity.
- Test your conditionals with various inputs to ensure they behave as expected.

4. `switch` statement

The `switch` statement is another conditional in Java that allows you to perform multi-way branching based on the value of an expression. It's designed for scenarios where you want to execute different code blocks depending on the value of a variable or expression.

```
switch (expression) {
    case value1:
        // Code to execute if expression equals value1
        break;
    case value2:
        // Code to execute if expression equals value2
        break;
    // Additional cases can follow...
    default:
        // Code to execute if expression doesn't match any case
}
```

Keywords in the code:

- **case**: Labels indicating the possible values of the expression.
- **break**: Terminates the switch block. Without it, execution continues to the next case.
- **default**: The default case is optional but serves as a catch-all if no other cases match the expression. It's commonly used to handle unexpected or undefined values.

Working:

- The expression is evaluated once.
- The value of the expression is compared to each case label.
- When a match is found, the corresponding code block is executed.
- Execution continues until a break statement is encountered or the switch block's end.
- If no match is found, the code block under default (if present) is executed.

5. Ternary Operators

- The ternary operator, also known as the conditional operator, provides a concise way to write conditional expressions in Java. It allows you to choose between two values or expressions based on a condition, all in a single line of code.
- **Basic Syntax**: condition ? valueIfTrue : valueIfFalse

Working:

- The condition is evaluated first.
- If the condition is true, the expression returns valueIfTrue.
- If the condition is false, the expression returns valueIfFalse.

```
int x = 10;
int y = 5;
int result = (x > y) ? x : y;

//if x is greater than y, the result will be assigned the value of x;
otherwise, it will be assigned the value of y.
```

Example of Nested Ternary Operator:

```
int x = 10;
int y = 20;
int z = 30;

// Find the maximum value among x, y, and z using a nested ternary
operator
int max = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z);

System.out.println("The maximum value is: " + max);
```

Working:

- The first level of the ternary operator compares x and y using (x > y). If x is greater than y, it evaluates the expression ((x > z) ? x : z); otherwise, it considers the expression ((y > z) ? y : z).
- In the first expression (x > z) ? x : z, it checks if x is greater than z. If it is, it returns x; otherwise, it returns z.
- In the second expression (y > z) ? y : z, it checks if y is greater than z. If it is, it returns y; otherwise, it returns z.
- The result of the first-level ternary operator is the maximum value among x, y, and z, which is stored in the max variable.
- Ultimately, the program prints the maximum value, which is determined using the nested ternary operator.

Conclusion

In conclusion, learning about conditionals in Java gives us essential tools for controlling program flow and making decisions based on conditions. These constructs enhance code readability, flexibility, and efficiency, allowing us to create more intelligent and adaptable Java programs. We can write cleaner and more maintainable code by choosing the correct conditional construct for the task at hand and following best practices.

- if, if-else, and if-else-if-else statements offer ways to handle different conditions and scenarios.
- The switch statement is effective for managing multiple cases with distinct code paths.
- The ternary operator simplifies writing simple conditional expressions efficiently.