KODNEST

ASSIGNMENT 7

Submitted By

Ragendu V S

E mail:ragendhuvs02@gmail.com

ELSE IF LADDER

The “else-if ladder,” also known as the “if-else-if-else” chain, is another control flow construct used in programming languages to make multiple conditional checks in sequence. It allows you to evaluate a series of conditions, and if any of them are true, the corresponding code block is executed. If found to be false, it continues to check the next else if ladder statement until the condition comes to be true or the control comes to end the else if ladder statement.

Syntax:

```
if(condition1){

    //code block to be executed if condition1 is true

}elseif(condition2){

    //code block to be executed if condition2 is true

}elseif(condition3){

    //code block to be executed if condition3 is true

}

else:

   //code block to be executed if none of the conditions are true

}
```

SWITCH STATEMENT

The “switch case” statement is a control flow mechanism found in many programming languages, including C, C++, Java, and others. It is particularly useful when you have multiple possible values for a single variable and want to execute different blocks of code based on the value of that variable.

DIFFERENCES BETWEEN ELSE IF LADDER AND SWITCH

STATEMENT:

1. **Syntax:**

- Switch case: It uses the `switch` keyword followed by an expression in parentheses. Cases are defined using the `case` keyword, and the block of code for each case is enclosed within curly braces.
- Else-if ladder: It uses multiple `if` statements followed by conditions. The conditions are checked one after the other, and the block of code associated with the first true condition is executed.

2. **Condition evaluation:**

- Switch case: The expression used in the switch statement is evaluated once, and the control jumps directly to the matching case. This means it is suitable for situations where you need to compare a single value against multiple constants.
- Else-if ladder: Each condition in the else-if ladder is evaluated one after the other until a true condition is found. This means all the conditions are checked sequentially, and the control passes through each condition, even if the earlier ones are true.

3. **Supported data types:**

- Switch case: It can only handle integral data types (e.g., int, char) and certain enumerated types.
- Else-if ladder: It can handle any expression that evaluates to a boolean value (true or false). This includes all data types that can be used in boolean expressions.

4. **Expression complexity:**

- Switch case: The expression inside the switch statement must result in a constant value, meaning it cannot be a complex expression or a range of values.
- Else-if ladder: The conditions in the else-if ladder can involve complex expressions, logical operations, and comparisons, allowing for more flexible conditions.

5. **Case matching:**

- Switch case: The case values must be constant and unique. The control will jump to the first matching case and execute the code within that case. If no match is found, an optional `default` case can be used.
- Else-if ladder: The conditions can be overlapping, meaning multiple conditions can be true for a given input, leading to multiple blocks of code being executed. The order of the conditions matters, and the first true condition is executed.

6. **Fall-through behaviour:**

- Switch case: If a case block does not end with a `break` statement, the control will fall through to the next case and continue executing the code in that case and any subsequent ones until a `break` statement is encountered or the switch block ends.

- Else-if ladder: Each `if` statement in the ladder is independent of others, and there is no implicit fall-through behaviour like in switch case.

7. **Readability and maintainability:**

- Switch case: It is more concise and readable when dealing with multiple constant values. It can be a good choice for simple, straightforward scenarios.
- Else-if ladder: For complex conditions or ranges of values, an else-if ladder may be more readable and maintainable as it allows expressing conditions more explicitly.

8. **Use cases**:

- Switch case: It is commonly used when you have a fixed set of values to compare against a single variable and when fall-through behaviour is needed (e.g., menu options, handling weekdays).
- Else-if ladder: It is used when you have a series of different conditions, each leading to a separate block of code execution, or when you need to evaluate complex expressions.

Nested simple if

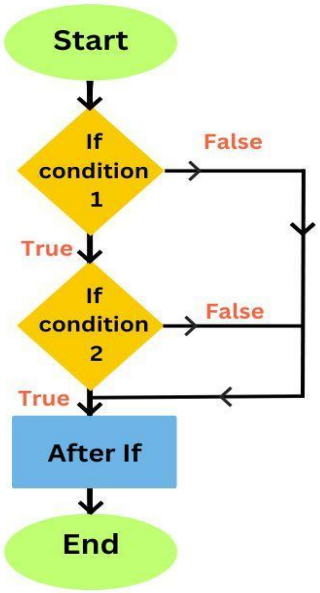
When an if block is defined within another if block, it is known as a nested if statement in Java, where the inner block of code is executed only if the condition of the outer if block returns true as well as the condition of the inner if block returns true.

The nested if statement in Java can be used to check for multiple conditions and execute statements depending on the condition associated with the if block.

Syntax:

```
if (condition) {
      if (condition) {
            // statements to be executed
}
}
```

Flowchart:



Example:

```
import java.util.Scanner;

public class Demo3 {
    public static void main(String args[])
    {
        Scanner scan=new Scanner(System.in);
         System.out.println("Enter the mark");
         int mark=scan.nextInt();
         System.out.println("Enter your family income");
         int income=scan.nextInt();
         if(mark>=1080){
              if(income<=200000){
System.out.println("You are eligible for Scholarship");
}
}}}
```

Nested if else

Java allows programmer to place if else block inside another if or else block. This is called as nested if else statement. Programmer can do any level of nesting in program which means you can place if else block inside another if or else block up to any number of times. Inner if block is executed only when outer if condition is true.

Syntax:

```
if(condition)
{
if(condition)
{
 //statements
}
else
 {
    //else block statement
  }
else
{
//statement
}
}
```

Example:

```
public class Demo4 {


    public static void main(String[] args) {

        int a = 2;
        int b = 2;
        int c = 2;

        if (a == b) {

         // nested if else condition to check if c is equal to a and b
            if (a == c) {
                // all are equal
                System.out.println("Equal");
            } else {
                // a!=c
                System.out.println("Not equal");
            }
        } else {
            // a!=b
            System.out.println("Not equal");
        }
    }
}
```