FACULTY OF ENGINEERING
Department of Telecommunications and Electronics Engineering

CONTROL STATEMENT

Basic Java Course

Advisor: Prof. Thap Thareoun

Present By : Ngouch HongCheng
: Saing LymChhun

CONTENT

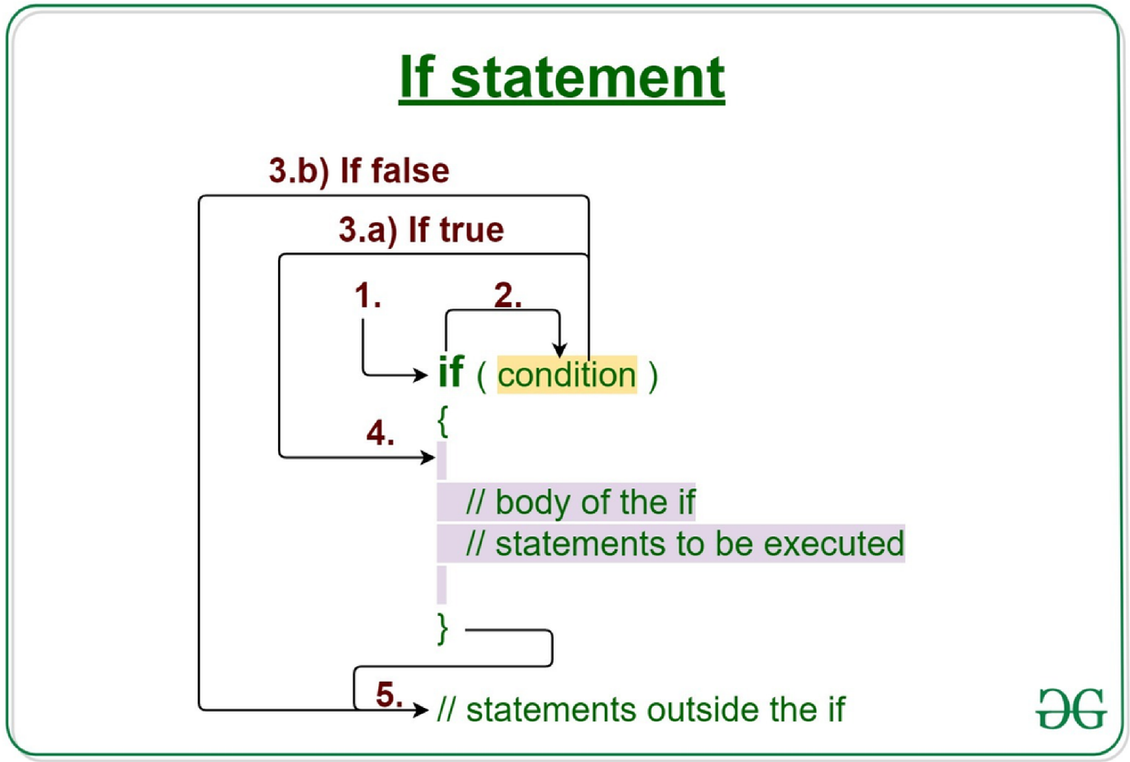
1 DECISION MAKING STATEMENTS

1. IF STATEMENT

The Java if statement is used to decide if a block of code should be executed based on whether a condition is true or false.

Syntax

```
if (condition) {
    // code to be executed if condition is true
} else {
    // code to be executed if condition is false
}
```



1 DECISION MAKING STATEMENTS

2. NESTED IF

In Java, nested if statements are if statements placed inside other if statements. They are used when you need to check multiple conditions in a sequence.

Syntax

```
if (outerCondition) {
    // code to be executed if outerCondition is true
    if (innerCondition) {
        // code to be executed if both outerCondition and innerCondition are true
    } else {
        // code to be executed if outerCondition is true but innerCondition is false
    }
} else {
    // code to be executed if outerCondition is false
}
```

1 DECISION MAKING STATEMENTS

3. IF-ELSE IF STATEMENT

In Java, The if-else if statement in Java allows you to evaluate multiple conditions sequentially. It's used when you have more than two possible outcomes to consider.

Syntax

```
public class Main {
    public static void main(String[] args) {
        int score = 75;

        if (score >= 85 && score <= 100) {
            System.out.println("You got grade A");
        } else if (score >= 75 && score < 85) {
            System.out.println("You got grade B");
        } else if (score >= 65 && score < 75) {
            System.out.println("You got grade C");
        } else if (score >= 55 && score < 65) {
            System.out.println("You got grade D");
        } else if (score >= 50 && score < 55) {
            System.out.println("You got grade E");
        } else {
            System.out.println("You fail your exam");
        }
    }
}
```

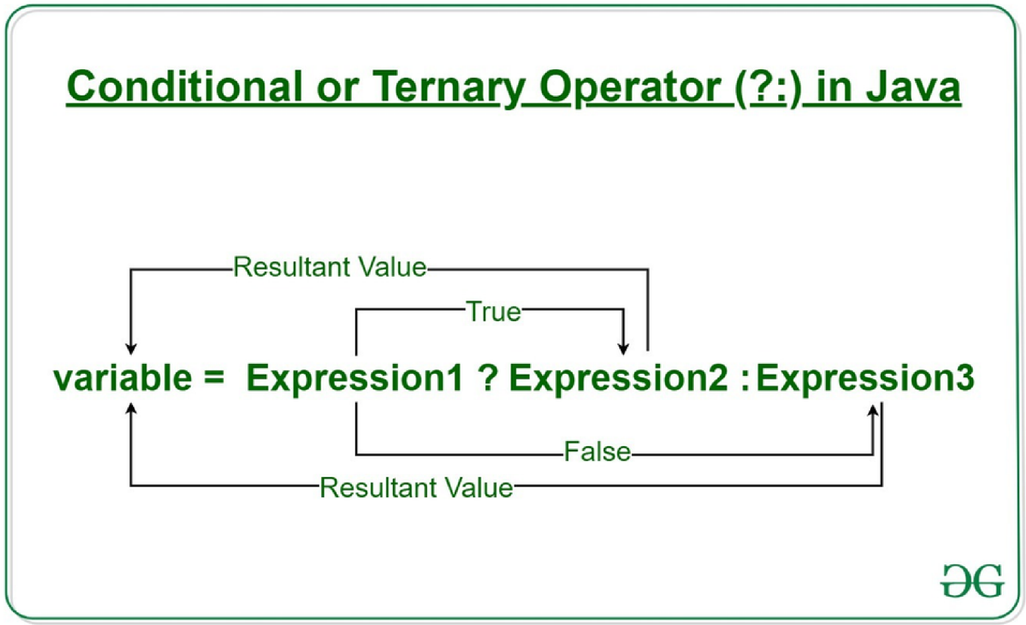
1 DECISION MAKING STATEMENTS

4. TERNARY OPERATOR

- The Java ternary operator is a shortcut for making decisions in code.
- It's like a compact version of an if-else statement, allowing you to assign values based on a condition with less typing. It's handy for making code shorter and easier to understand.

```
variable = (condition) ? expression1 : expression2;
```

Syntax



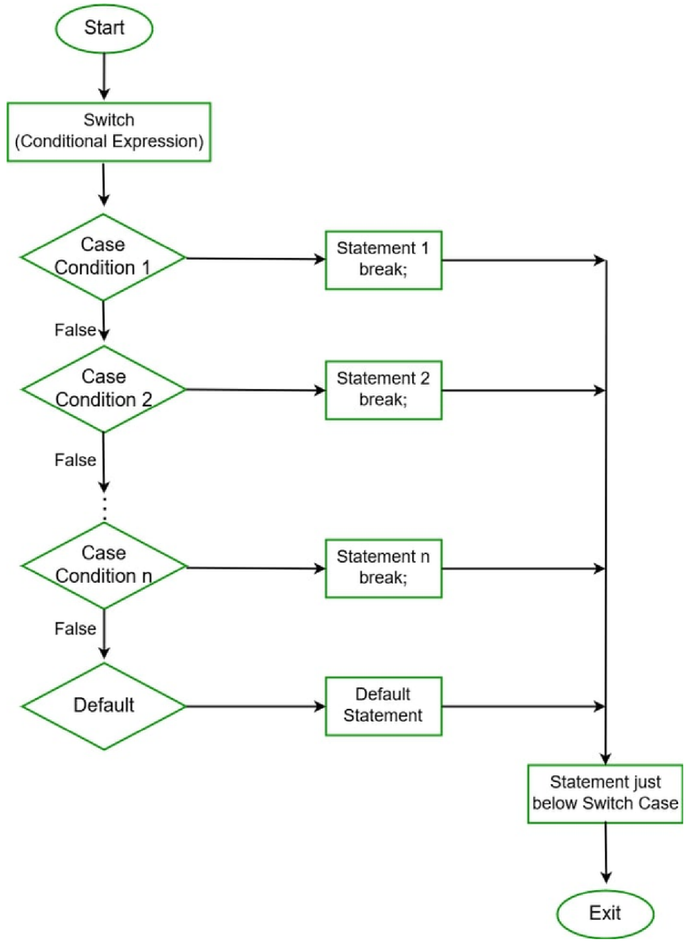
1 DECISION MAKING STATEMENTS

5. SWITCH STATEMENT

- In Java, the `switch` statement lets you choose one block of code to run from several options.
- It's like a simplified version of multiple `if-else` checks, comparing a variable against different values to decide which block of code to execute.
- It works with data types like `byte`, `short`, `char`, or `int`.

Syntax

```
switch (expression) {
    case value1:
        // code to be executed if expression equals value1
        break;
    case value2:
        // code to be executed if expression equals value2
        break;
    // additional cases as needed
    default:
        // code to be executed if expression doesn't match any case
}
```



1 DECISION MAKING STATEMENTS

6. NESTED SWITCH

- Nested switch statements in Java allow you to use one switch statement inside another.
- This helps in organizing and handling multiple levels of decision-making based on different variables or conditions.
- It provides a structured way to manage complex scenarios within your code.

Syntax

```
switch(ch1) {
     case 'A': System.out.println("This A is part of outer
     switch.");
          switch(ch2) {
          case 'A':
          System.out.println("This A is part of inner
          switch");
          break;
          case 'B': // ...
          } // end of inner switch
     break;
     case 'B': // ...
```

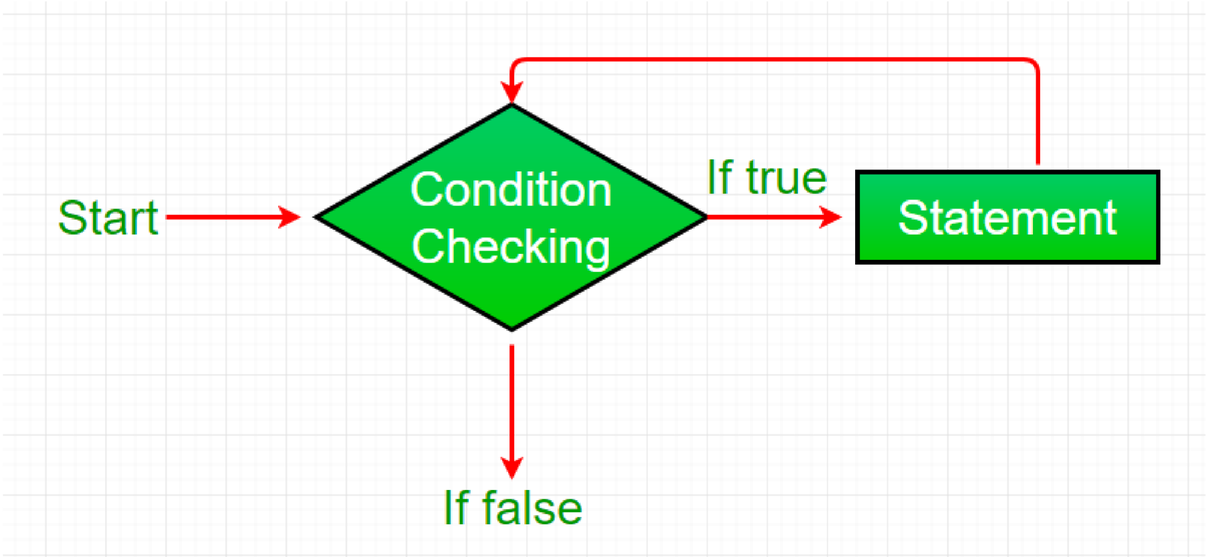
2 LOOP STATEMENT

1. WHILE LOOP

- The while loop in Java is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- The while loop can be thought of as a repeating if statement. If the condition is true, the loop executes the block of code inside the loop.
- This process continues until the condition becomes false.

Syntax

```
while (condition){
//code to be executed
I ncrement / decrement statement
}
```



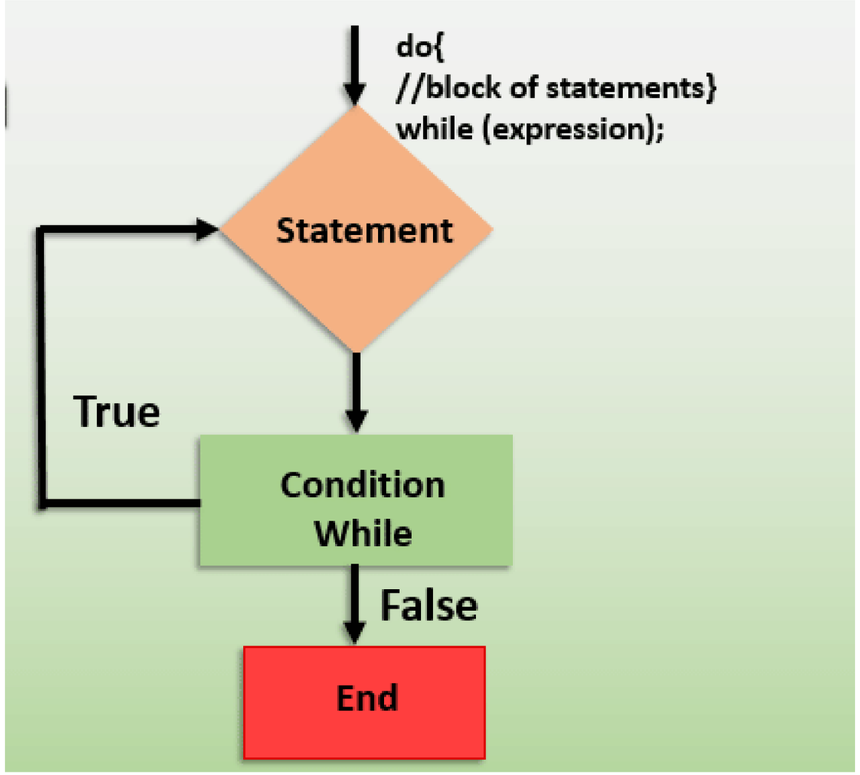
2 LOOP STATEMENT

2. DO WHILE LOOP

- A do-while loop in Java is a control flow statement that executes a block of code at least once and then repeatedly executes the block as long as a given condition is true.
- This loop guarantees that the code block is executed at least once, even if the condition is initially false.

Syntax

```
do{
//code to be executed / loop body
//update statement
}while (condition);
```



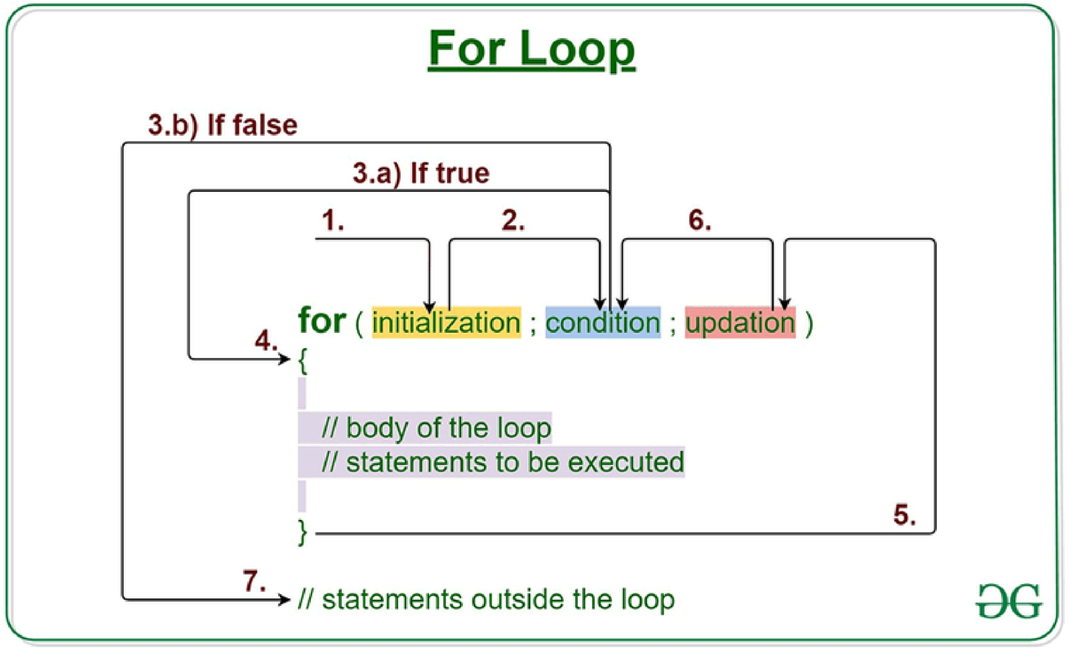
2 LOOP STATEMENT

3. FOR LOOP

- A for loop in Java is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- It is commonly used when the number of iterations is known beforehand.
- The for loop is composed of three parts: initialization, condition, and update, which provide a concise loop control structure.

Syntax

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```



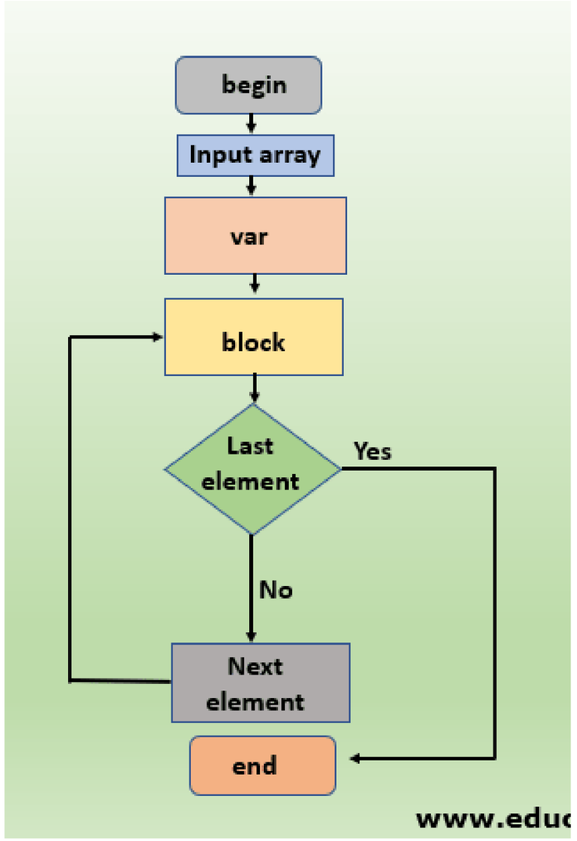
2 LOOP STATEMENT

4. FOR-EACH LOOP

- The for-each loop, also known as the enhanced for loop, is a simpler way to iterate over elements of arrays or collections in Java.
- It eliminates the need for explicitly managing loop counters and indices, making the code more readable and less error-prone.

Syntax

```
for(data_type variable : array_name){
//code to be executed
}
```



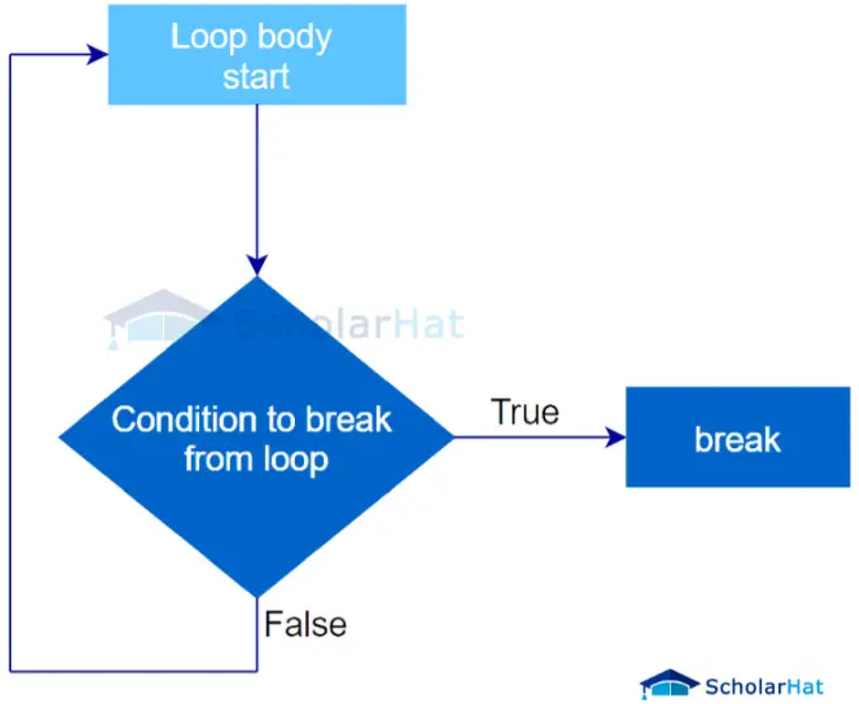
3 JUMP STATEMENT

1. BREAK

- The break statement in Java is used to terminate the nearest enclosing loop or switch statement immediately.
- When break is encountered inside a loop, the loop is exited, and the control moves to the next statement following the loop.

Syntax

```
jump-statement;
break;
```



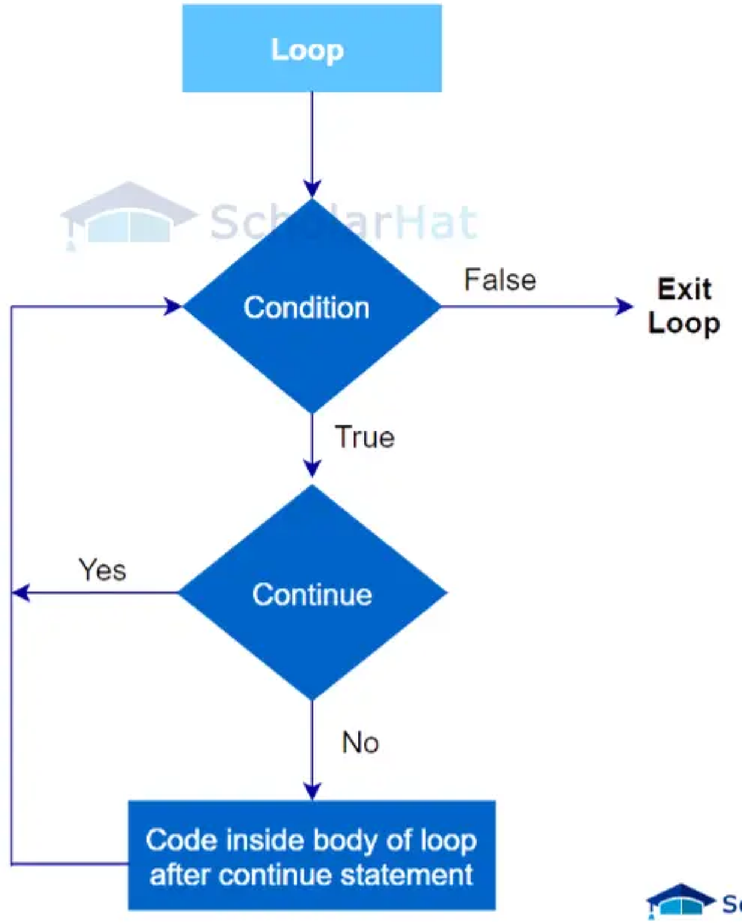
3 JUMP STATEMENT

2. CONTINUE

- In Java, the continue statement is a jump statement that is used to skip the rest of the current iteration of a loop and proceed to the next iteration.
- It is often used to avoid executing certain statements in a loop based on a condition, while still continuing with subsequent iterations.

Syntax

```
jump-statement;
continue;
```



3 JUMP STATEMENT

3. RETURN

- In Java, the return statement is a jump statement that is primarily used to exit from a method and optionally return a value back to the caller.
- Unlike break and continue, which are used within loops or switch statements to control the flow of execution within them, return is used at the method level to terminate the method execution.

Syntax

```
jump-statement;
continue;
```

END

THANK YOU

FOR YOUR ATTENTION