1. Flow of Control:
   The order in which statements are executed in a program.
2. Boolean Expression:
   An expression that evaluates to either true or false.
3. if Statement:
   Executes a block of code if a condition is true.
   Example:
   ```
   if (age > 18) {
       System.out.println("Adult");
   }
   ```
4. if-else Statement:
   Chooses one of two blocks based on a Boolean condition.
   Example:
   ```
   if (score >= 50) {
       System.out.println("Pass");
   } else {
       System.out.println("Fail");
   }
   ```
5. Nested if Statement:
   An if statement placed inside another if or else clause to make further decisions.
6. Block Statement:
   A group of statements enclosed in braces { } to form a single compound statement.
7. Relational Operator (==):
   Tests if two values are equal.
8. Relational Operator (!=):
   Tests if two values are not equal.
9. Relational Operator (<):
   Tests if one value is less than another.
10. Relational Operator (>):
    Tests if one value is greater than another.
11. Relational Operator (<=):
    Tests if one value is less than or equal to another.
12. Relational Operator (>=):
    Tests if one value is greater than or equal to another.
13. Logical NOT Operator (!):
    Negates a boolean value. For example, !true yields false.
14. Logical AND Operator (&&):
    Returns true only if both operands are true.
15. Logical OR Operator (||):
    Returns true if at least one operand is true.
16. Short-circuit Evaluation:
    Logical operators (&&, ||) stop evaluating as soon as the result is determined.
17. Truth Table:
    A table listing all possible values of a Boolean expression based on every input combination.

18. if Statement Example:
    if (age > 18) { System.out.println("Adult"); }
19. if-else Statement Example:
    if (score >= 50) { System.out.println("Pass"); } else { System.out.println("Fail"); }
20. Ternary (Conditional) Operator:
    A shorthand for if-else that returns a value.
    Syntax: condition ? expression1 : expression2
21. Ternary Operator Example:
    int max = (a > b) ? a : b;
22. switch Statement:
    Selects a block of code to execute based on the value of an expression.
    Example:

```
switch(day) {
    case 1: System.out.println("Monday"); break;
    default: System.out.println("Other day");
}
```

23. switch Expression:
    A variant of the switch statement that returns a value (newer Java versions).
24. do-while Loop:
    A post-test loop that executes its body at least once before checking the condition.
    Syntax:

```
do {
    statements;
} while (condition);
```

25. while Loop:
    A pre-test loop that executes as long as the condition is true.
    Syntax:

```
while (condition) {
    statements;
}
```

26. for Loop:
    A loop that includes initialization, condition, and increment/decrement in one statement.
    Syntax:

```
for (initialization; condition; increment) {
    statements;
}
```

27. for-each Loop:
    A simplified loop for iterating over arrays or collections.
    Syntax:

```
for (Type item : collection) {
    statements;
}
```

28. Infinite Loop:
    A loop that never terminates because its condition always evaluates to true.

29. Sentinel Value:
    A special value used to indicate the end of input within a loop.
30. Input Validation Loop:
    A loop that continues to prompt the user until valid input is received.
    Example:
    ```
    do {
        System.out.print("Enter a positive number: ");
        input = scanner.nextInt();
    } while (input <= 0);
    ```
31. Nested Loops:
    Loops inside loops, useful for multi-dimensional data processing.
    Example (nested while):
    ```
    int i = 1;
    while (i <= 3) {
        int j = 1;
        while (j <= 2) {
            System.out.println(i + "," + j);
            j++;
        }
        i++;
    }
    ```
32. Comparing Floating Point Numbers:
    Due to precision issues, use a tolerance when comparing floats.
33. Tolerance in Floating Point Comparison:
    A small value (e.g., 0.00001) used to determine if two floats are “close enough.”
    Example:
    ```
    if (Math.abs(f1 - f2) < 0.00001) { … }
    ```
34. Comparing Characters:
    Uses relational operators; characters are compared based on their Unicode values.
35. Unicode Ordering:
    The natural order of characters in Java: digits come first, then uppercase letters, then lowercase letters.
36. Comparing Strings using equals():
    Method to check if two strings are identical in content.
    Example:
    ```
    if (str1.equals(str2)) { … }
    ```
37. Comparing Strings using compareTo():
    Method to determine lexicographic order between strings.
    Example:
    ```
    int cmp = str1.compareTo(str2);
    if (cmp < 0) { … }
    ```
38. Lexicographic Ordering:
    Ordering based on dictionary sequence, which is affected by case and length.

39. Comparing Objects:
    By default, equals() compares object references, but it can be overridden to compare object content.
40. Assignment Operator (=) vs Equality Operator (==):
    '=' assigns a value, while '==' checks if two values are equal.
41. Indentation Importance:
    Proper indentation is essential for code readability, though it does not affect how Java executes the code.
42. Boolean Literal: true
    The literal representing a true value.
43. Boolean Literal: false
    The literal representing a false value.
44. Operator Precedence:
    Defines the order in which operators are evaluated (arithmetic > relational > logical).
45. Compound Conditions:
    Combining multiple expressions using logical operators (&&, ||, !).
46. Code Sample Using &&:
    ```
    if (x > 0 && y > 0) { System.out.println("Both positive"); }
    ```
47. Code Sample Using ||:
    ```
    if (x < 0 || y < 0) { System.out.println("At least one negative"); }
    ```
48. Code Sample: Nested Conditions:
    ```
    if (a > b) {
        if (a > c) { System.out.println("a is largest"); }
    }
    ```
49. Code Sample: while Loop Counting:
    ```
    int count = 1;
    while (count <= 10) {
        System.out.println(count);
        count++;
    }
    ```
50. Code Sample: do-while Loop Counting:
    ```
    int count = 1;
    do {
        System.out.println(count);
        count++;
    } while (count <= 10);
    ```
51. Code Sample: for Loop Iteration:
    ```
    for (int i = 0; i < 10; i++) {
        System.out.println(i);
    }
    ```
52. Code Sample: for-each Loop Iteration:
    ```
    for (String item : items) {
        System.out.println(item);
    }
    ```

53. Code Sample: Ternary Operator Usage:
    String result = (score >= 60) ? "Pass" : "Fail";
54. Code Sample: switch Statement:
    switch(day) {
        case 1: System.out.println("Monday"); break;
        case 2: System.out.println("Tuesday"); break;
        default: System.out.println("Other day");
    }
55. Flowchart for if Statement:
    A conceptual diagram that shows the decision process in an if statement.
56. Decision Making in Java:
    Using conditionals (if, if-else, switch) to control program flow.
57. Conditional Statements Overview:
    Statements that let you choose which code to execute based on conditions.
58. Repetition Statements Overview:
    Loops that allow code to be executed repeatedly.
59. Loop Control Statements:
    Keywords like break and continue (not detailed here) that alter loop behavior.
60. Pre-test Loop:
    A loop (while, for) that tests its condition before executing the loop body.
61. Post-test Loop:
    A loop (do-while) that tests its condition after executing the loop body.
62. Increment Operator (++):
    Increases a variable's value by one.
    Example: a++;
63. Decrement Operator (--):
    Decreases a variable's value by one.
    Example: a--;
64. Code Sample: Increment Operator:
    int a = 5;
    a++; // a becomes 6
65. Code Sample: Decrement Operator:
    int a = 5;
    a--; // a becomes 4
66. Infinite Loop Problem:
    Occurs when a loop's condition is never false, causing non-termination.
67. Debugging Loop Termination:
    The process of ensuring that loop conditions will eventually evaluate to false.
68. Calculating Loop Iterations:
    Determining how many times a loop will execute, especially in nested loops.
69. Code Sample: Nested while Loops:
    int i = 1;
    while (i <= 3) {
        int j = 1;
        while (j <= 2) {

```
                System.out.println(i + "," + j);
                j++;
            }
            i++;
        }
```

70. Code Sample: Nested for Loops:

```
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 2; j++) {
                System.out.println(i + "," + j);
            }
        }
```

71. Using Logical Operators in Conditions:

    Combining multiple conditions with &&, ||, and !.

72. Code Sample: if with && and ||:

```
        if ((x > 0 && y > 0) || z == 0) {
            System.out.println("Condition met");
        }
```

73. Code Sample: Input Validation using while:

```
        Scanner sc = new Scanner(System.in);
        int input;
        do {
            System.out.print("Enter a positive number: ");
            input = sc.nextInt();
        } while (input <= 0);
```

74. Short-Circuit Behavior Example:

    In the expression if (false && someMethod()), someMethod() is never called.

75. Code Sample: Avoiding Division by Zero:

```
        if (count != 0 && total / count > MAX) {
            System.out.println("Safe division");
        }
```

76. Comparing Data and Data Types:

    Understanding that comparisons differ based on data types (int, float, char, String, etc.).

77. Code Sample: Comparing Two Numbers:

```
        if (a < b) {
            System.out.println("a is less than b");
        }
```

78. Code Sample: Comparing Characters:

```
        if (ch1 < ch2) {
            System.out.println(ch1 + " comes before " + ch2);
        }
```

79. Code Sample: Using equals() for Strings:

```
        if (str1.equals(str2)) {
            System.out.println("Strings are equal");
        }
```

80. Code Sample: Using compareTo() for Strings:

```
int cmp = str1.compareTo(str2);
if (cmp < 0) {
    System.out.println(str1 + " comes before " + str2);
}
```

81. Importance of Tolerance for Floats:
    Always use a tolerance value when comparing floating-point numbers.
82. Code Sample: Floating Point Comparison:

```
if (Math.abs(f1 - f2) < 0.00001) {
    System.out.println("Floats are essentially equal");
}
```

83. for-each Loop in Arrays:
    Using for-each to iterate over array elements.
    Example:

```
for (int num : numbers) {
    System.out.println(num);
}
```

84. Iterator Interface Overview:
    An object that allows sequential access to elements in a collection.
85. Difference Between Pre-test and Post-test Loops:
    Pre-test loops check the condition before executing the loop; post-test loops execute the body first.
86. Use of Parentheses in Complex Boolean Expressions:
    Parentheses help clarify the order of evaluation in compound conditions.
87. Code Sample: if with Parentheses:

```
if ((a > b) && (c < d)) {
    System.out.println("Condition met");
}
```

88. Use of Curly Braces for Block Statements:
    Always use { } to group multiple statements, especially in conditionals and loops.
89. Code Sample: Block Statement Example:

```
if (x > y) {
    System.out.println("x is greater");
    x--;
} else {
    System.out.println("y is greater or equal");
}
```

90. Nested if-else Complexities:
    Understanding how else clauses pair with the nearest unmatched if.
91. Matching else Clause to Nearest if:
    A rule in Java where an else is associated with the closest preceding if that has not been paired with an else.
92. Common Pitfalls in Nested if Statements:
    Errors such as misaligned braces that lead to unexpected behavior.

93. Code Sample: Nested if Statement (MinOfThree):

```
if (num1 < num2) {
    if (num1 < num3) {
        min = num1;
    } else {
        min = num3;
    }
} else {
    if (num2 < num3) {
        min = num2;
    } else {
        min = num3;
    }
}
```

94. Self-Review Question: What is Flow of Control?

    It is the sequence in which statements are executed in a program.

95. Self-Review Question: What is a Truth Table?

    A chart that displays the output of a Boolean expression for every possible input combination.

96. Self-Review Question: How Do Logical Operators Work?

    They combine Boolean values: && returns true if both are true, || returns true if at least one is true, and ! negates a Boolean.

97. Self-Review Question: Difference Between while and do-while Loops?

    A while loop tests its condition before executing (and might not run at all), whereas a do-while loop executes its body at least once before checking the condition.

98. Self-Review Question: Write a Code Fragment to Count Occurrences of a Value.

    Example:

```
int count = 0;
for (int i = 0; i < array.length; i++) {
    if (array[i] == target) {
        count++;
    }
}
System.out.println(count);
```

99. Use of break Statement (Loop Control):

    Terminates the loop immediately when executed.

100. Use of continue Statement (Loop Control):

    Skips the remaining statements in the loop body and continues with the next iteration.

101. Code Sample: Using break in a Loop:

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

102. Code Sample: Using continue in a Loop:

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

```
        System.out.println(i);
    }
```

103. Repetition Statement Overview:
    A loop allows a program to execute a block of code repeatedly based on a condition.
104. Code Sample: Counting Using a for Loop with Custom Increment:

```
    for (int num = 100; num > 0; num -= 5) {
        System.out.println(num);
    }
```

105. Importance of Testing Loop Conditions:
    Ensure that the loop’s condition will eventually become false to avoid infinite loops.
106.