SIDDARTHA INSTITUTE OF SCIENCE AND TECHNOLOGY

Java Report

|| YEAR || SEMESTER

ISSUED BY
Sir Ravi Kumar

Done By
K.Naveen Kumar
224E1A1153
CAD

OBJECT ORIENTED PROGRAMMING THROUGH JAVA

COURSE OBJECTIVE:-

1. Gain knowledge about basic Java language syntax and semantics to write Java programs and use concepts such as variables, conditional and iterative execution methods etc
2. Understand the basic object oriented programming concepts and apply them in problem solving.
3. Illustrate inheritance concepts for reusing the program.
4. Demonstrate on multi-tasking by using multiple threads.
5. Understand the basics of java console and GUI based programming.

COURSE OUTCOMES:-

1. Gain knowledge about basic Java language syntax and semantics to write Java programs and use concepts such as variables, conditional and iterative execution methods etc
2. Understand the basic object oriented programming concepts and apply them in problem solving.
3. Illustrate inheritance concepts for reusing the program.
4. Demonstrate on multi-tasking by using multiple threads.
5. Understand the basics of java console and GUI based programming.

UNIT-01

The Java Language

INTRODUCTION TO JAVA

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.Java was developed by James Gosling at Sun Microsystems Inc in the year 1995 and later acquired by Oracle Corporation. It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs. Java is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible. A general-purpose programming language made for developers to write once run anywhere that is compiled Java code can run on all platforms that support Java.

FEATURES OF JAVA

1) Simple. Java is easy to learn and its syntax is quite simple, clean and easy to understand. ...
2) Object Oriented. In java, everything is an object which has some data and behaviour. ...
3) Robust. ...
4) Platform Independent. ...
5) Secure. ...
6) Multi Threading. ...
7) Architectural Neutral. ...
8) Portable.

IDE IN JAVA

An integrated development environment (IDE) is a software application that helps programmers develop software code efficiently. It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application. Just as writers use text editors and accountants use spreadsheets, software developers use IDEs to make their job easier.

Code editing automation:

Programming languages have rules for how statements must be structured. Because an IDE knows these rules, it contains many intelligent features for automatically writing or editing the source code.

Syntax highlighting

An IDE can format the written text by automatically making some words bold or italic, or by using different font colours. These visual cues make the source code more readable and give instant feedback about accidental syntax errors.

Intelligent code completion

Various search terms show up when you start typing words in a search engine. Similarly, an IDE can make suggestions to complete a code statement when the developer begins typing.

Refactoring support

Code refactoring is the process of restructuring the source code to make it more efficient and readable without changing its core functionality. IDEs can auto-refactor to some extent, allowing developers to improve their code quickly and easily. Other team members understand readable code faster, which supports collaboration within the team.

Local build automation

IDEs increase programmer productivity by performing repeatable development tasks that are typically part of every code change. The following are some examples of regular coding tasks that an IDE carries out.

Compilation

An IDE compiles or converts the code into a simplified language that the operating system can understand. Some programming languages implement just-in-time compiling, in which the IDE converts human-readable code into machine code from within the application.

Testing

The IDE allows developers to automate unit tests locally before the software is integrated with other developers' code and more complex integration tests are run.

Debugging

Debugging is the process of fixing any errors or bugs that testing reveals. One of the biggest values of an IDE for debugging purposes is that you can step through the code, line by line, as it runs and inspect code behaviour. IDEs also integrate several debugging tools that highlight bugs caused by human error in real time, even as the developer is typing

TYPES OF ERRORS

There are several types of errors that occur in Java, including syntax errors, runtime errors, and logical errors. They are

- **Syntax Errors or Compilation Errors:**
  These occur when the code violates the rules of the Java syntax. These errors are usually caught by the Java compiler during the compilation phase.

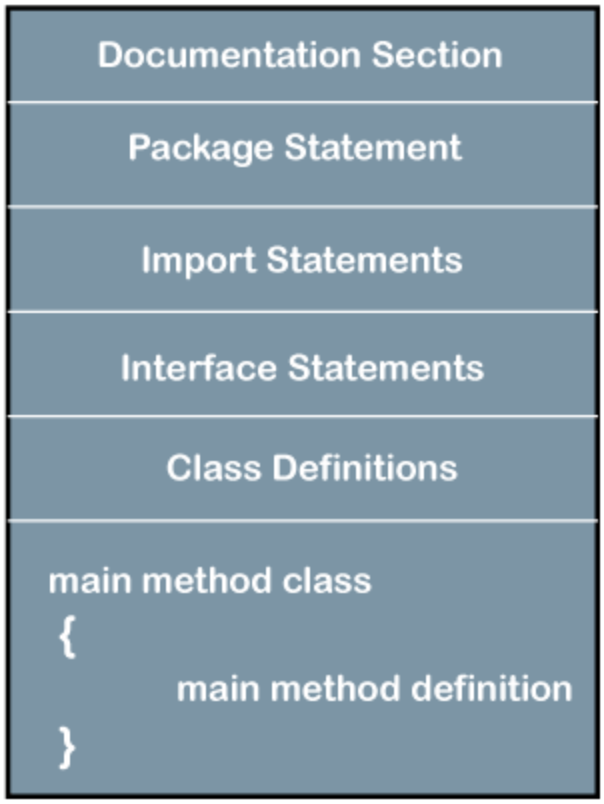
- **Runtime Errors:**
  These errors occur when the code encounters unexpected behaviour during its execution. These errors are usually caused by flawed logic or incorrect assumptions in the code and can be difficult to identify and fix.

- **Logical Errors:**
  These occur when the program is executing correctly, but the result is not what was intended. These errors can be difficult to identify and fix, as the program is running correctly but producing unintended results.

- **Linking Errors:**
  Linking errors occur when the Java Virtual Machine (JVM) tries to link a class file during runtime but encounters a problem. This can happen if a required class or library is missing, or if there is a problem with the syntax or format of the code.

- **Class Loading Errors:**
  Class loading errors occur when the JVM attempts to load a class into memory but encounters a problem. This can happen if the class file is missing or corrupted, or if there is a problem with the classpath.

STRUCTURE OF JAVA PROGRAM



Structure of Java Program

APPLICATIONS OF JAVA

- Desktop GUI Applications
- Mobile Applications
- Artificial intelligence
- Web applications
- Big Data technology
- Gaming applications

- Business applications
- Embedded systems
- Cloud applications
- Scientific applications

SOFTWARE OF JDK

The Java Development Kit (JDK) is one of three core technology packages used in Java programming, along with the JVM(Java Virtual Machine) and the JRE(Java Runtime Environment).It's important to differentiate between these three technologies and understand how they're connected:

- The JVM is the runtime that hosts running programs.
- The JRE is the on-disk part of Java that creates the JVM and loads programs into them.
- The JDK provides the tools necessary to write Java programs that can be executed and run by the JVM and JRE.

Developers new to Java often confuse the Java Development Kit and the Java Runtime Environment. The distinction is that the JDK is a package of tools for developing Java-based software, whereas the JRE is a package of tools for running Java code.

BEST JDK TO DOWNLOAD

- Oracle OpenJDK (limited free, GPL-licensed binaries at jdk.java.net)
- Oracle JDK (Oracle No-Fee commercial licence)
- Adoptium Eclipse Temurun (formally AdoptOpenJDK)
- Amazon Corretto
- Alibaba Dragonwell
- Azul Zulu (free)

STEPS TO DOWNLOAD JAVA

- Download JDK from the site .Go to the oracle site and open the java SE download page
- Install the JDK exe file
- Check the directory
- Update the environment variables
- Verify the Java installation

SET PATH IN JAVA

- Open the System Properties window.
- Click on the Advanced tab.
- Click on the Environment Variables button.
- In the System Variables section, find the PATH variable and click on the Edit button.
- Add the path to the JDK bin directory to the end of the PATH variable.
- Click OK to save the changes.

JAVA TOKENS

The program contains classes and methods. Further, the methods contain the expressions and statements required to perform a specific operation. These statements and expressions are made up of tokens. In other words, we can say that the expression and statement is a set of tokens. The tokens are the small building blocks of a Java program that are meaningful to the Java compiler. Further, these two components contain variables, constants, and operators. In this section, we will discuss what tokens are in Java.

1.Keywords:

These are the pre-defined reserved words of any programming language. Each keyword has a special meaning. It is always written in lowercase.

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

2.Identifiers:

Identifiers are used to name a variable, constant, function, class, and array. It is usually defined by the user. It uses letters, underscores, or a dollar sign as the first character. The label is also known as a special kind of identifier that is used in the goto statement. Remember that the identifier name must be different from the reserved keywords.

Rules to declare Identifiers:-

- The variable name must start with a letter, dollar sign ($), or underscore (_).
- The variable name can contain letters, digits, dollar signs, and underscores.
- The variable name cannot contain spaces.
- The variable name cannot be a Java keyword.

Examples:

1. PhoneNumber
2. PRICE

3. radius
4. a
5.a1

3.Literals:

In programming literal is a notation that represents a fixed value (constant) in the source code. It can be categorised as an integer literal, string literal, Boolean literal, etc. It is defined by the programmer. Once it has been defined cannot be changed.

4. Separators:-

The separators in Java are also known as punctuators. There are nine separators in Java, are as follows:
1. separator <= ; | , | . | (|) | { | } | [|]

- **Square Brackets []:**
  It is used to define array elements. A pair of square brackets represents the single-dimensional array, two pairs of square brackets represent the two-dimensional array.
- **Parentheses ():**

  It is used to call the functions and parsing the parameters.
- **Curly Braces {}:**

  The curly braces denote the starting and ending of a code block.
- **Comma (,):**

  It is used to separate two values, statements, and parameters.
- **Assignment Operator (=):**

  It is used to assign a variable and constant.

- **Semicolon (;):**

  It is the symbol that can be found at the end of the statements. It separates the two statements.

- **Period (.):**

  It separates the package name from the sub-packages and class. It also separates a variable or method from a reference variable.

5.Operators:

In programming, operators are the special symbol that tells the compiler to perform a special operation. Java provides different types of operators that can be classified according to the functionality they provide.

- Arithmetic Operators
- Assignment Operators
- Bitwise Operators
- Relational Operators
- Ternary Operators
- Logical Operators
- Unary Operators
- Shift Operators

ARITHMETIC OPERATORS

Arithmetic operators are the symbols used to perform basic mathematical operators data.

Operator	Description	Example
+ Addition	Adds values on either side of the operator	A+B=30
– Subtraction	Subtracts the right-hand operator with left-hand operator	A-B=-10
* Multiplication	Multiplies values on either side of the operator	A*B=200
/ Division	Divides left hand operand with right hand operator	A/B=0
% Modulus	Divides left hand operand by right hand operand and returns remainder	A%B=0

PROGRAM:

```
//Arithmetic operators

public class ArithmeticOperator {
   public static void main(String[] args) {
      int num1 = 10;
      int num2 = 5;
      System.out.println("K.Naveenkumar");
      int sum = num1 + num2;
      System.out.println("Sum: " + sum);
      int difference = num1 - num2;
      System.out.println("Difference: " + difference);
      int product = num1 * num2;
      System.out.println("Product: " + product);
      int quotient = num1 / num2;
      System.out.println("Quotient: " + quotient);
      int remainder = num1 % num2;
      System.out.println("Remainder: " + remainder);
   }
}
```

OUTPUT:

```
K.Naveenkumar
Sum: 15
Difference: 5
Product: 50
Quotient: 2
Remainder: 0
```

ASSIGNMENT OPERATORS

Assignment operators are used to assign values to variables.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b
+=	It adds right operand to the left operand and assigns the result to left operand	c += a
-=	It subtracts right operand from the left operand and assigns the result to left operand	c -= a
*=	It multiplies right operand with the left operand and assigns the result to left operand	c *= a
/=	It divides left operand with the right operand and assigns the result to left operand	c /= a
%=	It takes modulus using two operands and assigns the result to left operand	c %= a
^=	Performs exponential (power) calculation on operators and assign value to the left operand	c ^= a

PROGRAM:

```
//Assignment operator

public class AssignmentOperator {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 5;
        System.out.println("K.Naveenkumar");
        int result = num1;
        System.out.println("Result after assignment: " + result);
        result += num2;
        System.out.println("Result after addition assignment: " + result);
        result -= num2;
        System.out.println("Result after subtraction assignment: " + result);
        result *= num2;
        System.out.println("Result after multiplication assignment: " + result);
        result /= num2;
        System.out.println("Result after division assignment: " + result);
        result %= num2;
        System.out.println("Result after modulus assignment: " + result);
    }
}
```

OUTPUT:

```
K.Naveenkumar
Result after assignment: 10
Result after addition assignment: 15
Result after subtraction assignment: 10
Result after multiplication assignment: 50
Result after division assignment: 10
Result after modulus assignment: 0
```

BITWISE OPERATORS

Bitwise operators are used to performing the manipulation of individual bits of a number. They can be used with any integral type (char, short, int, etc.). They are used when performing update and query operations of the Binary indexed trees.

Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	op1 | op2
^	Bitwise Exclusive OR	op1 ^ op2
~	Bitwise Complement	~op

TRUTH TABLE:

X	Y	X&Y	X|Y	X^Y	~X
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

PROGRAM:

```
//Bitwise operator

public class BitwiseOperator {
```

```
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 5;
        System.out.println("K.Naveenkumar");
        int bitwiseAnd = num1 & num2; // Result: 0
        System.out.println("Bitwise AND: " + bitwiseAnd);
        int bitwiseOr = num1 | num2; // Result: 15
        System.out.println("Bitwise OR: " + bitwiseOr);
        int bitwiseXor = num1 ^ num2; // Result: 15
        System.out.println("Bitwise XOR: " + bitwiseXor);
        int bitwiseComplementNum1 = ~num1; // Result: -11
        System.out.println("Bitwise Complement of num1: " +
bitwiseComplementNum1);
        int bitwiseComplementNum2 = ~num2; // Result: -6
        System.out.println("Bitwise Complement of num2: " +
bitwiseComplementNum2);
        int leftShift = num1 << 2; // Result: 40
        System.out.println("Left Shift: " + leftShift);
        int rightShift = num1 >> 2; // Result: 2
        System.out.println("Right Shift: " + rightShift);
        int unsignedRightShift = num1 >>> 2; // Result: 2
        System.out.println("Unsigned Right Shift: " + unsignedRightShift);
    }
}
```

OUTPUT:

```
K.Naveenkumar
Bitwise AND: 0
Bitwise OR: 15
Bitwise XOR: 15
Bitwise Complement of num1: -11
Bitwise Complement of num2: -6
Left Shift: 40
Right Shift: 2
Unsigned Right Shift: 2
```

RELATIONAL OPERATORS

Java Relational Operators are a bunch of binary operators used to check for relations between two operands, including equality, greater than, less than, etc. They return a boolean result after the comparison and are extensively used in looping statements as well as conditional if-else statements and so on.

Java Symbol	Operator
==	Equal To (See Caution Below)
!=	Not Equal To
<	Less Than
<=	Less Than Or Equal To
>	Greater Than
>=	Greater Than or Equal To
&&	AND
||	OR

PROGRAM:

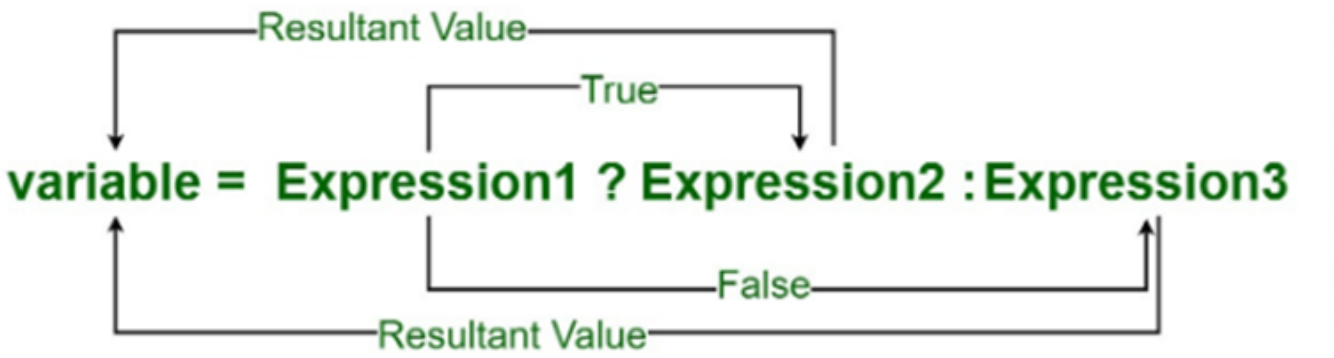
```
//Relational operator

class RelationalOperator {
   public static void main(String[] args) {
      int num1 = 10;
      int num2 = 5;
      System.out.println("K.Naveen Kumar");
      boolean equalTo = num1 == num2;
      System.out.println("Equal to: " + equalTo);
      boolean notEqualTo = num1 != num2;
      System.out.println("Not equal to: " + notEqualTo);
      boolean greaterThan = num1 > num2;
      System.out.println("Greater than: " + greaterThan);
      boolean lessThan = num1 < num2;
      System.out.println("Less than: " + lessThan);
      boolean greaterThanOrEqualTo = num1 >= num2;
      System.out.println("Greater than or equal to: " + greaterThanOrEqualTo);
      boolean lessThanOrEqualTo = num1 <= num2;
      System.out.println("Less than or equal to: " + lessThanOrEqualTo);
   }
}
```

OUTPUT:

```
K.Naveen Kumar
Equal to: false
Not equal to: true
Greater than: true
Less than: false
Greater than or equal to: true
Less than or equal to: false
```

CONDITIONAL OPERATORS

Java ternary operator is the only conditional operator that takes three operands. It's a one-liner replacement for the if-then-else statement and is used a lot in Java programming. We can use the ternary operator in place of if-else conditions or even switch conditions using nested ternary operators. Although it follows the same algorithm as the if-else statement, the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.



PROGRAM:

```
//Conditional operator

public class ConditionalOperator {
   public static void main(String[] args) {
      int num1 = 10;
      int num2 = 5;
       System.out.println("K.Naveen Kumar");
      // Conditional Operator
      String result = (num1 > num2) ? "num1 is greater" : "num2 is greater";
      System.out.println(result);
```

```
        // Nested Conditional Operator
        int num3 = 15;
        String nestedResult = (num1 > num2) ? ((num1 > num3) ? "num1 is the greatest"
: "num3 is the greatest") : ((num2 > num3) ? "num2 is the greatest" : "num3 is the
greatest");
        System.out.println(nestedResult);
    }
}
```

OUTPUT:

```
K.Naveen Kumar
num1 is greater
num3 is the greatest
```

LOGICAL OPERATORS

Logical operators are used to perform logical “AND”, “OR” and “NOT” operations, i.e. the function similar to AND gate and OR gate in digital electronics. They are used to combine two or more condition

Operation	Operator	Description
Logical AND	&&	Returns True if both the conditions mentioned are true
Logical OR	||	Returns True if one or both the conditions mentioned are true
Logical NOT	!	Returns True if the condition mentioned is False

PROGRAM:

```
//Logical Operator

public class LogicalOperator {
   public static void main(String[] args) {
      boolean a = true;
      boolean b = false;
      System.out.println("K.NaveenKumar");
      boolean logicalAnd = a && b;
      System.out.println("Logical AND: " + logicalAnd);
      boolean logicalOr = a || b;
      System.out.println("Logical OR: " + logicalOr);
      boolean logicalNotA = !a;
      boolean logicalNotB = !b;
      System.out.println("Logical NOT of a: " + logicalNotA);
      System.out.println("Logical NOT of b: " + logicalNotB);
```

```
  }
}
```

OUTPUT:

```
K.NaveenKumar
Logical AND: false
Logical OR: true
Logical NOT of a: false
Logical NOT of b: true
```

UNARY OPERATORS

Java unary operators are the types that need only one operand to perform any operation like increment, decrement, negation, etc. It consists of various arithmetic, logical and other operators that operate on a single operand.

Types	Description	Example
Pre Increment	First it performs increment and then it assigns value	++i
Post Increment	First it assigns value and then it performs increment	i++
Pre Decrement	First it performs decrement and then it assign value	--i
Post Decrement	First it assigns value and then it performs decrement	i--

PROGRAM:

//Unary operator

```
public class UnaryOperatorsExample {
   public static void main(String[] args) {
      int num = 10;
      System.out.println("K.Naveen Kumar");
      int unaryPlus = +num;
      System.out.println("Unary Plus: " + unaryPlus);
      int unaryMinus = -num;
      System.out.println("Unary Minus: " + unaryMinus);
      int preIncrement = ++num;
      System.out.println("Pre-increment: " + preIncrement);
      int postIncrement = num++;
      System.out.println("Post-increment: " + postIncrement);
      System.out.println("Value of num after post-increment: " + num);
      int preDecrement = --num;
      System.out.println("Pre-decrement: " + preDecrement);
      int postDecrement = num--;
      System.out.println("Post-decrement: " + postDecrement);
      System.out.println("Value of num after post-decrement: " + num);
      int bitwiseComplement = ~num;
      System.out.println("Unary NOT (Bitwise Complement): " +
bitwiseComplement);
   }
}
```

OUTPUT:

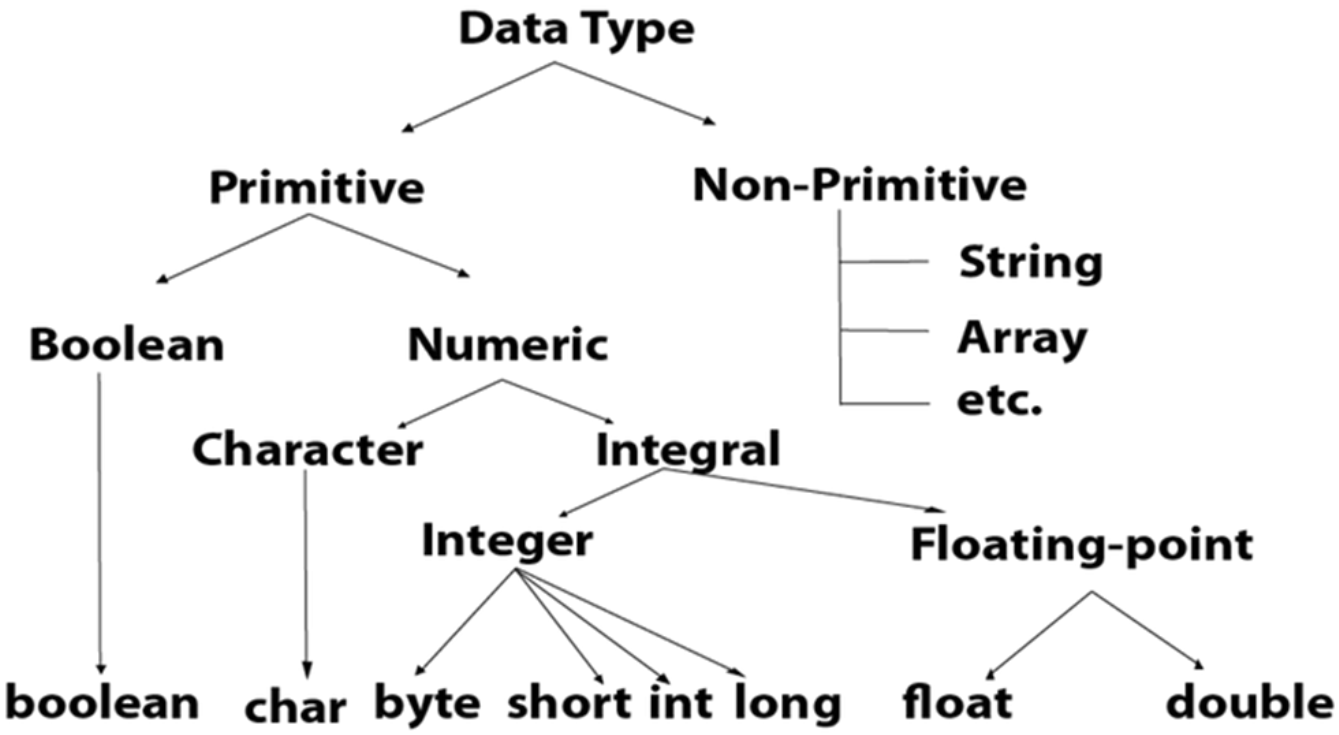
```
K.Naveen Kumar
Unary Plus: 10
Unary Minus: -10
Pre-increment: 11
Post-increment: 11
Value of num after post-increment: 12
Pre-decrement: 11
Post-decrement: 11
Value of num after post-decrement: 10
Unary NOT (Bitwise Complement): -11
```

JAVA VARIABLES

A variable is a container which holds the value while the Java program is executed.
A variable is assigned with a data type.
Variable is a name of memory location. There are three types of variables in java: local, instance and static.
A variable is the name of a reserved area allocated in memory. In other words, it is the name of the memory location. It is a combination of "vary + able" which means its value can be changed.

JAVA DATATYPES

Data types specify the different sizes and values that can be stored in the variable.
There are two types of data types in Java:
1. Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.
2. Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.



1.Primitive Data Types:

These are the basic data types provided by Java. They are not objects and do not have any methods. They are stored directly in memory and are passed by value. There are eight primitive data types in Java

- byte: 8-bit integer value.
- short: 16-bit integer value.
- int: 32-bit integer value.
- long: 64-bit integer value.
- float: 32-bit floating-point value.
- double: 64-bit floating-point value.
- char: 16-bit Unicode character.
- boolean: Represents true or false.

2.Reference Data Types:

These are data types that refer to objects. They are created using predefined classes or user-defined classes. They are stored in the heap memory, and

variables of reference types store references to the objects. Examples of reference data types include:

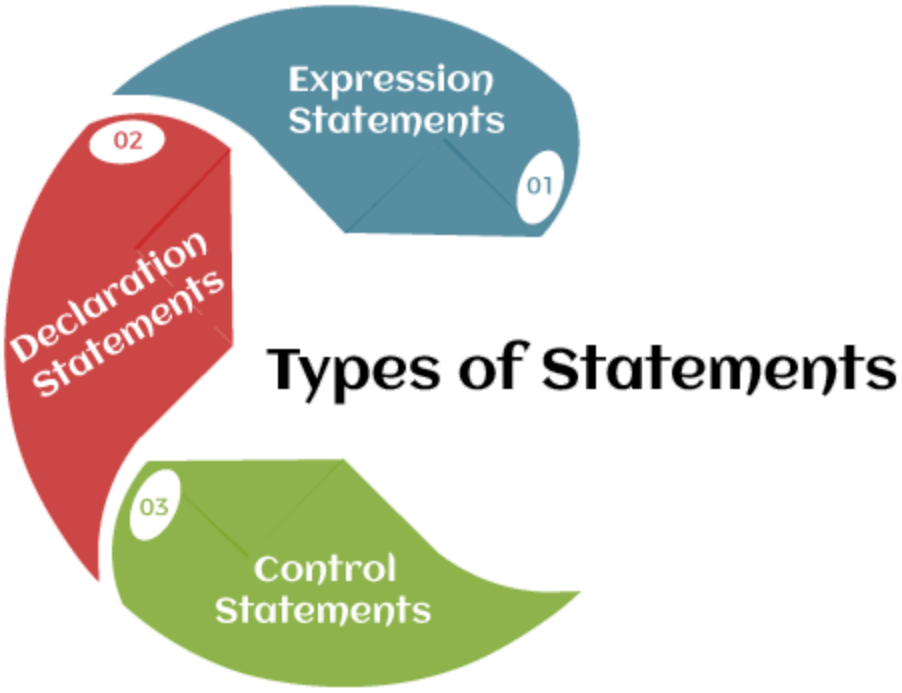
• Classes: Instances of user-defined classes.
• Interfaces: Interfaces can also be treated as reference types.
• Arrays: Arrays in Java are reference types.

JAVA STATEMENTS

In Java, a statement is an executable instruction that tells the compiler what to perform. It forms a complete command to be executed and can include one or more expressions. A sentence forms a complete idea that can include one or more clauses.

Java statements can be broadly classified into the following categories:

- Expression Statements
- Declaration Statements
- Control Statements



EXPRESSION STATEMENTS

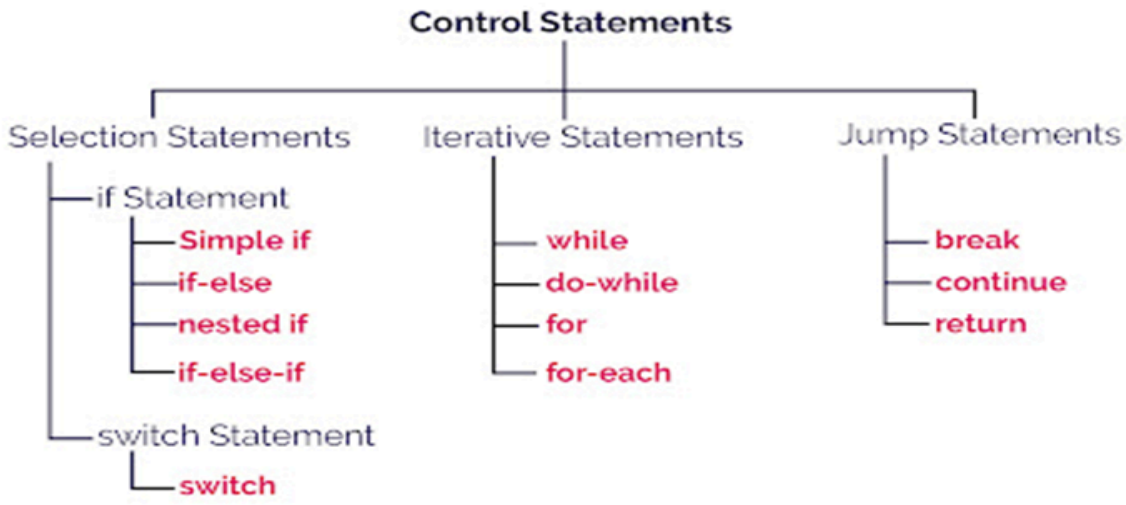
Expression is an essential building block of any Java program. Generally, it is used to generate a new value. Sometimes, we can also assign a value to a variable. In Java, expression is the combination of values, variables, operators, and method calls.

DECLARATION STATEMENTS

In declaration statements, we declare variables and constants by specifying their data type and name. A variable holds a value that is going to use in the Java program For example:

1. int quantity;
2. boolean flag;
3. String message;

CONTROL STATEMENTS



1. **Conditional or Selection Statements**
    - if Statement
    - if-else statement
    - if-else-if statement
    - Switch statement
2. **Loop or Iterative Statements**
    - for Loop
    - while Loop
    - do-while Loop
    - for-each Loop
3. **Flow Control or Jump Statements**
    - return
    - continue
    - break

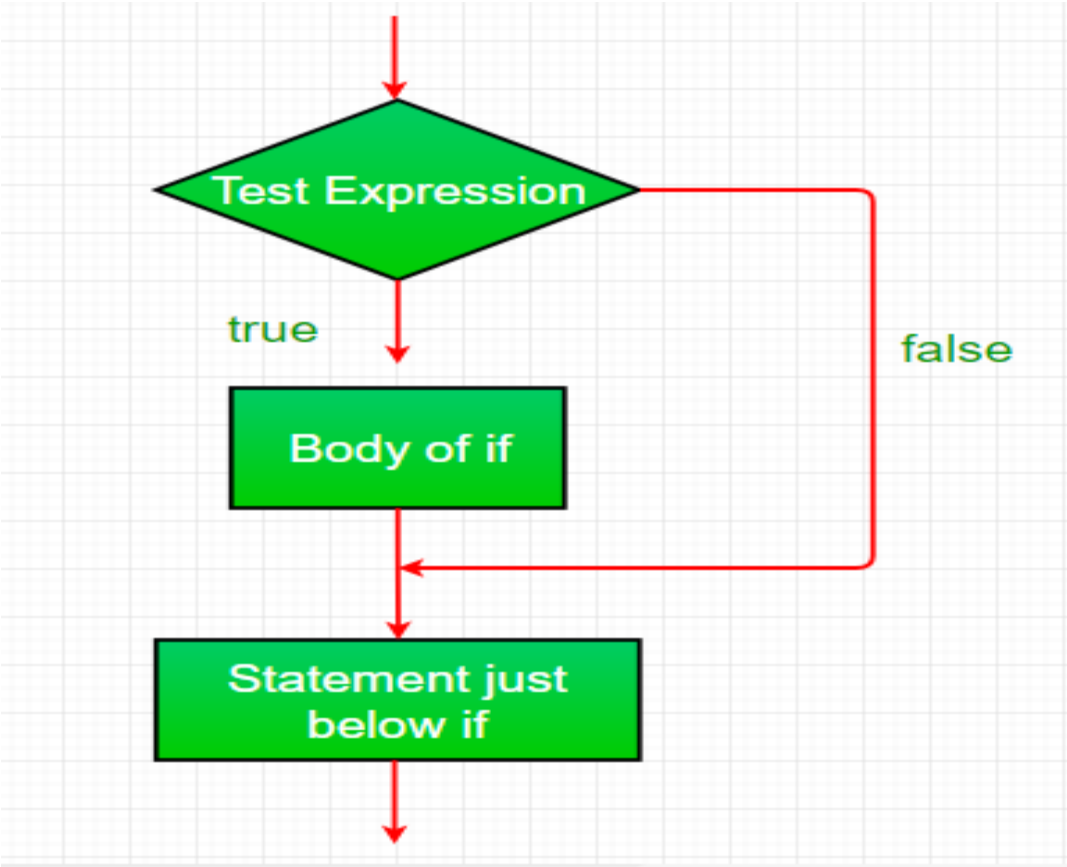
SIMPLE IF

The simple if statement is the most simple decision making statement. It is used to decide whether a certain block of statements will be executed or not.

SYNTAX :

```
if(condition)
{
Statements;
}
```

CONTROL FLOW:



PROGRAM:

```
//Simple if

import java.util.Scanner;
public class SimpleIfWithScanner {
   public static void main(String[] args) {
           Scanner scanner = new Scanner(System.in);
       System.out.println("Naveen kumar Enter a number: ");
      int number = scanner.nextInt();
      if (number > 10) {
         System.out.println("The number is greater than 10.");
      }
       scanner.close();
```

```
    }
}
```

OUTPUT:

```
java -cp /tmp/9B1TB4Qw9T SimpleIfWithScanner
Naveen kumar Enter a number:
11
The number is greater than 10.
```

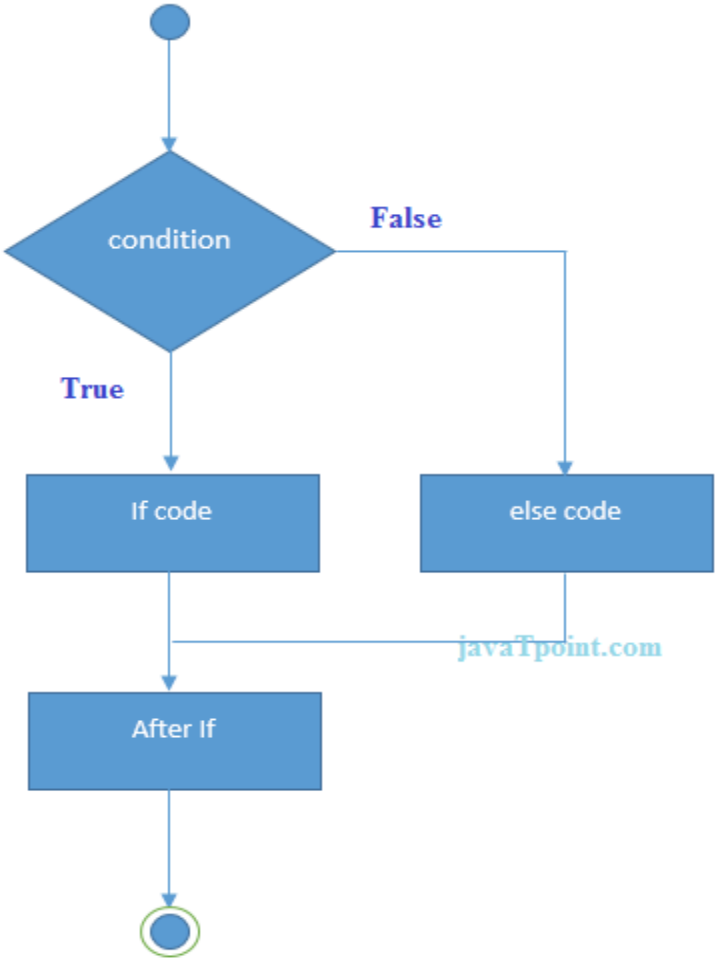
IF-ELSE

In the statement, if the specified is true, the if block is executed. Otherwise, the else block is executed.

SYNTAX :

```
if (condition)
{
Statements;
}
else
{
Statements;
}
```

CONTROL FLOW:



PROGRAM:

```
//If-else statement

import java.util.Scanner;
public class naveen {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Naveen Kumar Enter a number: ");
      int number = scanner.nextInt();
      if (number > 10) {
```

```
            System.out.println("The number is greater than 10.");
        } else {
            System.out.println("The number is not greater than 10.");
        }
            scanner.close();
    }
}
```

OUTPUT:

```
java -cp /tmp/9B1TB4Qw9T naveen
Naveen Kumar Enter a number:
5
The number is not greater than 10.
```

IF-ELSE-IF LADDER

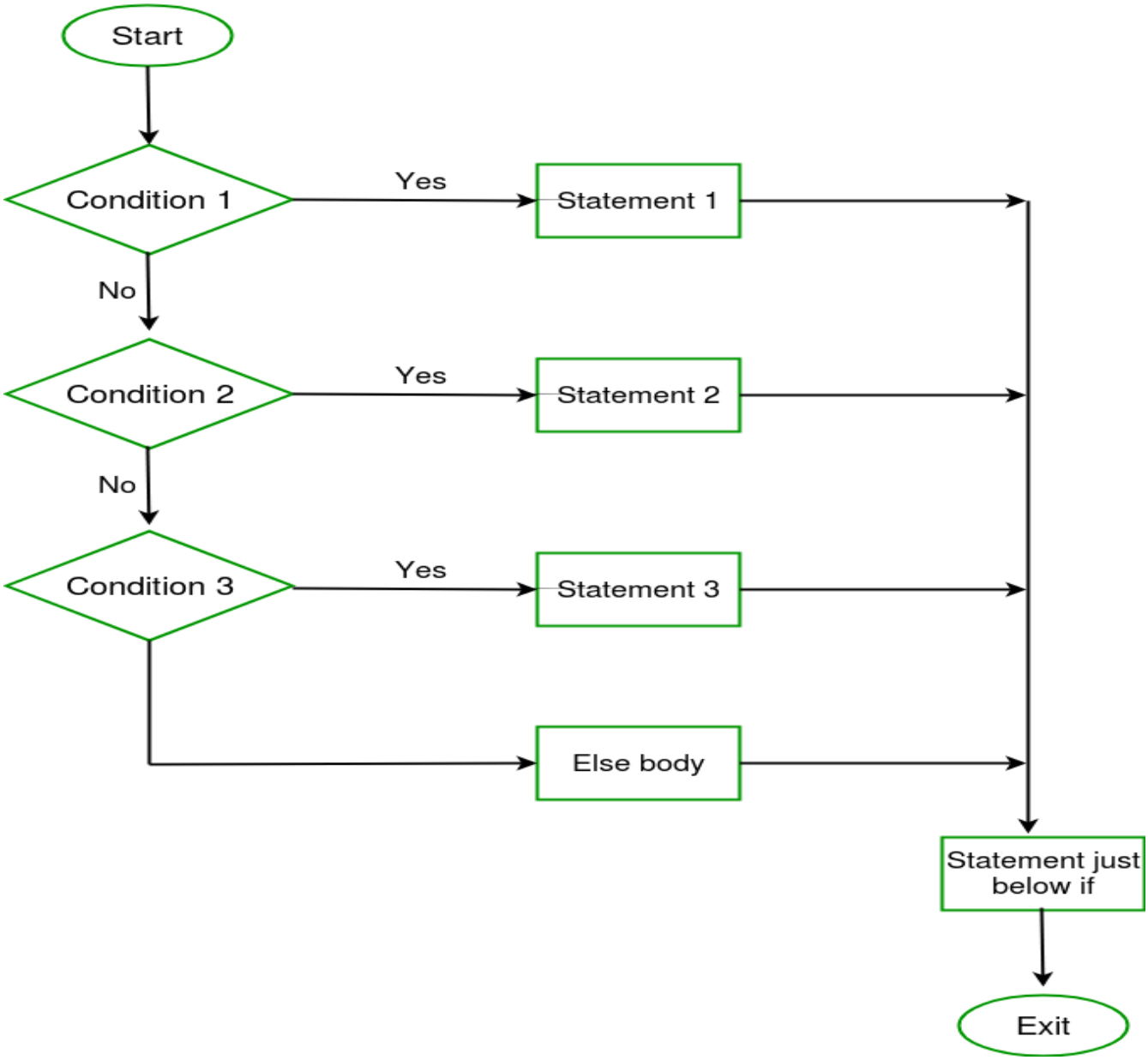
It is used to decide among multiple options. The if statements are executed from top down. As soon as one of the conditions controlling the if is true, the statements of that block are executed, and the rest of the ladder is bypassed.

SYNTAX :

```
if(condition)
{
Statements;
}
else if(condition)
{
Statements;
}
else
{
```

Statements;
}

CONTROL FLOW:



PROGRAM:

```
//if-else-if ladder

import java.util.Scanner;
public class java {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Naveen Kumar Enter a number: ")
      int number = scanner.nextInt();
        if (number > 10) {
         System.out.println("The number is greater than 10.");
      } else if (number < 10) {
         System.out.println("The number is less than 10.");
      } else {
         System.out.println("The number is equal to 10.");
      }
   }
}
```

OUTPUT:

```
java -cp /tmp/9B1TB4Qw9T java
Naveen Kumar Enter a number:
10
The number is equal to 10.
```

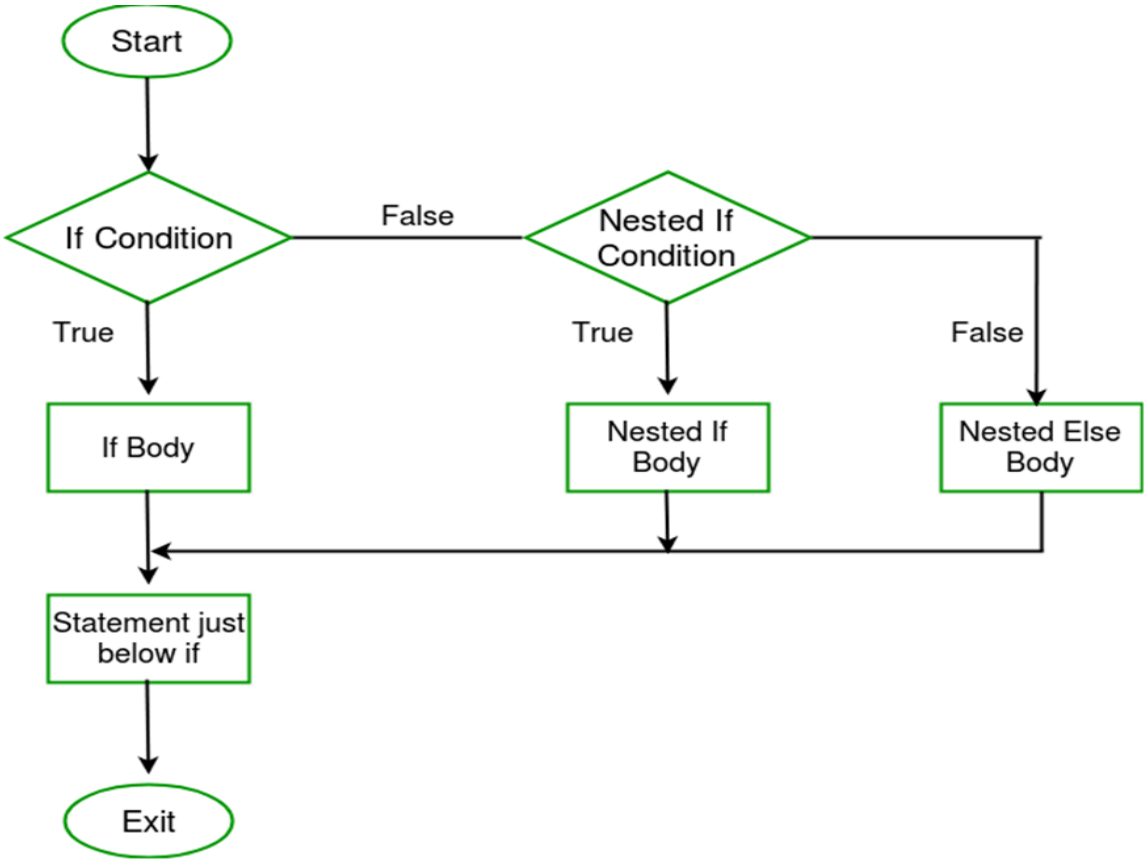
NESTED IF-ELSE

As if present inside an if block is known as a nested if block. It is similar to an if-else statement, except they are defined inside another if-else statement

SYNTAX :

```
if (condition)
{
        if (condition)
        {
        Statements;
        }
        else
        {
        Statements;
        }
}
else
{
Statements;
}
```

CONTROL FLOW:



PROGRAM:

```
//Nested if-else

import java.util.Scanner;
public class Next {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Naveen Enter your age: ");
      int age = scanner.nextInt();
      if (age >= 18) {
         System.out.println("You are eligible to vote.");
         System.out.print("Have you registered to vote? (yes/no): ");
         scanner.nextLine(); // consume the newline character left by nextInt()
         String response = scanner.nextLine();
         if (response.equalsIgnoreCase("yes")) {
            System.out.println("Great! You can participate in the elections.");
         } else {
            System.out.println("Please register to vote to participate in the elections.");
         }
      } else {
         System.out.println("You are not eligible to vote yet.");
      }
   }
}
```

OUTPUT:

```
java -cp /tmp/9B1TB4Qw9T Next
Naveen Enter your age: 19
You are eligible to vote.
Have you registered to vote? (yes/no): yes
Great! You can participate in the elections.
```

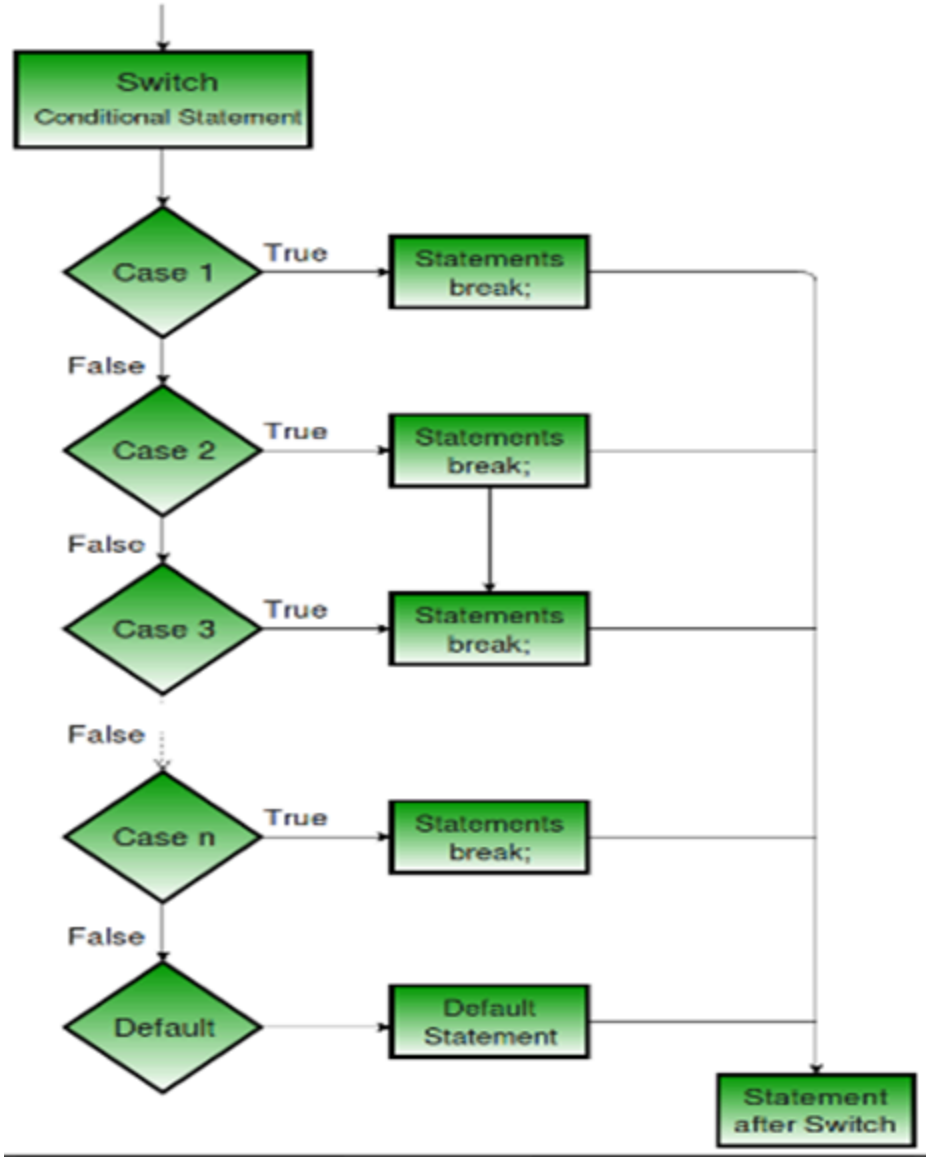
SWITCH STATEMENT

A switch statement in java is used to execute a single statement from multiple conditions. The switch statement can be used with short, byte, int, long, enum types, etc. One or N number of case values can be specified for a switch expression. Case values that are duplicate are not permissible. A compile-time error is generated by the compiler if unique values are not used. The case value must be literal or constant. Variables are not permissible. Usage of break statement is made to terminate the statement sequence. It is optional to use this statement. If this statement is not specified, the next case is executed.

SYNTAX :

```
switch(expression)
{
case 1: Statements;
        break;
case 2: Statements;
        break;
case n: Statements;
        break;
default: Statements;
        break;
}
```

CONTROL FLOW:



PROGRAM:

```
//To check given character is vowel or not

import java.util.Scanner;
```

```
public class VowelChecker {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Naveen  Enter a character: ");
      char ch = scanner.next().charAt(0);

      switch (ch) {
         case 'a':
         case 'A':
         case 'e':
         case 'E':
         case 'i':
         case 'I':
         case 'o':
         case 'O':
         case 'u':
         case 'U':
            System.out.println(ch + " is a vowel.");
            break;
         default:
            System.out.println(ch + " is not a vowel.");
      }
   }
}
```

OUTPUT:

```
java -cp /tmp/8iOv3iAKYm VowelChecker
Naveen  Enter a character: A
A is a vowel.
```

LOOPS

- while()
- do-while()
- for()
- for each()

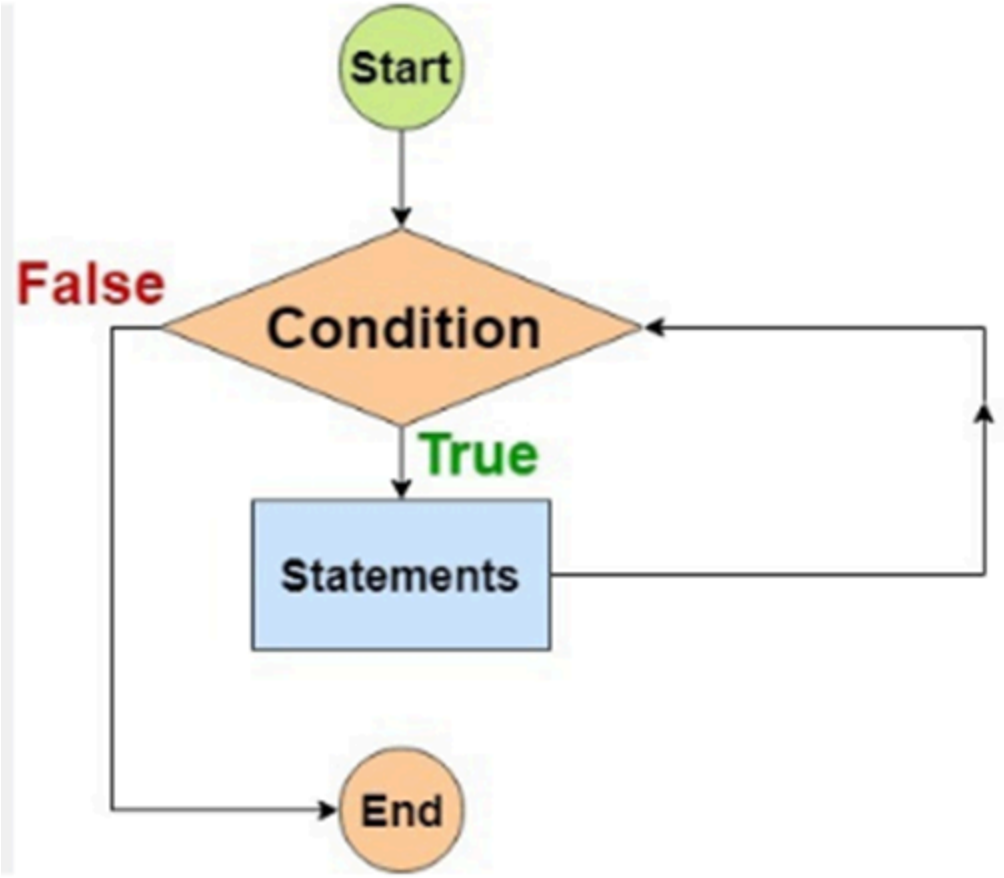
WHILE LOOP

while is known as the most common loop, the while loop evaluates a certain condition. If the condition is true, the code is executed. This process is continued until the specified condition turns out to be false.

SYNTAX :

```
initialization;
while(condition)
{
Statements;
increment/decrement;
}
```

CONTROL FLOW:



PROGRAM:

```
//To print n natural numbers

import java.util.Scanner;
public class NaturalNumbers{
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Naveen Enter the value of n: ");
      int n = scanner.nextInt();
```

```
        int i = 1;
        System.out.println("First " + n + " natural numbers:");
        while (i <= n) {
            System.out.println(i);
            i++;
        }
    }
}
```

OUTPUT:

```
java -cp /tmp/8iOv3iAKYm NaturalNumbers
Naveen Enter the value of n: 7
First 7 natural numbers:
1
2
3
4
5
6
7
```

PROGRAM:

```
//Factorial of a given number


import java.util.Scanner;
public class FactorialCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Naveen Enter a number: ");
```

```
        int n = scanner.nextInt();

        if (n < 0) {
            System.out.println("Please enter a number");
        } else {
            long factorial = 1;
            int i = 1;
            while (i <= n) {
                factorial *= i;
                i++;
            }
            System.out.println("Factorial of " + n + " is: " + factorial);
        }
    }
}
```

OUTPUT:

```
java -cp /tmp/8iOv3iAKYm Factorial
Naveen Enter a number: 10
Factorial of 10 is: 3628800
```

PROGRAM:

```
//Sum of n natural numbers

import java.util.Scanner;
```

```
public class SumOfNaturalNumbers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Naveen Enter a positive integer n: ");
        int n = scanner.nextInt();
        if (n <= 0) {
            System.out.println("Please enter a positive integer.");
        } else {
            int sum = 0;
            int i = 1;
            while (i <= n) {
                sum += i;
                i++;
            }
            System.out.println("Sum of first " + n + " natural numbers is: " + sum);
        }
    }
}
```

OUTPUT:

```
java -cp /tmp/8iOv3iAKYm SumOfNaturalNumbers
Naveen Enter a positive integer n: 20
Sum of first 20 natural numbers is: 210
```

PROGRAM:

//To display fibonacci series

```
import java.util.Scanner;
public class FibonacciSeries {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Naveen Enter the number for Fibonacci series: ");
      int limit = scanner.nextInt();
      int num1 = 0, num2 = 1;
      int count = 0;
      System.out.println("Fibonacci Series up to " + limit + " terms:");
      while (count < limit) {
         System.out.print(num1 + " ");
         int temp = num1 + num2;
         num1 = num2;
         num2 = temp;
         count++;
      }
   }
}
```

OUTPUT:

```
java -cp /tmp/8iOv3iAKYm FibonacciSeries
Naveen Enter the number for Fibonacci series: 13
Fibonacci Series up to 13 terms:
0 1 1 2 3 5 8 13 21 34 55 89 144
```

PROGRAM:

```
//To check given number is prime or not
```

```
import java.util.Scanner;
public class PrimeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Naveen Enter a positive integer: ");
        int number = scanner.nextInt();
        if (number <= 1) {
            System.out.println(number + " is not a prime number.");
        } else {
            int divisor = 2;
            boolean isPrime = true;
            while (divisor * divisor <= number) {
                if (number % divisor == 0) {
                    isPrime = false;
                    break;
                }
                divisor++;
            }
            if (isPrime) {
                System.out.println(number + " is a prime number.");
            } else {
                System.out.println(number + " is not a prime number.");
            }
        }
    }
}
```

OUTPUT:

```
java -cp /tmp/8iOv3iAKYm PrimeChecker
Naveen Enter a positive integer: 30
30 is not a prime number.
```

DO-WHILE LOOP

The do-while loop is similar to the while loop, the only difference being that the condition in the do-while loop is evaluated after the execution of the loop body. This guarantees that the loop is executed at least once.

SYNTAX :

Initialization;

do

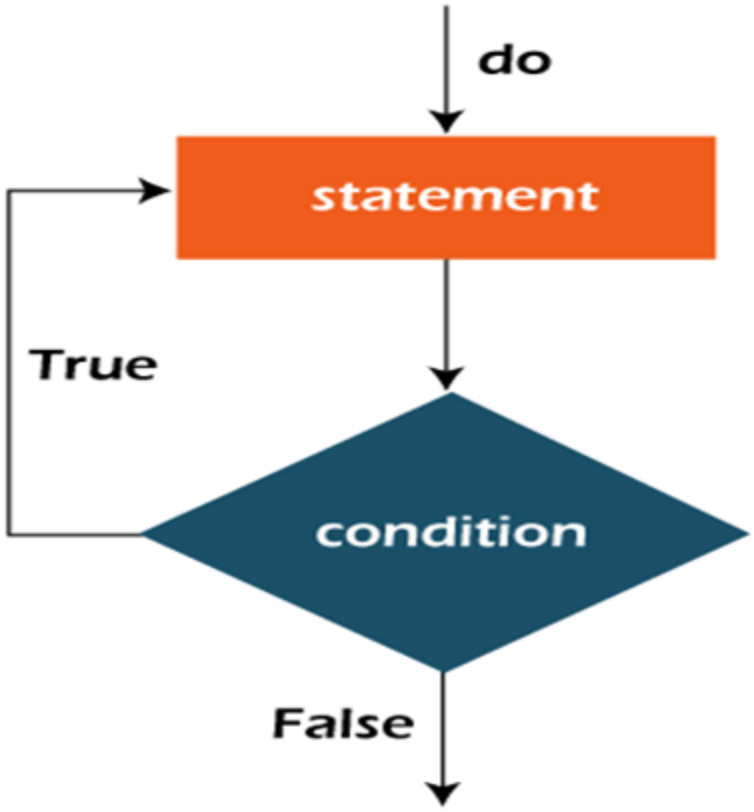
{

Statements;

increment/decrement;

}while(condition);

CONTROL FLOW:



PROGRAM:

```
import java.util.Scanner;
public class HelloPrinter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Naveen Enter the number of times to print 'Hello': ");
        int n = scanner.nextInt();
        int count = 0;
        do {
            System.out.println("Hello");
            count++;
        } while (count < n);
    }
}
```

OUTPUT:

```
java -cp /tmp/8iOv3iAKYm HelloPrinter
Naveen Enter the number of times to print 'Hello': 7
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```