Unit 1
The Basics of a Java Program

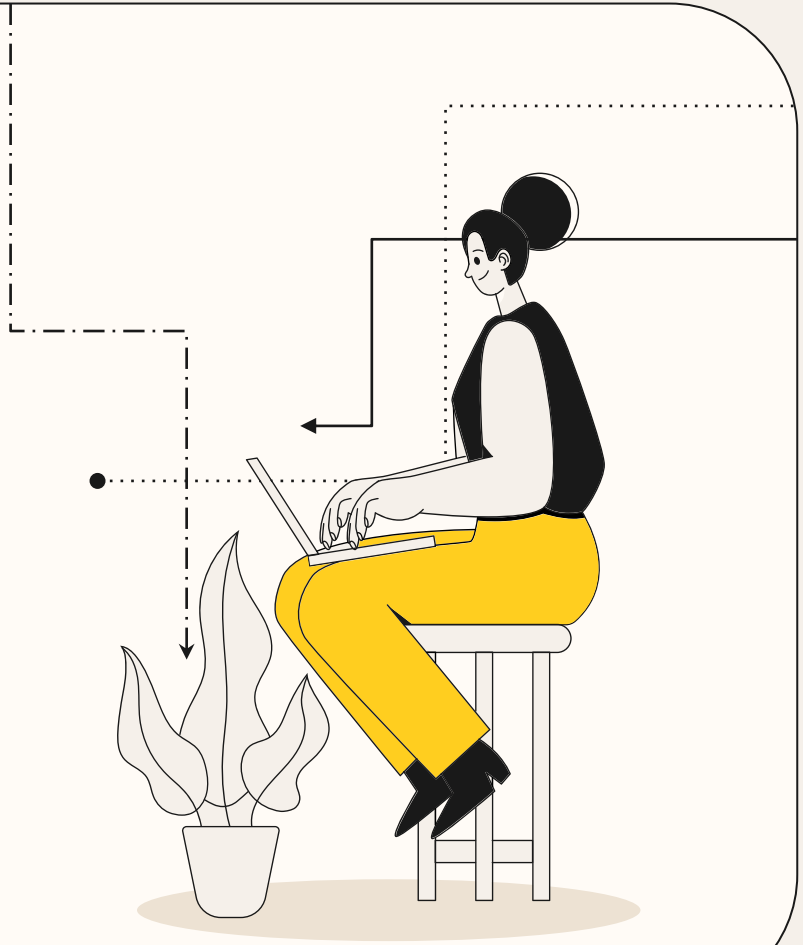
TABLE OF CONTENTS

Did you know?

Java is used by many large companies, including Amazon, Google, and Netflix, to develop their software systems.

1.

My First Java Program



Program Source Code

- It is a programming statement that is created by a programmer with a text editor or a visual programming tool and then saved in a file.

```
/* My first Java Program */

public class Hello{
     public static void main(String[] args){

          //prints the string hello world on the screen
          System.out.println("Hello World!");
     }
}
```

Types of Errors

Syntax Errors: are errors in form. These are encountered if a programmer inadvertently committed an error in typing the source code.

```
/* My first Java Program */

Public class Hello{

      public static void main(String[] args) {

            //prints the string hello world on the screen
            System.out.println("Hello World!")

      }

}
```

Capital letter P on keyword public

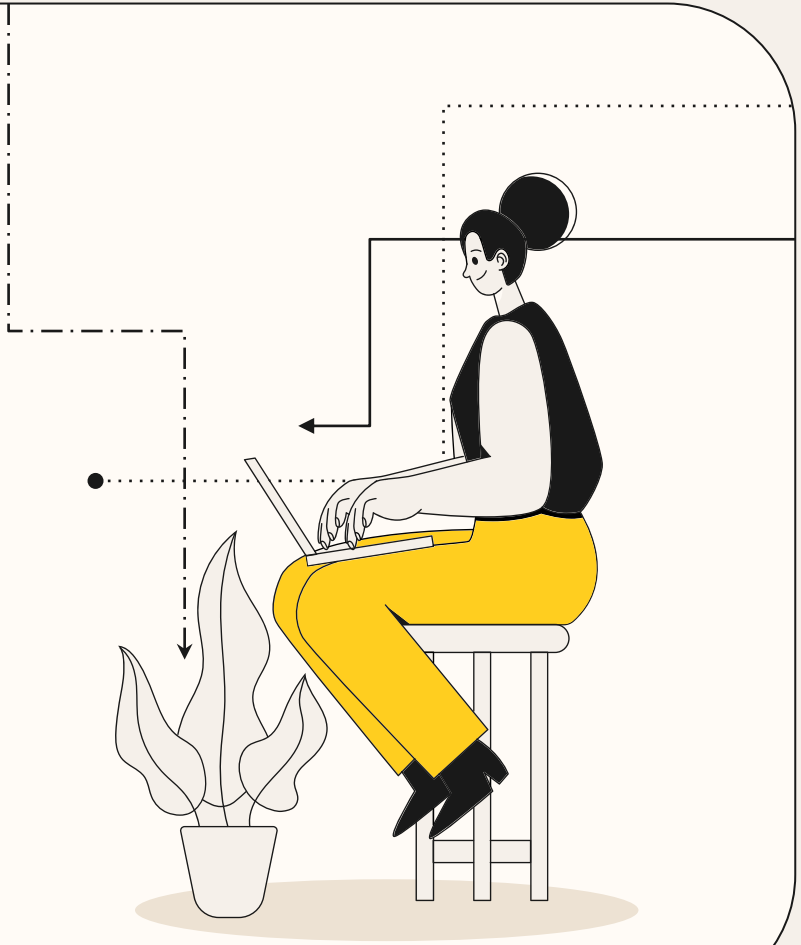
Omission of semicolon (;) at the end of the statement

Types of Errors

Run-time error: Run–time errors are errors in meaning (semantic). These are sometimes referred to as logical errors.

2.

The Java Program Structure



Comment/Remark

- A comment is used to document a part of the code. It is not part of the program itself and is only used for documentation to make it more readable.

```
/* My first Java Program */

public class Hello
{
      public static void main(String[] args){
            //prints the string hello world on the screen
            System.out.println("Hello World!");
```

Java Statement and Blocks

The statement or an instruction is one or more lines of code terminated by a semicolon.

A block is one or more statements bounded by open and closed curly braces that group the statements as one unit.

Statements

```
public static void main(String[] args){
 System.out.println("Hello");
 System.out.println("World!");
}
```

Block

Java Identifiers

- Identifiers are tokens. These are usually user-defined names that represent labels of variables, methods, and classes to name a few.

Rules for Identifiers in Java

1. The identifier should contain only

Letters [(a to z) and (A to Z)]
Digits (0 to 9)
Special characters (only $ or _)

Rules for Identifiers in Java

2. Identifiers should not start with a digit.

But a digit can be used from the second character onwards.

```
1number // invalid
n1 // valid
num1 // valid
temp122 // valid
```

Rules for Identifiers in Java

3. Java identifiers should not contain any special character except ‘$’ and ‘_’.

For example:- num$, number_one, num$_ are valid Java identifiers but num@, #num$_ are illegal Java identifiers.

Rules for Identifiers in Java

4. Java identifiers should not contain any space.

First Number // invalid
First_Number // valid
First$Number // valid

Rules for Identifiers in Java

5. Java identifiers are case sensitive.

The lowercase is different from the upper case.

```
int number = 10;
int NuMber = 20;
System.out.println(number); // 10
System.out.println(NuMber); // 20
```

Java Coding Convention

1. **For names of classes, capitalize the first letter of the class name.**

2. **The word’s first letter should start with a small letter for method and variable names.**

3. **For multi-word identifiers, capitalize the first letter of each word except the first word.**

Valid Identifiers

a
A
age
num1
xyz
final_grade

employee33
id_Number
firstName
sum
aReA5b3h1

Invalid Identifiers

6	Must start with a letter	
7	1st_Name	Must start with a letter
8	u&me	& symbol is not a letter or number
9	percent%	% symbol is not a letter of a number
10	last name	Space is not a letter or number

Java Keywords

Java Keywords				
abstract	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

5 Kinds of Java Literals

Literals: Literals in Java are a synthetic representation of boolean, character, numeric, or string data

1. **Integral Literals:** a numeric value(associated with numbers) without any fractional or exponential part

Decimal (base 10) //12
Hexadecimal (base 16) //0xC
Octal (base) //014

5 Kinds of Java Literals

2. **Floating Point Literals:** Represent decimals with fractional parts that can be expressed in standard or scientific notations.

Example:

```
3.1416
54.567
5.8234e2
10.2000e4
```

5 Kinds of Java Literals

3. Boolean Literals

Boolean literals have only two (2) values.
It is either *“True”* or *“False”.*

5 Kinds of Java Literals

4. Character Literals

Characters represent a single Unicode character.

Example:

```
'a'     Letter a
'Z'     Letter Z
'\n'    New line character
'\b'    Carriage return character
```

5 Kinds of Java Literals

5. String Literals

Represents multiple characters and are enclosed by double quotes.

Example:

```
"Hello World"
"Java Programming"
```

Primitive Data Type

- a primitive data type is a basic data type that is provided by the programming language and is not derived from any other type.

There are eight primitive data types in Java

- **byte:** A byte is an 8-bit signed integer.
- **short:** A short is a 16-bit signed integer.
- **int:** An int is a 32-bit signed integer.
- **long:** A long is a 64-bit signed integer.
- **float:** A float is a 32-bit floating-point number.
- **double:** A double is a 64-bit floating-point number.
- **boolean:** A boolean can have two possible values: true or false.
- **char:** A char is a 16-bit Unicode character.

Program Variables

- Variables are entities where data can be stored into it.
- Values stored in the variable can be changed anytime. It is an abstraction of the computer memory cell or collection of cells

Let’s try!

In your Java workbook, kindly practice and do the
2.6.1
2.6.2
2.6.3

In pages 16 to 18

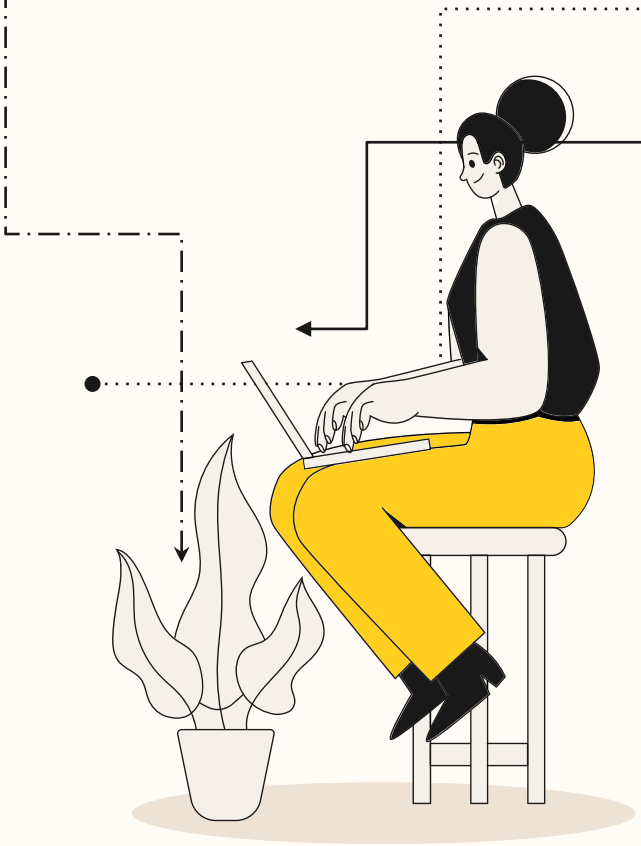
INTRODUCTION TO JAVA

UNIT 1

TABLE OF CONTENTS

1.

Brief History of Java



James Gosling: created the Oak programming language

Java: created as a platform-independent language

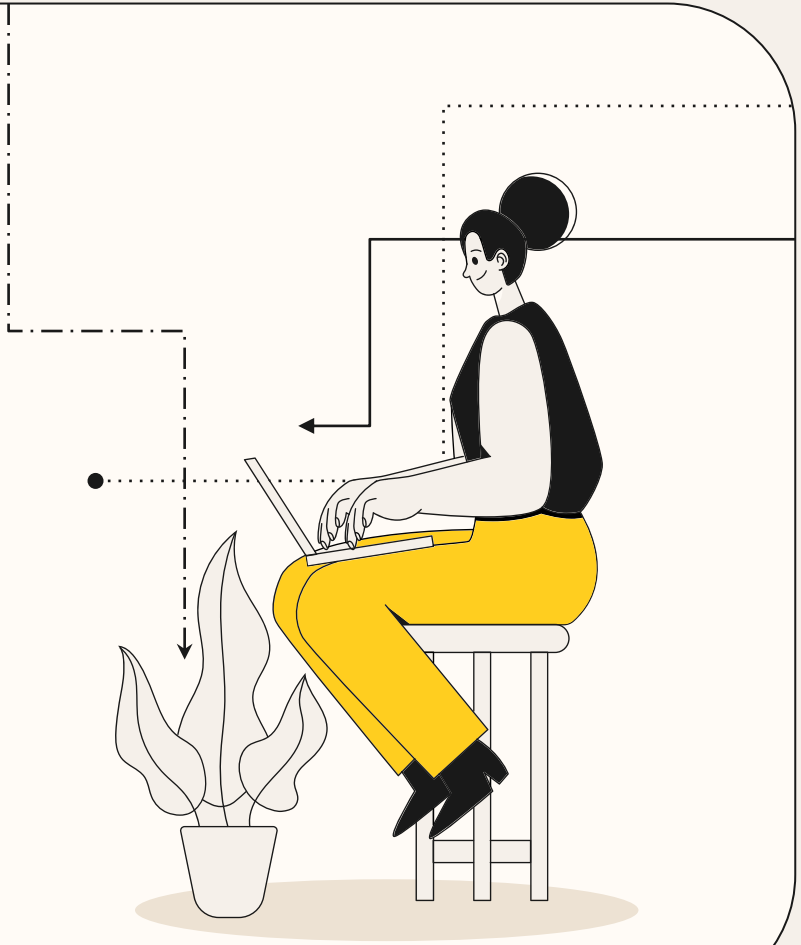
Star 7: first project developed using Java

Did you know?

Java runs on 3 billion devices worldwide!

2.

The Java Technology



Programming Language: Java can generate all kinds of applications that can be created using conventional programming languages.

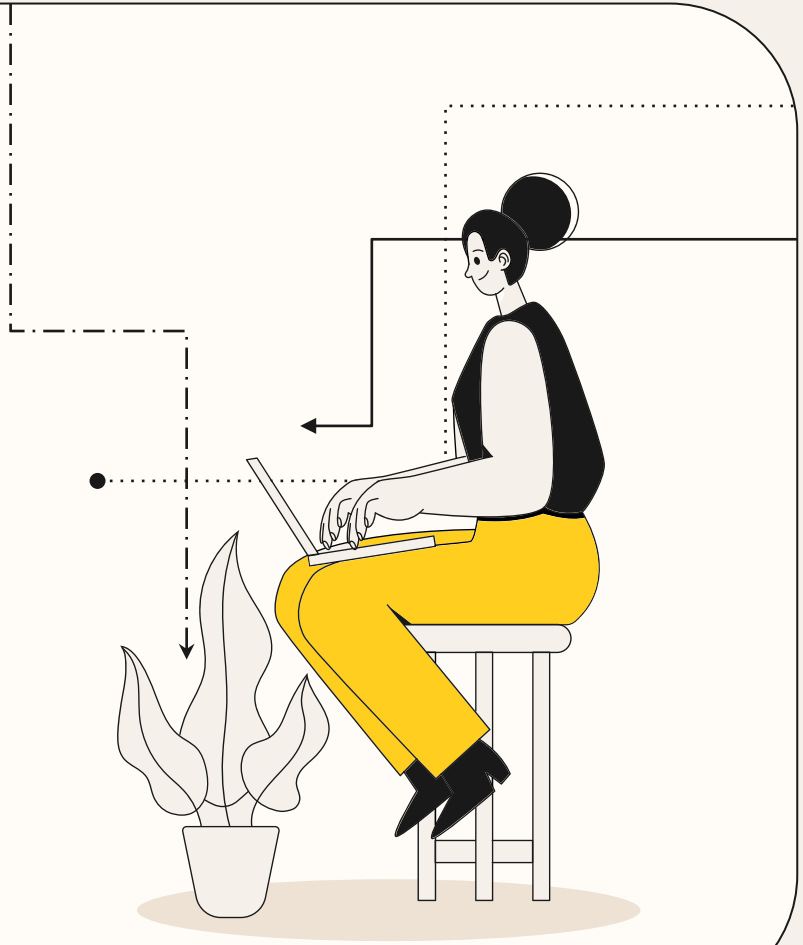
Development Language: Java provides the developer tools such as a compiler, an interpreter, a document generator, and a class file packaging tool.

Application Environment: Java applications are general-purpose programs running on a machine where the Java runtime environment (JRE) is installed.

Deployment Environment: refers to the environment or context in which a Java application is deployed or installed and made available for use by end-users.

3.

Features of Java



Java Virtual Machine (JVM)

is an interpreter that executes Java bytecode. When a Java program is compiled, it is converted into bytecode, which is a platform-independent code that can be executed by any JVM.

Garbage Collection

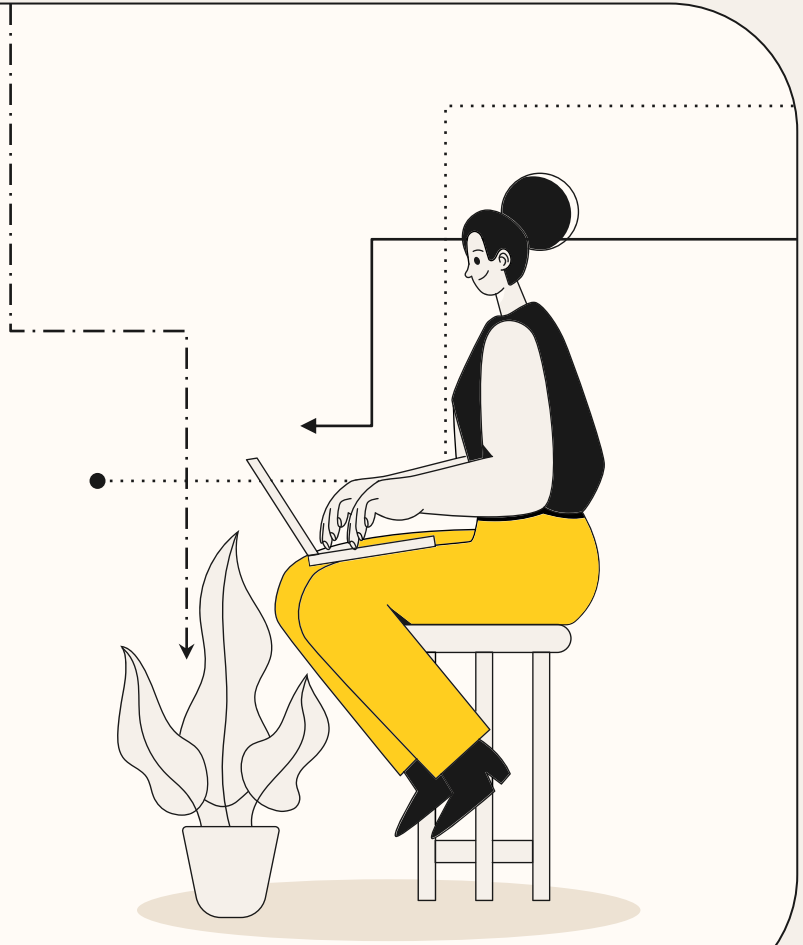
The garbage collection thread is responsible for deallocating memory previously allocated by the programmer.

Code Security

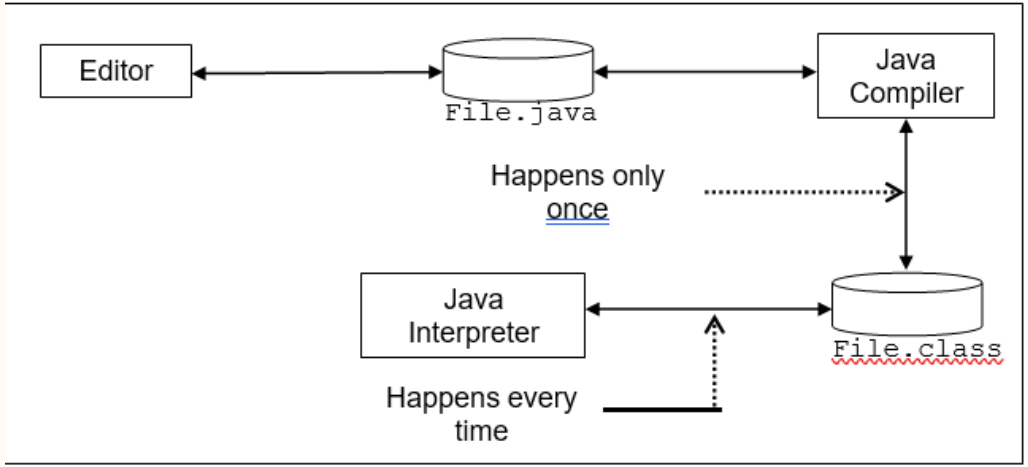
refers to the measures that are taken to ensure that Java code is secure and protected from unauthorized access or modification.

4.

Phases of Java



Phase 1: EDIT
Phase 2: COMPILE
Phase 3: LOAD
Phase 4: VERIFY
Phase 5: EXECUTE



Java Operators

Table of Contents

Introduction to Java Operators

- Operator – symbol(s) representing operation that can be performed on constants and variables
- Types of Operators
  - Assignment
  - Arithmetic
  - Increment and decrement
  - Relational
  - Logical
  - Conditional

Assignment Operator

Assignment Operator

- Uses an equal (=) symbol
- Stores or assigns a value from the right-hand side to the left-hand side
- Assignment operation is also called assignment statement, so it should end in a semicolon

```
<variable name> = <expression>;
```

Assignment Operator

- Left-hand side "gets the value" of the right-hand side
- Do not use "equal to" as it is not interchangeable
- It is also possible to assign a value of a variable to another variable of a compatible data type

```
X = 5;
Y = X;    // Y = 5
Z = Y;    // Z = 5
```

Assignment – Storing Values with Different Data Types

`int` to `double/float` and vice versa

- `int` to `double / float` – any decimal is dropped and the whole number is only retained; no rounding off

```
int i = 14. 50987;        // i = 14
```

- `double / float` to `int` – zeros are added according to the data type

```
double d = 14;        // d = 14.0
```

Arithmetic Operators

Arithmetic Operators

- Also known as mathematical operators

Symbol	Meaning	Compatible Data Types	Remarks
+	Addition	int, float, double	
-	Subtraction		
*	Multiplication		
/	Division		For `int`, any decimal is dropped
%	Modulo	int	

Arithmetic – Precedence and Associativity Rule

- Operators with higher precedence must be solved first
- Operators with the same precedence are evaluated from left to right

Operator	Associativity
*, / , %	Left to Right
+, -	Left to Right

Arithmetic – Precedence and Associativity Rule

$X = 3 + \underline{6 * 2} - 5 + 10 / 2 * 8 /2 - 3$

$X = 3 + 12 - 5 + \underline{10 / 2} * 8 /2 - 3$

$X = 3 + 12 - 5 + \underline{5 * 8} /2 - 3$

$X = 3 + 12 - 5 + \underline{40 / 2} - 3$

$X = \underline{3 + 12} - 5 + 20 - 3$

$X = \underline{15 - 5} + 20 - 3$

$X = \underline{10 + 20} - 3$

$X = \underline{30 - 3}$

$X = 27$

Increment and Decrement Operators

Pre-Increment Operator:

- If an Increment operator is used in front of an operand, it is called a Pre-Increment operator.

- ++x : which increments the value by 1 of ‘x’ variable.

```
x = 10;
y = ++x;
y = ?
```

Post Increment Operator:

- If an Increment operator is used after an operand, then is called Post Increment operator.

- x++ : which increase the value by 1 of variable ‘x’.

x = 10;

y = x++;

y = ?

Pre Decrement Operator:

- If a decrement operator is used in front of an operand, then it is called Pre decrement operator.

- –x : which decrease the value by 1 of variable ‘x’ .

x = 10;

y = --x;

Y = ?

Post Decrement Operator:

- If a decrement operator is used after an operand, then it is called Post decrement operator.

- x- – : which decrease the value by 1 of variable ‘x’ .

x = 10;

y = x--;

Y = ?

Relational Operators

Relational Operators

- Checks association of the left-hand side to right-hand side
- Yields a `TRUE` or `FALSE` condition only

Symbol	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Relational Operators

- Can be performed on all basic data types
- Can also be used in conjunction with other operators (conditional and logical)
- Must be enclosed in parenthesis
- Take note of the difference of the assignment (=) and relational (==) operators

Relational Operators

```
x = 8;
y = 13;

a = (x == y)          //result is FALSE
b = (x != y)    //result is TRUE
c = (x > y)           //result is FALSE
d = (x < y)           //result is TRUE
e = (x >= y)    //result is FALSE
f = (x <= y)    //result is TRUE
```

Logical Operators

Logical Operators

- Used to test multiple conditions
- Normally used in conjunction with relational operators

Symbol	Meaning
!	Logical NOT
&&	Logical AND
&	boolean Logical AND
||	Logical OR
|	boolean Logical Inclusive OR
^	boolean Logical Exclusive OR

Logical Operators

- Logical (NOT, AND, OR) supports short-circuit evaluations
- Boolen Logical (AND, Inclusive OR, Exclusive OR) evaluates every expression before a result is given
- General statement:

```
<expr1> <Logical Operator> <expr2>
```

```
(x == y) ! (a > b)
(x != y) && (a < b)
(x >= y) || (a <= b)
```

&& (Logical AND) and & (boolean Logical AND)

- The AND operator combines two expressions (or conditions) together into one condition group. Both expressions are tested separately by JVM and then && operator compares the result of both.

- If the conditions on both sides of && operator are true, the logical && operator returns true. If one or both conditions on either side of the operator are false, then the operator returns false.

&& (Logical AND) and & (boolean Logical AND)

- All expressions evaluated should be correct to be considered `TRUE`; otherwise the statement is `FALSE`

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

&& (Logical AND) and & (boolean Logical AND)

- Example:

```
if(x > y && y < z)
  System.out.println("Hello Java");
```

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

|| (Logical OR) and | (boolean Logical inclusive OR)

- The logical OR operator in Java combines two or more expressions or conditions together into a single condition group.

- The OR operator returns true if either one or both conditions returns true. If the conditions on both sides of the operator are false, the logical OR operator returns false.

|| (Logical OR) and | (boolean Logical inclusive OR)

- At least one expression evaluated should be correct for it to be considered `TRUE`; otherwise the statement is `FALSE`

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

|| (Logical OR) and | (boolean Logical inclusive OR)

- Example:

```
if(x == 1 || y == 1 || z == 1)
  System.out.println("Hello");
Syntax error..
```

expr1	expr2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

! (Logical NOT)

- The NOT operator is used to reverse the logic state of its operand. If the condition is correct, the logical NOT operator returns false. If the condition is false, the operator returns true.

! (Logical NOT)

- Unary operator
- Negates or gets the opposite of a certain result in a relational operation

expr	Result
TRUE	FALSE
FALSE	TRUE

! (Logical NOT)

- Example:

```
if(!( x > y ))
   System.out.println("Hello Java");
```

expr	Result
TRUE	FALSE
FALSE	TRUE

^ (boolean Logical exclusive OR)

- One statement must be `TRUE` and the other statement must be `FALSE`

expr1	expr2	Result
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Conditional Operators

Conditional Operator

- Conditional operators check the condition and decides the desired result based on both conditions.

Conditional Operator

Ternary operator (uses three arguments)

```
(Condition) ? expr1 : expr2
```

- `conditon` – boolean statement that determines what expression shall be used
- `expr1` – statement used if `expr1` is `TRUE`
- `expr2` – statement used if `expr1` is `FALSE`

Conditional Operator

- Example:

```
x = 20;
y = (x == 1) ? 61: 90;
System.out.println("Value of y is: " +  y);
y = (x == 20) ? 61: 90;
System.out.println("Value of y is: " + y);
```

- 1.) 2 * (3 + 4) = (2 * 3) + (2 * 4)
- 2.) 5 y + 6 * 10 / 8 * 9 + 1 – 4 + 7
- 3.) 8 – 10y / 6 * 10 + 34 – 20 * 2
- 4.) (6 x 4) ÷ 12 + 72 ÷ 8 – 9
- 5.) What is the “Motto” of UC

Getting Input from the Keyboard

CC1 – Computing Fundamentals

Table of Contents

Scanner

Scanner

is a class in the **java.util** package that allows for easy input parsing and retrieval of different data types from various input sources, such as the standard input, files, or strings.

Inputting Other Data Types

nextLine() Reads a single line value from the input.

Method	***Description***
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
next()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Steps in Using Scanner

1. We import the necessary class: Scanner from the java.util package.
2. Inside the main method, we create a Scanner object named scanner that reads from the standard input (System.in).
3. We prompt the user to enter their name by using System.out.print() to display the message.
4. We read the name input from the user using the nextLine() method of Scanner and store it in a String variable named name.

Steps in Using Scanner

4. We prompt the user to enter their age.
5. We read the age input from the user using the nextInt() method of Scanner and store it in an int variable named age.
6. We display a message that includes the entered name and age.
7. We close the Scanner using the close() method to release system resources.

```
import java.util.Scanner;

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

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        System.out.println("Hello, " + name + "! You are " + age + " years old.");

        scanner.close();
    }
}
```

Sample Program Using Scanner

```
import java.util.*; //Import the Scanner class

public class ScannerInput {
  public static void main(String[] args) {

        //Create Scanner object
      Scanner s = new Scanner(System.in);
      System.out.println("Enter username: ");

      //Read user input
      String userName = s.nextLine();
      System.out.println("Username is: " + userName);
  }
}
```

BufferedReader

BufferedReader

- a class provided in the java.io package that is used for efficient reading of characters from an input stream, such as a file, network socket, or other Reader implementations.

IOException

- is a checked exception, which means it must be declared in the method signature or caught within a try-catch block.

Features of BufferedReader

- **Efficient character reading:** BufferedReader is designed to efficiently read characters from an input stream by minimizing the number of system calls.
- **Line-by-line reading:** BufferedReader provides a convenient method called readLine() that reads an entire line of text from the input stream.
- **Read operations:** In addition to readLine(), BufferedReader offers other methods for reading characters or chunks of characters from the input stream, such as read(), read(char[] cbuf, int off, int len), and more.

Features of BufferedReader

- **Integration with other input sources:** BufferedReader can be used with various input sources. It accepts any Reader implementation, including FileReader, InputStreamReader, or any other class that extends Reader.

- **Error handling:** BufferedReader throws IOException for input/output-related errors.

Methods of BufferedReader

- **read():** This method reads a single character from the input stream and returns its integer representation.

```
int charValue = reader.read();
```

- **read(char[] cbuf):** This method reads characters into an array cbuf from the input stream and returns the number of characters read.

```
char[] buffer = new char[1024];
int numCharsRead = reader.read(buffer);
```

Methods of BufferedReader

- **read(char[] cbuf, int off, int len):** This method reads characters into an array cbuf starting at the given offset off, and reads at most len characters.

```
char[] buffer = new char[1024];
int numCharsRead = reader.read(buffer, 0, 100);
```

- **readLine():** This method reads a line of text from the input stream and returns it as a string.

```
String line = reader.readLine();
```

Note:

- **FileInputStream** and **InputStreamReader** are both classes in Java that are used for reading data from input sources, but they serve different purposes.
- If you are working with binary data or need to handle low-level byte operations, **FileInputStream** is appropriate.
- If you are working with text-based data and want to deal with characters and character encodings, **InputStreamReader** is more suitable.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class BufferedReaderExample {
    public static void main(String[] args) {
        BufferedReader reader = null;

        try {
            // Create a BufferedReader object to read from standard input (keyboard)
            reader = new BufferedReader(new InputStreamReader(System.in));

            // Read a line from the standard input
            String line = reader.readLine();
            // Process the line (or perform any desired operations)
            System.out.println("You entered: " + line);

        } catch (IOException e) {
            e.printStackTrace();
// Print the exception trace for debugging
        } finally {
            // Close the BufferedReader in the finally block to ensure it's always closed
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
```

Steps in Using BufferedReader

1. We import the necessary classes: BufferedReader, IOException, and InputStreamReader.
2. Inside the main method, we declare a BufferedReader variable called reader and initialize it as null.
3. We use a try-catch-finally block to handle any potential exceptions.
4. Within the try block, we create a BufferedReader object, reader, by wrapping an InputStreamReader object that reads from the standard input (System.in).
5. We read a line from the standard input using the readLine() method of BufferedReader and store it in a String variable called line

Steps in Using BufferedReader

6. We can process the line variable or perform any desired operations with it. In this example, we simply print the line with a message.

7. In the catch block, any IOException that occurs during the input reading process is caught and the exception trace is printed for debugging purposes.

8. In the finally block, we ensure that the BufferedReader is closed by calling its close() method. The close() method can potentially throw an IOException, so we handle it by printing the exception trace.

Scanner vs. BufferedReader

Scanner

- Scanner is primarily used for parsing and retrieving different types of input tokens.

BufferedReader

- BufferedReader is mainly used for reading lines of text efficiently.

Scanner vs. BufferedReader

Scanner

- Scanner provides built-in methods for parsing input tokens, such as nextInt(), nextDouble(), and nextLine(), which automatically handle different data types.

BufferedReader

- BufferedReader does not have built-in tokenization capabilities and mainly focuses on reading lines as strings.

Scanner vs. BufferedReader

Scanner

- Scanner does not have built-in buffering, so it may not be as efficient when reading large amounts of data.

BufferedReader

- BufferedReader provides buffering capabilities, which means it reads data from the underlying source in larger chunks, reducing the number of I/O operations and improving performance.

Scanner vs. BufferedReader

Scanner

- Scanner can read input from various sources, including the standard input (System.in), files (File or InputStream), or strings (String)

BufferedReader

- BufferedReader is typically used for reading from files or other character-based input streams.

Scanner vs. BufferedReader

Scanner

- Scanner is suitable for token-based parsing and retrieving different data types

BufferedReader

- BufferedReader is more focused on efficient line-by-line reading of text.

Introduction to Decision Control Structure

Unit 3

Table of contents

01

Introduction to algorithm and Flowchart

Algorithm

An algorithm is defined as sequence of steps to solve a problem (task)

Step 1: Start

Step 2: Create a variable to receive the user's email address

Step 3: Clear the variable in case it's not empty

Step 4: Ask the user for an email address

Step 5: Store the response in the variable

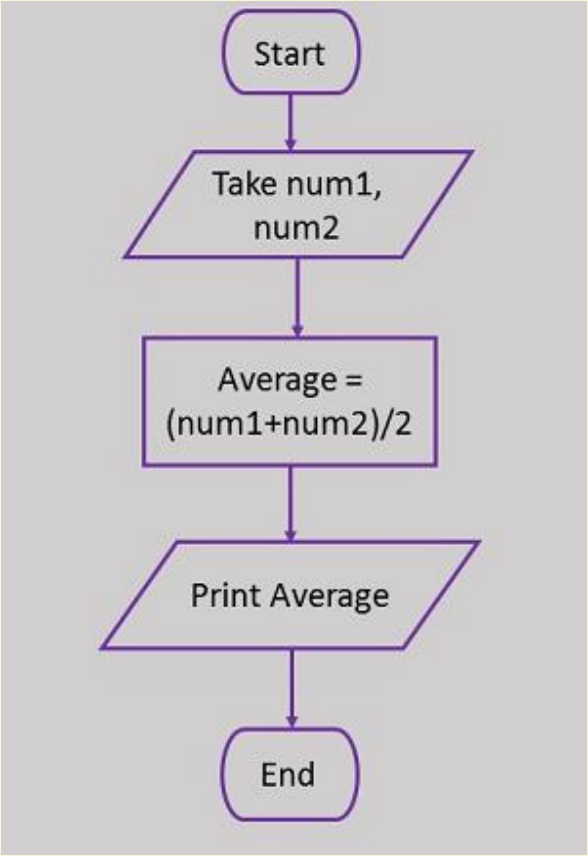
Step 6: Check the stored response to see if it is a valid email address

Step 7: Not valid? Go back to Step 3.

Step 8: End

Flowchart

A flow chart is a type of diagram that represents an algorithm, workflow or process. It shows the steps in the form of boxes of various kinds and their order by connecting them with arrows.



Flowchart building blocks

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectagle represents a process
	Decision	A diamond indicates a decision

Program

Set of instructions instructed to command to the computer to do some task.

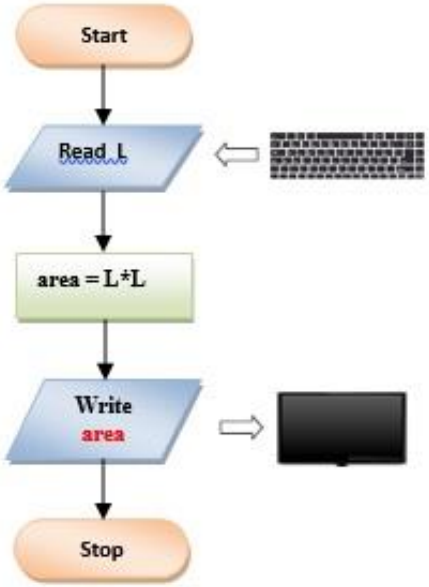
```
public class Demo {
    public static void main(String[] args) throws
        //declare new File and Scanner objects
        File file = new File("input.txt");
        Scanner inputFile = new Scanner(file);
        //loop through txt file
        while(inputFile.hasNext()){
            //read next line
            String line = inputFile.nextLine();
            System.out.print(line);
            //call check method to determine balan
            if(check(line))
                System.out.print("\t--> correct\n"
            else
                System.out.print("\t--> incorrect\
        }
        inputFile.close();
    }
```

Finding Area of the square

Algorithm

1. Start
2. Read length, L
3. area = L*L
4. Print or display area
5. Stop

Flowchart



Program

```
// Program to find area of a square

import java.util.Scanner;

public class AreaSquare{
    public static void main(String [] args){

    Scanner Ob1 = new Scanner(System.in);

    System.out.println("Enter length of sqaure L: ");
    int L = Ob1.nextInt();

    int area = L*L;

    System.out.println("Area of square is: " +area);
    }
}
```

02

Introduction to Decision Control Structure

Decision Control Structure

A statement or set of statements that is executed when a particular condition is True and ignored when the condition is False

There are the 6 ways of exercising decision making in Java:

1. if
2. if-else
3. nested-if
4. if-else-if
5. switch-case
6. jump-break, continue, return

03

If
If-else
If-else-if
Nested-if

If Statement

It encompasses a Boolean condition followed by a scope of code that is executed only when the condition evaluates to true.

However, if there are no curly braces to limit the scope of sentences to be executed if the condition evaluates to true, then only the first line is executed.

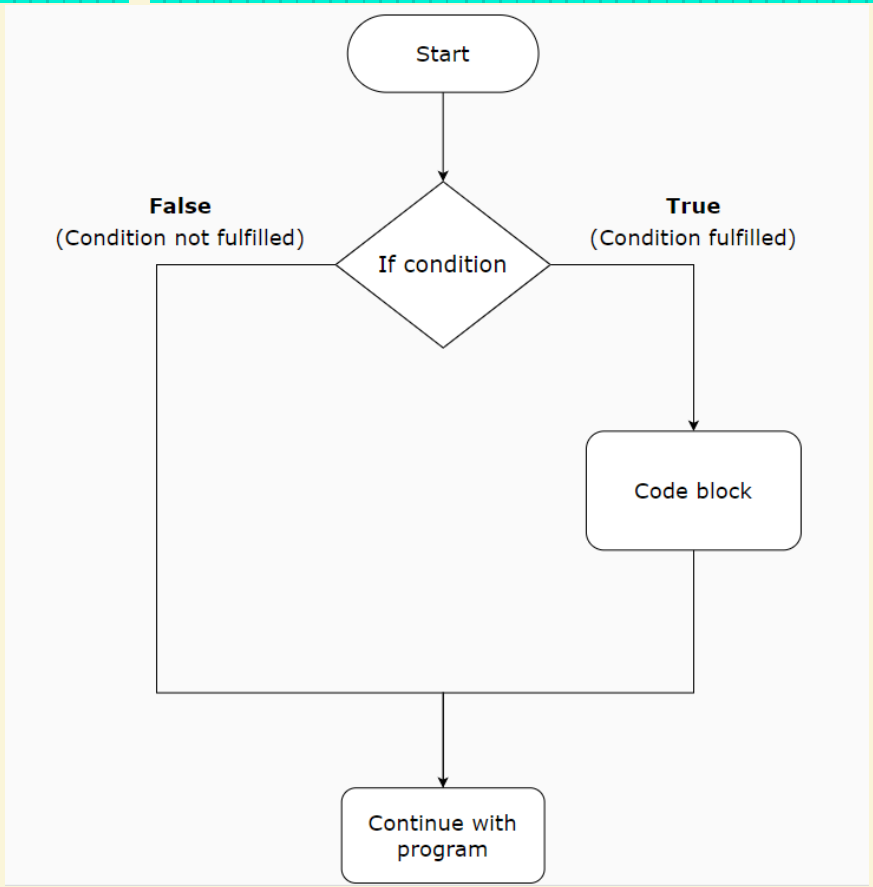
Syntax:

```
if(condition)
{

//code to be executed

}
```

If Statement



If Statement

```
if( grade >= 60 )
System.out.println( “Passed” );
```

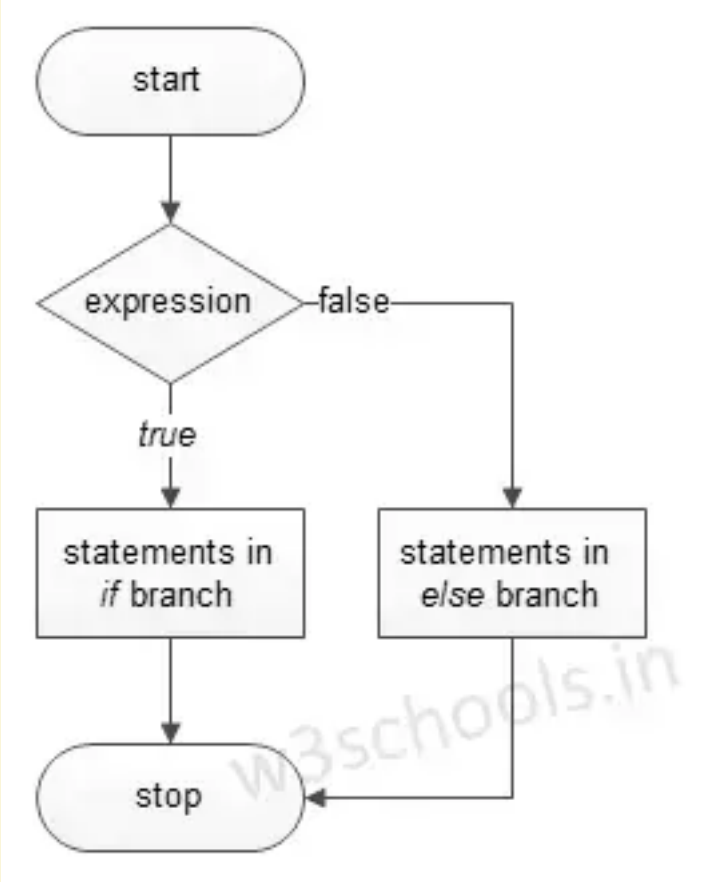
If-else Statement

This pair of keywords is used to divide a program to be executed into two parts, one being the code to be executed if the condition evaluates to true and the other one to be executed if the value is false.

Syntax:

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

If-else Statement



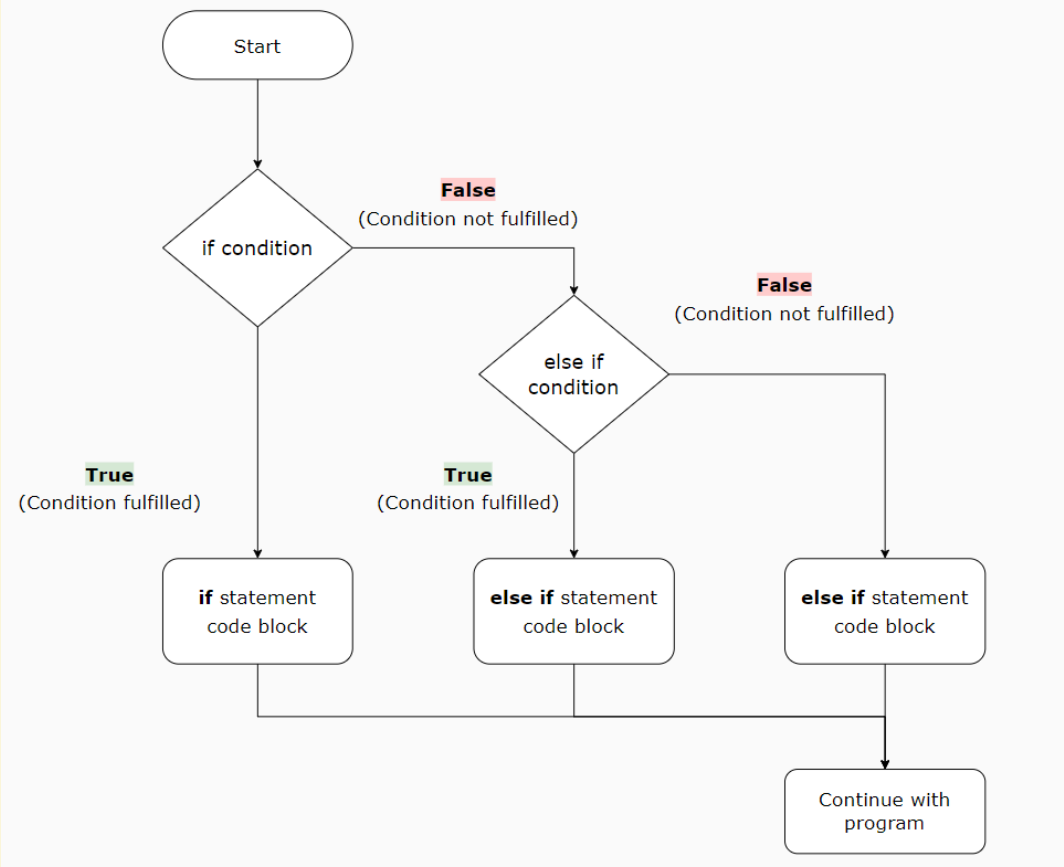
If-else-if ladder Statement

- The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that ‘if’ is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.
- There can be as many as ‘else if’ blocks associated with one ‘if’ block but only one ‘else’ block is allowed with one ‘if’ block.

Syntax:

```
if (logical expression) {
    // if statements code
    block
}
else if (logical expression) {
// else if statements code
    block
}
else {
// else statements code
    block
}
```

If-else-if Ladder Statement



Nested if Statements

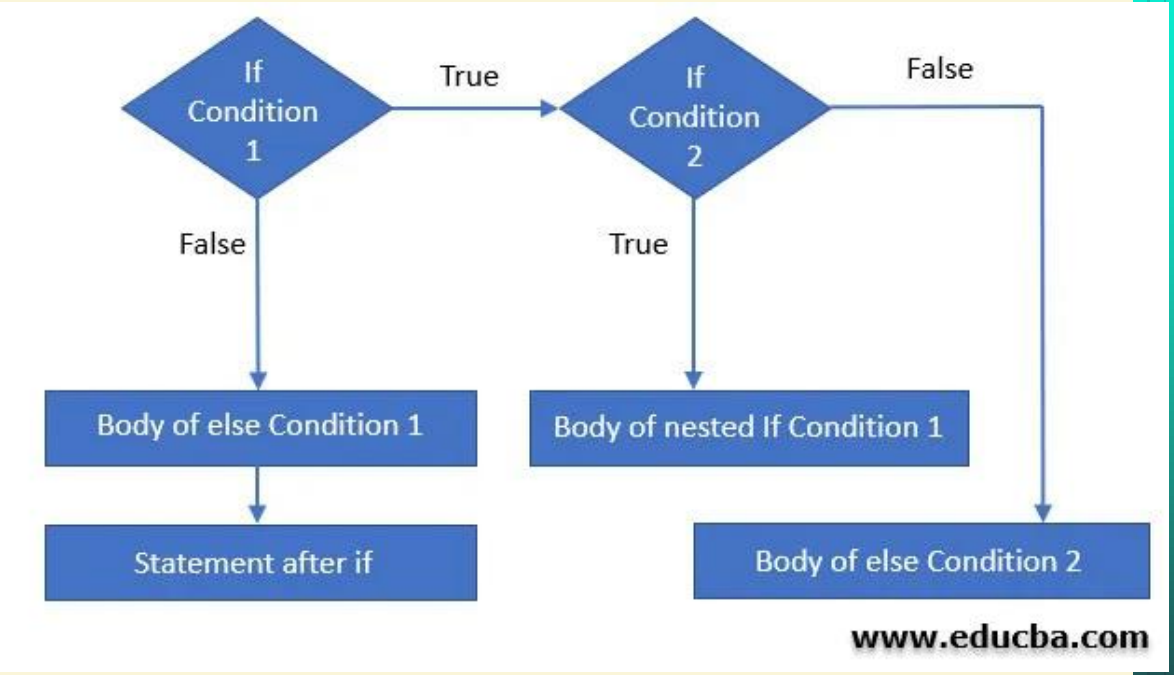
If the condition of the outer if statement evaluates to true then the inner if statement is evaluated.

Nested if's are important if we have to declare extended conditions to a previous condition

Syntax:

```
if (condition1)
{
   // Executes when
     condition1 is satisfied
   if (condition2)
   {
     // Executes when
     condition2 is satisfied
   }
}
```

Nested if Statements



Switch Statement

The switch statement is a multiway branch statement.
It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Used to execute different cases based on equality.

Syntax:

```
switch(expression)
{
case <value1>:
//code to be executed
break;
case <value2>:
//code to be executed
break;
default:
//code to be defaultly
    executed
}
```

References

https://data-flair.training/blogs/decision-making-in-java/

https://www.youtube.com/watch?v=O4KGYGQvHmw

https://javatutoring.com/java-switch-case-tutorial/