Java 17

Introduction

Welcome to the Java tutorial! Java is a high level language for building wide variety of applications. In this tutorial, we'll cover the basics of java and object oriented programming. By the end of this course, you'll have a solid understanding of Java and OOP concepts. Let's get started!

Types Of Applications

1. Console Applications

2. Desktop Applications
3. Mobile Applications
4. Web Applications
5. Web Services

Console Applications

These are text-based applications that run in a console or command-line interface. Console applications take input and produce output solely through text interaction. They are typically used for tasks that don't require a graphical user interface (GUI), such as command-line tools, batch processing, or simple text-based games. These applications can be created with Core Java libraries. Example zip/compressor.

Desktop Applications

A desktop application is a software program that runs on a user's computer and provides a graphical user interface (GUI) for interacting with the program's functionalities. Desktop applications are standalone programs that can be installed and executed directly on the operating system of a user's machine. These applications can be created with Java Swing and JavaFX Api's. Example: acrobat reader, media player, antivirus, etc

Mobile Applications

Mobile applications are used for a wide range of purposes, including games, social media, productivity tools, and more. With Java, you can develop Android applications using the Android Software Development Kit (SDK) and the Java programming language. Android apps are primarily built using Java, although newer frameworks like Kotlin are gaining popularity. Android apps run on mobile devices and can be distributed through app stores or installed directly on devices.

Web Applications

They are commonly used for e-commerce websites, content management systems (CMS), and enterprise applications. Users can access these application through web browsers. In Java web development, you can use frameworks like Java Servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF) to create dynamic and interactive web pages. Example: amazon, twitter, netflix web

Web Services

Web services enable interoperability between different platforms and facilitate integration between systems. Web services allow different applications to communicate and exchange data over a network. Java web services can be built using technologies like Java API for RESTful Web Services (JAX-RS) or Java API for XML Web Services (JAX-WS).

Software Developer Roles

1. Front-end Developer
2. Back-end Developer
3. Full-stack Developer
4. Mobile Developer
5. DevOps Engineer
6. Data Scientist

Front-end Developer

Front-end developers focus on the user interface (UI) and user experience (UX) aspects of an application. They are responsible for implementing the visual and interactive elements of a website or application using technologies like HTML, CSS, and JavaScript. Front-end developers work closely with designers to ensure that the UI design is translated into a functional and appealing user interface.

Back-end Developer

Back-end developers handle the server-side of applications, dealing with the logic and data processing that happens behind the scenes. They are responsible for developing the server-side code, managing databases, handling APIs, and integrating different systems. Back-end developers typically work with languages like Java, Python, Ruby, or PHP, and frameworks like Spring, Django, or Node.js.

Full-stack Developer

Full-stack developers have expertise in both front-end and back-end development. They can work on all layers of an application, including the UI, server-side logic, and database management. Full-stack developers are capable of handling end-to-end development tasks and have a broader understanding of how different components of an application interact.

Mobile Developer

Mobile developers specialise in creating applications for mobile devices, such as smartphones and tablets. They may focus on developing native mobile apps using platform-specific languages like Java or Kotlin for Android, or Swift or Objective-C for iOS. Alternatively, they may develop cross-platform mobile apps using frameworks like React Native or Flutter.

DevOps Engineer

DevOps engineers bridge the gap between development and operations, focusing on the deployment, integration, and automation of software systems. They work on tasks such as continuous integration and delivery (CI/CD), infrastructure management, configuration management, and monitoring. DevOps engineers ensure smooth collaboration between development and operations teams, aiming for efficient and reliable software delivery.

Data Scientist

Data scientists specialize in analyzing and extracting insights from large datasets. They have strong knowledge of statistical analysis, machine learning algorithms, and data visualization techniques. Data scientists use programming languages like Python or R to manipulate and analyze data, and they often work with tools and frameworks like TensorFlow, PyTorch, or scikit-learn.

What can you do with Java language ?

1. Console Applications
2. Desktop Applications
3. Mobile Applications
4. Web Applications
5. Web Services

What it needs to have a career in java tech stack ?

1. Beginners
    a. Language - Java
    b. Build life cycle tool - maven, gradle etc
    c. source version control - Git, subversion etc
    d. Databases
        i. SQL - Oracle, mysql
        ii. NOSQL - MongoDB, Casandra
2. Advanced
    a. Unit Testing - junit/TestNG
    b. Frameworks - Spring/ Springboot/ hibernate
    c. Design Patterns, SOLID Design principles
    d. Static Analysers - CheckStyle/ Findbugs/ PMD
    e. Web
        i. HTML/CSS/Javascript
        ii. Servlets, JSP, JavaServer Faces
        iii. MVC, RESTful Web Services
    f. Messaging and Integration: Apache Kafka or RabbitMQ
    g. Ci/CD - Jenkins
    h. Containerisation and Deployment: Docker, Kubernetes

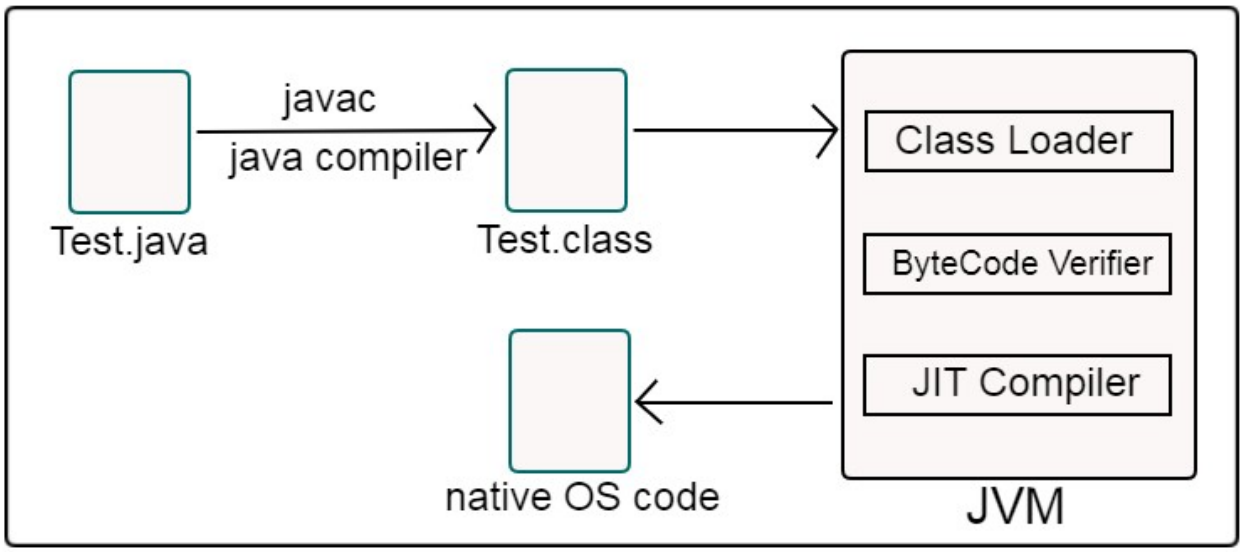
How to learn Java?

1. Learn java in high level
2. Learn logical programming (pattern printing, sorting etc)
3. Learn Data Structure & Algorithm (HackerRank, Leet code etc) - FAANG+ & Product based companies
4. Learn java in depth (Java constructs & rules) - Intermediate to Advanced (Service based companies)
5. Prepare for Java certification - Optional

What is Java ?

- Open source
- High level language
- Object oriented language
- Platform independent

Java Compilation Process



Is Java Compiled language or Interpreted language ?

Compilation: Java source code is initially compiled by the Java compiler (javac) into an intermediate form called bytecode. Bytecode is a platform-independent representation of the code and is not tied to any specific hardware or operating system.

Interpretation: The compiled bytecode is then executed by the Java Virtual Machine (JVM), which is responsible for interpreting and executing the bytecode. The JVM translates the bytecode into machine-specific instructions at runtime using Just-In-Time (JIT) compilation. This process enhances performance by translating frequently executed bytecode into native machine code.

Why java is platform independent ?

Same Java program run on different hardware and operating systems without any modification will produce the same output.

1. **Byte-code Compilation**: When you write a Java program, it's first compiled into an intermediate form called bytecode. Bytecode is a set of instructions that can be executed by the Java Virtual Machine (JVM).
2. **Java Virtual Machine (JVM)**: The JVM is a runtime environment that interprets and executes Java bytecode. **It is developed based on Java Specifications.** It is available for various operating systems, and each JVM implementation is responsible for translating bytecode into machine code specific to the underlying system.
3. **Platform-Specific JVM**: Java developers only need to write their code once in the Java programming language. The same bytecode can be executed on any JVM implementation, regardless of the underlying hardware and operating system. This means that as long as a compatible JVM exists for a particular platform, the Java program can run on it without modification.

Can you run more than one java application at the same time in a machine ?

Yes, Java code is execute in JVM. JVM is a virtual machine. Applications will run independently inside the JVM without any impact to other applications.

Install Java

https://www.oracle.com/in/java/technologies/downloads/

Java installation package has 2 components

1. Java Runtime Environment (JRE)
2. Java Development Kit (JDK)

What is JRE and JDK ?

1. JRE (Java Runtime Environment):
    a. The JRE is the package that allows you to run Java applications and applets on your machine.

    b. It includes the Java Virtual Machine (JVM), which is responsible for executing Java bytecode.
    c. **JRE doesn't contain tools and utilities for compiling or developing Java programs; it's purely for running Java applications.**
2. JDK (Java Development Kit):
    a. The JDK is a comprehensive package that includes everything you need for Java development.
    b. It contains the JRE, allowing you to run Java applications, but it also includes additional tools and libraries for writing, compiling, and debugging Java code.
    c. JDK includes the Java Compiler (javac) to convert Java source code into bytecode, and tools for packaging and distributing applications.
    d. JDK provides development resources such as header files, documentation, and examples.

Java Distributions

Java Development Kit (JDK) distributions are variations of the JDK provided by different vendors with additional features, optimisation's, or support.

1. **Oracle JDK:** This was historically the reference implementation of Java, provided by Oracle. It included both commercial and free versions. However, Oracle's new subscription-based model may require a paid license for certain use cases.
2. **OpenJDK:** This is the open-source reference implementation of the Java Platform, Standard Edition. It serves as the base for many other JDK distributions. OpenJDK is typically free to use and can be a good choice for those who prefer open-source solutions.
3. **Amazon Corretto:** Amazon Corretto is a no-cost, multi-platform, production-ready distribution of OpenJDK. It's designed to be compatible with other major JDK distributions.
4. **Azul Zulu:** Zulu is a certified, fully compliant OpenJDK build. It's available in both free and paid versions and offers features like high performance and support for various platforms.
5. **IBM JDK:** IBM provides its own JDK distribution with optimisations and features designed for enterprise use cases.

Which one to use ? Open JDK or Oracle JDK?

1. Oracle JDK - Free to use for non commercial, Provides support.
2. Open JDK - Free for commercial use, But no support.

Java versions

https://endoflife.date/java

Which version to use ?

Always upgrade to the recent LTS version.
Newer versions often come with performance improvements and enhanced security features.
Long Term Support (LTS) provides stable platform. It provides extended support for features/bugs and security fixes.

Set Path

Windows

Temporary:

```
set path=C:\Program Files\Java\jdk1.6.0_23\bin
```

Permanent:

```
Go to MyComputer properties -> advanced tab -> environment variables -> Add idk path in path variable
```

Mac & Linux

```
Create ~/.bash_profile or ~/.bashrc
export PATH="/path/to/java/bin:$PATH"
source ~/.bashrc
echo $PATH
```

Integrated development environment (IDE)

IDEs offer features like code completion, debugging, version control integration, and more, which can significantly boost your productivity as a Java developer. The choice of IDE often comes down to personal preference, team requirements, and specific project needs.

1. Eclipse: Eclipse is a highly customisable and extensible IDE. It supports a wide range of plugins, making it suitable for various Java development tasks.
2. IntelliJ IDEA: IntelliJ IDEA is known for its user-friendly interface and powerful features. It offers smart coding assistance, advanced debugging tools, and seamless integration with build systems.
3. NetBeans: NetBeans is an open-source IDE that provides a user-friendly interface and strong support for Java, JavaFX, and other languages.

Develop Hello world - Command line

Hello World

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

File: Main.java

Compilation:

javac Main.java

Execution

java Main

Day 2

Develop Hello world - IDE

1. Install the IDE
2. Create Project
3. Bind the JDK
4. Execute the project
5. Explore the project structure
6. Explore the settings

Display - Print to Console

Priint

```
public class Main {
    public static void main(String[] args) {
        // Using print to display output
        System.out.print("Hello, ");
        System.out.print("this is ");
        System.out.print("a demonstration ");
        System.out.print("of the print statement.");

                System.out.print("\n\n\n"); //newline character

        // Using println to add a newline
        System.out.println("Now let's use println for a new line.");
        System.out.println("This text is on a new line.");
        System.out.println("So is this text.");
    }
}
```

Java Comments

Comments can be used to add information about the code and to make it more readable.
Comments will be skipped and will not be executed.

1. Single-line Comments
2. Java Multi-line Comments
3. Java doc comments

Comments

```
/**
 * This class demonstrates different types of comments in Java.
 */
public class Main {

    /**
     * This method is the entry point of the program.
     * @param args Command-line arguments (not used).
     */
    public static void main(String[] args) {
        // Single-line comment
        System.out.println("Hello from single-line comment!");

        /*
         * Multi-line comment
         * This is a demonstration of
         * multi-line comments in Java.
         */
        System.out.println("Hello from multi-line comment!");

        /**
         * JavaDoc comment for the printMessage method.
         * This method prints a message to the console.
         * @param message The message to be printed.
         */
        printMessage("Hello from JavaDoc comment!");
    }

    /**
     * This method prints the given message.
     * @param message The message to be printed.
     */
    public static void printMessage(String message) {
        System.out.println(message);
    }
}
```

Logical Thinking & Programming

Scenario 1:

Find the largest number in the given list (small).

Largest Number

```
{10, 5, 20, 8, 15}
```

Scenario 2:

Find the largest number in the above list while blindfolded

Largest Number

```
{10, 5, 20, 8, 15}
```

Scenario 3:

Find the Second largest numbers in the given list (Large).

Largest Number
{10, 5, 20, 8, 15, 12, 7, 8, 9, 10}

Lab 02

Largest Number

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

        int[] numbers = new int[]{11, 2, 3, 4, 5, 20, 7, 8, 9, 10};

        int maxElement = numbers[0];

        for (int index = 1; index < numbers.length; index++) {
            if (numbers[index] > maxElement) {
                maxElement = numbers[index];
            }
        }
        System.out.println(maxElement);
    }
}
```

Lab 03

Largest Number

```
public class Main {
    public static void main(String[] args) {
        int[] numbers = {10, 5, 20, 8, 15};

        int firstLargest = Integer.MIN_VALUE;
        int secondLargest = Integer.MIN_VALUE;

        for (int num : numbers) {
            if (num > firstLargest) {
                secondLargest = firstLargest;
                firstLargest = num;
            } else if (num > secondLargest) {
                //does not handle duplicate
                secondLargest = num;
            }
        }
        if (secondLargest != Integer.MIN_VALUE) {
            System.out.println("The second largest number is: " + secondLargest);
        } else {
            System.out.println("There is no second largest number.");
        }
    }
}
```

Variables - Named Memory locations

Variable is a named memory location that holds a value of a specific data type
Each variable has a data type that determines what kind of value it can hold, such as integers, floating-point numbers, characters, etc.

Variables

```
int firstLargest = 0;
```

key components of a variable in Java:

Name or Identifier: The name of the variable is used to identify and access it within the program. Variable names should follow certain naming conventions, such as starting with a letter or underscore, and they are case-sensitive.

Data Type: The data type of a variable defines the kind of value it can hold. For example, int represents integer values, double represents floating-point numbers, char represents characters, and so on.

Value: The actual data stored in the variable is referred to as its value. This value can be assigned when the variable is declared or at a later point in the program.

Scope: The scope of a variable defines where in the program it can be accessed. Variables can have local scope (limited to a specific block of code) or broader scope (accessible throughout a class or method).

Declaration, Initialisation & Definition

1. Declare/ Declaration
    - Just Informing the compiler about the variable, Memory will not be allocated at this time.

    Declare

    ```
    int firstLargest; //No memory is allocated
    ```

2. Initialisation
    - Variable is already declared, Assigning a value for the first time. Memory gets allocated.

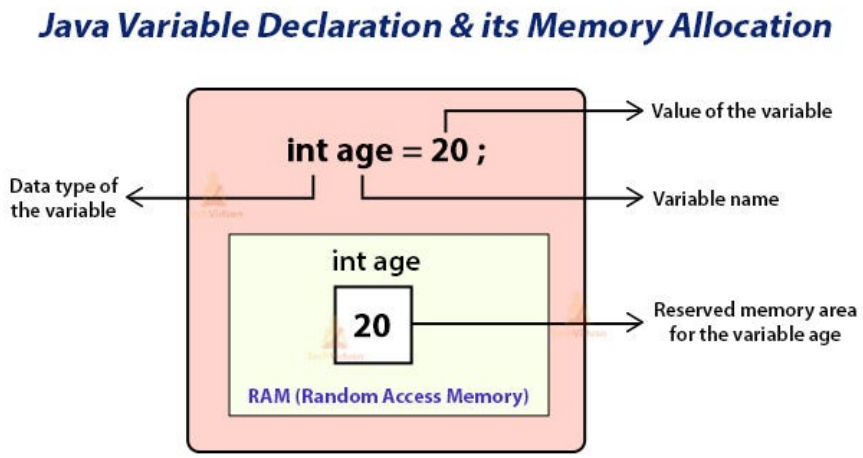
    Initilisation

    ```
    index=0; //assigning value for the first time.
    ```

3. Definition
    - Both declaration & initialisation in the same line. Memory will be allocated when this line is executed.

    Definition

    ```
    int index = 0;
    ```



Define Multiple Variables

Multiple Variables

```
int a = 5, b = 6, c = 50;
```

Identifiers

An identifier is a name used to identify a programming element such as a

1. variable
2. method
3. class or interface or other constructs

Rules for Identifiers:

- Identifiers can include letters (both uppercase and lowercase), digits (0-9), and underscores (_).
- They must start with a letter or underscore, but not with a digit.
- Identifiers are case-sensitive, meaning myVariable and MyVariable would be treated as distinct identifiers.
- Keywords (reserved words like int, class, if, etc.) cannot be used as identifiers.

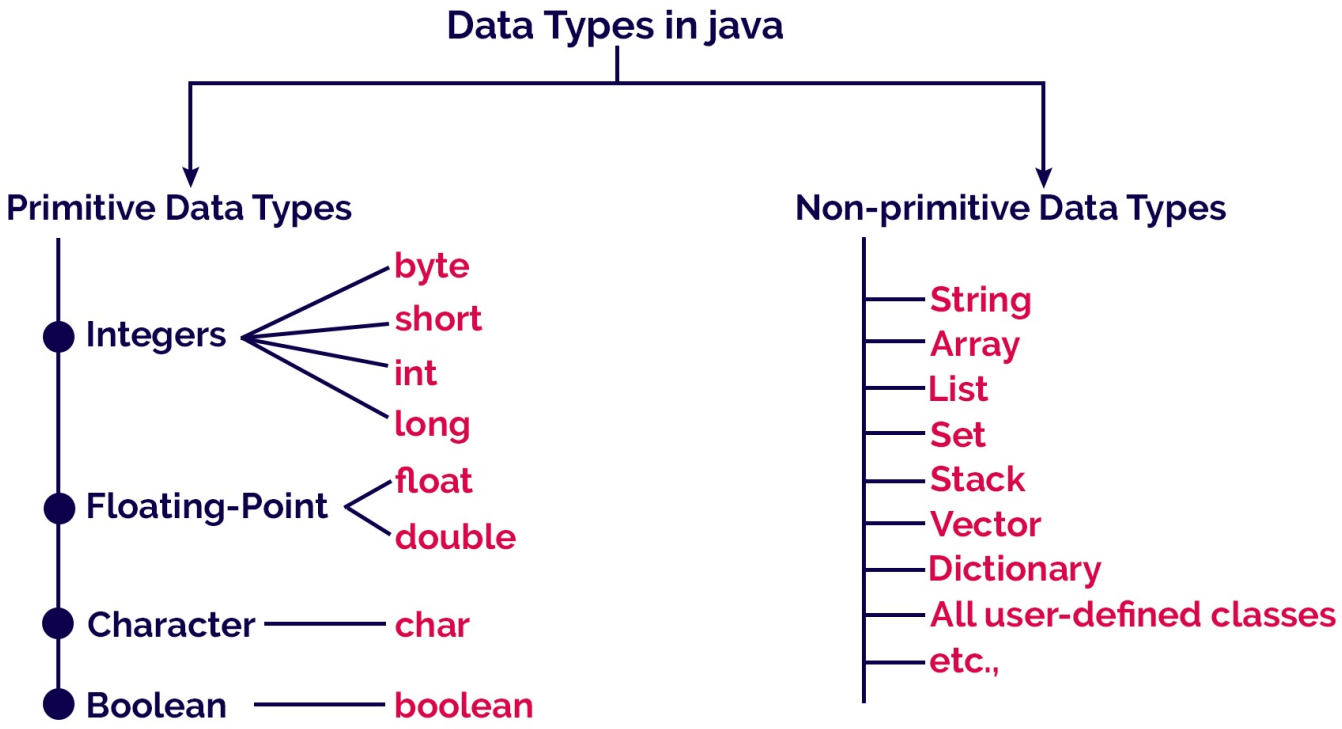
Conventions for Identifiers:

- Use camelCase for variables, methods, and parameter names (e.g., studentName, calculateTotal).
- Use PascalCase for class and interface names (e.g., StudentInfo, CalculatorUtility).
- Use all-uppercase letters with underscores for constant values (e.g., MAX_VALUE, PI).

Data Type

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.



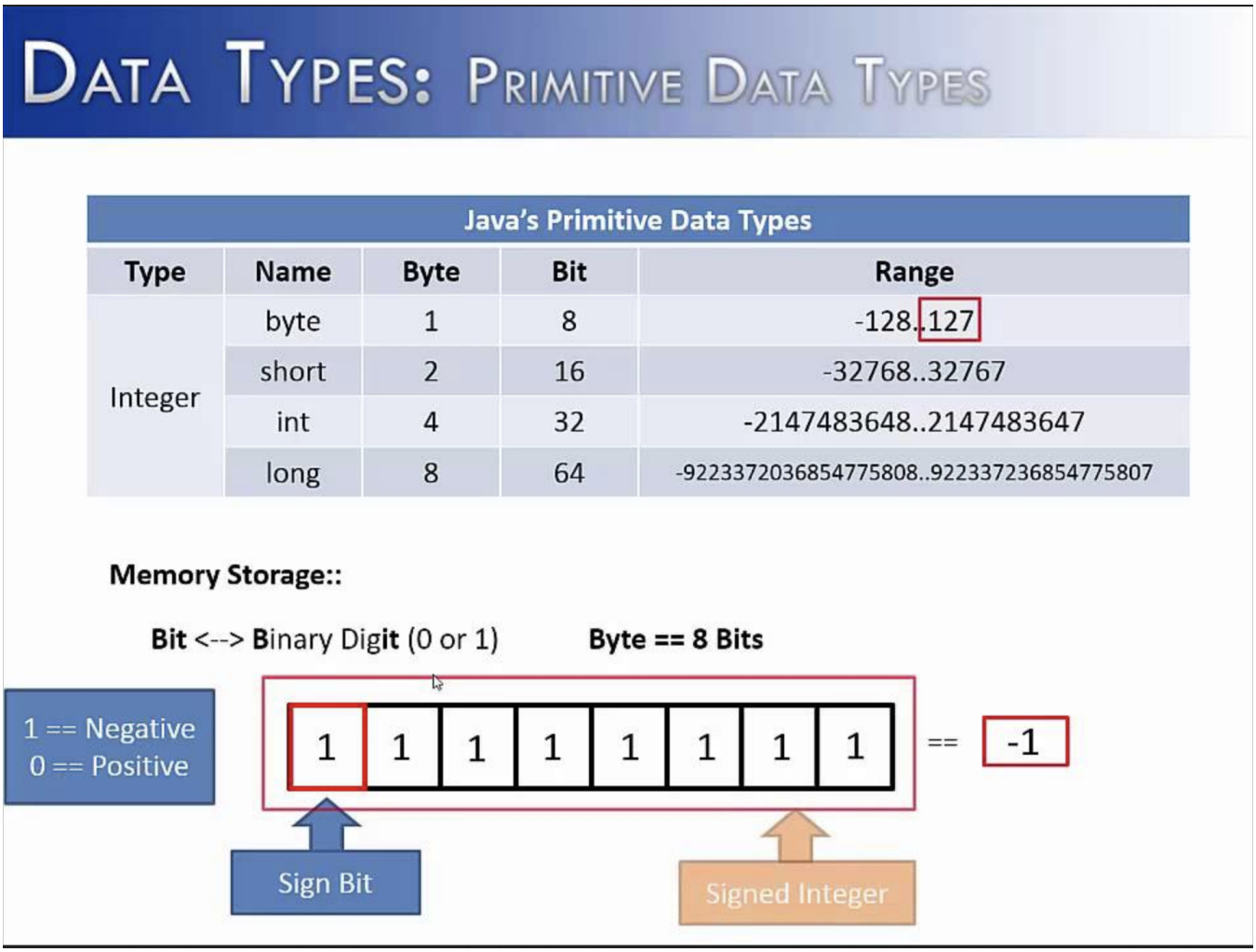
Data Type Values

Primitive Type Keyword

Type	Size in bytes	Range	Default Value
byte	1 byte	-128 to 127	0
short	2 bytes	-32,768 to 32,767	0
int	4 bytes	-2,147,483,648 to 2,147,483, 647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) Java implements IEEE 754 standard	0.0f
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)	0.0d
char	2 bytes	0 to 65,536 (unsigned)	'\u0000'
boolean	Not precisely defined*	true or false	false

Representation in Memory

```
byte someByteValue = -1;
```



Lab 04 - Data Type

Datatype

```
public class Main {
    static byte byteValue;
    static short shortValue;
    static int intValue;
    static long longValue;
    static float floatValue;
    static double doubleValue;
    static char charValue;
    static boolean booleanValue;
    public static void main(String[] args) {
        System.out.println("Default values for primitive data types:");
        System.out.println("byte: " + byteValue);
        System.out.println("short: " + shortValue);
        System.out.println("int: " + intValue);
        System.out.println("long: " + longValue);
        System.out.println("float: " + floatValue);
        System.out.println("double: " + doubleValue);
        System.out.println("char: " + charValue); // Empty output for char
        System.out.println("boolean: " + booleanValue); // false for boolean
    }
}
```

Lab 05 - Data Type - Overflow

Binary representation of 127 - 01111111

Adding 1 to it, makes it -10000000 with MSB (Sign bit) changed to 1

Data type

```
public class Main {
    public static void main(String[] args) {
        byte byteValue = 127; //Byte.MAX_VALUE;
        System.out.println("byte: " + byteValue);
        byteValue++; //byteValue = byteValue + 1;
        System.out.println("byte: " + byteValue);
    }
}
```

Java Type Casting

Assigning a value of one data type to another type.
In Java, there are two types of casting:

> **Widening Casting (automatically)** - converting a smaller type to a larger type size
> byte -> short -> char -> int -> long -> float -> double
>
> **Narrowing Casting (manually)** - converting a larger type to a smaller size type
> double -> float -> long -> int -> char -> short -> byte

Lab 06 - Casting

Implict - Widening

```
public class Main {
    public static void main(String[] args) {
        // Implicit Casting (Widening)
        int intValue = 100;
        long longValue = intValue; // Implicit casting from int to long
        float floatValue = longValue; // Implicit casting from long to float
        double doubleValue = floatValue; // Implicit casting from float to double
        System.out.println("Implicit Casting:");
        System.out.println("int to long: " + longValue);
        System.out.println("long to float: " + floatValue);
        System.out.println("float to double: " + doubleValue);
    }
}
```

Lab 07 - Casting

Casting - Narrowing

```
public class Main {
    public static void main(String[] args) {
        // Explicit Casting (Narrowing)
        double largeDouble = 12345.6789;
        // int intFromDouble = largeDouble;
        int intFromDouble = (int) largeDouble; // Explicit casting from double to int

        System.out.println("\nExplicit Casting:");
        System.out.println("double to int: " + intFromDouble);

        // Loss of Precision in Explicit Casting
        int largeInt = 1234567890;
        byte byteFromInt = (byte) largeInt; // Explicit casting from int to short

        System.out.println("\nLoss of Precision in Explicit Casting:");
        System.out.println("int to short: " + byteFromInt);
    }
}
```

Assessment -1

By now, You should know

1. The difference between JDK SDK
2. How to install JDK
3. How Java compilation process works
4. Why java is platform independent
5. Run a java program in command line
6. Run a java program in IntelliJ
7. How to Create a skeleton code (Class & main method) for performing some logic.
8. How to display Text, numbers
9. What is comment and how to add them
10. What is the need for Named memory location
11. What is declaration/initialisation and Definition
12. What happens when JVM executes the statement "int a = 20;"
13. What are identifiers and rules of identifiers.
14. What are data types
15. When to use which data type.

Things that are still not clear

1. What is "public"
2. What is "class"
3. What is "static"
4. What is "void"

Day 4

Java Operators

Operators are used to perform operations on variables and values.
Java divides the operators into the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Bitwise operators

Arithmetic operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	value = x + y
-	Subtraction	Subtracts one value from another	value = x - y
*	Multiplication	Multiplies two values	value = x * y
/	Division	Divides one value by another	value = x / y
%	Modulus	Returns the division remainder	value = x % y
++	Increment	Increases the value of a variable by 1	value = ++x
--	Decrement	Decreases the value of a variable by 1	value = --x

Lab 08

Operators

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

        // declare variables
        int a = 12, b = 5;

        // addition operator
        System.out.println("a + b = " + (a + b));

        // subtraction operator
        System.out.println("a - b = " + (a - b));

        // multiplication operator
        System.out.println("a * b = " + (a * b));

        // division operator
        System.out.println("a / b = " + (a / b));

        // modulo operator
        System.out.println("a % b = " + (a % b));

        // Increment operator - POST
        System.out.println("a++ = " + (a++));    //output 12

        // Increment operator - PRE
        System.out.println("++a = " + (++a));    //output 14

        // Decrement operator - POST
        System.out.println("b-- = " + (b--));    //output 5

        // Increment operator - PRE
        System.out.println("--b = " + (--b));    //output 3
    }
}
```

DivideAndFloatingPoint

```
public class DivideAndFloatingPoint {
    public static void main(String[] args) {
        int a = 5, b = 3;
        float c = 3.0F;
        double d = 3.0;
        System.out.println(a / b); //output 1
        System.out.println(a / c); //output 1.6666666
        System.out.println(a / d); //1.6666666666666667
    }
}
```

Assignment operators

Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Lab 09

Operators

```
class Main {
    public static void main(String[] args) {
        // create variables
        int a = 5;
        int b;

        // assign value using =
        b = a;
        System.out.println("b using =: " + b);

        // assign value using =+
        b += a;
        System.out.println("b using +=: " + b);

        // assign value using =*
        b *= a;
        System.out.println("b using *=: " + b);
    }
}
```

Comparison or Relational operators

Comparison operators are used to compare two values (or variables)
The return value of a comparison is boolean which is either true or false.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Lab 10

Operators

```
class Main {
    public static void main(String[] args) {
        // create variables
        int a = 7, b = 11;

        // value of a and b
        System.out.println("a is " + a + " and b is " + b);

        // == operator
        System.out.println(a == b);  // false

        // != operator
        System.out.println(a != b);  // true

        // > operator
        System.out.println(a > b);  // false

        // < operator
        System.out.println(a < b);  // true

        // >= operator
        System.out.println(a >= b);  // false

        // <= operator
        System.out.println(a <= b);  // true
    }
}
```

Java Logical Operators

Logical operators are used to determine the logic between variables or values or expressions.

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Bitwise operators

Bitwise operators perform operations on integer data at the individual bit-level

Operator	Name	Description	Example

|	Bitwise OR	Returns 1 if at least one of the operands is 1. Otherwise, it returns 0.	`int number1 = 12, number2 = 25, result;`
			`// bitwise OR between 12 and 25`
			`result = number1 | number2;`
			`System.out.println (result);  // prints 29`

&	Bitwise AND	Returns 1 if and only if both the operands are 1. Otherwise, it returns 0.	`result = number1 & number2;` `System.out.println (result);  // prints 8`
^	Bitwise XOR	Returns 1 if and only if one of the operands is 1. However, if both the operands are 0 or if both are 1, then the result is 0.	`result = number1 ^ number2;` `System.out.println (result);  // prints 21`

~	Bitwise Complement	Unary operator (works with only one operand). It changes binary digits 1 to 0 and 0 to 1.	`int number = 35, result;`
			`// bitwise complement of 35`
			`result = ~number;`
			`System.out.println (result);  // prints -36`

<<	Left Shift	Shifts all bits towards the left by a certain number of specified bits.	`int number = 2;`
			`// 2 bit left shift operation`
			`int result = number << 2;`
			`System.out.println (result);  // prints 8`

	-		

>>	Signed Right Shift	Shifts all bits towards the right by a certain number of specified bits.
		
		Least significant bits (rightmost) are discarded and the most significant position (leftmost) is filled with the sign bit
		
		Sign bit is preserved
			`int number1 = 8;`
			`int number2 = -8;`
			
			`// 2 bit signed right shift`
			`System.out.println (number1 >> 2);  // prints 2`
			`System.out.println (number2 >> 2);  // prints -2`

>>>	Unsigned Right Shift	Same as signed Right shift, But vacant leftmost position is filled with 0 instead of the sign bit.
		
		Sign bit is not preserved
			`int number1 = 8;`
			`int number2 = -8;`
			
			`// 2 bit signed right shift`
			`System.out.println (number1 >>> 2);  // prints 2`
			`System.out.println (number2 >>> 2);  // prints 1073741822`

Exercise - Lab 11

1. Write a java program to find the average of 3 subject scores of a student
    a. All the input for the marks will be in the range of 0 to 100
    b. Display the output

Java Input

Java provides different ways to get input from the user. Lets learn to get input from user using the object of Scanner class.
In order to use the object of Scanner, we need to import java.util.Scanner package.

Lab 12

Input

```
import java.util.Scanner;

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

        Scanner inputScanner = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int number = inputScanner.nextInt();
        System.out.println("You entered " + number);

        // Getting float input
        System.out.print("Enter float: ");
        float myFloat = inputScanner.nextFloat();
        System.out.println("Float entered = " + myFloat);

        // Getting double input
        System.out.print("Enter double: ");
        double myDouble = inputScanner.nextDouble();
        System.out.println("Double entered = " + myDouble);

        // Getting String input
        System.out.print("Enter text: ");
        System.out.println("Text entered = " + inputScanner.next());

        // closing the scanner object to release the resource
        inputScanner.close();
    }
}
```

Exercise - Lab 13

1. Write a java program to find the average of 3 subject scores of a student
    a. Receive the input from the user
    b. All the input for the marks should be in the range of 0 to 100
    c. Display the output

Statement & Blocks

Complete unit of execution is called a statement.
Unit of execution is identified by SEMI COLONS;

int score = 10;
int percentage = score * 100;

Java Blocks

A block is a group of statements (zero or more) that is enclosed in curly braces { }. For example,

Blocks

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

Day 5

Control flow statement

Control flow statements in Java are used to determine the sequence of execution in a program.
They enable you to make decisions, loop through code, and control the order in which statements are executed. Java provides three main categories of control flow statements:

1. Conditional Statements:
2. Looping Statements:
3. Jump Statements:

Conditional Statements:

- Conditional statement allows to execute different blocks of code based on certain conditions.
    - **if Statement:** Executes a block of code if a given condition is true.
    - **if-else Statement:** Executes one block of code if a condition is true and another block if the condition is false.
    - **if-else if Statement:** Executes multiple blocks of code based on different conditions.
    - **switch Statement:** Allows you to choose among a set of alternatives based on the value of an expression.
        - In Java 7, the switch statement was enhanced to support the use of String values as case labels. Before Java 7, switch only worked with integral types (char, byte, short, int, and enum constants).
        - In Java 12, the concept of switch was expanded to allow it to be used as an expression, which returns a value.
        - In Java 14, the yield statement was introduced to complement switch expressions. It allows you to both calculate a value and yield it in a single case branch.

Lab 14 - Program to find if score Grade is "PASS"

If-Else

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Find Grade");
        int input = 95;
        if(input >= 35){
            System.out.println("PASS");
        }
        }
}
```

Lab 15 - Program to find if score Grade is "PASS" or "DETAINED"

If-Else

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Find Grade");
        int input = 95;
        if(input >= 35){
            System.out.println("PASS");
        } else {
                        System.out.println("DETAINED");
                }
        }
}
```

Lab 16 - Program to find score grades

If-Else

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Find Grade");
        int input = 95;
        if(input >= 90){
            System.out.println("GRADE A");
        } else if(input >= 70){
                        System.out.println("GRADE B");
        } else if(input >= 50){
                        System.out.println("GRADE C");
                } else {
                        System.out.println("DETAINED");
                }
        }
}
```

Lab 17 - Program to find day of the week using switch

switch

```
public class Main {
    public static void main(String[] args) {
        int dayOfWeek = 3;
        switch (dayOfWeek) {
            case 1:
                System.out.println("Sunday");
                break;
            case 2:
                System.out.println("Monday");
                break;
            case 3:
                System.out.println("Tuesday");
                break;
            case 4:
                System.out.println("Wednesday");
                break;
            case 5:
                System.out.println("Thursday");
                break;
            case 6:
                System.out.println("Friday");
                break;
            case 7:
                System.out.println("Saturday");
                break;
            default:
                System.out.println("Invalid day");
        }
    }
}
```

Exercise

1. Write a java program to find if a number is Odd or Even
2. Write a java program to find greatest of 3 numbers.
3. Write a java program to find greatest of 4 numbers.
4. Write a program to perform the following operations.
    a. receive 2 numbers from the user
    b. Perform below operations on the above inputs based on user input.
        i. If user input is 1, then add the 2 inputs and display the output
        ii. If user input is 2, then subtract the 2 inputs and display the output
        iii. If user input is 3, then multiply the 2 inputs and display the output
        iv. If user input is 4, then divide the 2 inputs and display the output

Looping Statements:

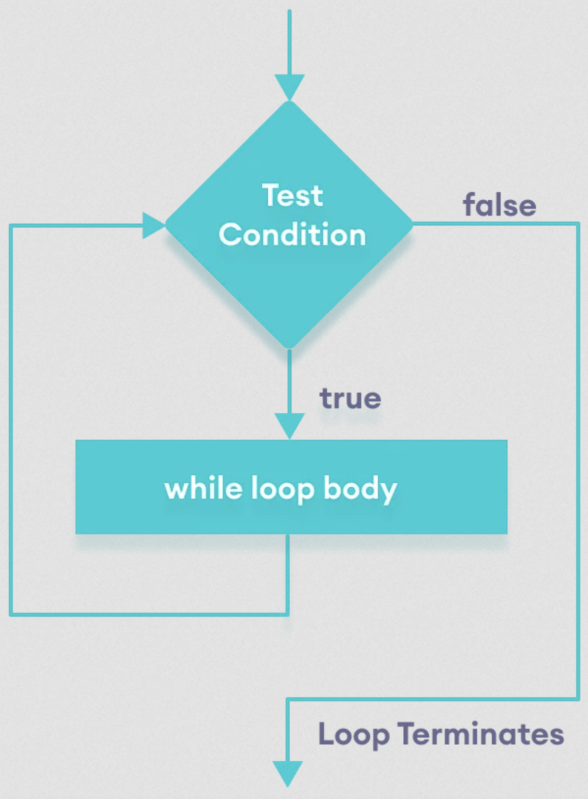
- Looping statements allow you to repeat a block of code multiple times. Java provides several types of loops:
    - **for Loop:** Repeats a block of code for a specified number of iterations.
    - **while Loop:** Repeats a block of code as long as a given condition is true.
    - **do-while Loop:** Repeats a block of code at least once, and then continues as long as a given condition is true.

While Loop

Java while loop is used to run a specific code until a certain condition is met. The syntax of the while loop is:

Syntax

```
while (condition) {
// body of loop
// optional break statement and its condition;
}
```



1. A while loop evaluates the "condition" inside the parenthesis ().
2. If the "condition" evaluates to true, the code inside the while loop is executed.
3. The "condition" is evaluated again.
4. This process continues until the "condition" is false.
5. When the "condition" evaluates to false, the loop stops.

Exercise

1. Write a program to perform the following operations. This is an extension to the previous exercise. Here the program should halt only when user inputs 0.
    a. receive 2 numbers from the user
    b. Perform below operations until user input 0.
        i. If user input is 1, then add the 2 inputs and display the output
        ii. If user input is 2, then subtract the 2 inputs and display the output
        iii. If user input is 3, then multiply the 2 inputs and display the output
        iv. If user input is 4, then divide the 2 inputs and display the output
        v. If user input is 0, then exit the loop and halt the program.

2. Write a program to print even numbers from with the range of 0 to 30
3. Write a program to print multiplication table of number of given number. Receive the input from user.
4. Write a program to count the number of digits in a given number.
5. Write a program to print Sum of Digits in a given number.
6. Write a program to generate the Fibonacci series up to a given number using a while loop.

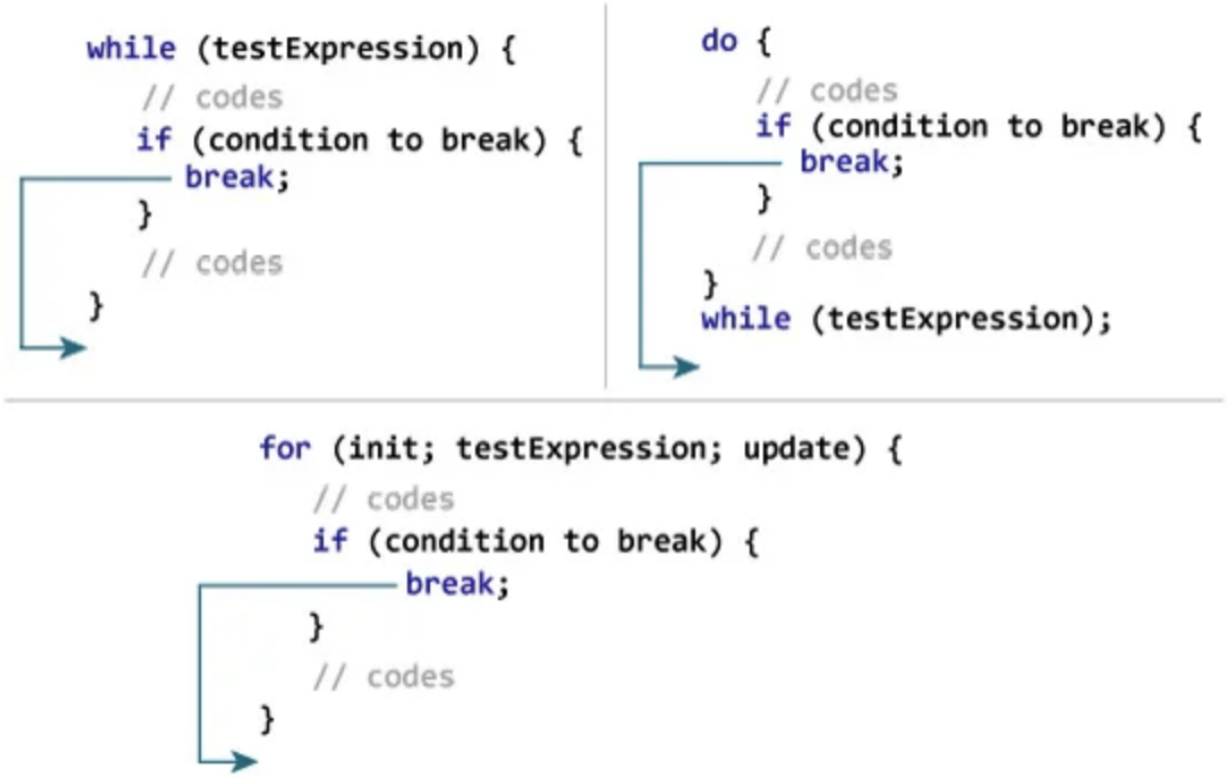
Jump Statements:

- Jump statements allow you to alter the normal sequence of execution by transferring control to a different part of the program. The main jump statements are:
    - **break Statement:** Terminates the current loop or switch statement and transfers control to the next statement.
    - **continue Statement:** Skips the remaining code in the current iteration of a loop and proceeds to the next iteration.
    - **return Statement:** Terminates the execution of a method and returns a value to the caller.

Break & Continue Statement

"break" and "continue" statements are used under 2 conditions.

1. Break - terminate the loop immediately without checking the test expression.
2. Continue - skip some statements inside the loop or



Note: Break statement will be used along with branching statements (If else). But it will only break the immediate looping control block or the switch block. There is an another version of break "Labeled break" which can break nested looping structures.

Lab: WhileLoopLab6Break

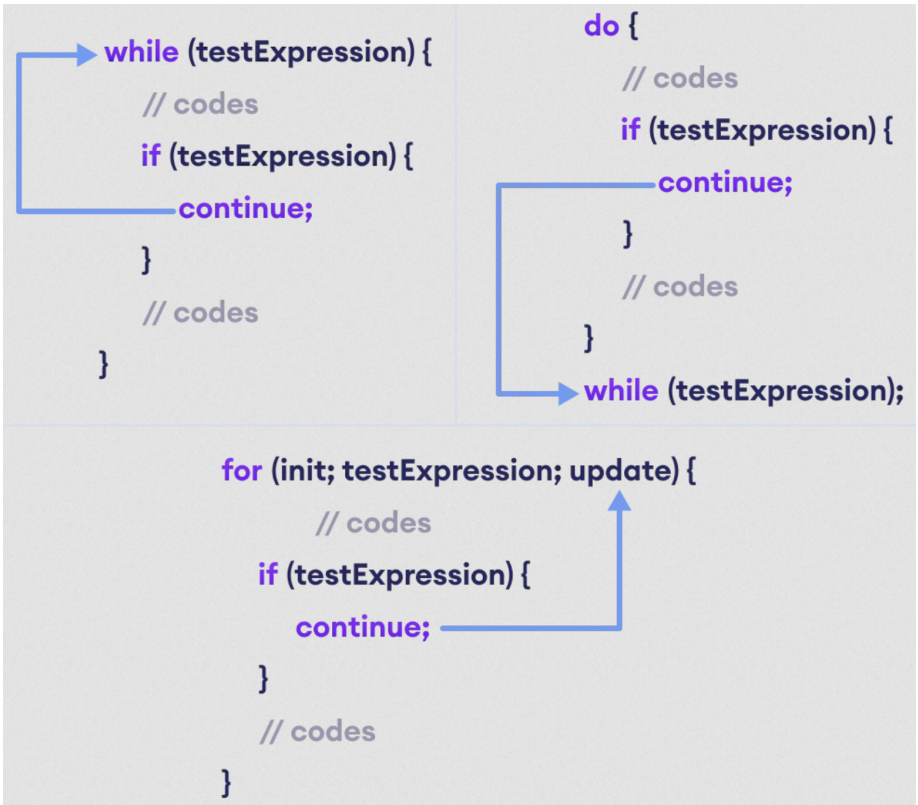
Receive input 1 to 5 from user, break if the input is not in correct sequence

WhileLoopLab6Break

```
import java.util.Scanner;
public class Day5_WhileLoopLab6Break {
    public static void main(String[] args) {

        int counter = 1;
        System.out.println("Enter numbers from 1 to 5 in correct order");
        Scanner inputScanner = new Scanner(System.in);
        while (counter <= 5) {
            int userInput = inputScanner.nextInt();
            if (userInput != counter) {
                break;
            }
            counter++; //counter = counter + 1;
        }
        inputScanner.close();
    }
}
```

Continue Statement

The continue statement skips the current iteration of a loop (for, while, do...while, etc).
After the continue statement, the program moves to the Beginning of the loop.



Exercise

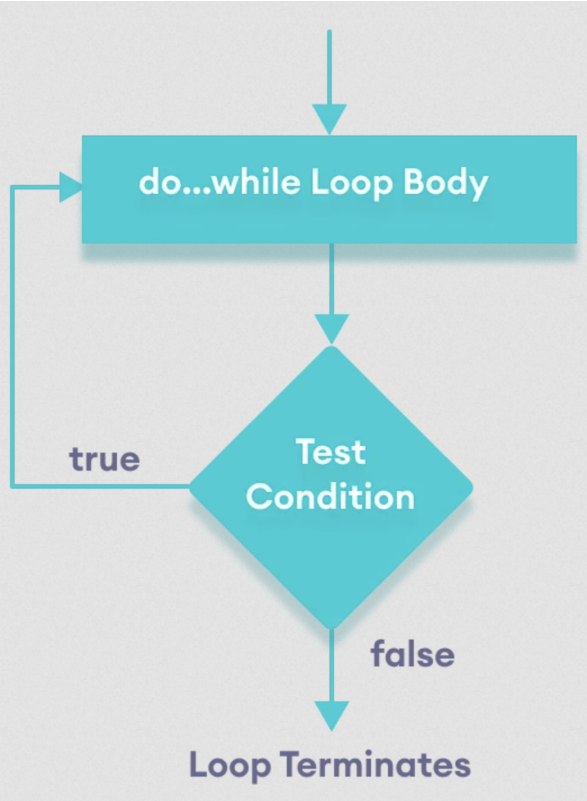
1. Write a program to print even numbers from with the range of 0 to 30 except number 10
2. Write a program to print even numbers from with the range of 0 to 30 except numbers in range 10 to 20

Do While Loop

"Do while" loop is same as "While" loop. The only difference is the "do while" loops gets executed at least once.
"The condition is checked after the execution. That makes it to execute the block at least once.

Syntax

```
do {
    // body of loop
} while(condition);
```



For Loop

The functionality of "for loop" is same as "while loop". But for loop is preferred when you know the number of times you want the loop to execute.

Syntax

```
for (initialization; condition; iteration) {
    // Code to be executed in each iteration
}
```

1. Initialization: This part is used to initialize the loop control variable(s) and is executed only once before the loop starts. It's where you can declare and initialize variables that will be used within the loop.
2. Condition: This is a boolean expression that defines the condition that must be true for the loop to continue executing. As long as the condition remains true, the loop will continue to execute.
3. Iteration: This part is executed after each iteration of the loop. It's where you can update the loop control variable(s) or perform other operations that should happen after each iteration.

For Loop

```
public class ForLoopExample {
    public static void main(String[] args) {
        for (int index = 0; index < 10; index++) {
            System.out.println(index);
        }
    }
}
```

Nested Loops:

Control Loops like (While, Do While & For Loop) can be mixed and nested to solve complex problems.

Nested Loops

```
public class NestedLoopLab1 {
    public static void main(String[] args) {
        //For Loop implementation
        System.out.println("For loop implementation");
        for (int outerLoopIndex = 1; outerLoopIndex <= 10; outerLoopIndex++) {
            System.out.println("Starting Outer Loop Iteration - " + outerLoopIndex);
            for (int innerLoopIndex = 1; innerLoopIndex <= 10; innerLoopIndex++) {
                System.out.print(innerLoopIndex + "\t");
            }
            System.out.println(); // Move to the next line after each row
        }
    }
}
```

Day 6:

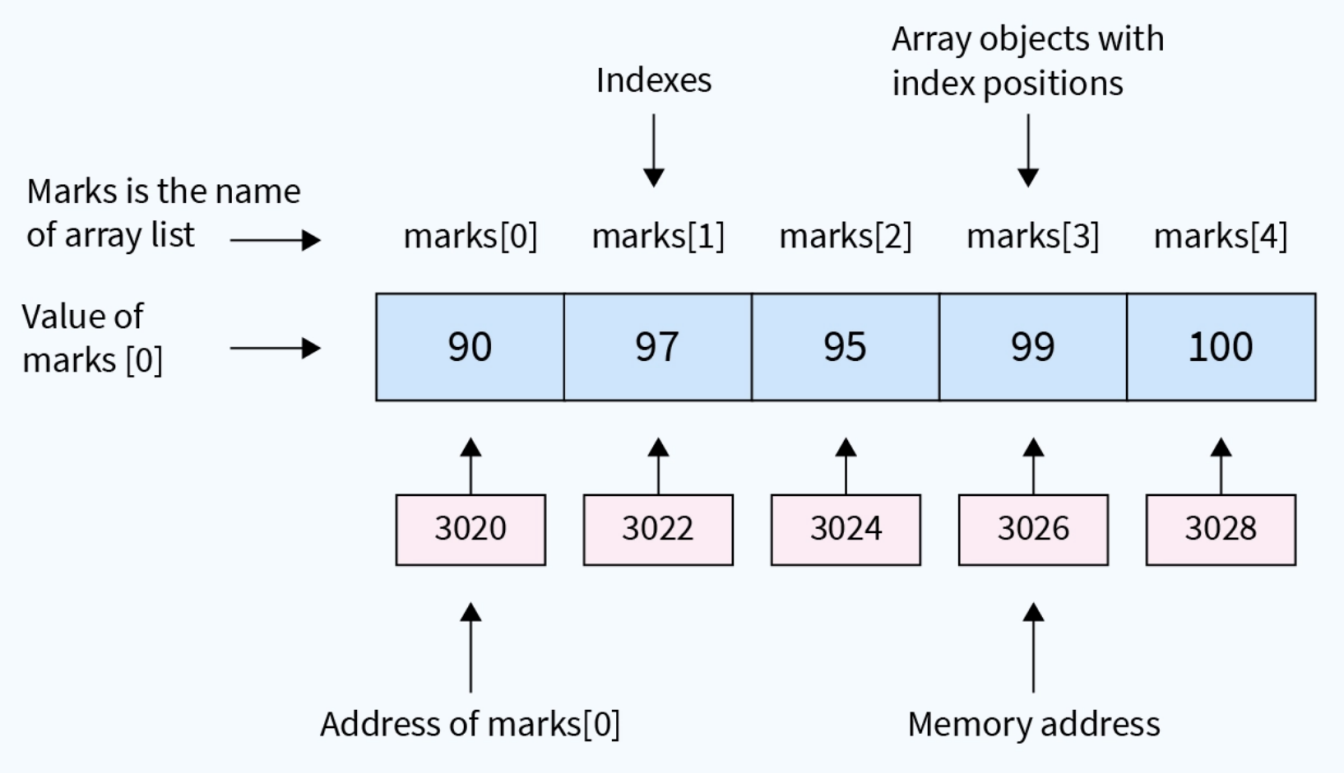
Arrays

An array is a collection of similar types of data.
For example,
if we want to store the marks of 5 students then we can create an array of the type "int" that can store 100 marks.

Syntax

```
//Declaration
int[] marks;

//Initialisation
marks = new int[5]; //size is must

//Initialisation
marks = new int[] {90, 97, 95, 99, 100}; //size is not required

//Definition
int[] marks = new int[] {90, 97, 95, 99, 100};
```

How array works?



Accessing the array elements

We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

- We can get the value from the array index
- We can also set the value to the index position

Lab: ArrayLab1

ArrayLab1

```
public class Day6_ArrayLab1 {
    public static void main(String[] args) {
        // create an array
        int[] marks = {90, 97, 95, 99, 100};

        System.out.println("Array size" + marks.length);

        // access each array elements
        System.out.println("Accessing Elements of Array:");
        System.out.println("First Element: " + marks[0]);
        System.out.println("Second Element: " + marks[1]);
        System.out.println("Third Element: " + marks[2]);
        System.out.println("Fourth Element: " + marks[3]);
        System.out.println("Fifth Element: " + marks[4]);

        //Modify it
        marks[0] = 10;
        System.out.println("\nFirst Element after modifying: " + marks[0]);
    }
}
```

Iteration or Looping through the Array using Index position

Lab2ForLoop

```
public class Day6_ArrayLab2ForLoop {
    public static void main(String[] args) {
        // create an array
        int[] marks = {90, 97, 95, 99, 100};

        System.out.println("Array size" + marks.length);

        // access each array elements
        System.out.println("Accessing Elements of Array:");
        for(int index=0; index < marks.length; index++){
            System.out.println("element at index " + index + " : " + marks[index]);
        }
        //Modify it
        marks[0] = 10;
        System.out.println("\nFirst Element after modifying: " + marks[0]);
    }
}
```

Iteration or Looping through the Array Values without Index position

For Each Loop is used to iterate the Array elements value. Index information will not be available in this mode.

ArraLab3ForEach

```
public class Day6_ArrayLab3ForEachLoop {
    public static void main(String[] args) {
        // create an array
        int[] marks = {90, 97, 95, 99, 100};

        System.out.println("Array size: " + marks.length);

        // access each array elements
        System.out.println("Accessing Elements of Array:");
        for(int mark : marks){
            System.out.println(mark);
        }
        //Modify it
        marks[0] = 10;
        System.out.println("\nFirst Element after modifying: " + marks[0]);
    }
}
```

Multi Dimensional Arrays - Need

How do we store and process complex data?

Lab - Day6_ArrayMultiDimLab1

Write a Program to calculate total and average of a 4 student and their marks in 3 subjects.

- There are 4 students
- Each Student has 3 subjects
- Calculate and print the total and average of the student marks.
- No of students are likely to grow.

How will you design the array structure.

1. Do you define an array for each subject and collect all student scores or
2. Do you define an array for each student and collect his scores in that array.

		Student0	Student1	Student2	Student3
sub1-Array	allStudentScores1	10	20	25	30
sub2-Array	allStudentScores2	40	50	55	60
sub3-Array	allStudentScores3	70	80	85	90

		Score0	Score1	Score2
Student1-Array	student1AllScores	10	40	70
Student2-Array	student2AllScores	20	50	80
Student3-Array	student3AllScores	25	55	85
Student4-Array	student4AllScores	30	60	90

Multi Dimensional Array

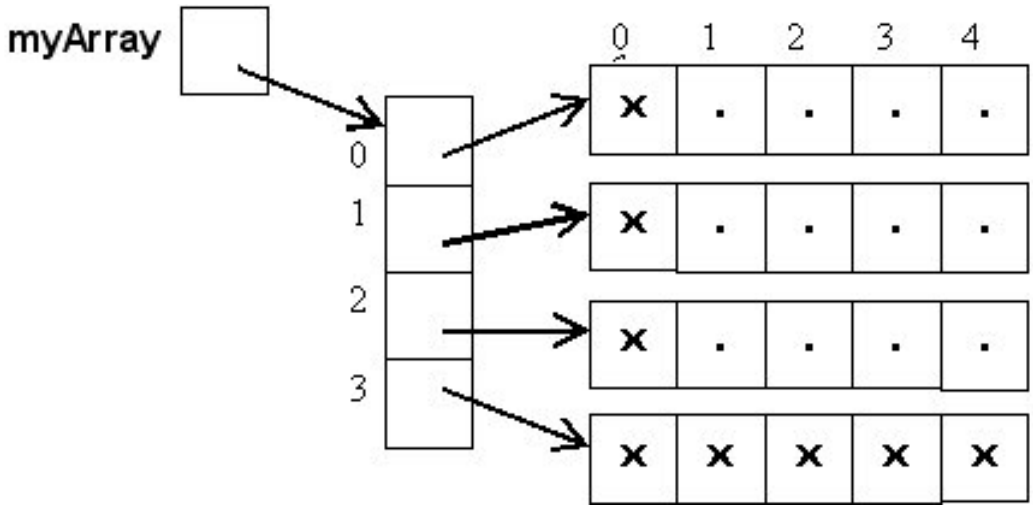
```
public class Day6_ArrayMultiDimLab1 {
    public static void main(String[] args) {
        //Variation 1
        int totalNoOfStudents = 4;
        int totalNoOfSubjects = 3;

        System.out.println("Variation 1");
        int[] allStudentScores1 = new int[]{10, 20, 25, 30};
        int[] allStudentScores2 = new int[]{40, 50, 55, 60};
        int[] allStudentScores3 = new int[]{70, 80, 85, 90};

        int[] allStudentTotalScores = new int[totalNoOfStudents];
        double[] allStudentAverageScores = new double[totalNoOfStudents];

        //Iterate through the array and calculate the total and average
        for (int studentIndex = 0; studentIndex < totalNoOfStudents; studentIndex++) {
            allStudentTotalScores[studentIndex] += allStudentScores1[studentIndex];
            allStudentTotalScores[studentIndex] += allStudentScores2[studentIndex];
            allStudentTotalScores[studentIndex] += allStudentScores3[studentIndex];

            //All the 3 subjects of students are added, Finding and storing the average
            allStudentAverageScores[studentIndex] = allStudentTotalScores[studentIndex] / ((double)
totalNoOfSubjects);
        }

        //Iterate through the array and print the total and average of all the students
        for (int studentIndex = 0; studentIndex < totalNoOfStudents; studentIndex++) {
            System.out.println("Total of student" + studentIndex + ": " + allStudentTotalScores[studentIndex]);
            System.out.println("Average of student" + studentIndex + ": " + allStudentAverageScores
[studentIndex]);
        }
    }
}
```

Multi Dimensional Arrays

A multidimensional array is an array of arrays. That is, each element of a multidimensional array is an array itself. For example,

Syntax

```
//Declaration
int[][] marks;

//Initialisation
marks = new int[3][4]; //size is must for the first array

scores = new int[5][];
scores[0] = new int[10];
scores[1] = new int[3];

//Initialisation
scores = new int[][]{
                {10, 20, 25, 30},
                {40, 50, 55, 60},
                {70, 80, 85, 90}
        };


//Definition
                int[][] scores = new int[5][];


        int[][] scores = new int[][]{
                {10, 20, 25, 30},
                {40, 50, 55, 60},
                {70, 80, 85, 90}
        };

//Accessing
scores[0][1]
```



Copy Array

Copying arrays using assignment operator will copy the reference only. It means the new variable will also refer the same memory location. And any change in either of array variables will be reflected in both.

Copy Array

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

        int [] numbers = {1, 2, 3, 4, 5, 6};
        int [] positiveNumbers = numbers;    // copying arrays

        // change value of first array
        numbers[0] = -1;

        // printing the second array
        for (int number: positiveNumbers) {
            System.out.print(number + ", ");
        }
    }
}
```

To copy Arrays, Perform Deep copy or use below API's.

- System.arraycopy()
- Arrays.copyOfRange()

Sorting

Sorting is the process of arranging a collection of elements in a particular order. It is classified as follows. Choosing the right sorting algorithm depends on the characteristics of your data and the specific requirements of your application. It's important to consider factors such as the size of the dataset, the distribution of elements, memory usage, stability, and the desired time complexity when selecting a sorting method.

Based on Nature:

Comparison Sorts:

These algorithms compare elements using comparison operators like <, >, <=, >=. They determine the relative order of elements based on comparisons.
Examples: **Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort.**

Non-Comparison Sorts:

These algorithms do not solely rely on comparisons to sort elements. They take advantage of specific properties of the data.
Examples: **Counting Sort, Radix Sort, Bucket Sort.**

Based on Function:

In-Place Sorts:

These algorithms sort elements within the given array without requiring extra memory space proportional to the input size.
Examples: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort (with some variants).

Out-of-Place Sorts:

These algorithms require additional memory space proportional to the input size to store intermediate results or auxiliary data structures.
Examples: Merge Sort, Heap Sort.

Based on Stability:

1. Stable Sorts: These algorithms maintain the relative order of equal elements as they appear in the original input.
    - Examples: Bubble Sort, Merge Sort.
2. Unstable Sorts: These algorithms do not guarantee the relative order of equal elements in the sorted output.
    - Examples: Selection Sort, Quick Sort, Heap Sort.

Must know sorting algorithms

1. Simple Sort - Bubble sort
2. nLogn sort - Quick sort, Merge sort
3. Uniform distribution/ fixed range - Rank sort / bucket sort
4. top N - Heap sort

Searching

Searching is a techniques used to locate a specific element within a collection of data, such as an array or a list.

Linear Search:

Involves checking each element in the collection sequentially until the target element is found.
Simple and works for both sorted and unsorted data.
Time complexity: O(n) in the worst case (where 'n' is the number of elements).

Binary Search:

Requires the collection to be sorted.
Divides the search space in half with each comparison, reducing the search range exponentially.
Time complexity: O(log n) in the worst case, where 'n' is the number of elements. Very efficient for large datasets.

Big-O cheat sheet

https://www.bigocheatsheet.com/

Methods or Functions

The term "Method" or "Function" are interchangeably used.

A method is a block of code which only runs when it is called.
Methods are used to perform some specific actions.

Advantages:
Avoid duplicate Code and promote code reuse: define the code once, and use it many times.
Complex logics can be modularised.

Syntax

```
returnType methodName(datatype parameter1, datatype parameter2, ...) {
  // method body
  // return output
}
```

Lab - Day7MethodLab1

MethodLab1

```
public class Day7_MethodLab1 {
    public static void main(String[] args) {
        int length1 = 5;
        int width1 = 10;

        int length2 = 8;
        int width2 = 15;
        calculateArea(length1, width1);
        calculateArea(length2, width2);
    }
    public static void calculateArea(int length, int width) {
        //int area = length * width;
        System.out.println("Area :" + (length * length));
    }
}
```

Caller

Method which calls/invoke a method is caller the caller method.

Method Parameters

Methods may have parameters. Caller method must pass the parameter when calling a method.

Method without Parameter

MethodLab2

```
import java.util.Scanner;

public class Day7_MethodLab2 {
    public static void main(String[] args) {
        addTwoNumber();
    }

    public static void addTwoNumber() {
        Scanner inputScanner = new Scanner(System.in);
        System.out.println("Enter 2 numbers");
        int a = inputScanner.nextInt();
        int b = inputScanner.nextInt();
        System.out.println("Sum of 2 numbers : " + (a + b));
        inputScanner.close();
    }
}
```

Method with primitive datatypes parameters - Pass by value

MethodLab3

```
public class Day7_MethodLab3 {
    public static void main(String[] args) {
        int a = 2;
        int b = 5;
        printSquaredSumOfTwoNumbers(a, b);
    }

    public static void printSquaredSumOfTwoNumbers(int a, int b) {
        a *= a;
        b *= b;
        System.out.println("Sqaured Sum of 2 numbers : " + (a + b));
    }
}
```

Method with array/object as parameter - Pass by reference

MethodLab4PassByReference

```
import java.util.Scanner;

public class Day7_MethodLab4 {
    public static void main(String[] args) {
        int[] inputArray = new int[]{1, 2, 3, 4};
        printArray(inputArray);
        squareTheNumbers(inputArray);
        printArray(inputArray);
    }
    public static void squareTheNumbers(int[] inputArray) {
        for (int index = 0; index < inputArray.length; index++) {
            inputArray[index] *= inputArray[index];
        }
    }
    public static void printArray(int[] inputArray) {
        for (int index = 0; index < inputArray.length; index++) {
            System.out.print(inputArray[index] + "\t");
        }
        System.out.println();
    }
}
```

Return values

A method may or may not return data back to the caller.
Void - indicates that method will not return any data.
To return any data, its data type must be mentioned in the method signature

MethodLab5ReturnInt

```
public class Day7_MethodLab5ReturnInt {
    public static void main(String[] args) {
        int a = 2;
        int b = 5;
        int sum = printSquaredSumOfTwoNumbers(a, b);
        System.out.println("Sqaured Sum of 2 numbers : " + sum);
    }

    public static int printSquaredSumOfTwoNumbers(int a, int b) {
        a *= a;
        b *= b;
        return a + b;
    }
}
```

Scope of Variables

In Java, variables are only accessible inside the region they are created. This is called scope.
Method Scope - Variables created within a method, are available anywhere within that method.
Block Scope - Variables created within a block (If, while, for etc), are available anywhere within that block.

Recursion

A method may call itself recursively and this technique is called recursion.

Exit Condition

Every recursive method should have an exit condition. A recursive method has to stop at some point when the task is completed. This is achieved with an appropriate condition to stop the method call.

RecursionLab1

```
public class Day7_RecursionLab1 {
    public static void main(String[] args) {
        int result = sum(10);
        System.out.println(result);
    }

    public static int sum(int number) {
        if (number > 0) {
            return number + sum(number - 1);
        } else {
            return 0;
        }
    }
}
```

Note:
Always try to avoid recursion.
Method calls are expensive that control statements. This is due to the stack management of every method call.

Debugging

Debugging is the process of identifying and fixing errors, bugs, and unexpected behavior in your code. It involves stepping through your code line by line, observing variables, expressions, and program flow to understand what is happening at each point and to pinpoint the source of issues.

All modern Integrated development environment provide debugging tools to inspect your code.

ToDo

Identifiers

DIVISION & INTEGER

INFINITE LOOP

Label

Labelled Break

For Each

Bitwise Operators

Break a loop statement without using "break" keyword

Print 1 to 10 without using Loop statements