WEEK 1 MODULE: INTRODUCTION TO JAVA AND ITS FEATURES

TOPICS

- Introduction and History of Java
- Features of Java
- Setting Up IDE and JDK
- Writing simple Java Programs
- Stages of Java Development: Editing, Compiling and Running
- Java naming convention
- Java data types
- Operator Precedence

LEARNING OUTCOMES:

By the end of this module, students will be able to:

- Know the History and Importance of Java
- Features of Java
- Coding in an IDE
- Writing, compiling and executing Java programs
- How to name Class, Variables, Methods and Constants
- Types of data in Java; primitive types and reference types
- Operators in Java; their precedence and writing clean codes.

DAY 1

What is a Computer Program

A computer program is a clear, step-by-step, finite set of instructions.

History of Java

Java is an object-oriented programming language that was developed in the early 1990s by James Gosling. It has been the foundation of millions of applications across many platforms such as Windows, Macintosh, UNIX, Android-based handheld devices, Embedded Systems, and corporate solutions. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. Java has significantly impacted the world of software development since its inception in the mid-1990s. Java is widely used for supercomputers and data centers.

The Java programming language is perhaps one of the most popular object-oriented languages today.

Java programming language can be used to build desktop applications, web applications and mobile (Android) applications. It is a very powerful language that has been in use for a long time. You can also build applications with GUIs (graphical user interface) using Java. Java and Kotlin (used for Android app development) are compatible; most Java libraries or framework can be used by Kotlin. For this course, we'll limit our learnings to Java. With a good understanding of Java, you'll be able to learn Kotlin EASILY on your own.

Features of Java

- **Object-Oriented**: Java is a fully object-oriented programming language. It follows the principles of encapsulation, inheritance, and polymorphism, allowing developers to create modular and reusable code.
- **Platform Independence**: Java programs are compiled into bytecode, which can be executed on any platform with a Java Virtual Machine (JVM). This platform independence makes Java highly portable, enabling developers to write code once and run it anywhere.
- **Automatic Memory Management**: Java features automatic memory management through its built-in garbage collection mechanism. Developers do not need to manually allocate and deallocate memory, as the JVM takes care of memory management, freeing the developers from memory-related errors such as memory leaks.
- **Robust and Secure**: Java includes various features designed to enhance program robustness and security. It enforces strong type checking during compilation, includes

exception handling mechanisms to deal with errors, and provides a security model that protects against unauthorized access.

- **Rich Standard Library**: Java comes with a comprehensive standard library that provides a wide range of classes and methods for common programming tasks. This library includes utilities for file I/O, networking, database access, graphical user interface (GUI) development, and more, enabling developers to build complex applications efficiently.
- **Multi-threading Support**: Java has built-in support for concurrent programming through its multithreading capabilities. Developers can create and manage multiple threads within a single program, allowing for concurrent execution and better utilization of system resources.
- **High Performance**: While Java is often criticized for its performance compared to lower-level languages like C++, it still offers good performance thanks to the optimization techniques employed by modern JVM implementations. Additionally, Java provides just-in-time (JIT) compilation, which translates +bytecode into machine code at runtime for improved execution speed.
- **Developer Community and Ecosystem**: Java has a large and active developer community, which means there is extensive documentation, tutorials, and online resources available. It also has a vast ecosystem of third-party libraries, frameworks, and tools that enhance productivity and simplify development tasks.

Let's examine Platform Independence, OOP concept and Automatic memory management in-depth.

Platform Independent

Architecture neutral

A Java program will run identically on every platform. A java program written on whatever platform (Windows, Linux etc.) will run same way when moved from platform to platform. This is possible due to the Java Virtual Machine. Here are ways the JVM handles our codes:

- Instead of producing a processor-specific code, Java compilers produce an intermediate code called bytecode.
- The bytecode is also a binary code but is not specific to a particular CPU.
- A Java compiler will produce exactly the same bytecode no matter what computer system is used.
- The Java bytecode is then interpreted by the Java Virtual Machine (JVM) interpreter.
- Notice that each type of computer system has its own Java interpreter that can run on that system.
- This is how Java achieves compatibility.

- It does not matter on what computer system a Java program is compiled, provided the target computer has a Java Virtual machine. See below image.

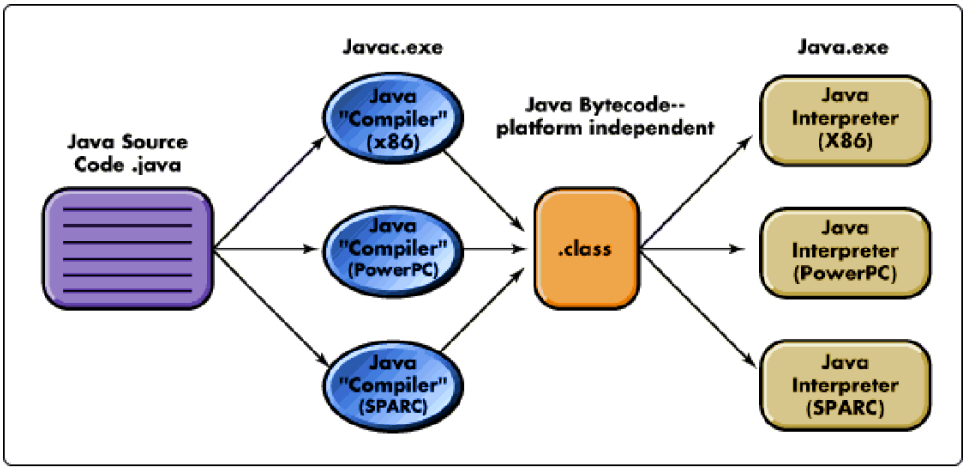

Figure 1: JVM Explained in picture.

Object Oriented Programming Concept in Java

Object-oriented programming (OOP) is a programming paradigm that organizes code around the concept of objects, which are instances of classes. Java is an object-oriented programming language that fully supports OOP principles. Let's dive into the key concepts of OOP in Java:

- **Classes and Objects**: A class is a blueprint or template that defines the properties (attributes) and behaviors (methods) of objects. An object, on the other hand, is an instance of a class. For example, you can have a class called "Car" that defines the characteristics and actions of a car. You can then create multiple car objects based on this class.
- **Encapsulation**: Encapsulation refers to the bundling of data and methods within a class, and the restriction of access to the internal state of an object from outside the class. It helps in achieving data hiding and abstraction. In Java, you can use access modifiers such as public, private, and protected to control the visibility of class members.
- **Inheritance**: Inheritance enables you to create a new class (derived or child class) from an existing class (base or parent class). The derived class inherits the properties and behaviors of the parent class, allowing code reuse and the creation of class hierarchies. Java supports single inheritance (one class extends another class) and multiple inheritance through interfaces (a class implements multiple interfaces).
- **Polymorphism**: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables you to perform a single action in different ways, depending on the objects involved. Polymorphism in Java is typically achieved through method overriding and method overloading.
- **Abstraction**: Abstraction focuses on defining the essential characteristics of an object, while hiding the implementation details. It allows you to create abstract classes and

interfaces that provide a common interface for a group of related classes. Abstract classes cannot be instantiated, but they can be subclassed. Interfaces, on the other hand, define a contract that implementing classes must adhere to.

- **Association, Aggregation, and Composition**: These are relationships between classes. Association represents a relationship between two or more classes, where each class has its own independent lifecycle. Aggregation is a specialized form of association, where one class is composed of one or more other classes, but they can exist independently. Composition is a stronger form of aggregation, where the composed class cannot exist without the container class.

These are the fundamental concepts of OOP in Java. By leveraging these concepts, you can create modular, reusable, and maintainable code that models real-world entities in a structured manner.

Automatic Memory management

Java automatic memory management refers to the management of memory resources within a Java program. In Java, memory management is handled automatically by the Java Virtual Machine (JVM), which is responsible for allocating and releasing memory for objects and data structures used in your program.

Here are some key concepts related to Java memory management:

- **Heap memory**: In Java, objects are created and stored in a part of memory called the heap. The heap is a large, shared memory area that is used to allocate memory for objects dynamically at runtime. The heap memory is managed by the JVM and automatically allocated and deallocated as needed.
- **Garbage collection**: Java uses a garbage collector to automatically reclaim memory occupied by objects that are no longer needed. The garbage collector identifies and frees the memory occupied by objects that are no longer referenced by any part of the program. This automatic memory reclamation helps simplify memory management and reduces the risk of memory leaks.
- **Automatic memory allocation**: In Java, you don't need to explicitly allocate memory for objects using manual memory allocation techniques like **malloc()** in languages like C or C++. Instead, memory allocation for objects is handled automatically when you use the **new** keyword to create objects.
- **No explicit deallocation**: Java relieves you from the burden of explicitly deallocating memory when objects are no longer needed. The garbage collector automatically identifies and reclaims memory occupied by unused objects, freeing you from manual memory deallocation responsibilities. This feature helps prevent common memory-related issues such as dangling pointers and memory leaks.
- **Memory management best practices**: While Java handles memory management for you, it's still important to write efficient code and follow good coding practices. This

includes avoiding unnecessary object creation, minimizing the scope of variables, and using appropriate data structures and algorithms to optimize memory usage.

By providing automatic memory management, Java helps simplify the process of managing memory and reduces the likelihood of memory-related errors. It allows you to focus more on writing the application logic and less on manual memory management tasks, making Java a popular choice for building robust and memory-safe applications.

Before we proceed, we need to all have an IDE (Integrated Development Environment) and JDK (Java Development Kit)/SDK (Software Development Kit) installed on our system. These are some of the requirements to be able to write your code and run it on your system. Below is a step-by-step guide on how to download the required software.

Installation of JDK/SDK and IDE (IntelliJ)

Step 1: Download the JDK:

1. Open a web browser and go to the Oracle JDK download page: Java Downloads | Oracle
2. Click on the "Java SE Development Kit 20 Downloads" section.
3. Accept the license agreement by selecting the checkbox next to "Accept License Agreement".
4. Choose the JDK installer according to your operating system. For Windows, download the appropriate JDK installer for your system (x64 for 64-bit or x86 for 32-bit).
5. Click on the download link to start downloading the JDK installer.

Step 2: Install the JDK:

6. Once the JDK installer is downloaded, locate the file and double-click on it to run the installer.
7. Follow the prompts in the installer to complete the installation. You can typically accept the default settings unless you have specific requirements.

Step 3: Download IntelliJ IDEA:

8. Open a web browser and go to the JetBrains IntelliJ IDEA download page: Download IntelliJ
9. On the download page, click on the "Download" button below the IntelliJ IDEA Community edition (free version) or IntelliJ IDEA Ultimate edition if you are willing to pay (paid version).
10. Once the download is complete, locate the downloaded installer file and double-click on it to run the installer.

Step 4: Install IntelliJ IDEA:

11. Follow the prompts in the installer to complete the installation. You can typically accept the default settings to continue.
12. On the "Choose Install Location" screen, you can select the destination folder where IntelliJ IDEA will be installed. You can keep the default location or choose a different one.
13. Once the installation is complete, launch IntelliJ IDEA.

Step 5: Configure IntelliJ IDEA:

14. On the initial startup, IntelliJ IDEA may prompt you to import settings or choose a theme. You can select the appropriate options based on your preferences.
15. Next, you'll be prompted to select a JDK for IntelliJ IDEA. Choose the JDK you installed earlier from the list, or you can specify the JDK path manually if it's not listed.
16. Follow the remaining prompts to complete the initial setup and configuration.

Now that you've completed the installation of the required software; it's time to let your fingers play on the keyboard while you start your journey to becoming a world class software developer/engineer.

Best of Luck!

DAY 2

Structure of Java Program

```
public class ClassName {
    public static void main(String[] args ){
     statement 1
     statement 2
      * * *
     statement N
    }
}
```

Let's break it down:

i. **Package Declaration:** This section is used to specify the package that the program belongs to. A package is a way of grouping related classes together. You'll see examples when we start writing our java codes.

ii. **Import Statements:** This section is used to import classes from other packages. This allows you to use the classes in your program without having to fully qualify their names.

iii. **Class Definition:** This section is used to define a class. A class is a blueprint for an object. It defines the data and methods that an object will have.

iiii. **Main Method Class:** This section is used to define the main method. The main method is the entry point for a Java program. It is where the execution of the program begins.

iv. **Methods and Behaviors:** This section is used to define the methods and behaviours of the class. Methods are the actions that an object can perform. Behaviors are the properties of an object.

Java Class, Objects, Methods and Main method (In-depth)

Java Class: A Java class is defined using the class keyword followed by the class name. It encapsulates data (in the form of variables) and behaviors (in the form of methods) related to a specific concept or entity. Here's an example of a simple Java class:

```
public class Person {

   // Variables (or fields)

   String name;

   int age;


   // Methods

   public void sayHello() {
```

```
        System.out.println("Hello, my name is " + name + " and I am " + age + " years old.");
    }
}
```

Objects: An object is an instance of a class. It represents a specific entity or a real-world object. You can create objects by using the new keyword followed by the class name and parentheses. Here's an example of creating objects from the Person class:

```
Person person1 = new Person();
Person person2 = new Person();
```

In this example, person1 and person2 are two separate objects of the Person class.

Methods: Methods are blocks of code that define the behavior of a class or perform specific actions. They encapsulate a set of instructions that can be executed on objects of that class. In the Person class example above, the sayHello() method is defined. It prints a greeting message using the name and age variables. Any method that will be called directly from the main method must be declared static.

Main Method: The main method is a special method that serves as the entry point for a Java program. It is called by the Java Virtual Machine (JVM) when the program starts running. The main method must have the following signature:

```
public class Main {
    public static void main(String[] args) {
        // Program execution starts here
        System.out.println("Hello, World!");
    }
}
```

In this example, when the program is executed, the main method is called, and the message "Hello, World!" is printed to the console.

The main method acts as the entry point for your program, and you can invoke other methods, create objects, and perform various operations within it.

Syntax of Java

Basically, the syntax of java is:

datatype variableName = expression.

The **datatype** specifies the type of value a variable can hold; the expression is the content being held in the variable.

The variableName can be any letter, followed by a number or other letters. We are not allowed to have operators (+, -, /, %) in our variable names. See example below:

a, name, studentName, hourlyRate, start001

Below are some java statements (variables and expressions).

- int a = 5;
- b = a;
- b = b + 12; // valid: assignment operator is not equals operator
- c = a + b;
- a + b = c; // invalid: left side not a variable

Hello, World (A simple Java Program)

```
public class HelloWorld {
    public static void main(String[] args) {
        // Program execution starts here
        System.out.println("Hello, World!");
    }
}
```

The 3 Stages of Java Program Development

1. **Coding (Editing):** This is the process of writing java statements with the correct syntax and rules. The program must be saved with an extension ".java".
2. **Compiling:** This is the process of converting the java program into byte codes. This can be accomplished by using the javac compiler to create .class file which contains the byte codes.
3. **Execution (Running): To** execute a Java program, you use the java command followed by the name of the main class. For example, if your main class is named Main, you would run

"java Main" in the command line. **The JVM plays** a very important role in execution of the program; it starts up and loads the byte code of the specified class, and begins executing the main method, which serves as the entry point of the program.

JVM Role in Java Execution

The Java Virtual Machine (JVM) is a software program that interprets Java bytecode and executes it on the host platform. The JVM is responsible for the following tasks:

Here is a more detailed description of what it does:

- **Class loading**: When a Java program is executed, the JVM loads the Java classes that the program needs into memory. The JVM uses a class loader to do this. The class loader is responsible for finding the Java classes on the classpath and loading them into memory.
- **Verification**: Once the Java classes are loaded into memory, the JVM verifies them to ensure that they are safe to run. The verification process ensures that the Java bytecode is well-formed and that it does not contain any malicious code.
- **Execution**: Once the Java bytecode has been verified, the JVM executes it. The execution process is responsible for translating the Java bytecode into machine code and running it on the host platform.
- **Garbage collection**: The JVM manages memory for Java objects using a garbage collector. The garbage collector is responsible for identifying objects that are no longer being used and freeing up the memory that they are using.
- **Security**: The JVM provides security features to protect Java programs from malicious code. These features include sandboxing, code verification, and exception handling.

The JVM is a complex piece of software, but it is essential for running Java programs. It is responsible for ensuring that Java programs are safe, secure, and efficient.

Practical Sessions

- A program using println; printf, print.
- A tabulated program output to the console: \t \n %s %d special characters.
- Using variables to print strings and manipulation of strings
- Addition program using direct value and variables.
- Show control flow through debugger.
- Use scanner to take user input
- Introduce arrays

DAY 3

Java Naming Convention

Naming in Java follows a particular unwritten rule which makes Java codes easily readable to every programmer. It specifies best practices in class names, variable names and method names.

Classes: Noun or Noun phrase, every first letter of each word must be capitalized. Can not contain underscore or hyphen. It contains Identity (class name), attributes (class variables) and methods (class behavior).

Examples: Tree, DatePalm, Student, GraduateStudent, BankAccount, InputStreamReader etc.

Variables: Noun or Noun phrase or adjective. It describes one of the attributes of the class. First word/only word will be all lowercase. For phrases, every first letter of subsequent word(s) must be capitalized.

Examples: name, email, emailAddress, firstName, annualSalary, yearOfEmployment etc.

Constants: The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)

Examples: ***static final int MIN_WIDTH = 4;***

static final double MAX_WIDTH = 999.90;

static final String DB_NAME = “customer”;

Methods: Verb or verb phrase. It describes an action that can be performed by the method. For phrases, every first letter of subsequent word(s) must be capitalized.

Examples: getName(), getEmail(), getEmailAddress(), getFirstName(), getAnnualSalary(), setGraduationYear(int year), finalizeTransaction(long id, double amount) etc.

The above naming convention are very essential for code readability and adherence to best standards. If you fail to follow above rules, your code will run perfectly but you may not be able

to read the code if you return after even a day. Always adhere to the naming principles to be a **World Class** developer.

Java Data Types

There are two main types of data in Java; Primitive Data types and Reference Data types. Let's examine the difference and the different types in each category.

- **Primitive Data Types**

There are eight primitive data types in Java.

1. **byte**: Represents a 8-bit signed integer. Range: -128 to 127.
2. **short**: Represents a 16-bit signed integer. Range: -32,768 to 32,767.
3. **int**: Represents a 32-bit signed integer. Range: -2,147,483,648 to 2,147,483,647.
4. **long**: Represents a 64-bit signed integer. Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
5. **float**: Represents a 32-bit floating-point number.
6. **double**: Represents a 64-bit floating-point number.
7. **boolean**: Represents a boolean value, which can be either true or false.
8. **char**: Represents a single character. Range: 0 to 65,535 (unsigned).

- **Reference Data Types**

1. **Class/Object:** A class is a blueprint for an object. It defines the data and methods that an object will have. It could be user developed on inbuilt e.g. String, Integer, Double etc.
2. **Interface:** An interface is a blueprint for a class. It defines the methods that a class must implement.
3. **Array:** An array is a data structure that can store a fixed number of values of the same type.
4. **String:** A string is a sequence of characters. It can be as long as possible and may contain numbers, special characters etc.

Casting

i. A cast is an explicit conversion of a value from its current type to another type.
ii. The syntax for a cast is: (type) expression

We have two main types of casting: Widening and Narrowing.

Widening refers to where you convert a smaller type to a bigger type. This may not need the type specification in parenthesis as shown in bullet two above. See example below;

```
int x = 30, y = 45;
double sum = x + y; // widening casting
```

Narrowing refers to conversion of a bigger type downwards to a smaller type. See example below;

```
double height = 3.5;
double weight = 8.0;
int total = (int) height + weight; //narrowing casting
```

Practical Session

Debugging - pending

- Debugging Steps
- Watch points
- Step in
- Step through
- Break points
- Variable inspection

DAY 4

Day 1 to Day 3 Recap on all subjects.

More about Java Programs

Arithmetic Operators

Operator	Description
+	Adds op1 and op2
-	Subtracts op2 from op1
*	Multiplies op1 by op2
/	Divides op1 by op2
%	Remainder of dividing op1 by op2

Let's look at some examples using the operators above.

Arithmetic expression	Value
`1 / 2`	`0`
`86 / 10`	`8`
`86 / 10.0`	`8.6`
`86.0 / 10`	`8.6`
`86.0 / 10.0`	`8.6`

86 % 10	6

Arithmetic Operator Priority

- An expression is a sequence of variables, constants, operators, and method calls that evaluates to a single value.
- Arithmetic expressions are evaluated according to the priority rules.
- All binary operators are evaluated in left to right order.
- In the presence of parenthesis, evaluation starts from the innermost parenthesis.

Operators	Priority (Precedence)
+ - (unary)	1
* / %	2
+ - (binary)	3

Expression	Value
3 + 7 % 2	4
(2 – 5) * 5 / 2	-7
2 – 5 + 3	0

The Math Class

Many mathematical functions and constants are included in the Math class of the Java library. Some are:

Function /constant	**Meaning**
sqrt(x)	Returns the square root of x.
abs(x)	Returns the absolute value of x, x can be double, float, int or long.

cos(a), sin(a), tan(a)	Returns the trigonometric cosine/sine/tangent of an angle given in radians
exp(x)	Returns the exponential number e raised to the power of x
log(x)	Returns the natural logarithm (base e) of x
max(x, y) , min(x, y)	Returns the greater/smaller of two values, x and y can be double, float, int or long
pow(x, y)	Returns x^y
PI	The approximate value of PI

Short Hand Assignment Operators

Java provides several shorthand assignment operators:

Example:

```
a += 5;   // equivalent to a = a + 5;
```

Short-Form	**Equivalent to**
`op1 += op2`	op1 = op1 + op2
`op1 -= op2`	op1 = op1 – op2
`op1 *= op2`	op1 = op1 * op2
`op1 /= op2`	op1 = op1 / op2
`op1 %= op2`	op1 = op1 % op2

Increment and Decrement Operators

Increment/decrement operations are very common in programming. Java provides operators that make these operations shorter.

Operator	**Use**	**Description**
++	op++	Post Increments op by 1;
++	++op	Pre-Increments op by 1;

--	op--	Post Decrements op by 1;
--	--op	Pre-Decrements `op` by 1;

Relational Operators

Operator	Meaning
==	equal
!=	not equal
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal

The result of a comparison is boolean.

Boolean Operators

Operator	Meaning
&&	logical and
||	logical or
!	negation

EXTERNAL LINKS

1. History of Java
2. Java Virtual Machine
3. OOP in Java
4. Automatic Memory Management in Java
5. Structure of Java Program
6. Stages of Java Program Development
7. Naming Convention in Java Programming Language
8. Data types in Java Programming Language
9. Debugging in Java Program