Module : 1

1. Introduction to Java

* Theory :
* History of Java :
* Java is a programming language originally developed by James Gosling at Sun Microsystems.
* and released in 1995 as a core component of Sun Microsystems' Java platform.
* The language derives much of its syntax from C and C++.
* Java is currently one of the most popular programming languages in use, particularly for client server web applications.
* Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.
* Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.
* Java's motto was **"Write Once, Run Anywhere"** (WORA), highlighting its platform independence using the Java Virtual Machine (JVM).
* Oak was renamed to **Java** (the name was inspired by Java coffee).
* Summary :
*  **Creator**: James Gosling and team at Sun Microsystems
*  **First Released**: 1995
*  **Current Owner**: Oracle Corporation
*  **Key Features**: Platform independence, robustness, security, object-oriented
*  **Modern Usage**: Continues to evolve with regular updates and broad community support.
* Features of Java :
* Simple
* Object – Oriented
* Portable
* Distribute
* High Performance
* Multithreaded
* Robust
* Dynamic
* Secure
* Object – Oriented :
* Java is fully Object-oriented programming language. & Object oriented concepts are following.
* Class : A blueprint for creating objects. It defines properties (fields) and behaviors (methods).
* Object : A real-world entity created from a class. It has a state and behavior.
* Inheritance : Bundling data and methods that operate on that data within a class, and hiding internal details from outside access.
* Encapsulation : A mechanism where one class acquires properties and behavior of another class.
* Polymorphism : Ability to take many forms. There are two types:   
   • **Compile-time** (method overloading)   
   • **Runtime** (method overriding)
* Platform Independent :
* **Platform independence** means that a Java program can run on any operating system or hardware platform **without modification**.
* Allows Java applications to run across different operating systems without rewriting code.
* **Source Code Compilation:**
* Java code (.java file) is compiled by the Java compiler (javac) into an intermediate format called **bytecode** (.class file).
* This bytecode is **not** platform-specific.
* **Java Virtual Machine (JVM):**
* The bytecode is executed by the **JVM**, which is platform-specific.
* JVM interprets the bytecode and runs it on the native machine.
* Understanding JVM, JRE, and JDK
* JVM : Java Virtual Machine
* It is a platform-independent execution environment that converts the Java bytecode into machine language and executes it.
* Most programming languages compile source code directly into machine code that is designed to run on a specific operating system, such as Windows or UNIX.
* JVMs are available for many hardware and software platforms.
* The use of the same bytecode for all JVMs on all platforms allows Java to be described as a "compile once, run anywhere" programming language, as opposed to "write once, compile anywhere", which describes crossplatform compiled languages.
* Thus, the JVM is a crucial component of the Java platform.
* JRE : Java Run-Time Environment
* Java Runtime Environment contains JVM, class libraries, and other supporting files.
* It does not contain any development tools such as compiler, debugger, etc.
* Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE.
* . If you want to run any java program, you need to have JRE installed in the system.
* Is a part of the Java Development Kit (JDK) that provides the necessary components to run Java applications.
* It acts as a layer between the Java application and the underlying operating system, handling the execution of Java bytecode.
* JDK : Java Development Kit
* is a comprehensive software development environment used for developing Java applications and applets.
* Java Developer Kit contains tools needed to develop the Java programs
* **Appletviewer:-** (For viewing java applets) this tool can be used to run and debug Java applets without a web browser
* **Javac:-** Javac means Java Compiler, which converts source code into Java bytecode.
* **Java :-** ( java interpreter) the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment.
* The old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader.
* **Javap :-** ( java disassembler) the class file disassemble.
* **Javah:-** (produce header files) the C header and stub generator, used to write native methods.
* **Java doc:-** (creating html document) the documentation generator, which automatically generates documentation from source code comments.
* **Jdb:-** java debugger.
* Setting up the Java environment and IDE (e.g., Eclipse, IntelliJ)
* Step 1: Install Java Development Kit (JDK)
* **Download and Install:**
* Go to the official [Oracle JDK download page](https://www.oracle.com/java/technologies/javase-downloads.html) or use [OpenJDK](https://jdk.java.net/).
* Download the latest LTS (Long Term Support) version (e.g., Java 17 or Java 21).
* Install it and note the installation path.
* **Set Environment Variables (if not automatically set):**
* **On Windows :**
* Open System Properties > Environment Variables.
* Add a new system variable:
  + **Name:** JAVA\_HOME
  + **Value:** C:\Program Files\Java\jdk-XX
* Edit the Path variable and add: %JAVA\_HOME%\bin
* **Check installation:**
* bash
* java -version
* javac -version

Step 2: Install an IDE

* **Option A: Eclipse IDE**
* **Download & Install:**
* Download from: https://www.eclipse.org/downloads/
* Choose “Eclipse IDE for Java Developers” and install it.
* **First-Time Setup:**
* Launch Eclipse.
* Choose a workspace folder (your projects will be stored here).
* Install any updates it recommends.
* Create a new Java project via:
* File > New > Java Project
* **Option B: IntelliJ IDEA**
* **Download & Install:**
* Download from: <https://www.jetbrains.com/idea/download/>
* Choose the **Community Edition** (free) or **Ultimate** (paid).
* **First-Time Setup:**
* Run IntelliJ and complete the first-time setup wizard.
* Create a new project:
* New Project > Java > Select SDK
* (You can add your installed JDK here.)
* Start coding in the **src** folder.
* IDEs support Maven or Gradle for managing dependencies.
* Java Program Structure (Packages, Classes, Methods)
* **1. Document Section :-** In Documentation section one can write author name, definition of class or a program , description of a program and how this algorithm works and other details, and this is OPTIONAL part and this part write in (/\* \*/)multiline comment.
* **2. Package Statement :-** In java program first statement one can write is a package statement. This statement declares a package name and tells the compiler that the classes included here belongs to this package; the package statement is optional in any java program For ex:-package first; .
* **3. Import Statement:-** Once a package is declared one can write any number of import statements. import statement can be write after package statement and before the class definition. For ex:-import java.util.\*; Import first.\*; .
* **4. Class Declaration :-** In java program there may be more than one class. In this class definition statement we have to provide keyword class along with the class name. A class can include variable as well as methods.
* **5. Main Method :-** In a java program there may be more than one class one class can have main method. This is the essential part of java program. In main method one can create objects of different classes and with the use of there
* **Lab Exercise:**
* Install JDK and set up environment variables.
* Write a simple "Hello World" Java program.
* Compile and run the program using command-line tools (javac, java).
* **Program :**

**public** **class** hello {

**public** **static** **void** main(String[] args)

{

System.***out***.println("hello world");

}

}

1. **Data Types, Variables, and Operators**

* **Theory :**
* **Primitive Data Types in Java (int, float, char, etc.)**
* The primitive data types are predefined data types, which always hold the value of the same data type, and the values of a primitive data type don't share the state with other primitive values.
* These data types are named by a reserved keyword in Java programming language.
* There are eight primitive data types supported by Java programming language:
* **byte :**
* The byte data type is an 8-bit signed two's complement integer. It ranges from -128 to127 (inclusive).
* This type of data type is useful to save memory in large arrays.
* We can also use byte instead of int to increase the limit of the code.
* Syntax : byte b = 5;
* **short:**
* The short data type is a 16-bit signed two's complement integer.
* It ranges from -32,768 to 32,767.short is used to save memory in large arrays.
* **Syntax :** short s = 2;
* **int :** The int data type is used to store the integer values not the fraction values.
* It is a 32-bit signed two's complement integer data type.
* It ranges from -2,147,483,648 to 2,147,483,647 that are more enough to store large number in your program.
* However for wider range of values use long.
* **float :**
* The float data type is a single-precision 32-bit IEEE 754 floating point.
* It ranges from 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative).
* Use a float (instead of double) to save memory in large arrays.
* **double :**
* This data type is a double-precision 64-bit IEEE 754 floating point.
* It ranges from 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative).
* This data type is generally the default choice for decimal values.
* **char :**
* The char data type is a single 16-bit, unsigned Unicode character.
* It ranges from 0 to 65,535.
* They are not integral data type like int, short etc. i.e. the char data type can't hold the numeric values.
* **Boolean :**
* The boolean data type represents only two values: true and false and occupy is 1-bit in the memory.
* These values are keywords in Java and represent the two boolean states: on or off, yes or no.
* We use boolean data type for specifying conditional statements as if, while, do, for.
* Variable Declaration and Initialization
* A **variable** is a name that refers to a memory location used to store data.
* **Instance Variables (Non-static fields):**
* In object oriented programming, objects store their individual states in the "non-static fields" that is declared without the static keyword.
* **Class Variables (Static fields):** These are collectively related to a class and none of the object can claim them its soleproprietor.
* The variables defined with static keyword are shared by all objects.
* **Local Variables:** The variables defined in a method or block of code is called local variables.
* **Parameters:** Parameters or arguments are variables used in method declarations.
* Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise
* **Operators**
* + Additive operator (also used for String concatenation)
* - Subtraction operator
* \* Multiplication operator
* / Division operator
* % Remainder operator
* **Unary Operators**
* ++ Increment operator; increments a value by 1
* -- Decrement operator; decrements a value by 1
* ! Logical complement operator
* **Relational Operator**
* = = Equal to
* != Not equal to
* > Greater than
* > = Greater than or equal to
* < Less than
* < = Less than or equal to
* **Conditional Operators**
* && Conditional-AND
* || Conditional-OR
* ?: Ternary (shorthand for if-then-else statement)
* **Bitwise and Bit Shift Operators**
* ~ Unary bitwise complement
* << Signed left shift
* >> Signed right shift
* >>> Unsigned right shift
* & Bitwise AND
* ^ Bitwise exclusive OR
* | Bitwise inclusive OR
* Type Conversion and Type Casting
* (Widening / Implicit Casting)
* In Java, **Type Conversion** and **Type Casting** are processes used to convert one data type into another. Though often used interchangeably, they refer to slightly different concepts depending on whether the conversion is done **automatically** or **manually**.
* This is the automatic conversion of a smaller data type to a larger data type. It happens when there is no risk of data loss.
* **Also known as:** Widening Conversion or Implicit Casting
* When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:
* The two types are compatible.
* The destination type is larger than the source type.
* Although the automatic type conversions are helpful, they will not fulfill all needs.
*  Done automatically by the compiler
*  No data loss
*  Safe and preferred when moving from smaller to larger data types
* Type Casting(Narrowing / Explicit Casting)
* This is the manual conversion of a larger data type to a smaller data type. It must be explicitly done by the programmer because it may result in data loss.
* **Also known as:** Narrowing Conversion or Explicit Casting
* **Ex.:**
* double d = 10.5;
* int num = (int) d;
* **Key Points:**
* Requires explicit syntax: (targetType) value
* Can lead to data loss (e.g., truncating decimal values)
* Must be used carefully
* **Lab Exercise:**
* Write a program to demonstrate the use of different data types.

**public** **class** DatatypeDemo {

**public** **static** **void** main(String[] args)

{

**byte** b= 5;

**short** s = 2;

**int** i = 10;

**float** f = 12.1f;

**double** d = 6677.60;

**char** ch = 'c';

**boolean** bl = **false**;

System.**out**.println("This byte : "+b);

System.**out**.println("This short : "+s);

System.**out**.println("This integer : "+i);

System.**out**.println("This float : "+f);

System.**out**.println("This double : "+d);

System.**out**.println("This char : "+ch);

System.**out**.println("This boolean : "+b);

}

}

* Create a calculator using arithmetic and relational operators.
* Demonstrate type casting (explicit and implicit).