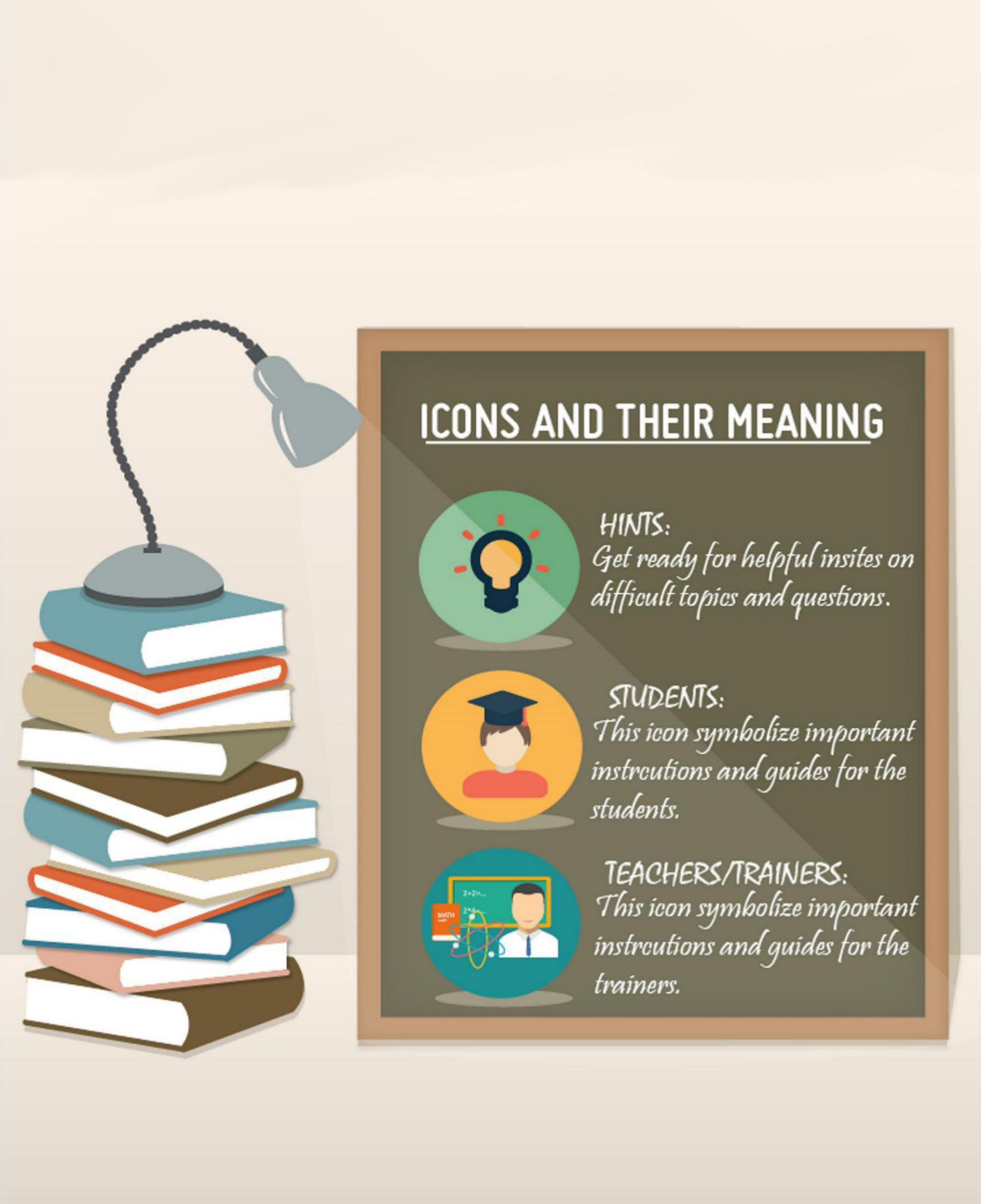
CORE JAVA

MANUAL V8.3

MODULE CODE:

ANUDIP FOUNDATION



ICONS AND THEIR MEANING

HINTS:
Get ready for helpful insites on difficult topics and questions.

STUDENTS:
This icon symbolize important instrcutions and guides for the students.

TEACHERS/TRAINERS:
This icon symbolize important instrcutions and guides for the trainers.

Module 1: Java Fundamental and Programming Concepts

Chapter 2

Objective: After completing this lesson you will be able to : Understand the architecture of a JVM Know how to install Java Gain an idea about some basic Java commands	**Materials Required:** 1. Computer 2. Internet access
Theory Duration: 90 minutes	**Practical Duration:** 30 minutes
Total Duration: 120 minutes	

Chapter 2

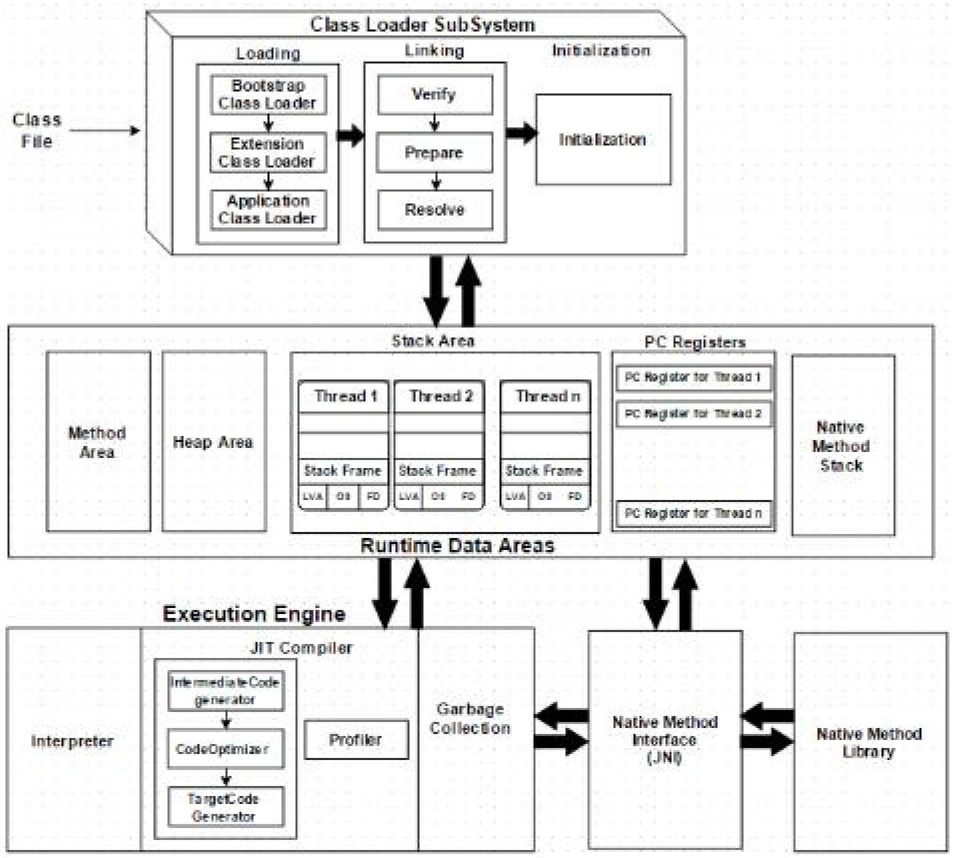
2.1 Overall Architecture of JVM with JIT and JRE

A Java Virtual Machine (JVM) is the software-based engine necessary for running Java bytecode applications. It is responsible for performing bytecode analysis, understanding the code and then running it. Gaining awareness about the architecture of JVM helps programmers write code systematically.

A JVM is responsible for loading a file that has been written and compiled into a .class file. Execution of class files is performed in a stepwise manner by the different components of a Java Virtual Machine.

The architecture of a JVM

A Java Virtual Machine (JVM) has three main subsections. Take a look at this diagram to gain an understanding of the components.



As evident from the above diagram, the three sections are the –

1. **ClassLoader Subsystem**

A classloader subsystem is responsible for performing the dynamic class loading functions of Java. It is responsible for loading, linking and initialising compiled .class files.

A classloader subsystem has designated sections for loading, linking and initialisation. Take a look at the components of each section below.

* **Loading** – The loading section of a classloader subsystem consists of three different loaders. These are –
i) **Extension ClassLoader** – Loads classes located inside an ext folder (jre/lib)
ii) **BootStrap ClassLoader** – Typically loads rt.jar classes and other core libraries from the bootstrap classpath. This loader is prioritised over others.
iii) **Application ClassLoader** – Loads application-level classpaths.

The Delegation Hierarchy Algorithm is utilised by the loaders whilst loading Java class files.

* **Linking** – The linking section of the ClassLoader Subsystem performs three integral steps –
i) **Verification** – The linking bytecode verifier makes sure if the correct bytecode is being used. The verifier issues an error message if the code fails verification.
ii) **Preparation** – This component is responsible for memory allocation and assigning default values to static variables.
iii) **Resolving** – It replaces symbolic references with original Method Area references.

* **Initialisation** – This section of a ClassLoader designates original values to all static variables. Java static block code is executed in the initialisation phase.

2. **Runtime Data Area**

* **Method Area** – It is a shared resource of the JVM that stores class-level data.

* **Heap Area** – It stores all objects, along with their respective arrays and variables.

* **Stack Area** – Separate runtime stacks are created for individual threads. When a new method is initiated, a new entry is made into the Stack Frame or memory (see above diagram for reference). A Stack Frame is further divided into –

i) **Local variable array** – Stores local variables and their respective values

ii) **Operand stack** – Is a runtime for performing intermediate actions

iii) **Frame data** – Stores method-related symbols

* **PC Registers** – Individual threads are assigned their designated PC registers, to be able to store executing instruction address. After execution, the PC register moves on to another instruction.

* **Native Method stacks** – They store native method information. Separate methods are created for each thread.

3. **Execution Engine** – It executes bytecode allocated to the Runtime Data Area. This engine reads and runs bytecode. It has two sections –

* **Interpreter** – It performs fast bytecode reading but slow execution. It has a drawback as fresh interpretations are required upon multiple-time initiations of a single method.

* **JIT Compiler** – The Just in Time (JIT) compiler is used in cases where the engine finds repeated code. It compiles bytecode into native code. Performance is enhanced as the converted native code is utilised for repeated calls. The JIT has –

i) **Intermediate Code generator** – Generates intermediate code

ii) **Code Optimiser** – Optimises generated intermediate code

iii) **Target code generator** – It generates native code

iv) **Profiler** – This component detects if a method is being called multiple times.

* **Garbage Collector**: Eliminates unreferenced Java objects, and collects created objects.

JRE (Java Runtime Environment)

JVM is implemented through the JRE or Java Runtime Environment. It provides software tools for application development. JRE acts as a runtime environment for JVM and consists of libraries and additional files.

2.2 Installation of Java

The process of installing Java varies slightly for Windows, Mac and Linux systems. But it involves a few simple steps.

Java installation steps include –

* Downloading a Java Software Development Kit (JDK) installer for macOS, Windows or Linux

* Try finding the latest available version of the JDK.

* Download the installer relevant to your operating system

* Run the installer and follow the installation steps

* Use an online testing method to determine if Java has been successfully installed.

2.3 Command and tools (javac, java)

*** Javac** – Javac is the commonly used Java compiler. It is responsible for compiling bytecode to be executed by the JVM.

*** Java** – It is a command required to launch a Java application.

*** Javap** – Used to disassemble single or multiple class files

*** Javadoc** – Uses Java source files to create API documentation HTML pages

*** Jmod** – Used for creating JMOD files

*** Java -version** – Command for checking Java version

Practical (30 minutes) – Search and install a Java Development Kit (JDK) on a computer system. Then use the command line to execute a Hello World programme.

Instructions: The progress of students will be assessed with the exercises mentioned below.

MCQ

1. Java Virtual Machine is a software-based _____.

a) system

b) engine

c) class

d) none of the mentioned

2. A JVM loads a Java ______ file.

a) installation

b) dash

c) class

d) hash

3. A JVM has ________ fundamental subsections.

a) 6

b) 10

c) 4

d) 3

4. What type of .class files can a classloader subsystem initialise?

a) standalone

b) compiled

c) disassemble

d) None of the mentioned

5. From where does the Extension ClassLoader load files?

a) ext folder

b) dir folder

c) data folder

d) All of the mentioned

6. The ClassLoader Subsystem linking section is responsible for verifying _______.

a) JavaScript

b) Java array

c) bytecode

d) All of the mentioned

7. What does the method area of Runtime Data Area store?

a) class-level data

b) objects

c) run commands

d) None of the above

8. The Operand stack is a ________

a) Applet

b) executable file

c) storage

d) runtime

9. The JIT of JIT compiler stands for _____ _____ ______.

a) Java Infrastructure Testing

b) Just in Time

c) Java Independent Testing

d) None of the above

10. Javac is a ______

a) Library

b) Directory

c) Class

d) None of the above