Disclaimer:
This lecture material is prepared using the book *JAVA: The Complete Reference* by *Herbert Schildt*

Chapter 2 Introduction to JAVA

History of JAVA

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

James Gosling

Oak Tree

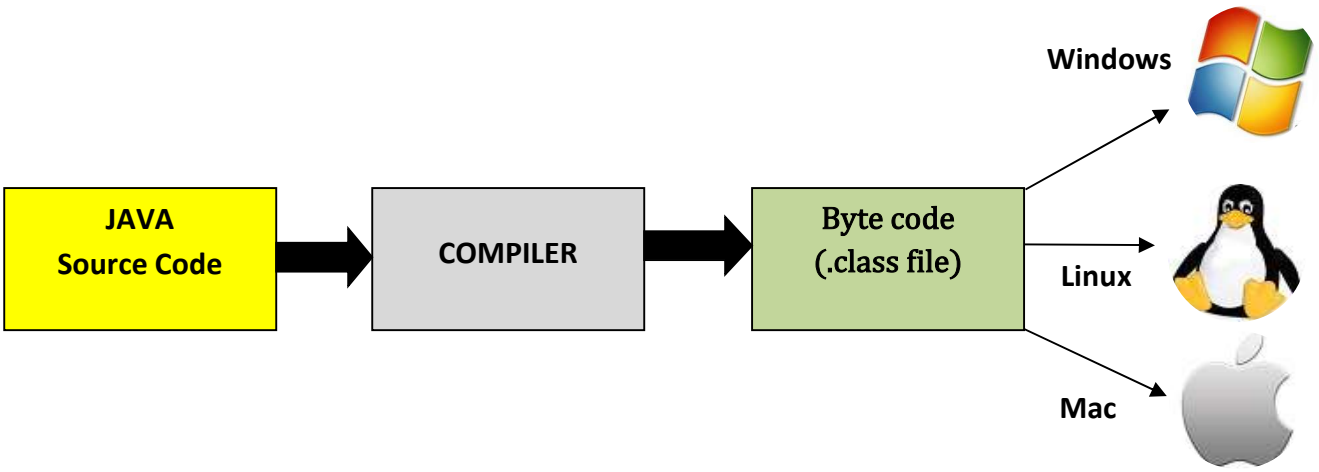
JAVA Coffee

- **James Gosling** is the founder of java.
- *James Gosling*, *Mike Sheridan*, and *Patrick Naughton* initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- Originally designed for small, embedded systems in electronic appliances like set-top boxes. Firstly, it was called "*Greentalk*" by James Gosling, and file extension was .gt. After that, it was called Oak and was developed as a part of the Green project.
- **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.
- In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.
- Java is an island of Indonesia where first coffee was produced (called java coffee).
- Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**. JDK 1.0 released in January 23, 1996.

JAVA Byte Code

The intermediate machine independent codes which are formed after the successful compilation of a java program are called byte code.

Byte code of a java program is always saved in the extension **.class** file. The Java byte code is not completely compiled, but rather just an intermediate code sitting in the middle because it still has to be interpreted and executed by the **JVM** installed on the specific platform such as Windows, Mac or Linux. Due to the above reason, java is called a platform independent language or write once run anywhere (**WORA**) language.



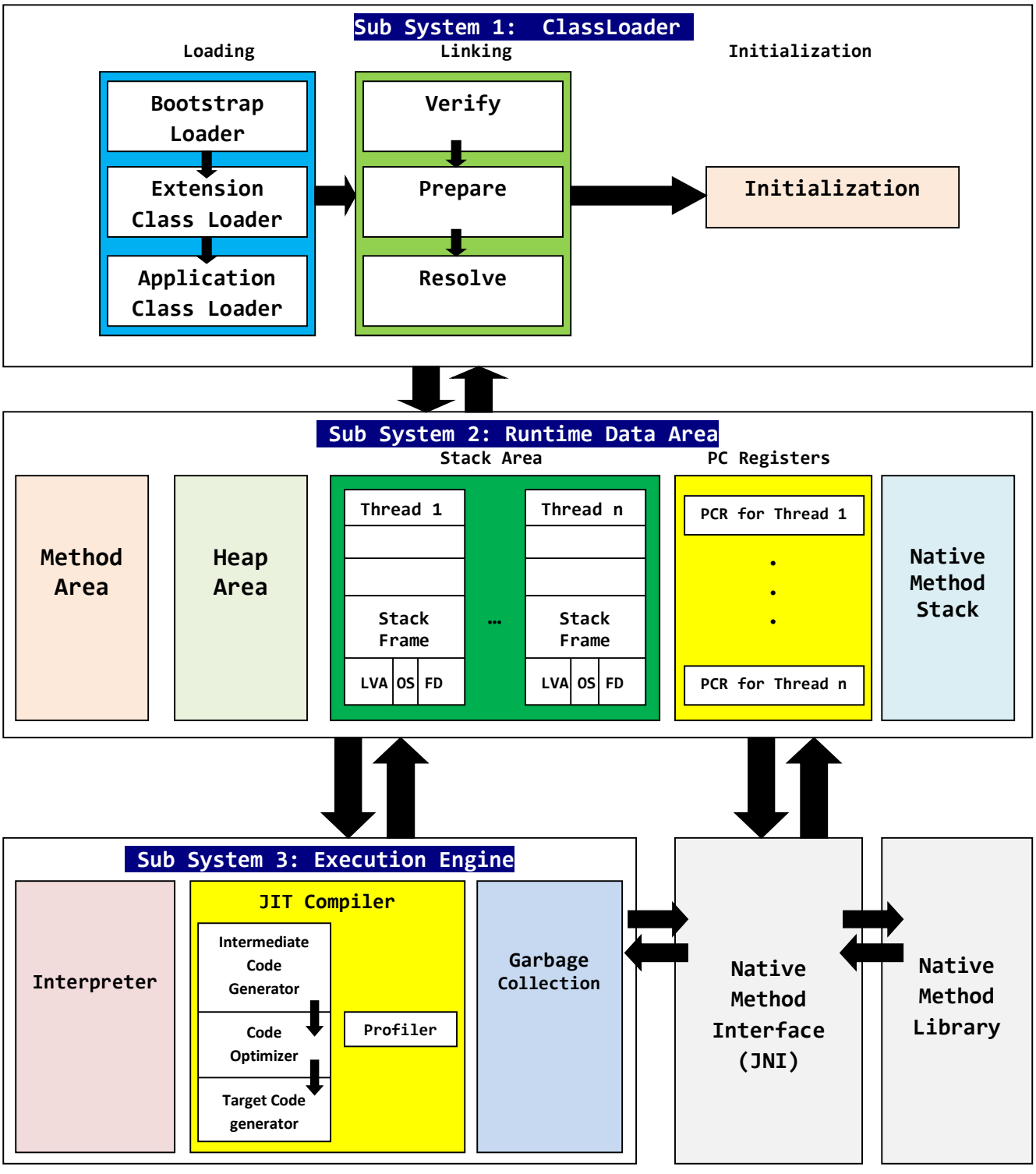
JVM- Java Virtual Machine

A Virtual Machine is a software implementation of a physical machine. Java was developed with the concept of WORA (Write Once Run Anywhere), which runs on a VM. The compiler compiles the Java file into a Java .class file, then that .class file is input into the JVM, which loads and executes the class file. Below is a diagram of the Architecture of the JVM.

JVM Architecture

As shown in the below architecture diagram, the JVM is divided into three main subsystems:

- ClassLoader
- Runtime Data Area
- Execution Engine



[Diagram of JVM Architecture]

Subsystem 1: ClassLoader

Java's dynamic class loading functionality is handled by the ClassLoader subsystem. It loads, links and initializes the class file when it refers to a class for the first time at runtime, not compile time.

Loading

Classes will be loaded by this component. BootStrap ClassLoader, Extension ClassLoader, and Application ClassLoader are the three ClassLoaders that will help in achieving it.

- **BootStrap ClassLoader** – Responsible for loading classes from the bootstrap classpath, nothing but **rt.jar.** Highest priority will be given to this loader.
- **Extension ClassLoader** – Responsible for loading classes which are inside the ext folder **(jre\lib).**
- **Application ClassLoader** –Responsible for loading Application Level Classpath, path mentioned Environment Variable, etc.

The above ClassLoaders will follow Delegation Hierarchy Algorithm while loading the class files.

Linking

- **Verify** – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error.
- **Prepare** – For all static variables memory will be allocated and assigned with default values.
- **Resolve** – All symbolic memory references are replaced with the original references from Method Area.

Initialization

This is the final phase of ClassLoading; here, all static variables will be assigned with the original values, and the static block will be executed.

Subsystem 2: Runtime Data Area

The Runtime Data Area is divided into five major components:

1. **Method Area** – All the class-level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.

2. **Heap Area** – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.
3. **Stack Area** – For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three sub entities:
    - **Local Variable Array** – Related to the method how many local variables are involved and the corresponding values will be stored here.
    - **Operand stack** – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.
    - **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data.
4. **PC Registers** – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.
5. **Native Method stacks** – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

Execution Engine

The bytecode, which is assigned to the **Runtime Data Area,** will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.

1. **Interpreter** – The interpreter interprets the bytecode faster but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.
2. **JIT Compiler** – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to

native code. This native code will be used directly for repeated method calls, which improve the performance of the system.

- **Intermediate Code Generator** – Produces intermediate code
- **Code Optimizer** – Responsible for optimizing the intermediate code generated above
- **Target Code Generator** – Responsible for Generating Machine Code or Native Code
- **Profiler** – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.

3. **Garbage Collector**: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.

Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.

Native Method Libraries: This is a collection of the Native Libraries, which is required for the Execution Engine.

Java Buzz words: Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java *buzzwords*. A list of most important features or buzz words of Java language is:

- **Simple**
- **Secure**
- **Portable**
- **Object-Oriented**
- **Robust**
- **Multi threaded**
- **Architecture neutral**
- **Distributed**
- **Dynamic**

Now let us briefly explain the above buzz words.

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

Java syntax is based on C++ (so easier for programmers to learn it after C++).

Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.

There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Secure

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because no explicit pointer is used in java &java programs run inside a virtual machine sandbox, which is insulated from virus.

Portable

Java is portable because it facilitates you to carry the Java byte code to any platform. It doesn't require any implementation.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Robust

Robust simply means strong. Java is a robust language due to following reason:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Multi-threaded

Java programs can execute many tasks simultaneously by defining multiple threads thereby achieving multitasking. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Architecture Neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. In C programming, *int* data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e. C and C++. Java supports dynamic compilation and automatic memory management (garbage collection).

Comparison between C++ & Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Parameter	C++	Java

Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.

Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for same.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

✍ **Points To Remember :**

- Java doesn't support default arguments like C++.
- Java does not support header files like C/C++. Java uses the **import** keyword to include different classes and methods.

Review Questions of Chapter 2

MCQ Type

1. Choose the most appropriate answer:

a) A platform is the hardware or software environment in which a program runs. Which of the following is/are java platform component(s)?

a) HTML b) Java Virtual machine
c) Hot Java d) Java API

b) Which of the following provides run time environment for java byte code to be executed?

a) JVM b) JRE
c) JDK d) JAVAC

c) Which component is used to compile, debug and execute java program?

a) JVM b) JRE
c) JDK d) JIT

d) Which component is responsible for converting byte code into machine specific code?

a) JVM b) JRE
c) JDK d) JIT

e) Which component is responsible to run java program?

a) JVM b) JRE
c) JDK d) JIT

f) Which component is responsible to optimize byte code to machine code?

a) JVM b) JRE
c) JDK d) JIT

g) Which statement is true about java?

a) Platform dependent b) Platform independent
c) Code dependent d) Sequence dependent

h) What is the extension of java code files?

a) .java b) .class
c) .binary d) .obj

i) What is the extension of java byte code files?

a) .java b) .class
c) .binary d) .obj

j) Java language is

a) Robust b) Purely object oriented
c) Portable d) All of these

Short Type

2. Answer briefly

a) Define byte code.
b) Define JVM.
c) Why java is called a portable language?

d) Why java is called a robust language?
e) Why java is called a dynamic language?
f) Why java is architecture neutral?
g) What is JDK, JIT & JRE?
h) Define multithreading.
i) Who is the father of JAVA?
j) Why java is called a platform independent language?
k) Why java is called write once run anywhere language (WORA)?
l) Why java programs are secure over internet?

Long Type

3. Answer in details

a) Discuss the features of java which makes it a powerful language.
b) What is JVM? Explain the architecture of JVM with the help of a neat diagram.
c) Differentiate between C++ and JAVA in details.

****