Chapter 12: Compilation and Execution of Java Programs

The topics

1. Java Programming Language

The Java programming language was designed keeping in mind the changing nature of computing environments. With the rapid growth of the Internet and the World Wide Web, software is no longer used on a single computer. Programs now need to run on different machines connected through networks, especially for online services and electronic commerce. Therefore, Java was created to support secure, fast, and reliable applications that can work efficiently in such network-based environments.

Since Java programs are expected to run on multiple platforms in a heterogeneous network, traditional methods of distributing different binary versions for different machines are no longer practical. To overcome this problem, Java was designed to be architecture-neutral and portable. This allows the same Java program to run on different operating systems and hardware without modification. Java is also dynamically adaptable, meaning programs can be updated and used easily over networks.

To meet these requirements, Java was designed to be simple and familiar so that programmers can learn and use it easily. It follows the object-oriented approach, which helps in organizing programs better and makes them suitable for large, distributed applications. Java also supports multithreading, which allows a program to perform multiple tasks at the same time, improving performance. Being an interpreted language, Java provides high portability and flexibility, making it ideal for modern Internet-based applications.

2. Characteristics Of Java

2.1 Simple, Object-Oriented, and Familiar

Java is designed to be a **simple programming language** that can be learned and used easily, even by beginners. It does not require extensive training, and its basic concepts can be understood quickly. As a result, programmers can start writing useful programs from the very beginning. Java follows modern software development practices, which makes it suitable for today's applications.

Java is **object-oriented by design**, meaning it is built completely around objects and classes. Object-oriented concepts such as encapsulation and message passing are especially useful in **distributed and client–server applications**. As software systems become more complex and network-based, object-oriented programming becomes essential. Java provides a clean, efficient, and well-structured platform for developing such applications.

Java also provides a **large library of pre-written and tested classes**. These libraries support basic data types, input/output operations, networking, and graphical user interfaces. Programmers can use these libraries directly or extend them to add new functionality, which saves time and improves reliability.

Although Java was not implemented using C++, its syntax is intentionally kept like C++. This makes Java a **familiar language** for programmers who already know C or C++. At the same time, Java removes many complex and unsafe features of C++, allowing programmers to migrate easily to Java and become productive quickly.

2.2 Robust and Secure

Java is designed to produce **robust and reliable programs**. It performs extensive error checking at compile time and during program execution. These checks help programmers detect errors early and encourage good programming practices.

Java uses a **simple and safe memory management system**. Objects are created using the new keyword, and memory is automatically released using garbage collection. There are no explicit pointers or pointer arithmetic, which removes many common errors found in C and C++ programs. As a result, Java programs are less likely to crash due to memory-related problems.

Since Java programs often run in **network and distributed environments**, security is extremely important. Java has built-in security features in both the language and run-time system. These features protect systems from unauthorized access, viruses, and malicious code, making Java suitable for Internet-based applications.

2.3 Architecture Neutral and Portable

Java is designed to work in **heterogeneous network environments**, where programs must run on different hardware and operating systems. To achieve this, the Java compiler converts source code into **bytecode**, which is an architecture-neutral intermediate form. This bytecode can run on any system that supports Java, solving problems related to software distribution and version compatibility.

Portability in Java goes beyond just bytecode. Java strictly defines the **size of data types and behavior of operators**, ensuring that programs behave the same way on all platforms. This eliminates inconsistencies that occur due to hardware or operating system differences.

The platform that enables this portability is called the **Java Virtual Machine (JVM)**. The JVM acts as an abstract machine that runs Java bytecode. Different implementations of the JVM exist for different systems, but all follow the same specification. This makes Java highly portable and easy to adapt to new platforms.

2.4 High Performance

uses efficient execution techniques that allow programs to run at high speed without frequent environment checks. Garbage collection runs in the background with low priority, ensuring that memory is available when needed without slowing down the program.

For applications that require heavy computation, performance-critical parts can be written in **native machine code** and connected to Java programs. In practice, Java applications respond quickly, especially interactive, and network-based programs.

2.5 Interpreted, Threaded, and Dynamic

Java programs are **interpreted**, meaning Java bytecode can be executed directly on any machine with a Java run-time environment. This makes program linking simple and lightweight, allowing faster development, testing, and modification compared to traditional compile–link–test cycles.

Java supports **multithreading at the language level**, allowing multiple tasks to run simultaneously within a program. It provides built-in thread classes and synchronization mechanisms to ensure safe and efficient execution. Java libraries are designed to be thread-safe, allowing multiple threads to access them without conflicts.

Java is also **dynamic**, meaning classes are loaded and linked only when needed. New classes can be added during program execution, even from network sources. This dynamic behavior makes Java flexible, adaptable, and well-suited for modern, distributed applications.

FEATURES OF JAVA

The prime reason behind creation of Java was to bring portability and security feature into a computer language. Beside these two major features, there were many other features that played an important role in molding out the final form of this outstanding language. Those features are :

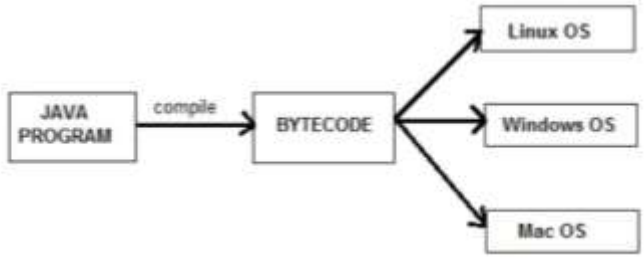
1) Simple : Java is easy to learn and its syntax is quite simple, clean, and easy to understand. The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.
Eg : Pointers and Operator Overloading are not there in java but were an important part of C++.

2) Object Oriented : In java, everything is an object which has some data and behavior. Java can be easily extended as it is based on Object Model. Following are some basic concepts of OOPs.

i. Class and Object
ii. Encapsulation
iii. Abstraction
iv. Inheritance
v. Polymorphism

3) Robust : Java tries to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic **Garbage Collector** and **Exception Handling**.

4) Platform Independent: Unlike other programming languages such as C, C++ etc. which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere (WORA) language.
On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



5) Secure: When it comes to security, Java is always the first choice. With java secure features it enables us to develop virus free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS; hence it is more secure. In java, we do not have pointers, so we cannot access out-of-bound arrays i.e. it shows **ArrayIndexOutOfBound Exception** if we try to do so. That is why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java. Also, java programs run in an environment that is independent of the OS (operating system) environment which makes java programs more secure.

6) Multi-Threading: Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU. Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along.

7) Architectural Neutral: Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to interpret on any machine.

8) Portable: Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types. As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.

9) High Performance: Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But Java enables high performance with the use of just-in-time compiler. Java architecture is defined in such a way that it reduces overhead during the runtime and at sometimes java uses Just in Time (JIT) compiler where the compiler compiles code on-demand basics where it only compiles those methods that are called making applications to execute faster.

10) Distributed: Java is also a distributed language. Programs can be designed to run on computer networks. Java has a special class library for communicating using TCP/IP protocols. Creating network connections is very much easy in Java as compared to C/C++. We can create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.

11) Dynamic flexibility: Java being completely object-oriented gives us the flexibility to add classes, new methods to existing classes, and even create new classes through sub-classes. Java even supports functions written in other languages such as C, C++ which are referred to as native methods.

12) Sandbox Execution: Java programs run in a separate space that allows user to execute their applications without affecting the underlying system with help of a bytecode verifier. Bytecode verifier also provides additional security as its role is to check the code for any violation of access.

13) Write Once Run Anywhere: As discussed above java application generates a '.class' file that corresponds to our applications(program) but contains code in binary format. It provides ease t architecture-neutral ease as bytecode is not dependent on any machine architecture. It is the primary reason java is used in the enterprising IT industry globally worldwide.

14) Power of compilation and interpretation: Most languages are designed with the purpose of either they are compiled language or they are interpreted language. But java integrates arising enormous power as Java compiler compiles the source code to bytecode and JVM executes this bytecode to machine OS-dependent executable code.

JAVA TERMINOLOGY

Before learning Java, one must be familiar with these common terms of Java.

1. Java Virtual Machine(JVM): This is generally referred to as JVM. There are three execution phases of a program. They are written, compile and run the program.

- Writing a program is done by a java programmer like you and me.
- The compilation is done by the **JAVAC** compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.
- In the Running phase of a program, **JVM** executes the bytecode generated by the compiler.

Now, we understood that the function of Java Virtual Machine is to execute the bytecode produced by the compiler. Every Operating System has a different JVM but the output they produce after the execution of bytecode is the same across all the operating systems. This is why Java is known as a **platform-independent language.**

2. Bytecode in the **Development Process:** As discussed, the Javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. It is saved as **.class** file by the compiler. To view the bytecode, a disassembler like javap can be used.

3. Java Development Kit(JDK): While we were using the term JDK when we learn about bytecode and JVM. So, as the name suggests, it is a complete Java development kit that includes everything including compiler, Java Runtime Environment (JRE), java debuggers, java docs, etc. For the program to execute in java, we need to install JDK on our computer to create, compile and run the java program.

4. Java Runtime Environment (JRE): JDK includes JRE. JRE installation on our computers allows the java program to run, however, we cannot compile it. JRE includes a browser, JVM, applet support, and plugins. For running the java program, a computer needs JRE.

5. Garbage Collector: In Java, programmers cannot delete the objects. To delete or recollect that memory JVM has a program called Garbage Collector. Garbage Collectors can recollect the objects that are not referenced. So, Java makes the life of a programmer easy by handling memory management. However, programmers should be careful about their code whether they are using objects that have been used for a long time. Because Garbage cannot recover the memory of objects being referenced.

6. ClassPath: The classpath is the file path where the java runtime and Java compiler look for **.class** files to load. By default, JDK provides many libraries. If you want to include external libraries they should be added to the classpath.

PARTS OF A JAVA PROGRAM CODE:

```
class Sample { //declaring the class
   public static void main(String args[]) //declaring the main( ) method
   {
      int a, b, sum=0, product=0; //variables required in the program
      a=10; //input value 1
      b=20; //input value 2
      sum=a+b; //adding two values stored in a and b
      product=a*b; //multiplying two values stored in a and b
      System.out.println("Sum="+sum); //output 1
      System.out.println("Product="+product); //output2
   }
}
```

- **class** : class keyword is used to declare classes in Java
- **public** : It is an access specifier. Public means this function is visible to all.
- **static** : static is again a keyword used to make a function static. To execute a static function, you do not have to create an Object of the class. The main() method here is called by JVM, without creating any object for class.
- **void** : It is the return type, meaning this function will not return anything.
- **main** : main() method is the most important method in a Java program. This is the method which is executed; hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error.
- **String[] args** : This statement is used to signify that the user may opt to enter parameters to the Java Program at command line. The word **args** refers to argument. We can use both String[] args or String args[]. Java compiler would accept both forms.
- **System.out.println()** : This statement is used to print anything (output) on the screen.

3. Compilation Of Java Programs

Java, being a platform-independent programming language, does not work on the one-step compilation. Instead, it involves a two-step execution, first through an OS-independent compiler; and second, in a virtual machine (JVM) which is custom-built for every operating system.
The two principal stages are explained below:

Principle 1: Compilation
First, the source '.java' file is passed through the compiler, which then encodes the source code into a machine-independent encoding, known as Bytecode. The content of each class contained in the source file is stored in a separate '.class' file.

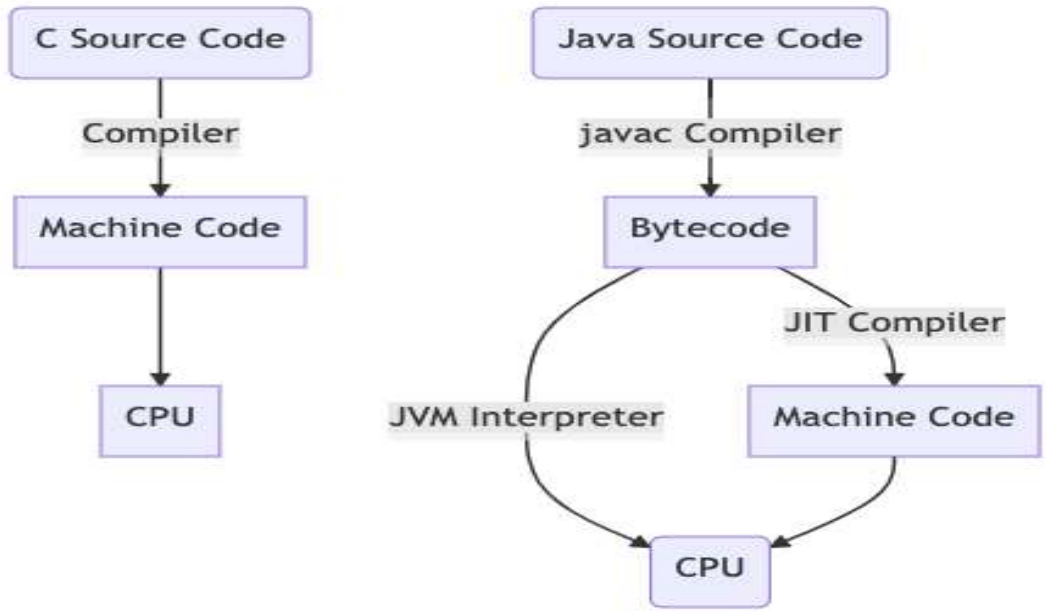
Principle 2: Execution
The class files generated by the compiler are independent of the machine or the OS, which allows them to be run on any system. To run, the main class file (the class that contains the method main) is passed to the JVM and then goes through three main stages before the final machine code is executed. These stages are:

1. ClassLoader
2. Bytecode Verifier
3. Just-In-Time Compiler

4. Java Program Compilation Vs Other Compilation

The Java compilation process is specific to the Java programming language, and is used to convert Java source code into Java bytecode. The ordinary compilation process, on the other hand, refers to the process of converting source code into machine code that can be executed directly by a computer's central processing unit (CPU). In the Java compilation process, the source code is first compiled into Java bytecode by a Java compiler. This bytecode can then be executed on any platform that has a Java Virtual Machine (JVM) installed. The JVM interprets the bytecode and executes it on the target platform. The ordinary compilation process, on the other hand, requires that the source code be compiled specifically for the target platform's architecture and operating system. The resulting machine code is linked with any necessary libraries and is executed directly by the CPU. In summary, the Java compilation process results in bytecode that can be executed on any platform that has a JVM, while the ordinary compilation process results in machine code that is specific to the target platform.

To execute C code, you use 1 compiler. To execute Java code, you normally use 2 compilers and 1 interpreter.



C execution model	Java execution model
1. You write code in C (or other languages) 2. You feed your C code to a compiler, such as Clang or gcc 3. The compiler figures out what each instruction means and generates the corresponding instructions in machine code. The machine code is saved in an executable file 4. You click on the executable (or invoke it in the command line), and the operating system loads it in memory and directs the CPU to start executing them.	1. You write code in java (or kotlin, scala, clojure, etc…) 2. You feed your source code to a compiler, such as javac 3. The compiler figures out what each instruction means and generates the corresponding instructions in java bytecodes. The bytecodes are then saved into a .class file. Many .class files can be packed together into a .jar or .war file. There are no CPUs that understand java bytecode directly. 4. To execute it, you start a java virtual machine (JVM), which is just another program and you feed it the bytecodes. The JVM can choose how to execute the code. a. Read each bytecode, and perform the corresponding action. That is the JVM interpreter. b. Read each bytecode and generate the corresponding instructions in machine code and instruct the CPU to execute them. That is the JIT compiler.
Note that you can rerun the executable as many times as you want without involving the compiler. You can also send the executable to someone else and if they have a compatible CPU, they can run it without having to install a compiler.	Note that you can distribute the .class, .jar or .war file and other people can run them, even if they have a different operating system or CPU architecture. However, everyone will need a JVM installed to execute the code.

So, java involves the javac compiler, which compiles source code into bytecode, and then a second compiler that translates bytecode to machine code. Optionally, the JVM may choose to interpret the bytecode instead so no machine code is generated.
The JVM chooses between the interpreter and the JIT compiler to improve performance. If a function is executed only once, it is faster to just use the interpreter. If a function is executed a lot, it is worth investing time to JIT compile it and then use the compiled function in each execution.

5. Java Virtual Machine

Java :
1. Programming language (Software)

2. Platform to create a class, compile and execute

Java compilation: Source code is converted to Byte code through Java compiler (javac)
Java interpretation: Byte code is converted to machine code through Java interpreter (JVM)

Ordinary compilation - Source code directly converted to object code through compiler
e.g. C, C++, Python
Ordinary interpretation - Source code directly interpreted to machine line by line
e.g Basic, VB
Java compilation takes in two parts - compilation (byte code) and interpretation (machine code)

6. IDE For Creating & Running Java Programs

Execution of a Java program goes through three main stages before the final machine code is executed.
Stage 1: Class Loader
The main class is loaded into the memory bypassing its '.class' file to the JVM, through invoking the latter. All the other classes referenced in the program are loaded through the class loader. A class loader, itself an object, creates a flat namespace of class bodies that are referenced by a string name. The method definition is provided below illustration as follows:
There are two types of class loaders

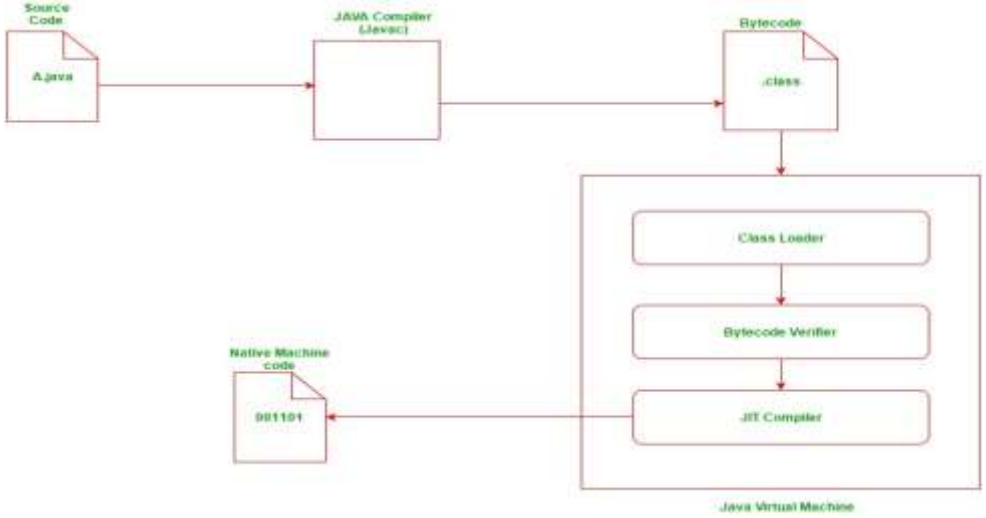
- primordial
- non-primordial

The primordial class loader is embedded into all the JVMs and is the default class loader. A non-primordial class loader is a user-defined class loader, which can be coded to customize the class-loading process. Non-primordial class loader, if defined, is preferred over the default one, to load classes.

Stage 2: Bytecode Verifier
After the bytecode of a class is loaded by the class loader, it must be inspected by the bytecode verifier, whose job is to check that the instructions do not perform damaging actions. The following are some of the checks carried out:

- Variables are initialized before they are used.
- Method calls match the types of object references.
- Rules for accessing private data and methods are not violated.
- Local variable accesses fall within the runtime stack.
- The run-time stack does not overflow.
- If any of the above checks fail, the verifier does not allow the class to be loaded.

Stage 3: Just-In-Time Compiler
This is the final stage encountered by the java program, and its job is to convert the loaded bytecode into machine code. When using a JIT compiler, the hardware can execute the native code, as opposed to having the JVM interpret the same sequence of bytecode repeatedly and incurring the penalty of a relatively lengthy translation process. This can lead to performance gains in the execution speed unless methods are executed less frequently.



The process can be well-illustrated by the following diagram given above as follows from which we landed up to the conclusion.

Conclusion: *Due to the two-step execution process described above, a java program is independent of the target operating system. However, because of the same, the execution time is way more than a similar program written in a compiled platform-dependent program.*

Difference between a Java Application and a Java Applet

Java Application is just like a Java program that runs on an underlying operating system with the support of a virtual machine. It is also known as an application program. The graphical user interface is not necessary to execute the java applications, it can be run with or without it.

Java Applet is a Java program that can be embedded into a web page. It runs inside the web browser and works on the client-side. An applet is embedded in an HTML page using the **APPLET** or **OBJECT** tag and hosted on a web server. Applets are used to make the website more dynamic and entertaining.

Java Application	Java Applet
Applications are just like a Java program that can be executed independently without using the web browser.	Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution.
The application program requires a main() method for its execution.	The applet does not require the main() method for its execution instead init() method is required.
The "javac" command is used to compile application programs, which are then executed using the "java" command.	Applet programs are compiled with the "javac" command and run using either the "appletviewer" command or the web browser.
Applications can access all kinds of resources and local files available on the system.	Applets can only access browser-specific services. They do not have access to the local system.
Applications can execute the programs from the local system.	Applets cannot execute programs from the local machine.
An application program is needed to perform some tasks directly for the user.	An applet program is needed to perform small tasks or part of them.
It cannot run on its own; it needs JRE to execute.	It cannot start on its own, but it can be executed using a Java-enabled web browser.

Advantages of Java:

1. Platform independent: Java code can run on any platform that has a Java Virtual Machine (JVM) installed, which means that applications can be written once and run on any device.
2. Object-Oriented: Java is an object-oriented programming language, which means that it follows the principles of encapsulation, inheritance, and polymorphism.
3. Security: Java has built-in security features that make it a secure platform for developing applications, such as automatic memory management and type checking.
4. Large community: Java has a large and active community of developers, which means that there is a lot of support available for learning and using the language.
5. Enterprise-level applications: Java is widely used for developing enterprise-level applications, such as web applications, e-commerce systems, and database systems.

Disadvantages of Java:

1. Performance: Java can be slower compared to other programming languages, such as C++, due to its use of a virtual machine and automatic memory management.
2. Memory management: Java's automatic memory management can lead to slower performance and increased memory usage, which can be a drawback for some applications.

Compilation & Execution Related

- **JVM – Java Virtual Machine**
  Executes Java bytecode and makes Java platform-independent.
- **JDK – Java Development Kit**
  Complete package used to **develop, compile, and run** Java programs.
- **JRE – Java Runtime Environment**
  Provides an environment to **run** Java programs but not to compile them.
- **JAVAC** – Java Compiler
  Compiles .java source files into .class bytecode files.
- **Bytecode** –
  Intermediate, platform-independent code generated by Java compiler.
- **JIT – Just-In-Time Compiler**
  Converts frequently used bytecode into machine code for better performance.

Java Program Structure

- **class** – Keyword used to define a class.
- **main()** – Entry point of a Java application.
- **public** – Access specifier that allows access from anywhere.
- **static** – Allows method to run without creating an object.
- **void** – Return type indicating no value is returned.
- **String[] args** – Command-line arguments.
- **System.out.println()** – Prints output to the console.

Memory & Security

- **Garbage Collector** –
  JVM component that automatically frees unused memory.
- **Exception Handling** –
  Mechanism to handle runtime errors safely.
- **Sandbox Execution** –
  Secure environment where Java programs run without harming the system.
- **Bytecode Verifier** –
  Checks bytecode for security violations before execution.

Multithreading & Performance

- **Thread** – Smallest unit of execution in a program.
- **Multithreading** – Execution of multiple threads simultaneously.
- **Thread-Safe** – Code that works correctly with multiple threads.

Portability & Architecture

- **Platform Independent** – Runs on any OS with JVM.
- **Architecture Neutral** – Bytecode not dependent on CPU architecture.
- **Portable** – Same program runs unchanged on different systems.
- **POSIX** – **Portable Operating System Interface**

Java Files & Tools

- **.java** – Java source code file.
- **.class** – Compiled bytecode file.
- **.jar** – **Java Archive**
- **.war** – **Web Application Archive**
- **Classpath** – Path where JVM looks for .class files.
- **javap** – Java disassembler tool.

Applet vs Application

- **Applet** – Java program embedded in a web page.
- **Application** – Standalone Java program.
- **HTML** – **HyperText Markup Language**

Execution Stages

- **Class Loader** – Loads class files into memory.
- **Primordial Class Loader** – Default JVM class loader.
- **Non-Primordial Class Loader** – User-defined class loader.
- **Interpreter** – Executes bytecode line by line.

Full forms related to Java:

- JVM – Java Virtual Machine
- JDK – Java Development Kit
- JRE – Java Run-time Environment
- API – Application Programming Interface
- IDE – Integrated Development Environment
- WORA – Write Once Run Anywhere

****