JAVA

What you learn ?

Java Introduction	
➢ What are Computer Programming Languages	➢ History of Computer Programming Languages
➢ Scope of Java	➢ Why do people use Java
➢ History of Java & Versions	➢ Java Features
➢ What is JRE,JDK,JVM	➢ Memory Management in JAVA
➢ Java Installation & Environment Variable Setup	
➢ Compiling & Running the Program	➢ Java Development Using Eclipse IDE

What are Computer Programming Languages

- Computer programming languages allow us to give instructions to a computer in a language the computer understands. Just as many human-based languages exist, there are an array of computer programming languages that programmers can use to communicate with a computer. The portion of the language that a computer can understand is called a “binary.” Translating programming language into binary is known as “compiling.” Each language, from C Language to Java, has its own distinct features, though many times there are commonalities between programming languages.
- These languages allow computers to quickly and efficiently process large and complex swaths of information. For example, if a person is given a list of randomized numbers ranging from one to ten thousand and is asked to place them in ascending order, chances are that it will take a sizable amount of time and include some errors.
- There are dozens of programming languages used in the industry today.

History of Computer Programming Languages

1883: Algorithm for the Analytical Engine				
1949: Assembly Language	1952: Autocode	1957: Fortran	1958: Algol	1959: COBOL
1959: LISP	1964: BASIC	1970: Pascal	1972: Smalltalk	1972: C
1972: SQL	1978: MATLAB	1983: Objective-C	1983: C++	1987: Perl
1990: Haskell	1991: Python	1991: Visual Basic	1993: R	**1995: Java**
1995: PHP	1995: Ruby	1995: JavaScript	2000: C#	2003: Scala
2003: Groovy	2009: Go	2010: Rust	2014: Swift	

Scope of Java

➢ Desktop GUI Applications	➢ Mobile Applications	➢ Enterprise Applications
➢ Scientific Applications	➢ Web-based Applications	➢ Embedded Systems
➢ Big Data Technologies	➢ Distributed Applications	➢ Cloud-based Applications
➢ Software Tools	➢ Gaming Applications	➢ Web servers and Application servers
➢ AI-ML		

Why do people use Java

- **Easy to Learn and Use**

  Java is easy to learn and use. It is developer-friendly and high level programming language.

- **Expressive Language**

  Java language is more expressive means that it is more understandable and readable.

- **Cross-platform Language**

  Java can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Java is a portable language.

- **Free and Open Source**

  Java language is freely available at offical web address.The source-code is also available. therefore it is open source.

Why do people use Java

- **Object-Oriented Language**

  Java supports object oriented language

- **Extensible**

  It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Java code.

- **Large Standard Library**

  Java has a large and broad library and provides rich set of module and functions for rapid application development.

- **GUI Programming Support**

  Graphical user interfaces can be developed using Java.

- **Integrated**

  It can be easily integrated with languages like C, C++ etc.

History of Java & Versions

Java is a high-level programming language that was first released in **1995 by Sun Microsystems**, which was later **acquired by Oracle Corporation**. **James Gosling, Mike Sheridan, and Patrick Naughton are credited with creating Java while working at Sun Microsystems.**

The idea behind Java was to create a programming language that would be portable across different computer platforms, meaning that a Java program written on one computer could run on another without any modifications. This was made possible by the use of the Java Virtual Machine (JVM), which allows Java code to be executed on any platform that has a JVM installed.

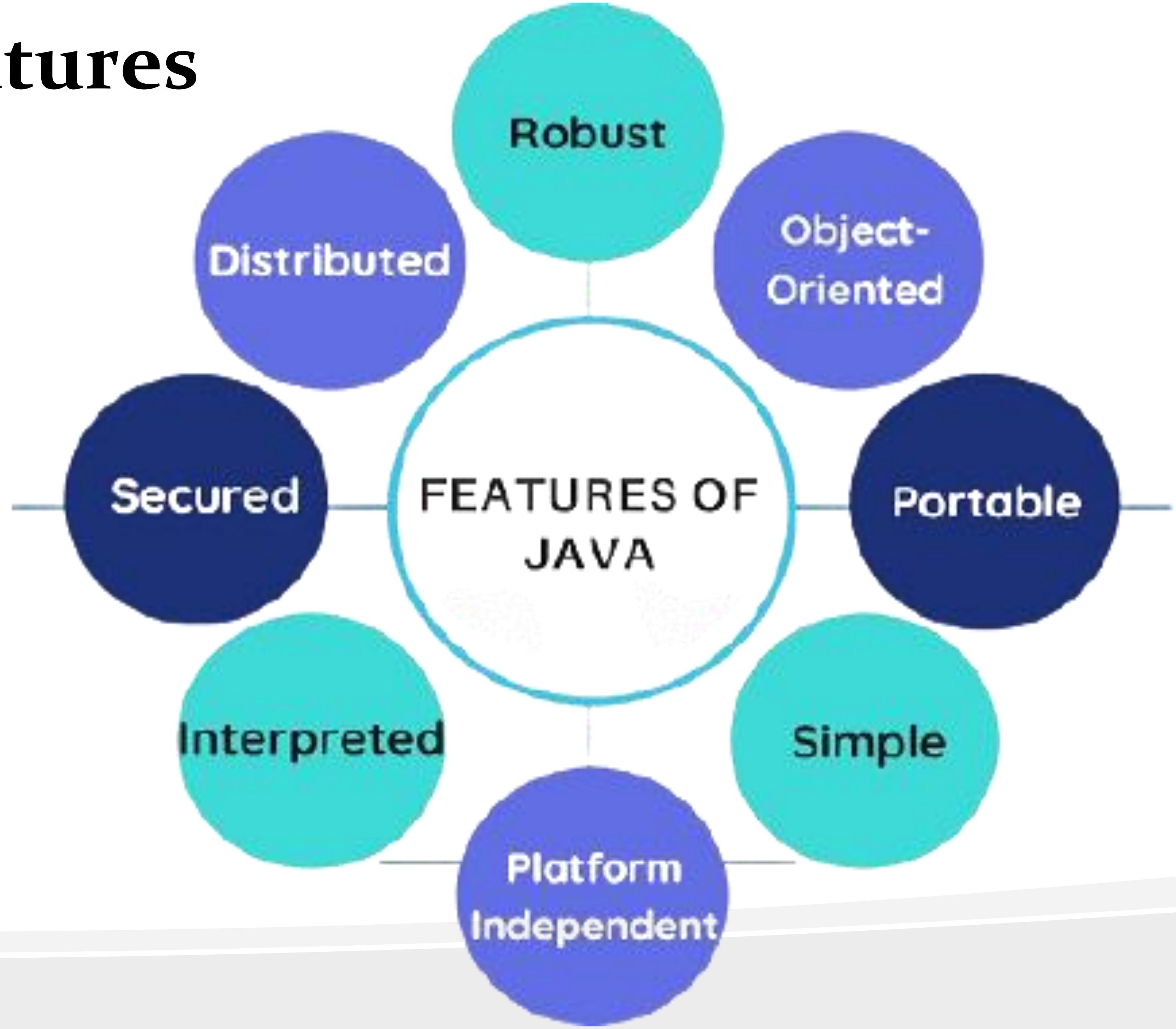
History of Java & Versions

In 2010, Oracle Corporation acquired Sun Microsystems and became the primary owner of Java. Since then, Java has continued to evolve, with new versions and updates being released regularly. Java 8, released in 2014, introduced several new features, including lambda expressions and the Stream API, which made it easier to write functional-style code.

Today, Java remains one of the most popular programming languages in use, with a wide range of applications across various industries. It is used for developing everything from desktop and mobile applications to enterprise-level software and web-based systems.

Oracle Java SE Support Roadmap*†				
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**	**Sustaining Support**
7 (LTS)	July 2011	July 2019	July 2022*****	Indefinite
8 (LTS)**	March 2014	March 2022	December 2030*****	Indefinite
9 (non-LTS)	September 2017	March 2018	Not Available	Indefinite
10 (non-LTS)	March 2018	September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	September 2026	Indefinite
12 (non-LTS)	March 2019	September 2019	Not Available	Indefinite
13 (non-LTS)	September 2019	March 2020	Not Available	Indefinite
14 (non-LTS)	March 2020	September 2020	Not Available	Indefinite
15 (non-LTS)	September 2020	March 2021	Not Available	Indefinite
16 (non-LTS)	March 2021	September 2021	Not Available	Indefinite
17 (LTS)	September 2021	September 2026****	September 2029****	Indefinite
18 (non-LTS)	March 2022	September 2022	Not Available	Indefinite
19 (non-LTS)***	September 2022	March 2023	Not Available	Indefinite
20 (non-LTS)***	March 2023	September 2023	Not Available	Indefinite
21 (LTS)***	September 2023	September 2028	September 2031	Indefinite

Java Features



Java Features

- **Object Oriented**

In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

- **Platform Independent**

Unlike many other programming languages including C and C++, when Java is compiled,
it is not compiled into platform specific machine, rather into platform-independent byte
code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

- **Simple**

Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Java Features

- **Secure**

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

- **Architecture-neutral**

Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

- **Portable**

Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

Java Features

- **Robust**

Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

- **Multithreaded**

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

- **Interpreted**

Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

Java Features

- **High Performance**

With the use of Just-In-Time compilers, Java enables high performance.

- **Distributed**

Java is designed for the distributed environment of the internet.

- **Dynamic**

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an
evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

1995_Java_whitepaper.pdf

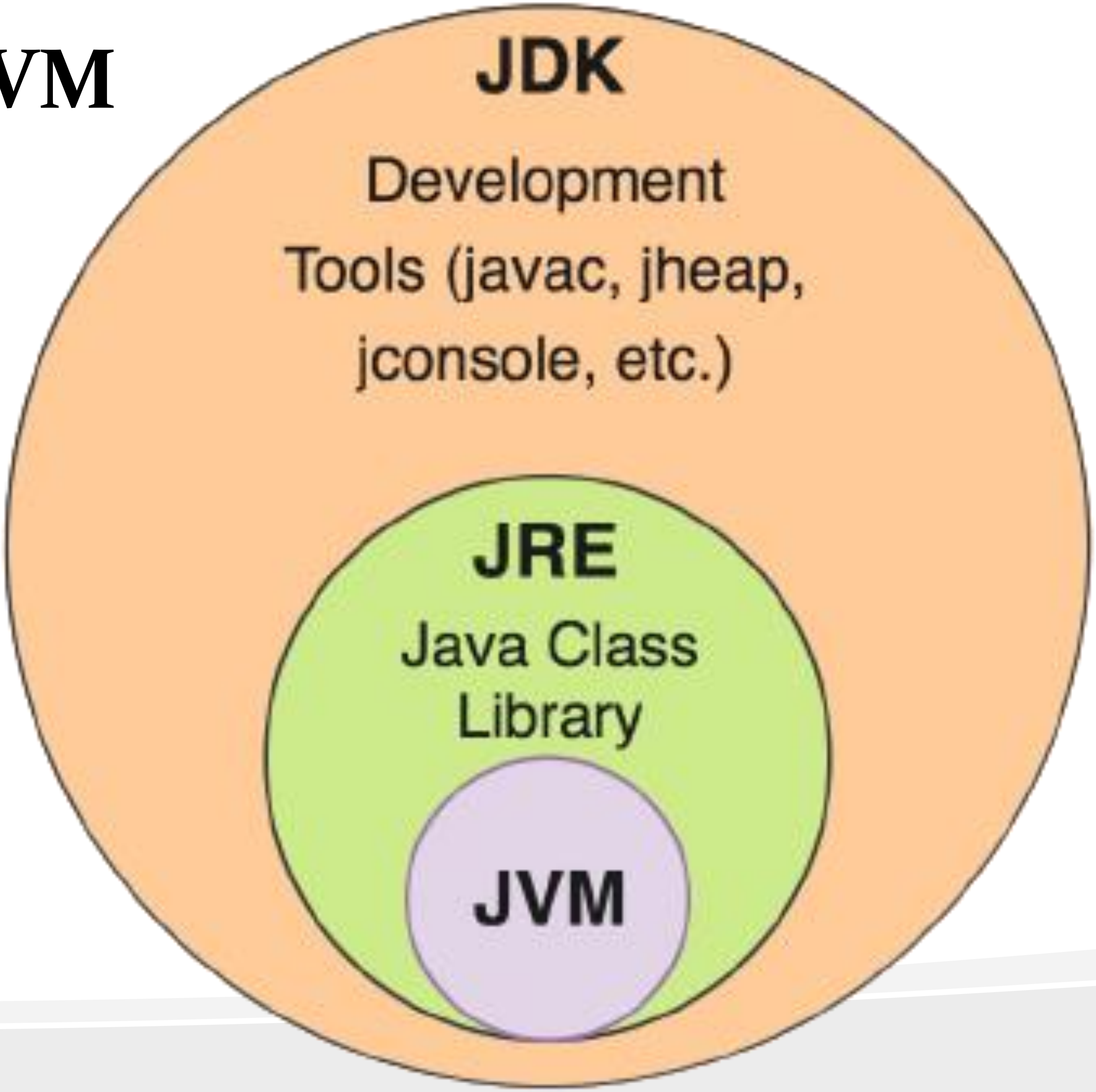
Java Features

Java's features do not end here, as every new version introduces new features.

- **The world’s most popular modern development platform**

 Java SE is the programming language of choice for enterprise applications. Java SE reduces costs, shortens development time, drives innovation, and improves application services. Protect your Java investment with Oracle Java SE Universal Subscription, which now includes GraalVM Enterprise and Java Management Service.

JRE-JDK-JVM


JDK
Development
Tools (javac, jheap,
jconsole, etc.)
JRE
Java Class
Library
JVM

JVM

Java Virtual Machine, or JVM, loads, verifies and executes Java bytecode. It is known as the interpreter or the core of Java programming language because it executes Java programming.

The role of JVM in Java

➢ JVM is specifically responsible for converting bytecode to machine-specific code and is

necessary in both JDK and JRE. It is also platform-dependent and performs many functions, including memory management and security. In addition, JVM can run programs written in other programming languages that have been translated to Java bytecode.

➢ Java Native Interface (JNI) is often referred to in connection with JVM. JNI is a programming framework that enables Java code running in JVM to communicate with (i.e., to call and be called by) applications associated with a piece of hardware and specific operating system platform. These applications are called native applications and can often be written in other languages. Native methods are used to move native code written in other languages into a Java application.

JVM

JVM consists of three main components or subsystems:

- Class Loader Subsystem is responsible for loading, linking and initializing a Java class file (i.e., “Java file”), otherwise known as dynamic class loading.

- Runtime Data Areas contain method areas, PC registers, stack areas and threads.

- Execution Engine contains an interpreter, compiler and garbage collection area.

JRE

Java Runtime Environment, or JRE, is a set of software tools responsible for execution of the Java program or application on your system.
JRE uses heap space for dynamic memory allocation for Java objects. JRE is also used in JDB (Java Debugging).

The role of JRE in Java

If a programmer would like to execute a Java program using the Java command, they should install JRE. If they are only installing (and not developing or compiling code), then only JRE is needed

JRE components

**Besides the Java Virtual Machine, JRE is composed of a variety of other supporting
software tools and features to get the most out of your Java applications.**

- **Deployment solutions:** Included as part of the JRE installation are deployment technologies like Java Web Start and Java Plugin that simplify the activation of applications and provide advanced support for future Java updates.

- **Development toolkits:** JRE also contains development tools designed to help developers improve their user interface. Some of these toolkits include the following:

Java 2D: An application programming interface (API) used for drawing two-dimensional graphics in Java language. Developers can create rich user interfaces, special effects, games and animations.

Swing: Another lightweight GUI that uses a rich set of widgets to offer flexible, user-friendly customizations.

JRE components

- **Integration libraries:** Java Runtime Environment provides several integration libraries and class libraries to assist developers in creating seamless data connections between their applications and services. Some of these libraries include the following:

Java IDL: Uses Common Object Request Brokerage Architecture (CORBA) to support distributed objects written in Java programming language.

Java Database Connectivity (JDBC) API: Provides tools for developers to write applications with access to remote relationship databases, flat files and spreadsheets.

Java Naming and Directory Interface (JNDI): A programming interface and directory service that lets clients create portable applications that can fetch information from databases using naming conventions.

JRE components

- **Language and utility libraries:** Included with JRE are Java.lang. and Java.util. packages that are fundamental for the design of Java applications, package versioning, management and monitoring. Some of these packages include the following

Collections framework: A unified architecture made up of a collection of interfaces designed to improve the storage and process of application data.
Concurrency utilities: A powerful framework package with high-performance threading utilities.
Preferences API: A lightweight, cross-platform persistent API that enables multiple users on the same machine to define their own group of application preferences.
Logging: Produces log reports — such as security failures, configuration errors, and performance issues — for further analysis.
Java Archive (JAR): A platform-independent file format that enables multiple files to be bundled in JAR file format, significantly improving download speed and reducing file

JDK

Java Development Kit, or JDK, is a software development kit often called a superset of JRE. It is the foundational component that enables Java application and Java applet development. It is platform-specific, so separate installers are needed for each operating system (e.g., Mac, Unix and Windows).

The role of JDK in Java

JDK contains all the tools required to compile, debug and run a program developed using the Java platform. (It's worth noting that Java programs can also be run using command line.)

JDK components

JDK includes all the Java tools, executables and binaries needed to run Java programs. This includes JRE, a compiler, a debugger, an archiver and other tools that are used in Java development.

JDK

The JDK (Java Development Kit) is a software development kit used for developing Java applications. There are different types of JDK available, each with its own set of features and capabilities. The most common types of JDK are:

- **Oracle JDK:** This is the official JDK provided by Oracle, which includes the latest features and security updates. It includes a full set of development tools, such as the Java compiler, debugger, and profiler.

- **OpenJDK:** This is an open-source version of the JDK that is maintained by the Java community. It is based on the same codebase as the Oracle JDK and provides a similar set of features and capabilities.

- **AdoptOpenJDK:** This is another open-source JDK distribution that provides pre-built binaries for multiple platforms. It is maintained by the AdoptOpenJDK community and provides long-term support for specific versions.

JDK

- **Amazon Corretto:** This is a no-cost, multi-platform, production-ready distribution of the OpenJDK that is maintained by Amazon. It includes a wide range of features, such as the Java Flight Recorder, Java Management Extensions, and the Java Advanced Management Console.
- **IBM JDK:** This is a JDK distribution provided by IBM, which includes a range of features such as JIT (Just-In-Time) compiler, Java Flight Recorder, and Java Mission Control.
- **Zulu JDK:** This is a JDK distribution provided by Azul Systems, which includes a range of features such as low-latency GC (Garbage Collection), Java Flight Recorder, and Java Mission Control.
- **RedHat JDK :** provides additional features and tools as part of its Red Hat Developer Toolset (DTS)

Java SE vs. Java EE

Java is synonymous with Java Standard Edition (Java SE) or Core Java. All three euphemisms refer to the basic Java specification that includes the act of defining types and objects. Java EE, on the other hand, provides APIs and is typically used to run larger applications. The content of this blog focuses on Java SE.

How JVM, JRE and JDK work together

The diagram provides an image of how JVM, JRE and JDK fit together in the Java landscape.
JVM, JRE and JDK all have very specific functions. Without all three, Java will not operate successfuly

JDK vs. JRE vs. JVM: Key differences

- JDK is the development platform, while JRE is for execution.
- JVM is the foundation, or the heart of Java programming language, and ensures the program’s Java source code will be platform-agnostic.
- JVM is included in both JDK and JRE – Java programs won’t run without it.

Complementary technologies

There are many complementary technologies that can be used to enhance JVM, JRE or JDK. The following technologies are among the most frequently used:

- **Just-in-time Compiler (JIT)** is part of JVM and optimizes the conversion of bytecode to machine code. It selects similar bytecodes to compile at the same time, reducing the overall duration of bytecode to machine code compilation.
- **Javac,** another complementary tool, is a compiler that reads Java definitions and translates them into bytecode that can run on JVM.
- **Javadoc** converts API documentation from Java source code to HTML. This is useful when creating standard documentation in HTML.

JVM and container technology

Java Virtual Machine (JVM) is used to create — you guessed it — virtual machines (VMs). VMs are servers that allow multiple applications to run on the same underlying physical hardware without impacting one another. This provides better use of resources and makes it much easier and cost-effective to scale than traditional infrastructure. VMs are also easily disposable because of their independence. When you no longer need the application, you simply take the VM down.

Containers take this abstraction to the next level and virtualize the OS kernel. The absence of the OS renders containers even more lightweight, fast and flexible than VMs.

Memory Management in JAVA

- **In Java, memory management is done automatically by the Java Virtual Machine (JVM). The JVM manages the memory using a process called garbage collection.**

When a Java program creates an object, the JVM allocates memory for the object. When the object is no longer needed, the JVM automatically deallocates the memory occupied by the object. This process of automatic memory management in Java is known as garbage collection

The JVM divides the memory into different regions, namely the **heap, the stack, and the**
Metaspace.
The **heap** is where objects are allocated, while the **stack** is used for local variables and method calls. The **Metaspace** is used to store metadata about classes and methods.

Memory Management in JAVA

Java provides a number of APIs for managing memory, including the **System.gc()** method, which requests that the JVM perform garbage collection, and the **Runtime.getRuntime().totalMemory() and Runtime.getRuntime().freeMemory()** methods, which can be used to get information about the total and free memory available to the JVM.

One important thing to keep in mind while programming in Java is to avoid memory leaks. A memory leak occurs when an object is no longer needed but is not properly deallocated. This can lead to the program consuming an excessive amount of memory and eventually crashing. To avoid memory leaks, it is important to properly manage the lifecycle of objects and to ensure that they are properly deallocated when no longer needed.

Heap area in Memory Management

The heap area is a region of memory that is used to store objects and arrays that are dynamically allocated during runtime. Some key points about the heap area in Java include:

- The heap is a shared resource that is used by all threads in a Java application.
- The size of the heap is determined at JVM startup and can be adjusted using command-line options such as **-Xms** (initial heap size) and **-Xmx** (maximum heap size).
- Objects and arrays are allocated on the heap using the new operator or by calling a constructor.
- The heap is divided into generations, with each generation having its own set of memory management policies.
- The garbage collector is responsible for reclaiming memory on the heap that is no longer being used by the application.

Stack area in Memory Management

The stack area is a region of memory that is used to store local variables and method call frames. Some key points about the stack area in Java include:

- Each thread in a Java application has its own stack, which is used to store the state of method invocations for that thread.
- The size of the stack is determined at thread creation time and cannot be adjusted during runtime.
- Local variables and method parameters are allocated on the stack and are automatically deallocated when the method returns.
- Method call frames are also allocated on the stack and are used to store information about the current method invocation, such as the return address and local variable values.
- The stack is a last-in, first-out (LIFO) data structure, which means that the most recently pushed items are the first to be popped off the stack.
- The stack area is typically much smaller than the heap area, and stack overflow errors can occur if a thread attempts to allocate more memory on the stack than is available.
- Because the stack area is managed by the JVM and has a fixed size, it is generally more efficient than the heap area for storing and accessing small amounts of data.

Metaspace area in Memory Management

In Java 8 and later versions, the Metaspace area is a region of memory in Java's memory management system that is used to store metadata about classes and class loaders. Here are some key points about the Metaspace area in Java:

- The Metaspace area replaces the PermGen area that was present in earlier versions of Java.
- Metaspace is a part of the native memory, rather than being part of the heap area.
- Metaspace is used to store metadata about classes and class loaders, including information such as class definitions, method definitions, and constant pool information.
- The size of the Metaspace area is not fixed and can grow or shrink dynamically based on the application's needs.
- Metaspace is garbage-collected just like the heap area, which means that unused metadata is automatically freed up and returned to the operating system.
- The Metaspace area is typically much more scalable than the PermGen area and can handle large and complex applications that dynamically load and unload classes at runtime.
- Metaspace is managed by the JVM and is not exposed to application code.

Garbage collector in JAVA

- Garbage collection in Java is the process of automatically reclaiming the memory occupied by objects that are no longer in use by the application. Garbage collection is an important feature of Java because it relieves the developer from the burden of manually deallocating memory.

- In Java, the garbage collector runs in the background and periodically checks which objects are still in use. Objects that are no longer being used by the application are identified and their memory is freed up for use by the JVM. This automatic memory management system makes Java programs more reliable and easier to develop than programs written in languages that require manual memory management.

- Java provides several different garbage collection algorithms that can be used to manage memory. The default algorithm is called the **Mark-and-Sweep algorithm**, which involves identifying which objects are in use by tracing references from the root objects (such as local variables and static fields) to other objects. Any objects that are not reachable from the root objects are considered garbage and are collected by the garbage collector.

Garbage collector in JAVA

Java also provides other garbage collection algorithms, such as the **Concurrent Mark-and-Sweep (CMS) algorithm and the Garbage-First (G1) algorithm**, which provide better performance and scalability in certain situations.

While garbage collection is automatic in Java, it is still important to write efficient and well-designed code to avoid creating unnecessary objects and to ensure that objects are properly released when they are no longer needed. This can help reduce the frequency and duration of garbage collection, which can improve the overall performance of the application.

Garbage collector in JAVA

Advantages of Garbage Collection :

- Simplified Memory Management: Garbage collection relieves developers from the burden of managing memory explicitly. They do not need to keep track of every object they create and manually free up the memory when it is no longer needed. This reduces the risk of memory leaks and makes development less error-prone.

- Increased Productivity: Since developers do not need to spend time on manual memory management, they can focus on writing application logic and improving code quality. This results in faster development cycles and more efficient use of development resources.

- Memory Efficiency: Garbage collection ensures that memory is used efficiently. It frees up memory that is no longer needed, reducing the chances of memory exhaustion and improving the overall performance of the application.

Garbage collector in JAVA

Advantages of Garbage Collection :

- Reduced Crashes: Garbage collection reduces the chances of crashes caused by memory-related issues. It automatically frees up memory and helps ensure that the application runs smoothly, even under heavy load.

- Scalability: Garbage collection makes it easier to write scalable applications. Developers do not need to worry about managing memory manually as the application grows in size and complexity, making it easier to write applications that can handle a large number of users and data.

Garbage collector in JAVA : System.gc() metho

- The **System.gc()** method is used to suggest that the Java Virtual Machine (JVM) perform a garbage collection. The JVM may or may not perform the garbage collection immediately after the method is called, as the actual timing of the garbage collection is left to the JVM's discretion. Calling **System.gc()** does not guarantee that garbage collection will occur, and it should not be relied upon for precise control over garbage collection.

Garbage collector in JAVA : finalize() method

- The **finalize()** method is a method defined in the **Object class, which is the superclass of all Java classes**. The finalize() method is called by the garbage collector before an object is about to be freed from memory. The purpose of finalize() is to provide an opportunity for the object to perform any necessary cleanup before it is destroyed.

- However, the use of **finalize()** is generally discouraged because it can cause performance issues and may not be reliable. Finalize() is not guaranteed to be called, and it is not predictable when it will be called. Moreover, the finalize() method can be expensive, and it can slow down the garbage collection process.

How to Install Java On Windows 11

- Download Zip file from OpenJDK -> **https://jdk.java.net/java-se-ri/17**

- Extract the zip file into a folder, e.g. C:\Program Files\Java\ and it will create a **jdk-17** folder (where the bin folder is a direct sub-folder).

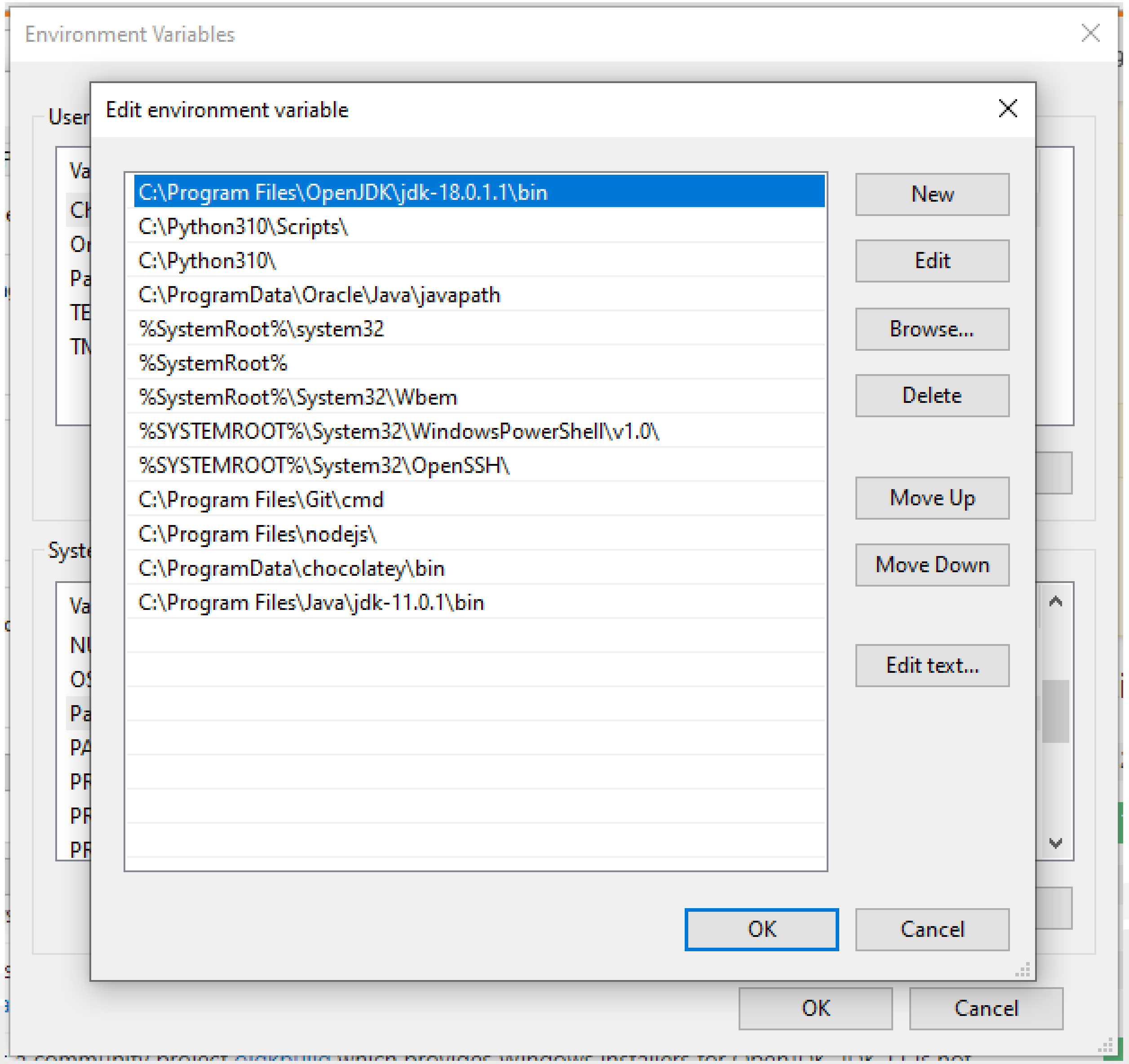
You may need Administrator privileges to extract the zip file to this location.

How to Install Java On Windows 11

Set a PATH:

- ➢ Open run utility using key combination -> Ctrl+R
- ➢ Enter
  -> sysdm.cpl
- ➢ Click Advanced Tab and then Environment Variables.
- ➢ Add the location of the bin folder of the JDK installation to the **PATH variable** in **System Variables**.

Click OK.



```
C:\Program Files\OpenJDK\jdk-18.0.1.1\bin
C:\Python310\Scripts\
C:\Python310\
C:\ProgramData\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\Program Files\nodejs\
C:\ProgramData\chocolatey\bin
C:\Program Files\Java\jdk-11.0.1\bin
```

How to Install Java On Windows 11

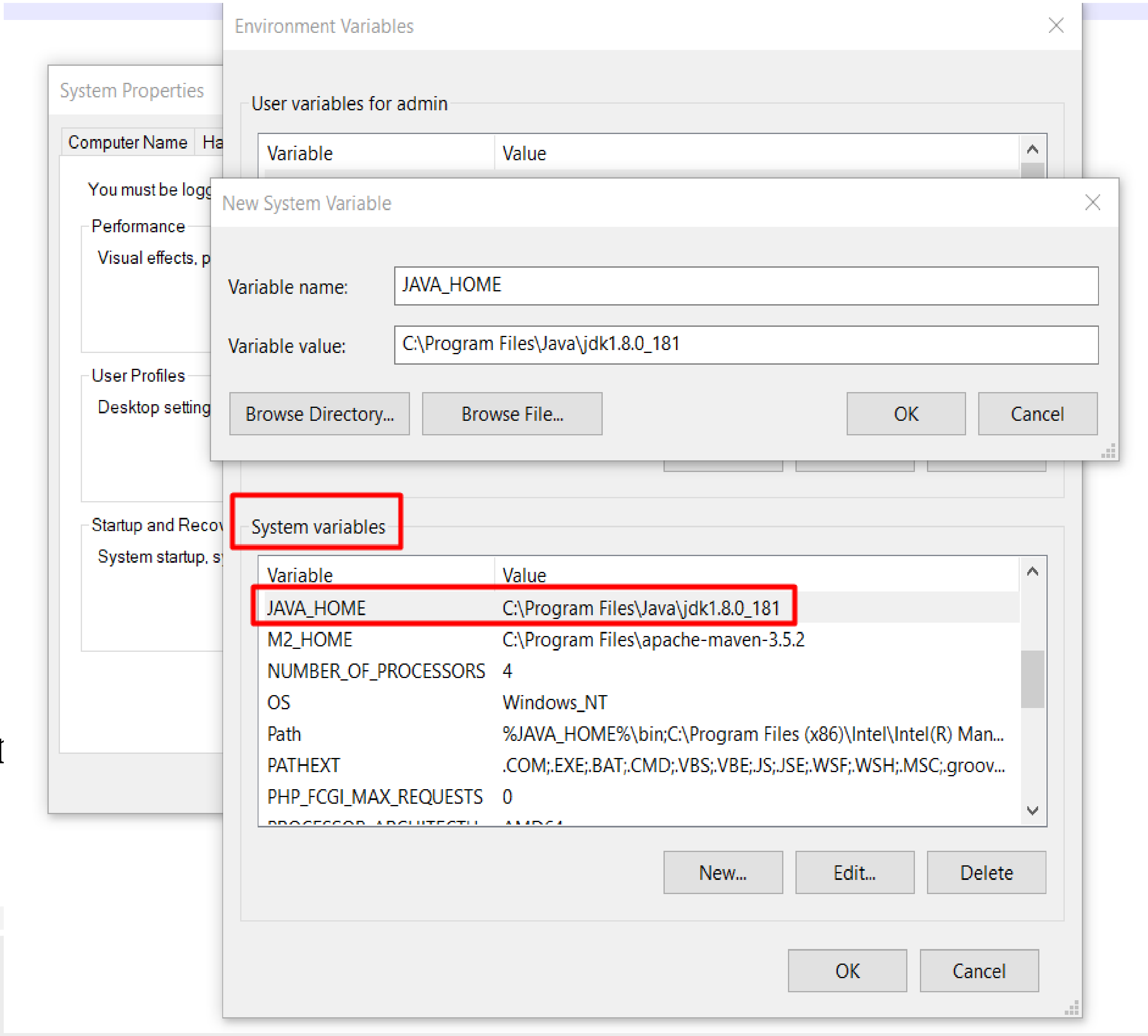
Set JAVA_HOME:

- Under System Variables, click New.
- Enter the variable name as **JAVA_HOME**
- Enter the variable value as the installation path of the JDK (without the bin sub-folder).

Click OK.

- Restart Windows & Check Java Versio
- Open Console & Enter below Command

```
-> java  -version
```



System Properties

Computer Name | Ha

You must be logg

Performance

Visual effects, p

User Profiles

Desktop setting

Startup and Reco

System startup, s

Environment Variables

User variables for admin

New System Variable

Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.8.0_181

System variables

Variable	Value
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_181
M2_HOME	C:\Program Files\apache-maven-3.5.2
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	%JAVA_HOME%\bin;C:\Program Files (x86)\Intel\Intel(R) Man...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.groov...
PHP_FCGI_MAX_REQUESTS	0

How to Install Java On Ubuntu Linux

All OpenJDK archive available on **https://jdk.java.net/archive/**

- Open a terminal window by pressing **Ctrl+Alt+T** or by searching for "**Terminal**" in the applications menu.

- Update the package index and upgrade any existing packages by running the following commands:

```
#> sudo apt-get update
#> sudo apt-get upgrade
```

- Add the OpenJDK package repository to your system by running the following command:

```
#>sudo add-apt-repository ppa:openjdk-r/ppa
```

How to Install Java On Ubuntu Linux

- Update the package index again by running the following command:

```
#> sudo apt-get update
```

- Install the OpenJDK 17 package by running the following command:

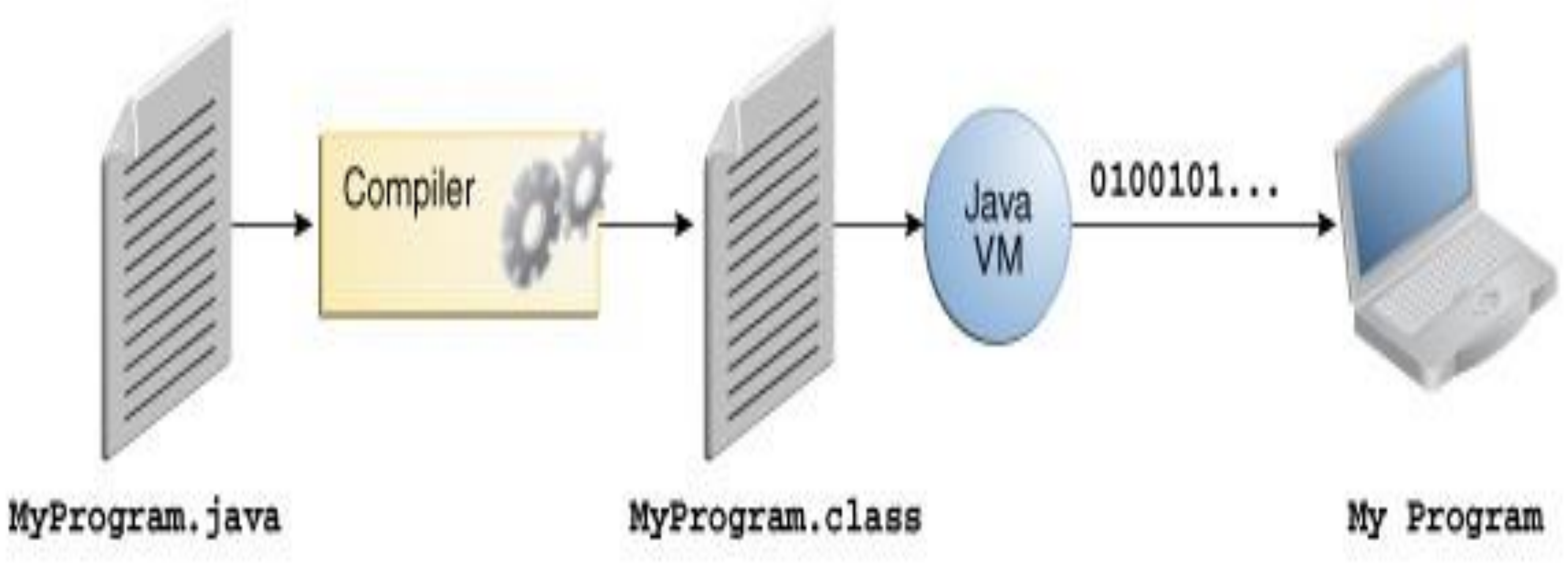
```
#> sudo apt-get install openjdk-17-jdk
```

This will install the JDK, which includes the JRE, the Java compiler, and other development tools.

- Verify that Java has been installed correctly by running the following command:

```
#> java -version
```

Compiling & Running the Program



Compiling & Running the Program

- **Create a source file**

A source file contains code, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and edit source files.

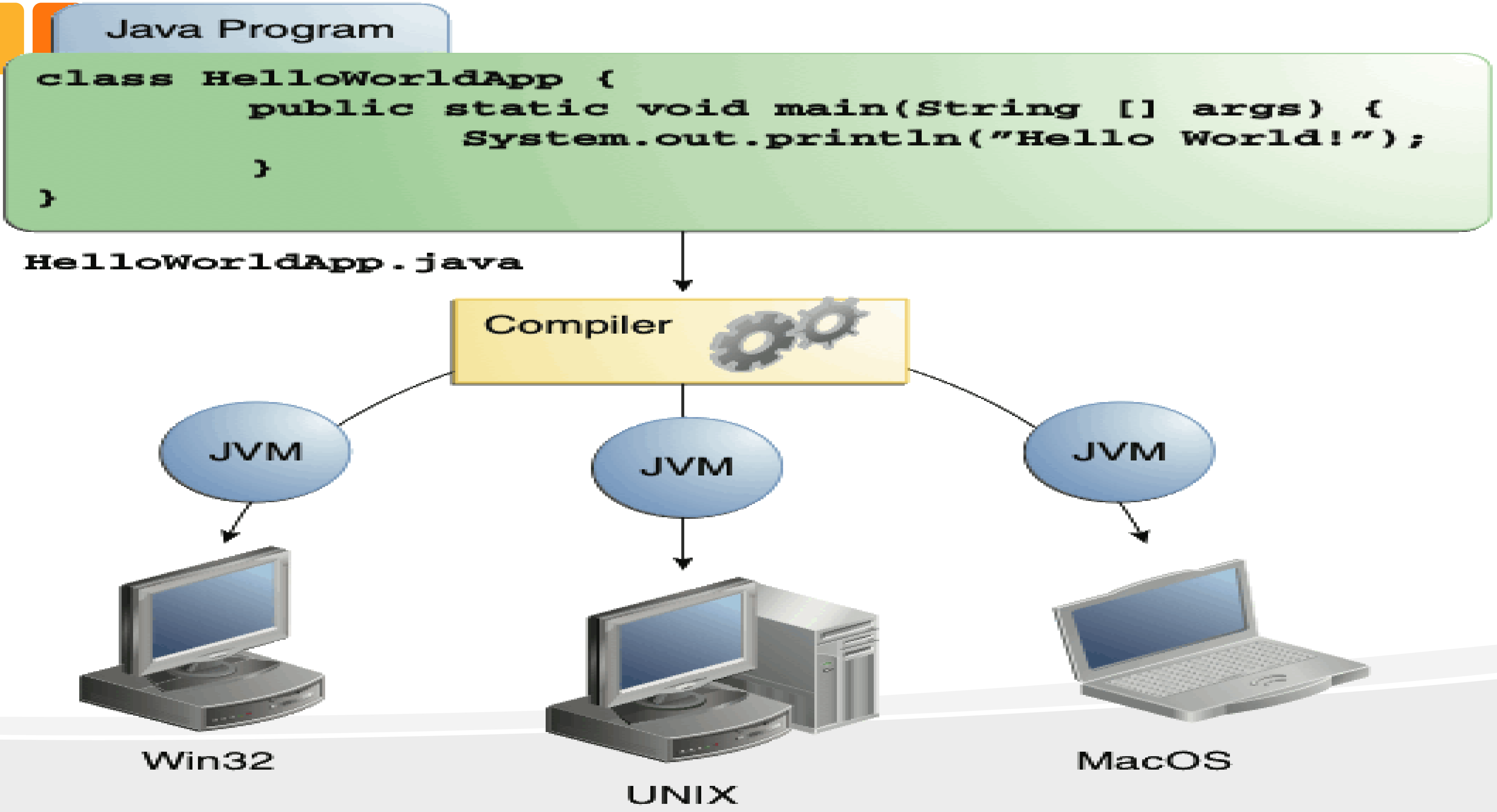
- **Compile the source file into a .class file**

The Java programming language compiler (javac) takes your source file and translates its text into instructions that the Java virtual machine can understand. The instructions contained within this file are known as **bytecodes**.

- **Run the program**

The Java application launcher tool (java) uses the Java virtual machine to run your application.

Compiling & Running the Program



```
class HelloWorldApp {
        public static void main(String [] args) {
                System.out.println("Hello World!");
        }
}
```

Java Development Using Eclipse IDE

Latest Eclipse IDE available with pre loaded JAVA ,so we don’t need to install java separately

To download visit following site

https://www.eclipse.org/downloads/packages/

from this site download
Eclipse IDE for Enterprise Java and Web Developers

as per your platform (windows,mac or linux)

Thanks

Anirudha Gaikwad