

Lab 01

Introduction to Java programming

Objectives:

1. What is JAVA?
2. Features of Java.
3. JAVA Basics.
4. Installing JDK and setting path.
5. Writing HelloWorld.java in Text Editor
6. JAVA variables & data types.
7. Input & Output
8. Java Variable Type Conversion & Type Casting
9. Lab tasks
10. Post Lab Questions

1: What is JAVA?

Java was developed by Sun Microsystems in 1995. It is a **programming language** as well as **platform**. Java is among the most popular programming languages out there, mainly because of how versatile and compatible it is. Java is general purpose programming language as it is used for software development, mobile applications, web servers, and client-side web applications. It is the native language of the Android operating system, which operates on Android phones and tablets.

Versions of Java:

There are many java versions that have been released.

Java SE Version	Version Number	Release Date
JDK	Alpha and Beta	(1995)
JDK 1.0 (Oak)	1.0	January 1996

JDK 1.1	1.1	February 1997
J2SE 1.2 (Playground)	1.2	December 1998
J2SE 1.3 (Kestrel)	1.3	May 2000
J2SE 1.4 (Merlin)	1.4	February 2002
J2SE 5.0 (Tiger)	1.5	September 2004
Java SE 6 (Mustang)	1.6	December 2006
Java SE 7 (Dolphin)	1.7	July 2011
Java SE 8	1.8	March 2014
Java SE 9	9	September 21st 2017
Java SE 10	10	March 20th 2018
Java SE 11	11	September 25th 2018
Java SE 12	12	March 19th 2019
Java SE 13	13	September 17th 2019
Java SE 14	14	March 17th 2020
Java SE 15	15	September 15th 2020
Java SE 16	16	March 16th 2021
Java SE 17	17	September 14th 2021
Java SE 18	18	March 22nd 2022
Java SE 19	19	September 20th 2022
Java SE 20	20	March 21st 2023
Java SE 21 (LTS)	21	September 19th 2023
Java SE 22	22	March 19th 2024
Java SE 23	23	September 17th 2024

Important changes in each version of Java:

Throughout its long history, Java SE has evolved with many changes and updates that enhance both developer productivity and the performance of Java applications. For your reference, below are the important changes in major releases of the Java SE platform:

Java SE 1.0 (1996)

- **Initial Release:** This was the first official release of the Java platform, introducing the basic elements of the language, including the core APIs, JVM, and the basic tools.

Java SE 1.1 (1997)

- **Inner Classes:** Support for inner classes was added.
- **JavaBeans:** Introduction of the JavaBeans component architecture.
- **JDBC:** The first version of Java Database Connectivity (JDBC) API.
- **Reflection:** Introduced for inspecting classes and interfaces at runtime.
- **RMI:** Remote Method Invocation (RMI) API was introduced.

Java SE 1.2 (1998) (also known as Java 2)

- **Collections Framework:** A new set of data structures such as List, Set, Map, etc.
- **Swing:** The introduction of the Swing graphical user interface (GUI) components.
- **JIT Compiler:** Just-In-Time (JIT) compiler integrated into the JVM for better performance.
- **Java Plug-in:** Enabled applets to run in web browsers.

Java SE 1.3 (2000)

- **HotSpot JVM:** The HotSpot JVM became the default virtual machine, improving performance.
- **Java Sound API:** Introduction of audio capabilities.
- **RMI over IIOP:** Allowed RMI to communicate with CORBA-compliant systems.

Java SE 1.4 (2002)

- **Assertions:** Added support for assertions, a debugging tool.
- **NIO (New I/O):** Enhanced input/output capabilities with non-blocking I/O.
- **Regular Expressions:** Built-in support for regular expressions.
- **Logging API:** Introduced for logging information.
- **XML Parsing:** The inclusion of APIs for processing XML (SAX, DOM).

Java SE 5.0 (2004)

- **Generics:** Added support for generic types, improving type safety.
- **Annotations:** Introduced metadata annotations.
- **Enhanced for-loop:** A new syntax for iterating over collections (for-each loop).
- **Autoboxing/Unboxing:** Automatic conversion between primitive types and their wrapper classes.
- **Enumerations:** Introduced the enum type.
- **Concurrency Utilities:** New API for managing threads and synchronization.

Java SE 6 (2006)

- **Scripting API:** Support for scripting languages like JavaScript.
- **Pluggable Annotations:** Compiler API for processing annotations.
- **Web Services:** Improved support for web services and the inclusion of JAXB and JAX-WS APIs.
- **Performance Improvements:** Various JVM and JIT enhancements.

Java SE 7 (2011)

- **Project Coin:** Small language changes such as the diamond operator, try-with-resources, multi-catch exceptions.
- **NIO.2:** Enhanced file I/O API, with support for file metadata and symbolic links.
- **Fork/Join Framework:** A framework for parallel programming.
- **Strings in Switch:** Strings became a valid argument for switch-case statements.
- **Automatic Resource Management:** Simplified the management of resources like files and sockets.

Java SE 8 (2014)

- **Lambda Expressions:** Introduced to support functional programming.
- **Stream API:** A powerful API for processing sequences of elements.
- **Optional Class:** A container object to avoid NullPointerException.
- **Default Methods:** Interfaces could now have default method implementations.
- **Date and Time API:** A new, comprehensive API for date and time management.
- **Nashorn JavaScript Engine:** Replaced the Rhino engine for JavaScript integration.

Java SE 9 (2017)

- **Modular System (Project Jigsaw):** Introduced the module system, allowing for more modular development and deployment.
- **JShell:** An interactive Java REPL tool.
- **Collection Factory Methods:** Simplified creation of immutable lists, sets, and maps.
- **Enhanced Process API:** Improved handling of system processes.
- **Private Interface Methods:** Allowed private methods within interfaces.

Java SE 10 (2018)

- **Local Variable Type Inference:** Introduction of the var keyword for local variables.
- **G1 Garbage Collector Enhancements:** Improved garbage collection performance.
- **Time-Based Release Model:** Shift to a predictable, time-based release cycle every six months.

Java SE 11 (2018)

- **LTS (Long-Term Support):** Java 11 was designated as an LTS release.
- **HTTP Client:** New HTTP client API to replace the legacy HttpURLConnection.
- **Local-Variable Syntax for Lambda Parameters:** Extended the use of var in lambda expressions.
- **Removed Modules:** Several modules, such as Java EE and CORBA, were removed from the standard distribution.
- **Nest-Based Access Control:** Simplified access control between nested classes.

Java SE 12 (2019)

- **Switch Expressions (Preview):** Enhanced switch statements to be used as expressions.
- **JVM Constants API:** Simplified the model for describing key class-file and runtime artifacts.

Java SE 13 (2019)

- **Text Blocks (Preview):** Multi-line string literals, simplifying the creation of strings with multiple lines.
- **Dynamic CDS Archives:** Reduced startup time and footprint through Class Data Sharing.

Java SE 14 (2020)

- **Switch Expressions (Standard):** Finalized the switch expressions introduced as a preview in Java 12.
- **Records (Preview):** Introduced a compact syntax for declaring data-carrying classes.
- **Pattern Matching for instanceof (Preview):** Simplified the pattern matching syntax in instanceof checks.

Java SE 15 (2020)

- **Text Blocks (Standard):** Finalized text blocks for multi-line strings.
- **Sealed Classes (Preview):** Restricted which classes or interfaces can extend or implement them.
- **Hidden Classes:** Support for creating classes not discoverable by classpath scanners.

Java SE 16 (2021)

- **Records (Standard):** Finalized records as a feature for simpler data classes.
- **Pattern Matching for instanceof (Standard):** Finalized pattern matching for the instanceof operator.
- **JEP 338:** Provided Vector API (Incubator) for expressing vector computations that compile to optimized machine code.

Java SE 17 (2021)

- **LTS (Long-Term Support):** Java 17 became the next LTS release.
- **Sealed Classes (Standard):** Finalized sealed classes for controlling inheritance.
- **Pattern Matching for Switch (Preview):** Extends pattern matching to switch statements.
- **Foreign Function & Memory API (Incubator):** Interacting with non-Java code and memory outside the JVM heap.

Java SE 18 (2022)

- **Simple Web Server:** A command-line tool for starting a minimal HTTP server.
- **UTF-8 by Default:** UTF-8 became the default charset for Java APIs.
- **Pattern Matching for Switch (Second Preview):** Continued development of pattern matching for switch.

Java SE 19 (2022)

- **Virtual Threads (Preview):** Lightweight threads for simplifying the writing of concurrent applications.
- **Structured Concurrency (Incubator):** Introduced a new model for structured concurrency in multithreaded programs.
- **Record Patterns (Preview):** Added patterns to Record classes for deconstruction.
- **Pattern Matching for Switch (Third Preview):** Further refinements for pattern matching with switch.

Java SE 20 (2023)

- **Record Patterns (Second Preview):** Continued refinement of record patterns for easier data access.
- **Pattern Matching for Switch (Fourth Preview):** More improvements and features for pattern matching.
- **Foreign Function & Memory API (Second Preview):** Further enhancements to interact with native code and memory.

Java SE 21 (2023) - Currently the Long Term Support (LTS) release

- **Virtual Threads:** Lightweight, JVM-managed threads that enhance scalability and efficiency in high-concurrency applications.
- **Sequenced Collections:** New interfaces for maintaining and manipulating ordered data collections.
- **Pattern Matching for Switch:** Simplifies conditional logic by extending pattern matching to switch statements.

Java SE 22 (2024)

- **Finalization of the Foreign Function & Memory API:** This feature allows Java programs to interact with native code and memory more safely and efficiently, replacing the older JNI approach.
- **String Templates (Second Preview):** Enhances string handling by allowing easier and safer embedding of expressions within string literals.
- **Structured Concurrency (Second Preview):** Improves the management of concurrent tasks, making code more reliable and easier to understand.

Java SE 23 (2024)

- **Markdown in JavaDoc Comments:** JavaDoc now supports Markdown, making documentation easier to write and read.
- **Generational Z Garbage Collector (ZGC):** JDK 23 makes generational mode the default for ZGC.
- **Features still In Preview:** Structured Concurrency, Flexible Constructor Bodies, Module Import Declarations, Primitive Types in Patterns, instanceof, and switch.

Platform: Any hardware or software environment in which a program runs is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Java Platforms

According to Oracle, there are four platforms of the Java programming language

Java Platform, Standard Edition (Java SE)

Java Platform, Enterprise Edition (Java EE)

Java Platform, Micro Edition (Java ME)

JavaFX/OpenJFX

2: Features of JAVA

The following are Java's main features:

- **Object Oriented** – In Java, everything is an object, creating objects that contain data and methods.
- **Architecture-neutral** – Traditionally, we would have to recompile a program for every system that it was going to run on because all systems have a different idea of what their machine code should look like. Java compiler generates an architecture-neutral object file format called as bytecode, which makes the compiled code executable on many machines but with the presence of Java runtime system.
- **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 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.
- **Portable** – Bytecode can be run by any system in which Java is installed. This is because when java is installed, Java virtual machine is also installed that is specific to that system.

It is this machine's responsibility to convert the bytecode into the final instructions of that particular machine.

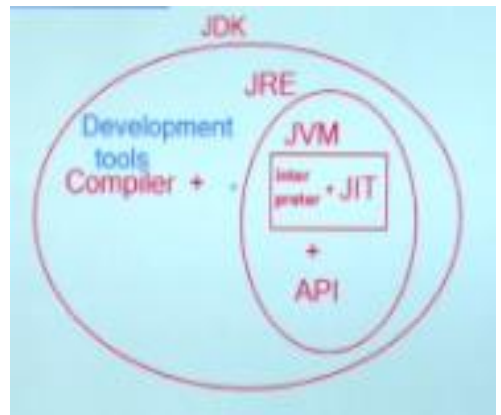
By making it the system's responsibility to do this final conversion, Java has created a write once, run anywhere language where anyone can hand you a Java program and you can run it on your machine

“Write once, run everywhere”

3: JAVA Basics

JDK

- It stands for Java Development Kit, is a software development environment used for developing Java applications and applets.
- It compiles and executes new and already built applications.
- It is a collection of development tools as well as Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.



JRE

- It stands for Java Runtime Environment. The Java Runtime Environment provides the minimum requirements for executing a Java application.
- It consists of the Java Virtual Machine (JVM), interpreter, JIT, core classes, and supporting files.

JVM

- It stands for Virtual Machine (JVM)
- It is responsible for executing bytecode where interpreter provides machine code for the current machine and has JIT as well.

JIT

- It stands for Just-in-time Compiler, is the part of the Java Virtual Machine (JVM) that is used to speed up the execution time.
- JIT interprets parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for full interpretation.

4: Installing JDK and setting path

The Java Development Kit (JDK) is a set of software tools for developing Java applications. The toolkit includes a compiler for converting Java code into bytecode, a collection of class libraries, documentation for integrating Java APIs, and the Java Runtime Environment (JRE) for executing Java code. The JDK provides developers with the resources they need to create Java applications and applets in one neat package.



Download Java Development Kit (JDK)

To download the Java Development Kit installation file:

1. Open a web browser and go to the link:
<https://www.oracle.com/java/technologies/downloads>
2. Select the latest JDK version. In this example, the latest available version is **JDK 21 LST**.
3. Navigate to the **Windows** tab.
4. Click the x64 Installer **download** link.

JDK 21 ① JDK 17 GraalVM for JDK 21 GraalVM for JDK 17

JDK Development Kit 21.0.2 downloads

JDK 21 binaries are free to use in production and free to redistribute, at no cost, under the [Oracle No-Fee Terms and Conditions \(NFTC\)](#).
JDK 21 will receive updates under the NFTC, until September 2026, a year after the release of the next LTS. Subsequent JDK 21 updates will be licensed under the [Java SE OTN License \(OTN\)](#) and production use beyond the limited free grants of the OTN license will require a fee.

Linux macOS **Windows ②**

Product/File description	File size	Download
x64 Compressed Archive	185.52 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.zip (sha256)
x64 Installer	163.91 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe (sha256) ③
x64 MSI Installer	162.07MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.msi (sha256)

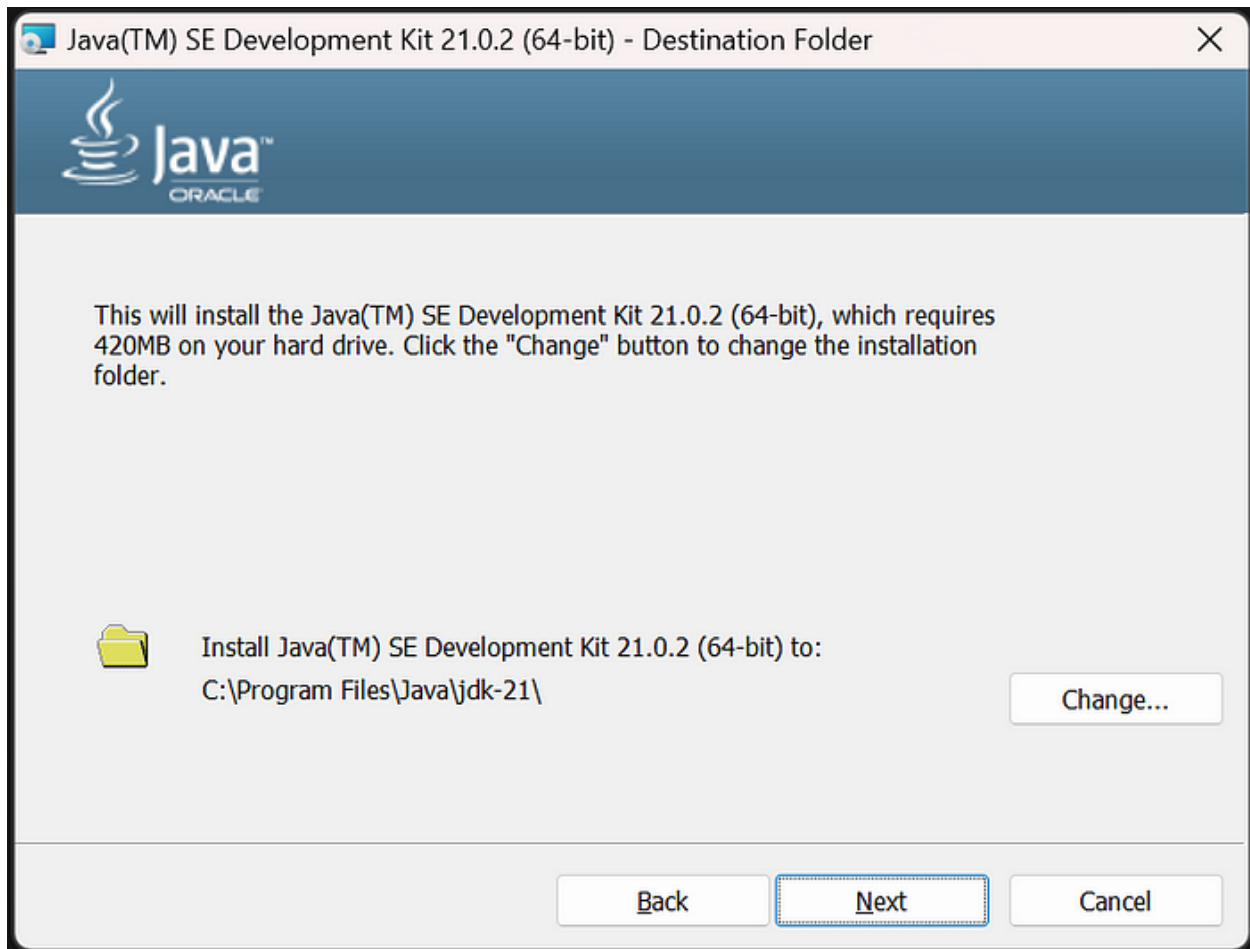
Install Java Development Kit

To install Java Development Kit on your Windows system:

1. Double-click the downloaded Java file to start the installation.
2. Once the installation wizard welcome screen appears, select **Next** to proceed.



3. Choose the destination folder for the Java installation files, or stick to the default path and click **Next**.



4. The installation process is complete when the Successfully Installed message appears. Click **Close** to exit the wizard.



The JDK 21 is successfully installed on your Windows system. To enable program compiling from any directory, you must set up Java environment variables.

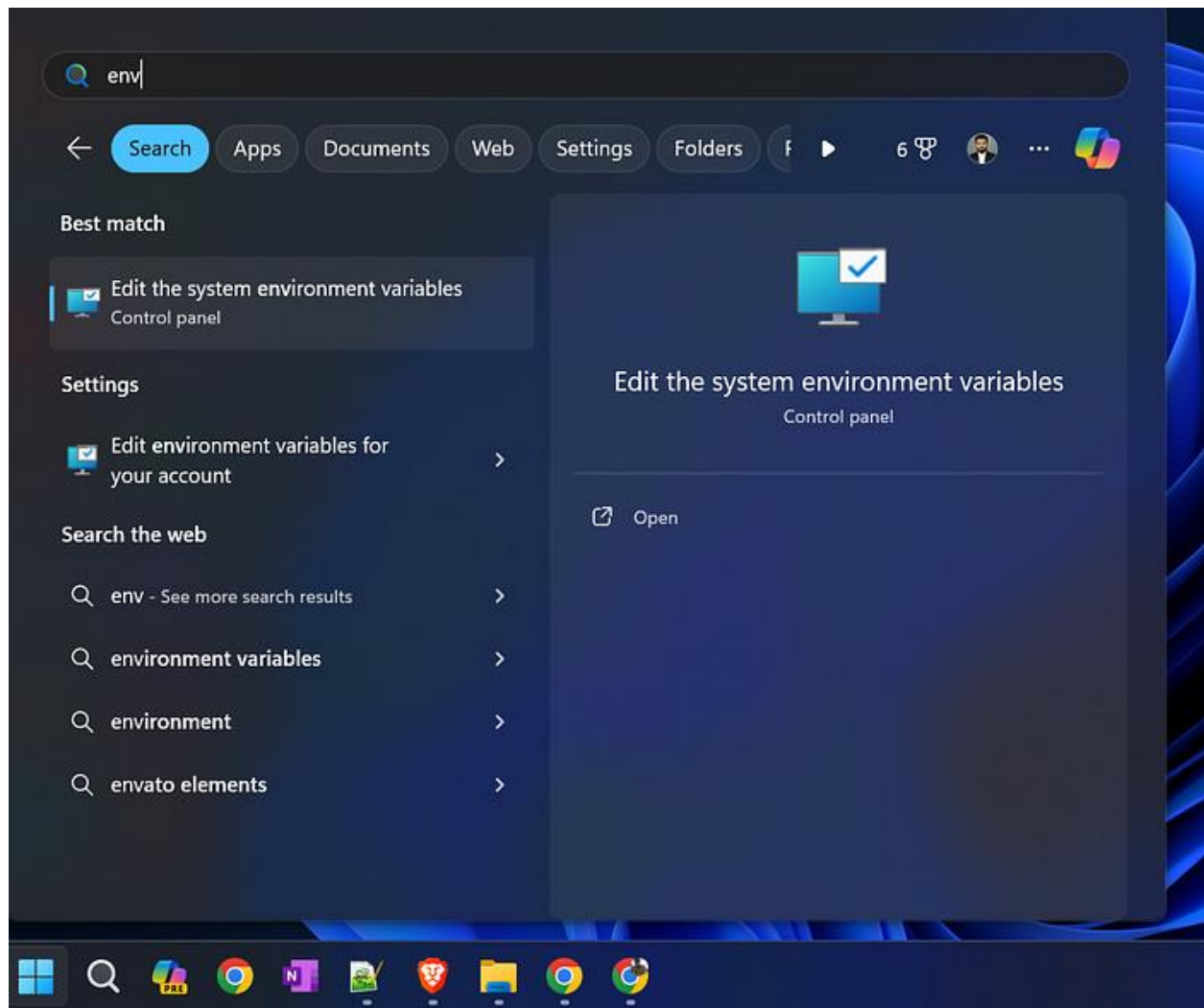
Set Environment Variables

To configure Java environment variables in Windows, follow the following steps

1. Add JAVA_HOME System Variable

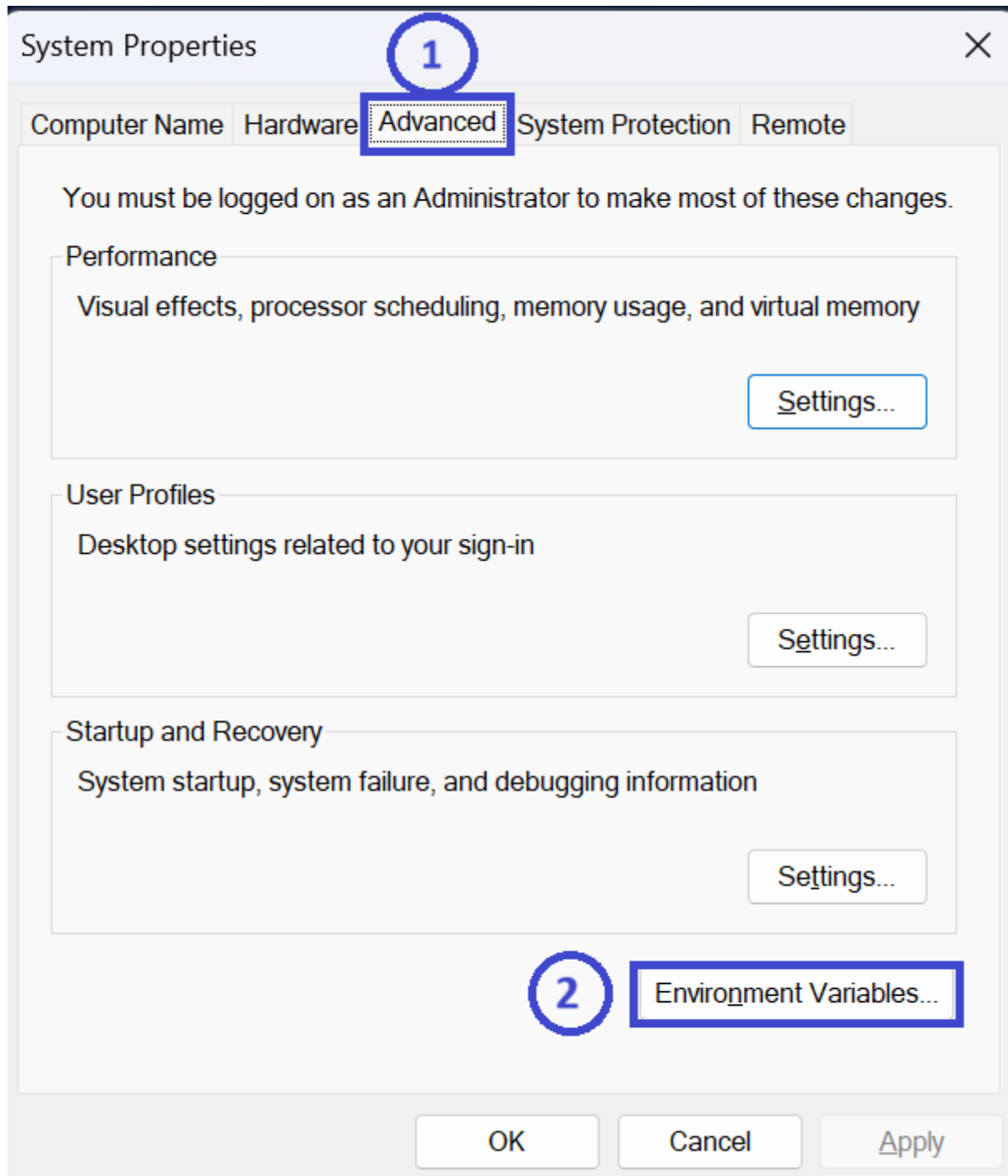
Some applications require the **JAVA_HOME** variable to point to the JDK installation directory. Follow the steps below to create the variable:

1. Open the **Start** menu and search for environment variables.
2. Select **Edit the system environment variables**.

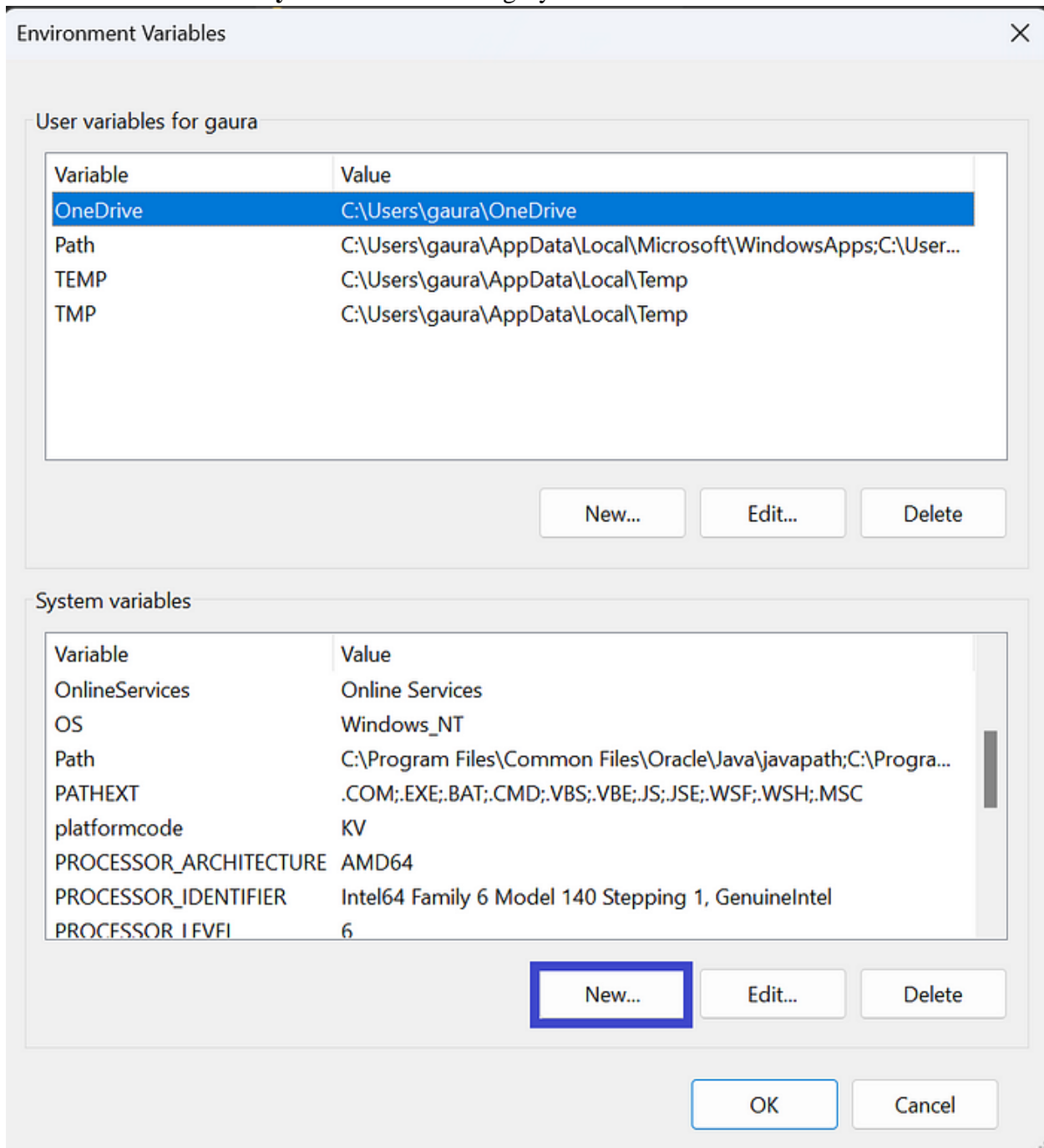


3. Select **Advanced** in the System Properties window.

4. Click **Environment Variables**.



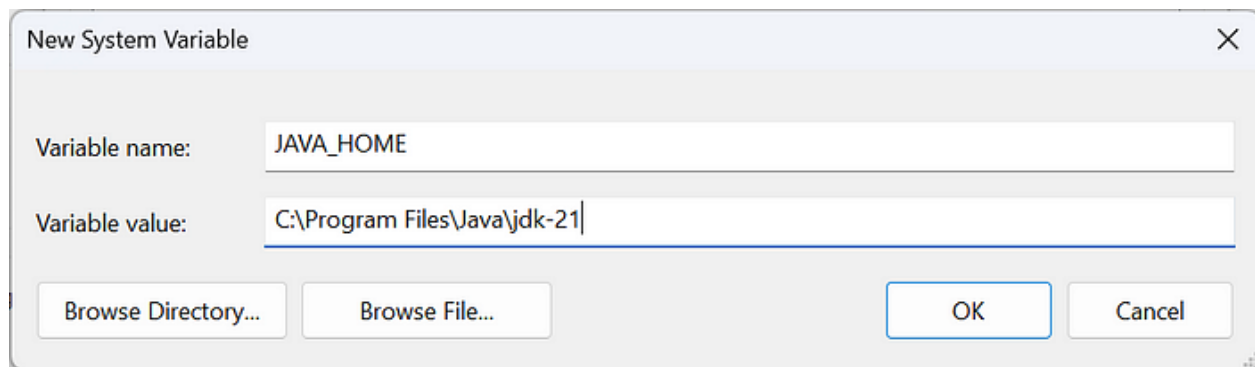
5. Click **New** under the **System variables** category to create a new variable.



6. Name the variable **JAVA_HOME**.

7. Enter the path to your Java JDK directory in the variable value field.

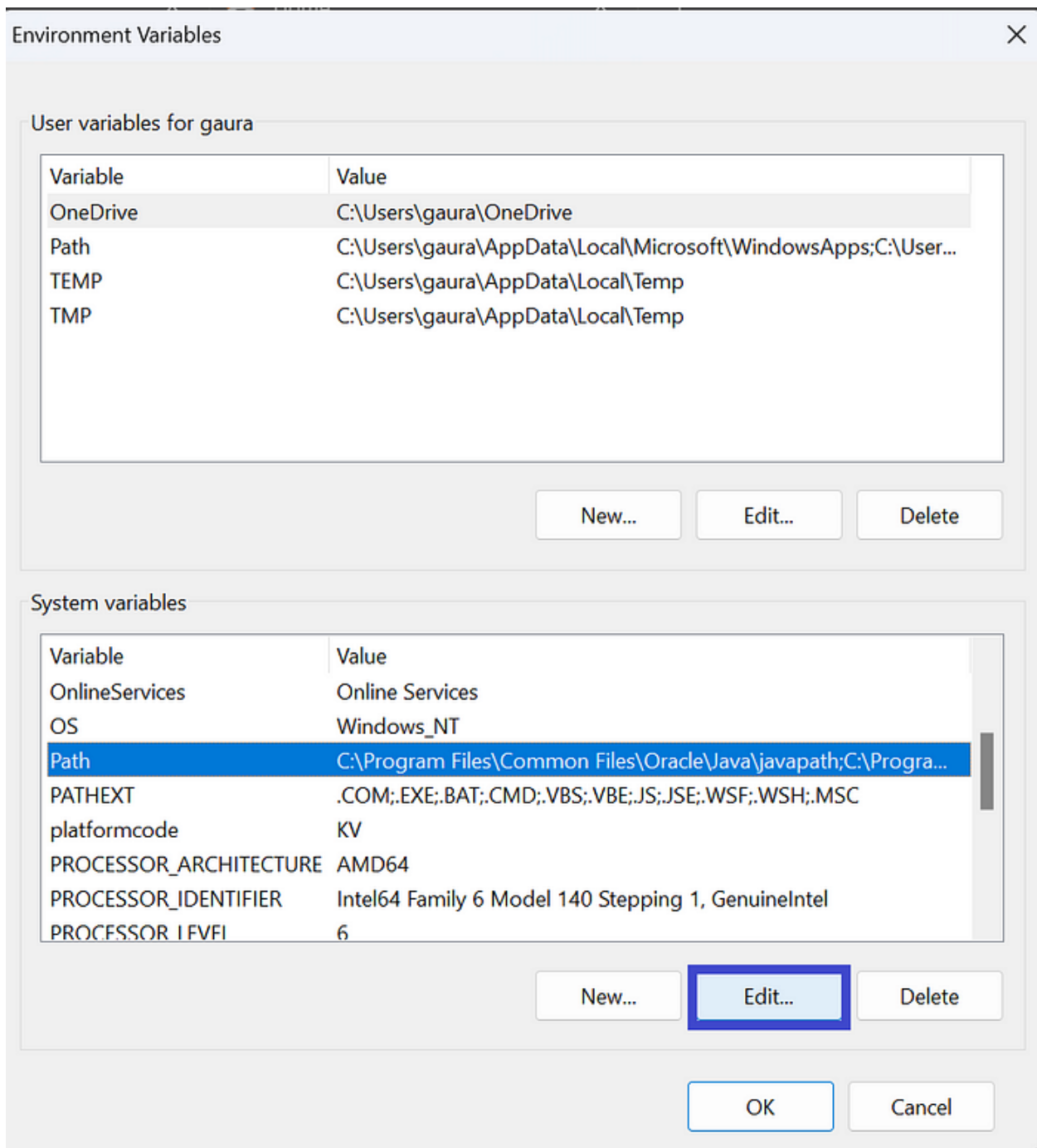
8. Click OK.



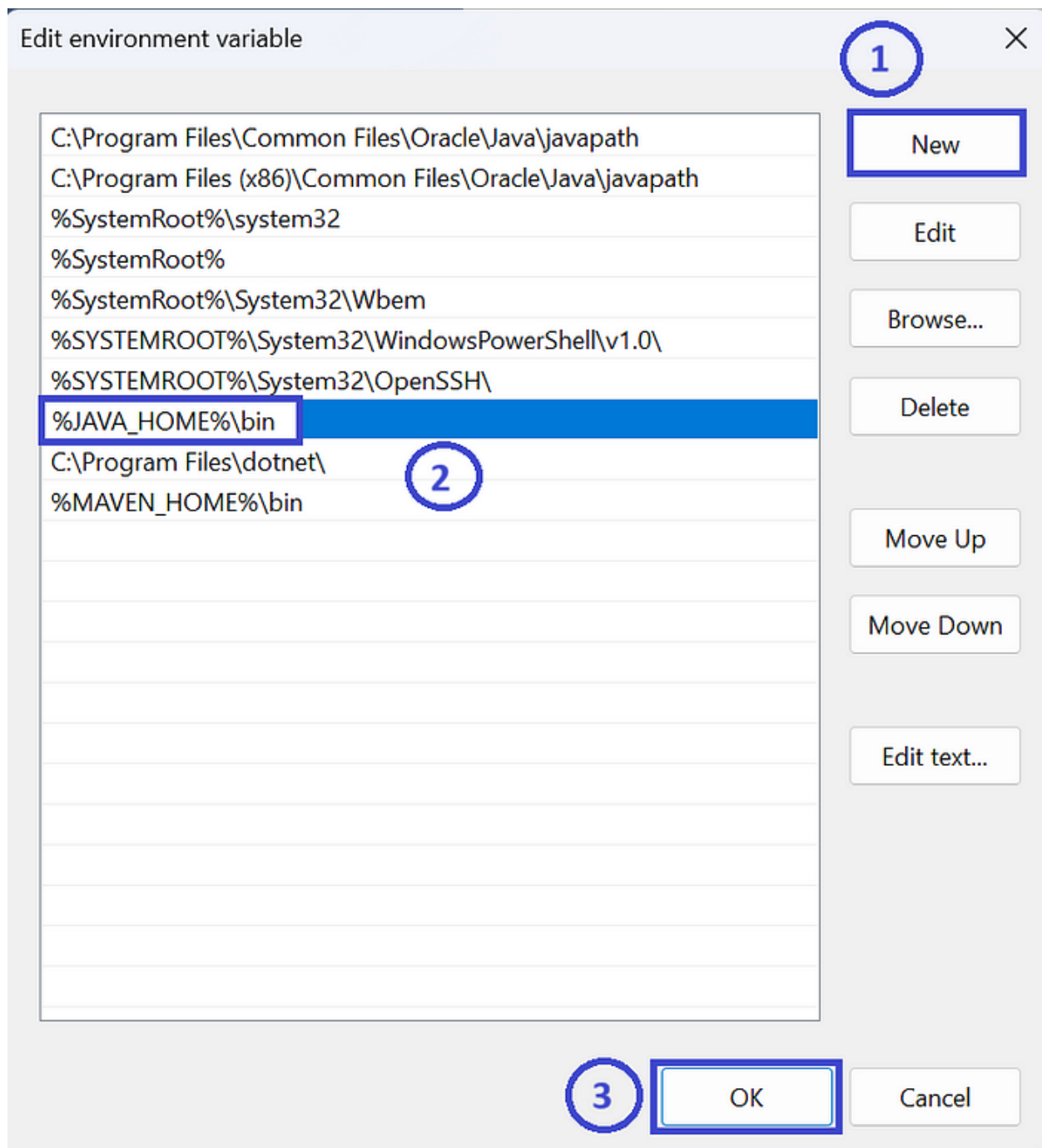
2. Add Java to Path Variables

Setting up Java System Variable ensures that Java is accessible from the command line in any directory.

1. In the Environment Variables wizard, Select the **Path** variable in the System variables category and click **Edit**.



2. Click New.
3. Enter the path as **%JAVA_HOME%\bin**.
4. Click OK to save the changes and exit the variable editing window.



Confirm the changes by clicking OK in the Environment Variables and System properties windows.

Test Java Installation

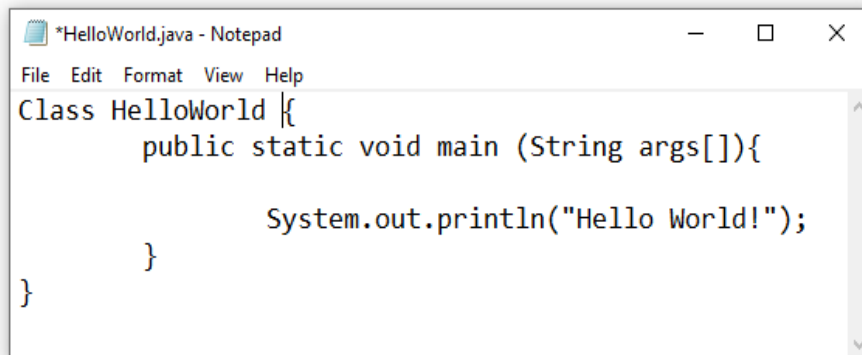
Verify that Java is installed by entering the **java -version** command in the command prompt. If installed correctly, the command outputs the Java version.

```
C:\Users\gaura>java -version
java version "21.0.2" 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
```

5: Writing HelloWorld.java in Text Editor

Follow the below given steps:

- i. Run the Notepad and enter below given code. Save this file with Class name and end it with “.java” extension. Save it on desktop for now!



```
*HelloWorld.java - Notepad
File Edit Format View Help
Class HelloWorld {
    public static void main (String args[]){
        System.out.println("Hello World!");
    }
}
```

- ii. Go to search bar in taskbar, write cmd and Open Command Prompt then write the following commands;
 - a. cd desktop //for going to desktop
 - b. javac HelloWorld.java
 - c. java HelloWorld

The Java programming language compiler (javac) takes your source file and translates the code into instructions known as bytecodes. “Java ClassName” will enable Java virtual machine to run your application/code.

6: Variables and data types in JAVA

In Java, there are three types of variables:

- Local Variables
- Instance Variables
- Static Variables

Local Variables

Local Variables are declared inside the body of a method.

Scope: Variables declared inside a method have method level scope and cannot be accessed outside the method

Instance Variables

Instance variables are defined without the 'static' keyword . They are defined outside a method within a class. Access modifiers can be given for instance variables. They are Object specific.

Scope: Dependent on the access modifier

Class/Static Variables

It is declared with the keyword 'static', outside the method within a class. Static variables are created when the program starts and destroyed when the program stops. There would only be one copy of each static variable per class, regardless of how many objects are created.

Scope: Visibility is like instance variables. However, most static variables are declared public since they must be available for users of the class

Example: Types of Variables in Java

```
class Variable {  
    int InsVarExam = 29; //instance variable  
    static int IsStatVar = 15; //static variable  
    void method() {  
        int IsLocalVar = 90; //local variable  
    }  
}
```

Data Types in Java

Data types classify the different values to be stored in the variable. In java, there are two types of data types:

- Primitive Data Types
- Non-primitive Data Types

Non primitive as arrays, strings

Primitive Data Types

Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values.

There are 8 primitive types: byte, short, int, long, char, float, double, and boolean

Integer data types

byte (1 byte)

short (2 bytes)

int (4 bytes)

long (8 bytes)

Floating Data Type

float (4 bytes)

double (8 bytes)

Textual Data Type

char (2 bytes)

Logical

boolean (1 bit) (true/false)

7: Input & Output

Output in Java Syntax:

```
System.out.println("Hello World");
```

Input in Java Syntax:

```
// import library
import java.util.Scanner;
// Creating scanner object
Scanner ip = new Scanner(System.in); //system.in represents that the input is
given via keyboard
```

Taking input from user

```
int ipFrmUser = ip.nextInt(); // Integer Input
double ipDbUser = ip.nextDouble(); // Double Input
```

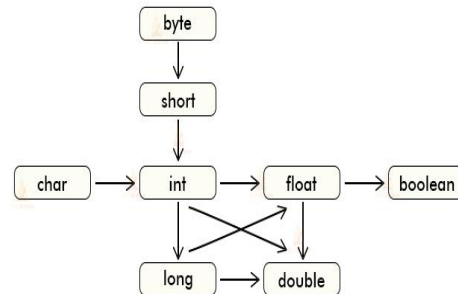
8: Java Variable Type Conversion & Type Casting

A variable of one type can receive the value of another type. Here are 2 cases.

Case 1: Variable of smaller capacity is be assigned to another variable of bigger capacity.

```
double d ;  
int i = 10;  
d = i;
```

Implicit Type Conversion in Java



This process is Automatic, and non-explicit is known as **type conversion**

Case 2: Variable of larger capacity is be assigned to another variable of smaller capacity.

```
double d = 10;  
int i;  
i = (int) d
```

Type Cast
Operator

In such cases, you have to explicitly specify the type cast operator. This process is known as **type casting**.

In case, you do not specify a type cast operator; the compiler gives an error. Since this rule is enforced by the compiler, it makes the programmer aware that the conversion he is about to do may cause some loss in data and prevents accidental losses.

Example: To Understand Type Casting

```
class Demo {  
    public static void main(String args[]) {  
        byte x;  
        int a = 270;  
        double b = 128.128;  
        System.out.println("int converted to byte");  
    }  
}
```

```

        x = (byte) a;
        System.out.println("a and x " + a + " " + x);
        System.out.println("double converted to int");
        a = (int) b;
        System.out.println("b and a " + b + " " + a);
        System.out.println("\ndouble converted to byte");
        x = (byte)b;
        System.out.println("b and x " + b + " " + x);
    }
}

```

Output:

int converted to byte

a and x 270 14

double converted to int

b and a 128.128 128

double converted to byte

b and x 128.128 -128

Lab Tasks

Marks: 10, All Questions carry equal marks

Exercise 1 (JAVA Environment Installation & Error Messages)

Set up a Java development environment. In the main() method of your program try to compile the following invalid Java code snippets. Record the error messages you receive. What do you think each error message indicates?

```
System.out.println("Hello World")
```

```
System.out.println(Hello World)
```

```
System.out.println"Hello World";
```

```
println("Hello World);
```

To generate one final error message, remove one of the brackets from the end of your program. Now what message do you receive?

Exercise 2 (Mathematical Expressions)

Write Java code to identify if the given input by the user is even or odd.

Exercise 3 (Type casting)

Perform division using two double variables and store the result in int variable and print the results

Exercise 4 (Operators)

Find largest among three numbers using if..else as well as operators statement

Enter value a:30

Enter value b:10

Enter value c:70

1. Write a Java comment saying 'OOP! Java Object Oriented Programming'
2. Declare a first name variable and assign a value to it
3. Declare a last name variable and assign a value to it
4. Declare a full name variable and assign a value to it
5. Declare a country variable and assign a value to it
6. Declare a city variable and assign a value to it
7. Declare an age variable and assign a value to it
8. Declare a year variable and assign a value to it
9. Declare a variable isMarried and assign a value to it
10. Declare a variable isTrue and assign a value to it
11. Declare a variable isLightOn and assign a value to it
12. Declare multiple variables on one line

Post lab questions to ponder

1. Can you cast string into int?
2. Why JAVA when there are other OOP languages?