# Content of this File

- Introduction to Java Programming
- JVM, JRE & JDK
- Java Setup in the system
- Introduction to IDEs and Comments and Documentation
- Java First Program

## Introduction to Java Programming

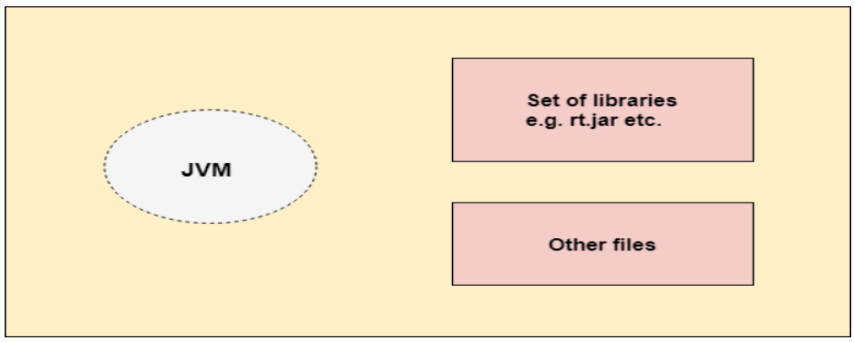
- Developed by Sun Microsystem Inc in 1991, later acquired by Oracle Corporation.
  - Oracle Corporation and Sun Microsystems announced to enter into a definitive agreement under which Oracle will acquire Sun common stock for $9.50 per share in cash. The transaction is valued at approximately $7.4 billion net of Sun's cash and debt.
  - Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time.
  - James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.
  - The language was initially called *Oak* after an oak tree that stood outside Gosling's office. Later the project went by the name *Green* and was finally renamed *Java*, from Java coffee, a type of coffee from Indonesia.

- **Java** is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.
  - It is a general-purpose programming language intended to let programmers *write once, run anywhere* (WORA), meaning that compiled Java code can run on all platforms that support Java without the need to recompile.
  - Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.
  - The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them.
  - The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages.

- As of March 2024, Java 22 is the latest version. Java 8, 11, 17, and 21 are previous LTS versions still officially supported.
- Java is used to create application software such as Desktop application, Enterprise Application, Mobile Application and much more.
- Principles:
  - It must be simple, object-oriented, and familiar.
  - It must be robust and secure.
  - It must be architecture-neutral and portable.
  - It must execute with high performance.
  - It must be interpreted, threaded, and dynamic.
- Sun has defined and supports four editions of Java targeting different application environments and segmented many of its APIs so that they belong to one of the platforms. The platforms are:
  - Java Card for smart-cards.
  - Java Platform, Micro Edition (Java ME) – targeting environments with limited resources.
  - Java Platform, Standard Edition (Java SE) – targeting workstation environments.
  - Java Platform, Enterprise Edition (Java EE) – targeting large distributed enterprise or Internet environments.

# JVM, JRE & JDK

- JVM- Java Virtual Machine - specification
  - JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist.
  - It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.
  - JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other.
  - However, Java is platform independent.
  - There are three notions of the JVM: *specification*, *implementation*, and *instance*.
  - The JVM performs the following main tasks:
    - Loads code
    - Verifies code
    - Executes code
    - Provides runtime environment
- JRE- Java Runtime Environment- It is the implementation of JVM
  - The Java Runtime Environment is a set of software tools which are used for developing Java applications.
  - It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.



❖ JDK- Java Development Kit - Kit that contain Development tools, JRE

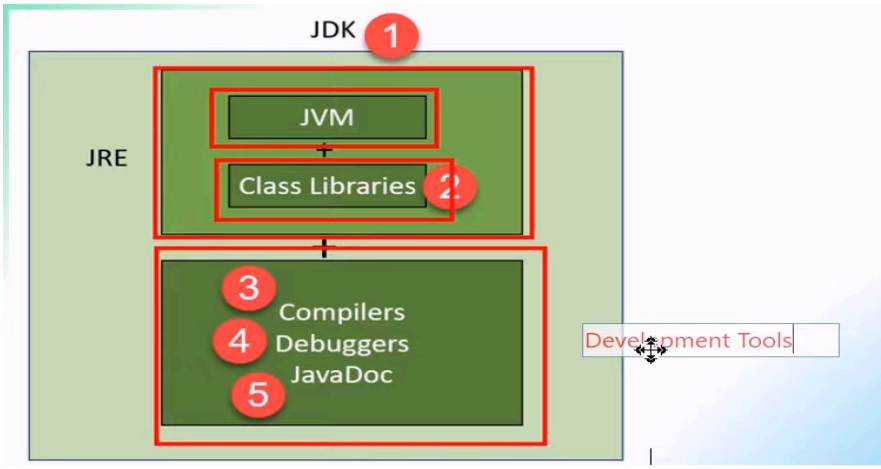
➢ The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets.

➢ It physically exists. It contains JRE + development tools.

➢ JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

➢ The JDK contains development tools which includes a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



## Java Setup in the System

- Download and Install JDK (try to download the updated version or above version 8)
- Download IntelliJ IDEA / or any other IDE you are comfortable with
    - If you are starting to learn Java it is recommended to use notepad and cmd to run your java code.
    - But if you are downloading for development it is recommended to use IDE.
- Don’t forget to set the path of your JDK in your system variable.
- Ready to learn the Java programming language.

# Introduction to IDEs and Comments and Documentation

## Integrated Development Environments (IDEs)

- What is an IDE?
  - An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.
  - It typically consists of a source code editor, build automation tools, and a debugger.
  - Some popular Java IDEs include:
    - IntelliJ IDEA: Known for its intelligent code completion, deep static analysis, and powerful refactoring tools.
    - Eclipse: A widely-used IDE with a rich set of features and a large plugin ecosystem.
    - NetBeans: An open-source IDE with strong support for Java, as well as other languages like PHP and HTML5.
- Why use an IDE?
  - **Code Completion**: IDEs provide smart code completion, which helps in writing code faster and with fewer errors.
  - **Syntax Highlighting**: Helps in easily identifying different parts of the code with color-coded elements.
  - **Debugging Tools**: IDEs come with integrated debugging tools that help in identifying and fixing bugs efficiently.
  - **Refactoring**: Helps in restructuring existing code without changing its external behavior, making it cleaner and more readable.
  - **Integrated Version Control**: Many IDEs have built-in support for version control systems like Git, facilitating collaborative development.

## Comments in Java

- What are Comments?
  - Comments are statements that are not executed by the compiler and interpreter. They are used to make the code more readable.
  - There are three types of comments in Java:
    - **Single-line Comments**: Used for brief comments, starting with `//`

      ```
      // This is a single-line comment
      ```
    - **Multi-line Comments**: Used for longer comments, enclosed between `/*` and `*/`.

      ```
      /*
         This is a multi-line comment
      ```

```
   It can span multiple lines
*/
```

- **Documentation Comments**: Used to create API documentation, starting with `/**` and ending with `*/`. These are processed by the Javadoc tool.

```
/**
 * This is a documentation comment
 * It is used by Javadoc to generate API
documentation
 */
public class MyClass {
    // class code here
}
```

❖ Best Practices for Using Comments

- ➢ Use comments to explain the purpose of the code and any complex logic.
- ➢ Avoid obvious comments that do not add any value.
- ➢ Keep comments up-to-date with the code changes to avoid discrepancies.
- ➢ Use documentation comments for public APIs to provide useful information to other developers.

Documentation in Java

❖ Java Documentation (Javadoc)

- ➢ Javadoc is a tool used for generating API documentation in HTML format from Java source code.
- ➢ It uses special documentation comments (starting with `/**` and ending with `*/`) to extract information.
- ➢ **Tags in Documentation Comments**:
  - @author: Specifies the author of the class or method.
  - @version: Specifies the version of the class or method.
  - @param: Describes a parameter for a method.
  - @return: Describes the return value of a method.
  - @see: Provides a reference to another method or class.
  - @since: Specifies when a class or method was added.
  - @deprecated: Indicates that a method or class should no longer be used.
- ➢ Example of a Documentation Comment

```
/**
 * The Calculator class provides methods to perform basic
arithmetic operations.
 *
 * @author John Doe
 * @version 1.0
 * @since 2024-07-30
```

```
 */
public class Calculator {

    /**
     * Adds two integers.
     *
     * @param a the first integer
     * @param b the second integer
     * @return the sum of a and b
     */
    public int add(int a, int b) {
        return a + b;
    }

    // Other methods and class details
}
```

- Generating Javadoc
  - To generate Javadoc, use the `javadoc` tool provided with the JDK.
  - Command to generate Javadoc from the command line:
    javadoc -d doc -sourcepath src com.example.myproject
  - The above command will generate HTML documentation in the `doc` directory for the source files located in the `src` directory and belonging to the package `com.example.myproject`.

# Java First Program

Writing Your First Java Program

- Steps to Create a Java Program:
  - **Create a Source File**: Write the Java code in a text editor and save it with a `.java` extension.
  - **Compile the Source File**: Use the Java compiler (`javac`) to compile the source file into bytecode, which is saved in a `.class` file.
  - **Run the Bytecode**: Use the Java interpreter (`java`) to run the bytecode on the Java Virtual Machine (JVM).

Example: HelloWorld Program

**Step 1: Create a Source File**

- Open a text editor (e.g., Notepad, Notepad++, or any IDE).
- Type the following code and save it as `HelloWorld.java`:

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

**Step 2: Compile the Source File**

- Open the command prompt (cmd) or terminal.
- Navigate to the directory where `HelloWorld.java` is saved.
- Compile the source file using the `javac` command:

```
javac HelloWorld.java
```

- This will generate a `HelloWorld.class` file in the same directory.

**Step 3: Run the Bytecode**

- Run the compiled bytecode using the `java` command:

```
java HelloWorld
```

- You should see the following output:

```
Hello, World!
```

## Understanding the HelloWorld Program

**Code Explanation:**

```
public class HelloWorld {
```

`public`: An access modifier that makes the class accessible from other classes.

`class`: Keyword used to declare a class.

`HelloWorld`: The name of the class. It should match the filename (`HelloWorld.java`).

```
public static void main(String[] args) {
```

`public`: An access modifier that makes the method accessible from other classes.

`static`: Allows the method to be called without creating an instance of the class.

`void`: The method does not return any value.

`main`: The name of the method. It's the entry point of any Java application.

`String[] args`: Parameter that accepts command-line arguments as an array of `String` objects.

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

`System`: A built-in class that provides access to system-related resources.

`out`: A static member of the `System` class, which is an instance of `PrintStream`.

`println`: A method of `PrintStream` that prints the argument passed to it and terminates the line.

Summary:

- The `main` method is the entry point of the program.
- `System.out.println("Hello, World!");` prints "Hello, World!" to the console.

Common Errors and Troubleshooting

1. **Class Name Mismatch**:
    - Ensure that the class name matches the filename.
    - For example, if the file is named `HelloWorld.java`, the class name should be `HelloWorld`.
2. **Missing `main` Method**:
    - The program must contain a `main` method with the correct signature:

      ```
      public static void main(String[] args)
      ```
3. **Compilation Errors**:
    - Ensure that the Java Development Kit (JDK) is correctly installed and the `javac` command is available in the system path.
4. **Runtime Errors**:
    - Ensure that the compiled `.class` file is in the correct directory and the `java` command is used with the correct class name.