Introduction to Java

- Java is a **high level, object-oriented and platform independent** programming language that is designed to have as few implementation dependencies as possible.

- Java is developed by **James Arthur Gosling** at **Sun MicroSystems** and released in **1995**. Later It was acquired by Oracle Corporation and is widely used for developing applications for desktop, web, and mobile devices.

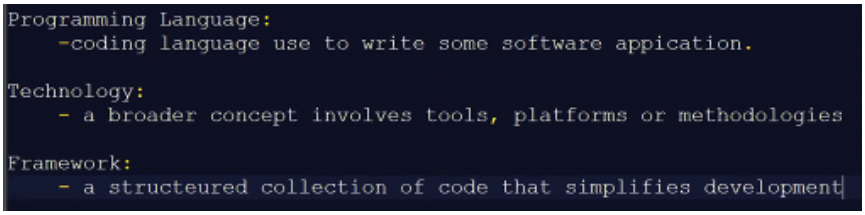
- Java is so popular because it is a **platform independent language, versatile language(used for wide range of applications), largest community support, simplicity, many opportunities available for Java developers in industry.**

- Java follows the principle of **"write once run anywhere" (WORA),** meaning programs can run on any platform with a **Java Virtual Machine(JVM).** Java code can run on all platforms that support Java without the need for recompilation.

- Java makes **writing, compiling, and debugging programming easy**. It helps to **create reusable code** and **modular programs**.

- Java has **both interpreter and compiler**. Java source code is compiled to produce a platform independent bytecode and JVM then interprets this bytecode to execute the code.

- Java is an object oriented language but it is **not purely object oriented language** as it supports the primitive data types as well not just classes and objects.



Features of Java

- **Platform Independent:** Compiler converts source code to byte code and then the JVM executes the bytecode generated by the compiler. This byte code can run on any platform be it Windows, Linux, or macOS which means if we compile a program on Windows, then we can run it on Linux and vice versa. Each operating system has a different JVM, but the output produced by all the OS is the same after the execution of the byte code. That is why we call java a platform-independent language. Give a real-time example like Minecraft or candy crush saga.

- **Object-Oriented Programming:** Java is an object-oriented language, promoting the use of objects and classes. Organizing the program in the terms of a collection of objects is a way of object-oriented programming, each of which represents an instance of the class. The four main concepts of Object-Oriented programming are:
    - Abstraction
    - Encapsulation
    - Inheritance
    - Polymorphism

- **Simplicity:** Java's syntax is simple and easy to learn, especially for those familiar with C or C++. It eliminates complex features like pointers and multiple inheritances, making it easier to write, debug, and maintain code. It provides hardware abstraction.

- **Robustness:** Java language is robust which means it is a reliable language. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, exception handling, and memory allocation. It prevents the system from collapsing/crashing with help of concepts like exception handling, garbage collector.

Features of Java

- **Security:** In java, we don't have pointers, so we cannot access out-of-bound arrays i.e. it shows ArrayIndexOutOfBound Exception if we try to do so. That's 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. It uses garbage collector to dereference the unused instances, making java a secure language.

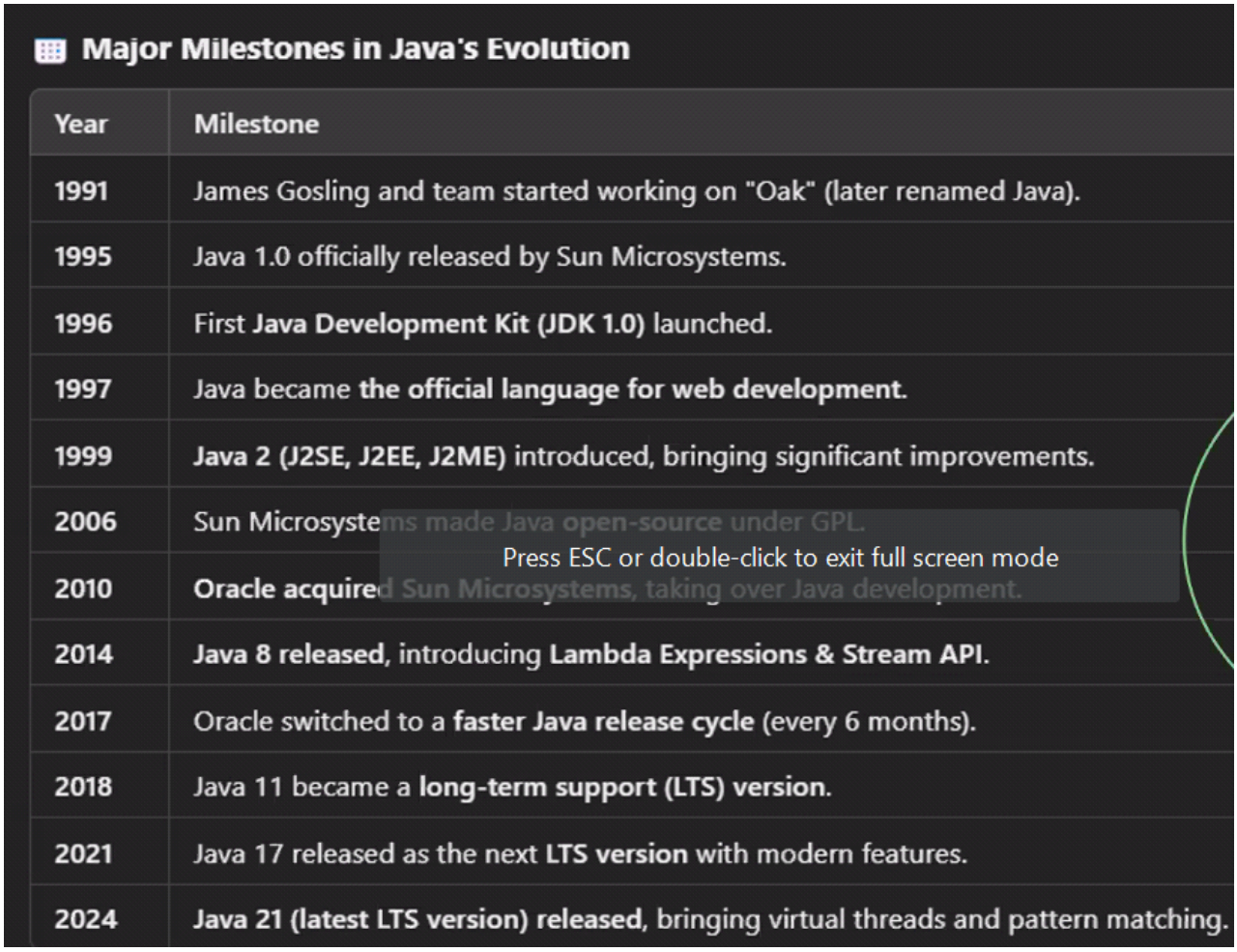
- **Distributed:** We can create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java.

- **Multithreading:** Java supports multithreading, enabling the concurrent execution of multiple parts of a program. This feature is particularly useful for applications that require high performance, such as games and real-time simulations.

- **High Performance:**
  Java architecture is defined in such a way that it reduces overhead during the runtime and sometimes java uses Just In Time (JIT) compiler where the compiler compiles code on-demand basis where it only compiles those methods that are called making applications to execute faster.

- **Portable:**
  Java code is portable as java byte code can be executed on various platforms irrespective of the platform on which it is written.

Major milestones in Java Evolution

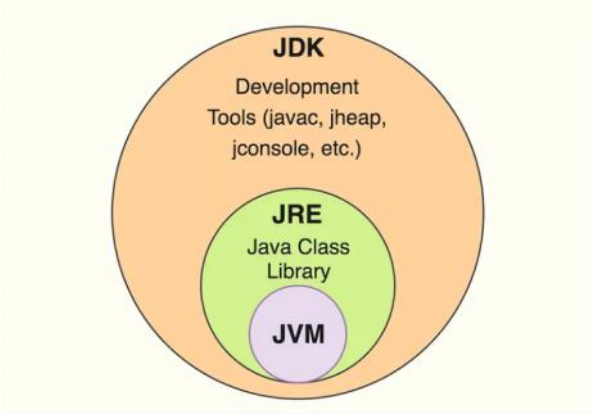


Major Milestones in Java's Evolution

Year	Milestone
1991	James Gosling and team started working on "Oak" (later renamed Java).
1995	Java 1.0 officially released by Sun Microsystems.
1996	First **Java Development Kit (JDK 1.0)** launched.
1997	Java became **the official language for web development.**
1999	**Java 2 (J2SE, J2EE, J2ME)** introduced, bringing significant improvements.
2006	Sun Microsyste**ms made Java open-source under GPL.**
2010	**Oracle acquire**d Sun Microsystems, taking over Java development.
2014	**Java 8 released**, introducing **Lambda Expressions & Stream API**.
2017	Oracle switched to a **faster Java release cycle** (every 6 months).
2018	Java 11 became a **long-term support (LTS) version.**
2021	Java 17 released as the next **LTS version** with modern features.
2024	**Java 21 (latest LTS version) released**, bringing virtual threads and pattern matching.

Java Architecture (JVM, JDK, JRE)

- Java follows a "Write Once, Run Anywhere" (WORA) principle, meaning Java applications can run on any platform without modification. This is possible due to Java Architecture, which consists of three main components:
  - Java Development Kit (JDK)
  - Java Runtime Environment (JRE)
  - Java Virtual Machine (JVM)



- **JDK:** It is a software development environment that is used to develop java applications. The JDK is a complete package that allows developers to write, compile, and debug Java applications. It includes the JRE, along with development tools.

Component	Description
JRE (Java Runtime Environment)	Required to run Java applications.
Compiler (javac)	Converts Java source code (.java) into bytecode (.class).
Java Virtual Machine (JVM)	It executes the compiled bytecode.
Java Runtime Environment (JRE)	It includes libraries and JVM which are necessary to run a java applications.
Debugger (jdb)	Helps debug Java programs.
JavaDoc (javadoc)	Generates documentation from Java comments.
Java Archive (jar)	Packages multiple .class files into a single .jar file.
API libraries	Predefined classes and methods for java development
Other Development Tools	Profilers, monitoring tools, jheap, jconsole, etc.

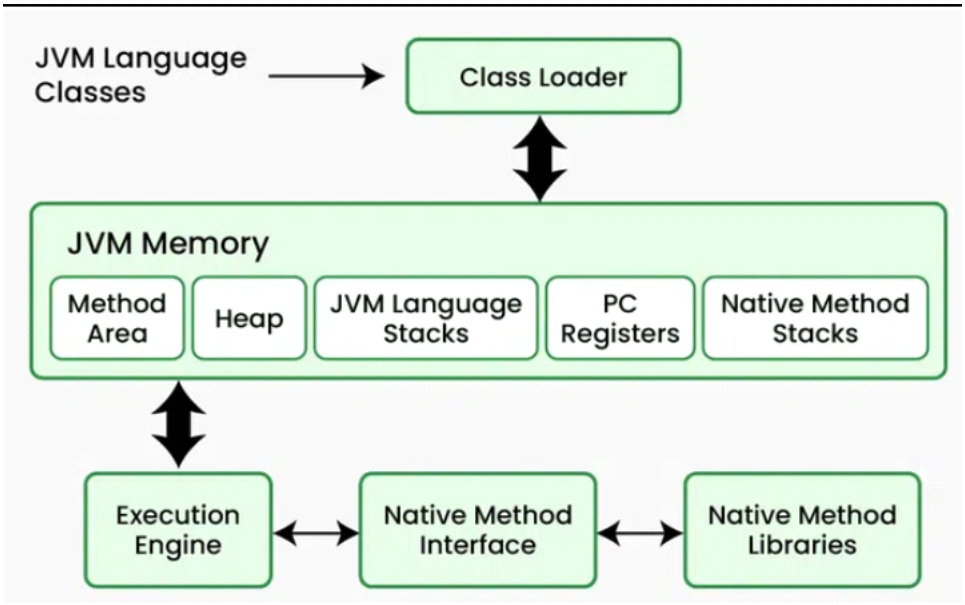
- **JRE:** JRE is a part of JDK and it builds a runtime environment where the Java program can be executed. It contains the libraries and software needed by the Java programs to run. It takes the Java code and integrates with the required libraries, and then starts the JVM to execute it. Based on the Operating System, JRE will deploy the relevant code of the JVM. The JRE provides everything needed to run Java applications but does not include development tools like a compiler.

Component	Description
JVM (Java Virtual Machine)	Executes Java bytecode.
Core Libraries (rt.jar)	Essential Java class libraries (e.g., java.lang, java.util).
Java ClassLoader	Loads Java classes into memory.
Garbage Collector	Manages memory automatically.

- **JRE vs JDK vs JVM**
  - JDK = JRE + Development Tools (compiler, debugger, Interpreter, etc.)
  - JRE = JVM + Core Libraries (Only for running Java apps), runtime files (no compiler)
  - JVM = runs java applications by converting bytecode to machine code.

- **JVM:**
  - JVM is the core of Java's architecture. It is responsible for loading, verifying, and executing Java bytecode.
  - It serves as a bridge between Java programs and the underlying operating system.
  - It is an engine that provides a run-time environment to run the Java applications and it is part of JRE.
  - It runs java applications by converting bytecode to machine code.

- Java uses the combination of both (compiler and interpreter). source code (.java file) is first compiled into byte code and generates a class file (.class file). Then JVM converts the compiled binary byte code into a specific machine language. In the end, JVM is a specification for a software program that executes code and provides the runtime environment for that code.



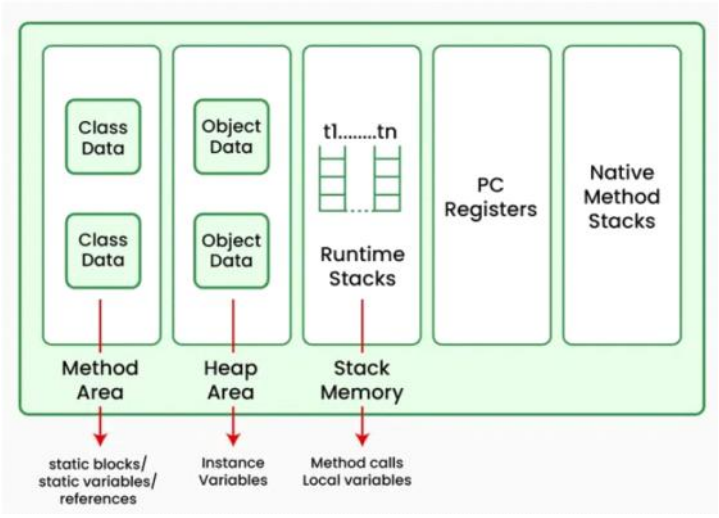
- JVM consists of three main subsystems:
  - Class Loader Subsystem
  - JVM Memory Areas
  - Native Method Interface
  - Execution Engine
  - Native Method Libraries

- **The Class Loader subsystem:**
  - It is responsible for loading Java bytecode (.class files) into JVM memory when a program starts. Three Types of Class Loaders:
    - Bootstrap ClassLoader – Loads core Java classes from rt.jar (like java.lang.Object).
    - Extension ClassLoader – Loads classes from the ext directory (java.ext.dirs).
    - Application ClassLoader – Loads application-specific classes from the classpath.

- **Class Loading Process:**
  - **Loading:** Reads .class files from disk or network.
  - **Linking:**
    - Verification: Ensures bytecode follows Java standards.
    - Preparation: Allocates memory for static variables.
    - Resolution: Converts symbolic references to actual memory locations.
  - **Initialization:** Executes static initializers and assigns values.

- **<u>JVM Memory Areas</u>:**
  - JVM divides memory into different areas:

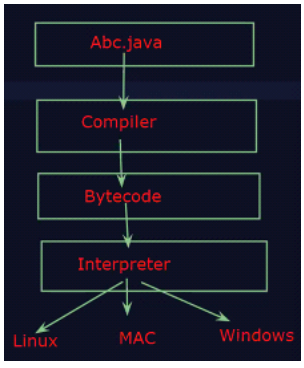
Memory Area	Description
Method Area	Stores class metadata, static variables, references and method code.
Heap	Stores objects and instance variables (shared across threads).
Stack	Stores method call frames and local variables (separate for each thread).
PC Register	Holds the address of the currently executing instruction.
Native Method Stack	Manages native (non-Java) method calls.



- **Execution Engine:**
  - Execution engine executes the ".class" (bytecode). It reads the byte-code line by line, uses data and information present in various memory area and executes instructions. It can be classified into three parts:
    - **Interpreter:** It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.
    - **Just-In-Time Compiler(JIT):** It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native/machine code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.
    - **Garbage Collector:** It destroys un-referenced objects. Automatically manages memory by reclaiming unused objects.

- **Java Native Interface (JNI):** It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution. It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

- **Java Method Libraries:** These are collections of native libraries required for executing native methods. They include libraries written in languages like C and C++.



Java Development platforms

- **J2SE- Java Platform Standard Edition: (local purpose, used by learners)**
  - It has concepts for developing software for Desktop based (standalone) CUI (command user interface) and GUI (graphical user interface) applications, applets, database Interaction application, distributed application, and XML parsing applications.

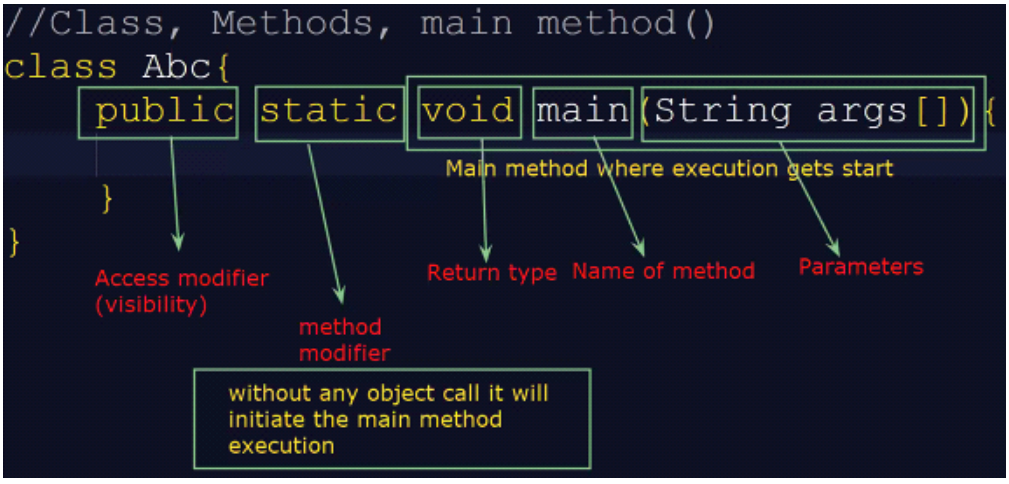
- **J2EE -Java Platform Enterprise Edition: (Advanced Java, Java for Web development, corporate purpose)**
  - It has concepts to develop software for Web applications, Enterprise applications, and Interoperable applications. These applications are called high-scale applications. Some examples are:- banking and insurance-based applications.

- **J2ME - Java Platform Micro Edition: (Android development, Service Platform)**
  - It has concepts to develop software for consumer electronics, like mobile and electronic level applications. Java ME was popular for developing mobile gaming applications. This edition was called micro because these edition programs are embedded in small chips. The program embedded in the chip is called micro (small).

- **JavaFX - Java Platform Effects:**
  - Used for creating rich internet application (where multimedia is used), designing lightweight user interface applications.
  - Java FX stands for Java platform Effects (Eff=F, ects=X). It provides concepts for developing rich internet applications with more graphics and animations. It's an extension concept to swing applications of Java SE. The Java FX API is included as part of Java SE software. Just by installing Java SE, we will also get Java FX API.

Main method Signature

- **Main method Signature:**



- **Pilot Program:**

```java
class demo{
    public static void main (String[] args){
        System.out.println("Hello, World!");
    }
}
//compile javac <filename.java>
//Run: java classname
```

```
C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>javac demo.java

C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>java demo
Hello, World!

C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>
```

- **You can define multiple classes in single java file. If class to be executed doesn't have a similar name as filename, make sure proper class name while running the bytecode.**

```java
class abc{
    public static void main(String[] args){
        System.out.println("abc class");
    }
}

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

```
C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1> javac demo.java

C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>java demo
demo class

C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>java abc
abc class
```

- Suppose the file is saved as demo1.java and we want to execute demo class. In this instance if filename (i.e. demo1) is used instead of class name (i.e. demo), JVM will throw a runtime error called **"Couldn't find or load main class"**. This happens because when demo1.java was compiled, two .class files(bytecodes) were generated with name similar to the classes in the program. Since there is no class with name demo1.java in this code, demo1.class file isn't created, triggering an error.

```
C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>java demo1
Error: Could not find or load main class demo1
Caused by: java.lang.ClassNotFoundException: demo1
```

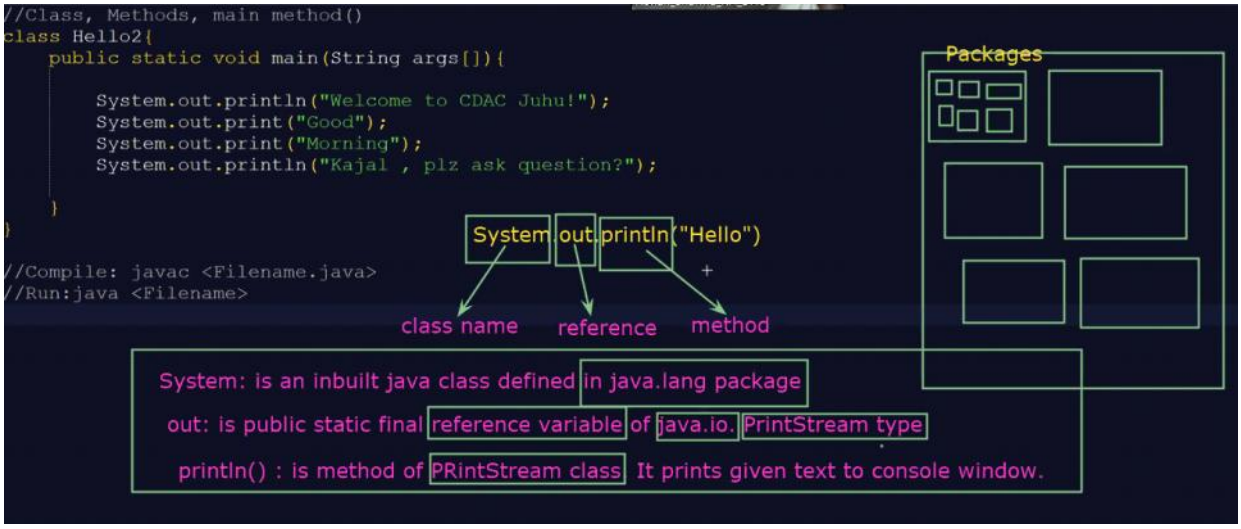
- **Note: If class is declared with public access modifier, class name containing main method must be same as file name.**

```java
public class demo{
    public static void main (String[] args){
        System.out.println("demo class");
    }
}
//compile javac <filename.java>
//Run: java classname
```

- **Above program will only be executed successfully, if filename is demo.java. Anything else will result in a compile time error.**

```
C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>javac demo1.java
demo1.java:1: error: class demo is public, should be declared in a file named demo.java
public class demo{
       ^
1 error
```

println() method



- There are 3 methods in PrintStream class
  - **print()** - prints the data on the current line and doesn't move the cursor to next line.
  - **println()** - prints the data on the current line and moves the cursor to the next line.
  - **printf()** - This method is used to get formatted output.

    Syntax:- System.out.println(format, argument)

```
Format specifiers:
----------------------
%d: Integers
%f:Floating point
%s:String
%c:Character
%b:Boolean
%n:New Line
```

- Example:

```
public class PRINT{
    public static void main(String[] args){
        System.out.println("Hello");
        System.out.print("Hi");
        System.out.println("Same Line");
        System.out.println("New Line");

        double num = 100.23468562;
        System.out.printf("Number = %.2f%n", num);
    }
}
```

```
C:\Users\SHRIRAM SABADE\OneDrive\Desktop\FEB25\OOPJ\Day 1>java PRINT
Hello
HiSame Line
New Line
Number = 100.23
```

- **JIT (Just in Time) Compiler** converts bytecode into native machine code at runtime to improve performance.
- **Purpose:** Speed up the execution process for frequently used code.
- **Working:**
  - JVM first interprets the bytecode line by line.
  - JIT detects **hotspot methods (frequently used methods)**
  - Convert those methods into **native machine code.**
  - **Stores compiled code** for future reuse.

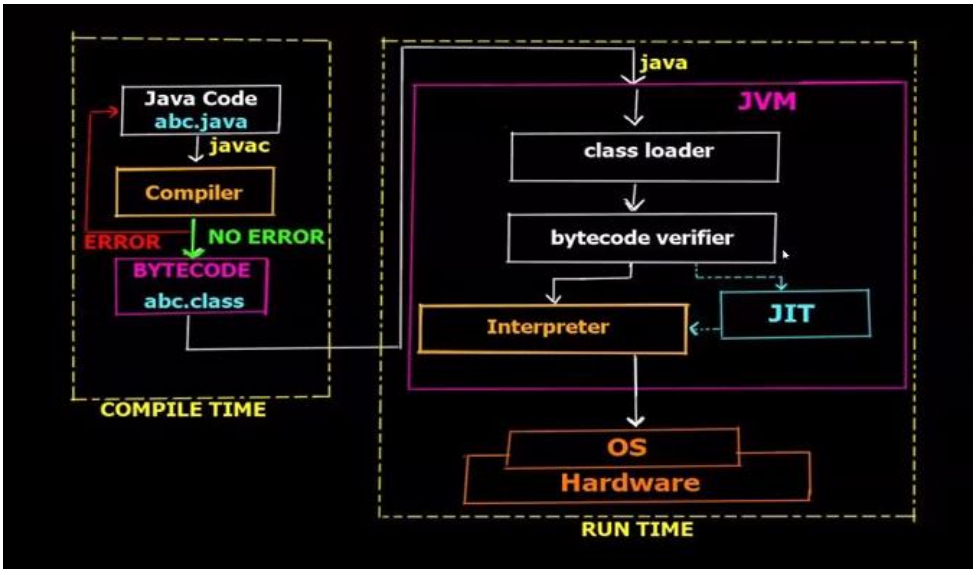


Fig. Java file execution

Tokens in Java