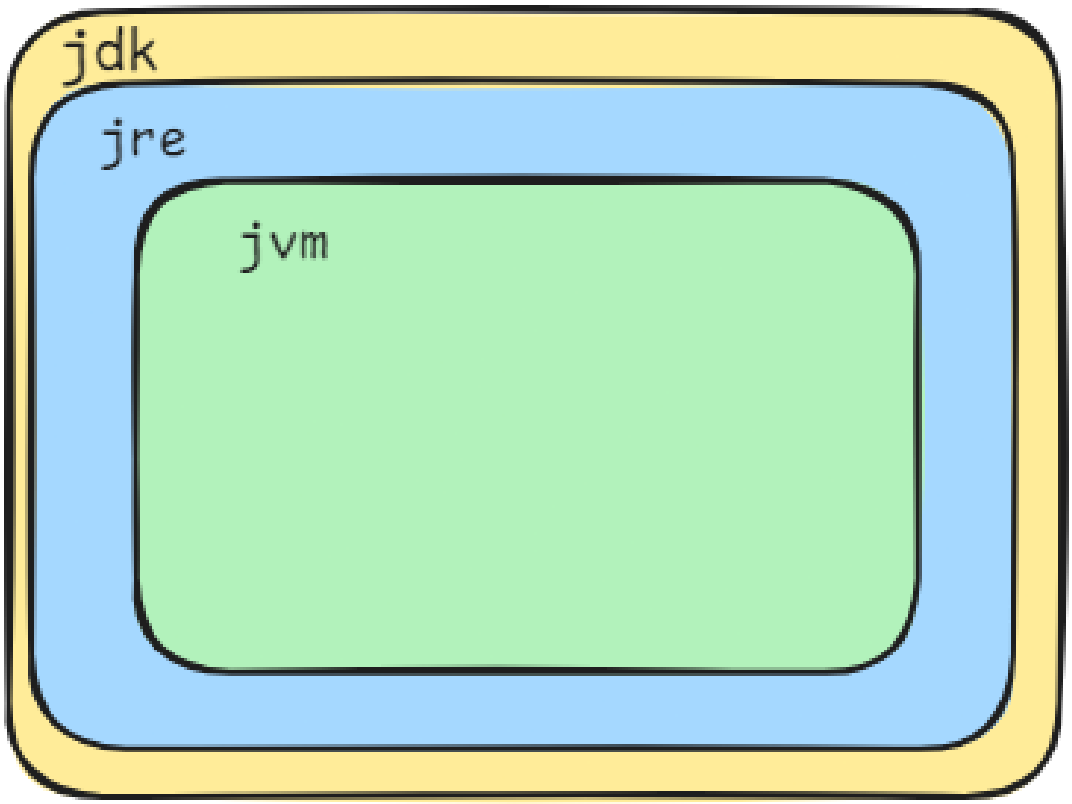
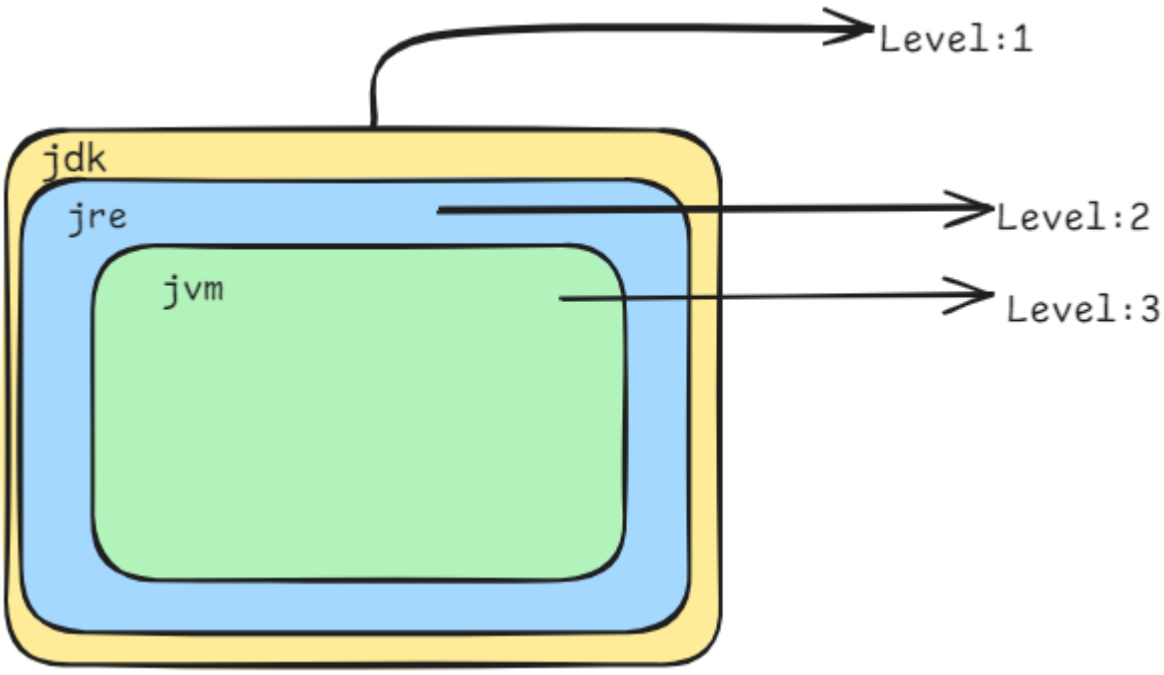
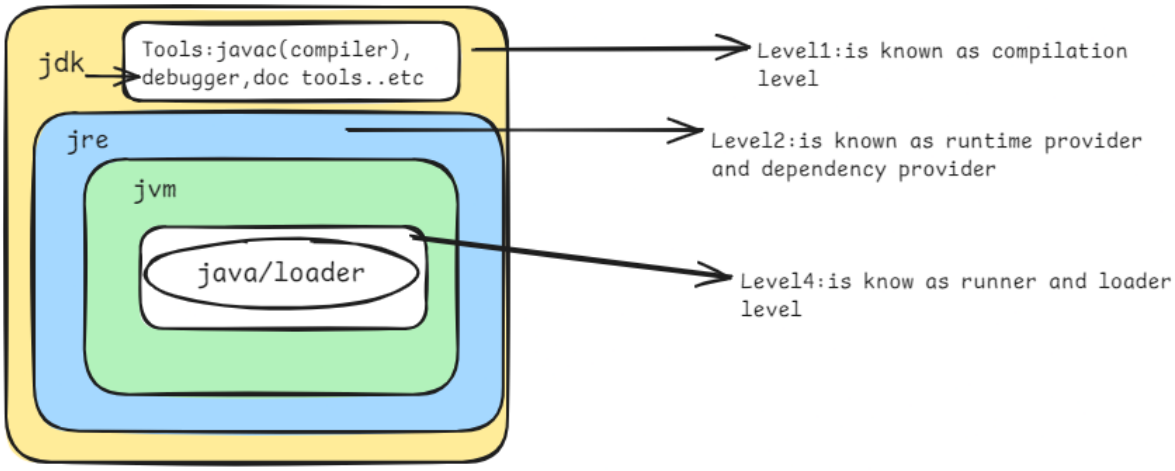
Introduction To Java Architechture





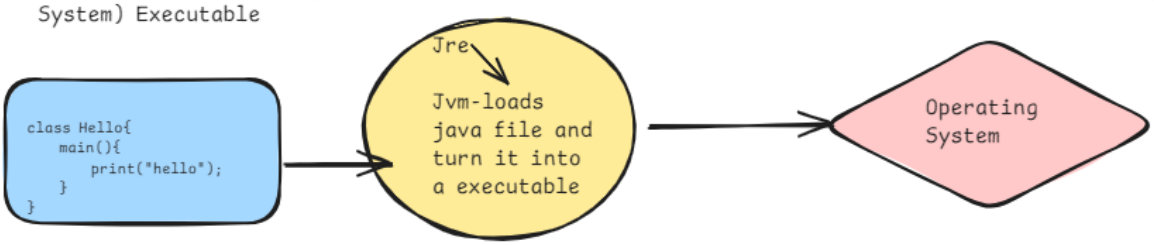


Diving Deep into Java Terminology
---
1)Java Development Kit
:-JDK is a software development kit which bundels the java compiler
and its runtime enviorment together for developing java application.
It includes Java runtime enviorment(JRE) which is responsible for providing
runtime libraries and class file required to run the java program and jre
comes with JVM(Java Virtual Machine) which is responsible for loading the
byte file into Runtime and executing it

2)Java Runtime Enviorment(JRE)
:-Now to run the java program we need an enviorment to make it run and that where JRE comes
into the play it helps in providing the virtual machine and dependency which are required by
a simple or complex java Program

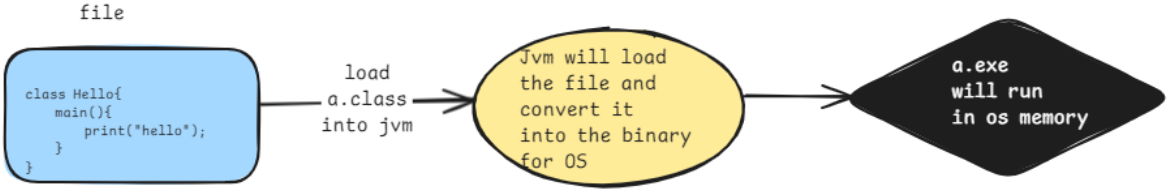
->JRE consist of jvm,core classes and supporting file and packages

->JRE is the underlying technology that communicates b/w the .java program and os(Operating
System) Executable



3)Java Virtual Machine(JVM)
:-The JVM is the core part of the JRE that actually executes Java bytecode.
It provides a runtime environment for Java applications, handling tasks like memory management, garbage collection, and executing the compiled Java code.

->Jvm loads the .class file also known as the byte code into its enviorment and start's processing it into a executable for that perticular Operating system like for windows the executable will be .exe and for mac or linux it will be .out file



a.java-->a.class via javac

->Note:JVM is available for many hardware and software.That is JDK,JRE and JVM are platform Dependent because the configuration of each os is different from each other.However Java program is platform independent as it relies on jvm so as long as Computer posses a jvm we can load java program or class file into it and run it on any os

->JVM performs the following main task:
*Load code
*Verifies code
*execute code

Few Important questions to address

Q1.Is JDK is necessary to run java program
->No,JDK is necessary to compile java program into .class file or byte code ,JRE is necessary to run java program

Follow up Question

Q2.Can we run JRE without JDK
->

Yes, you can run the **Java Runtime Environment (JRE)** without the **Java Development Kit (JDK)**. Here's a breakdown of how they work:

JRE vs. JDK

- **JRE (Java Runtime Environment)**:
  - This is what you need to run Java applications. It includes the Java Virtual Machine (JVM), core libraries, and other components required to execute Java programs.
  - You can run Java applications and applets using just the JRE.
- **JDK (Java Development Kit)**:
  - This is a more comprehensive toolkit that includes everything in the JRE, plus development tools like the Java compiler (`javac`), debuggers, and other utilities for building Java applications.
  - You need the JDK if you want to develop Java applications.

Running Java Applications

- **Using Only JRE**: If you have the JRE installed, you can run compiled Java programs (i.e., `.class` files or JAR files) using the `java` command. For example:

  bash Copy code
  ```bash
  java -jar MyApplication.jar
  ```

- **Using JDK**: If you have the JDK installed, you can both compile and run Java applications. For instance:

  bash Copy code
  ```bash
  javac MyProgram.java   // Compiles the Java source code
  java MyProgram         // Runs the compiled program
  ```

Summary

- You **do not need** the JDK to run Java applications if you have the JRE installed.
- However, if you want to develop Java applications (compile source code), you need the JDK.

->So if level 2 JRE is not dependent on
Level 1 JDK does that mean Level 3 JVM is
also not dependent on Level 2 JRE

Q3.Can JVM run without JRE ?
->Answer is no JVM needs jre
The Java Virtual Machine (JVM) is part of
the Java Runtime Environment (JRE),but JRE is not a part of jdk
JDK just bundle JRE with other tool

->JRE is a seprate entity but JVM is a part of JRE and not seprate
entity JVM need JRE dependencies to run a java file

Java Runtime Environment (JRE)

- The JRE is a software package that provides the necessary libraries and components to run Java applications.
- It includes the JVM, along with the Java class libraries and other components that Java programs need to run.

Java Virtual Machine (JVM)

- The JVM is the core part of the JRE that actually executes Java bytecode.
- It provides a runtime environment for Java applications, handling tasks like memory management, garbage collection, and executing the compiled Java code.

Summary

- **JRE contains the JVM**: The JRE provides the environment to run Java applications and includes the JVM as a component.
- **JVM is a part of JRE**: The JVM is specifically responsible for executing Java bytecode and managing the execution environment.

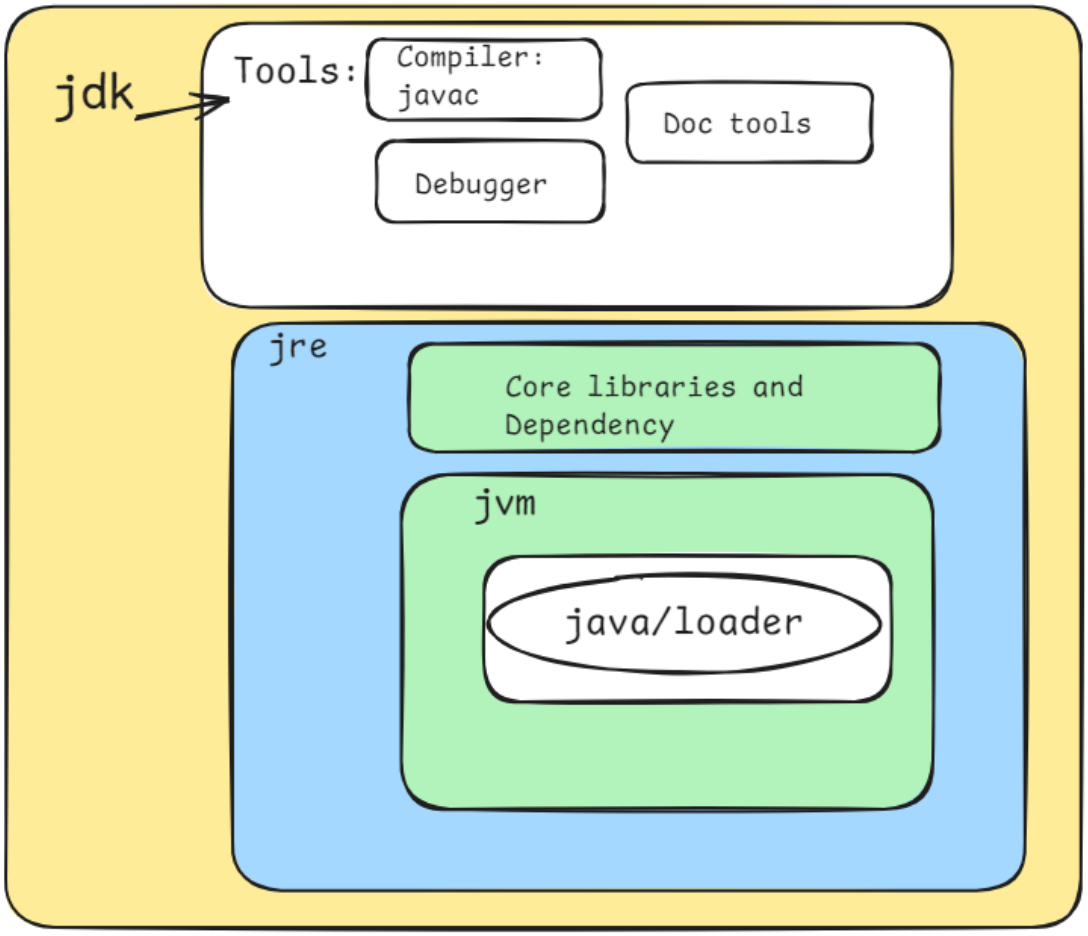
Visual Representation

lua Copy code

```lua
+-----------------+
|      JRE        |
| +-------------+ |
| |     JVM     | |
| +-------------+ |
+-----------------+
```

Let Understand JDK-Architecture:

Here's a simplified flow diagram to illustrate the architecture of the Java Development Kit (JDK):

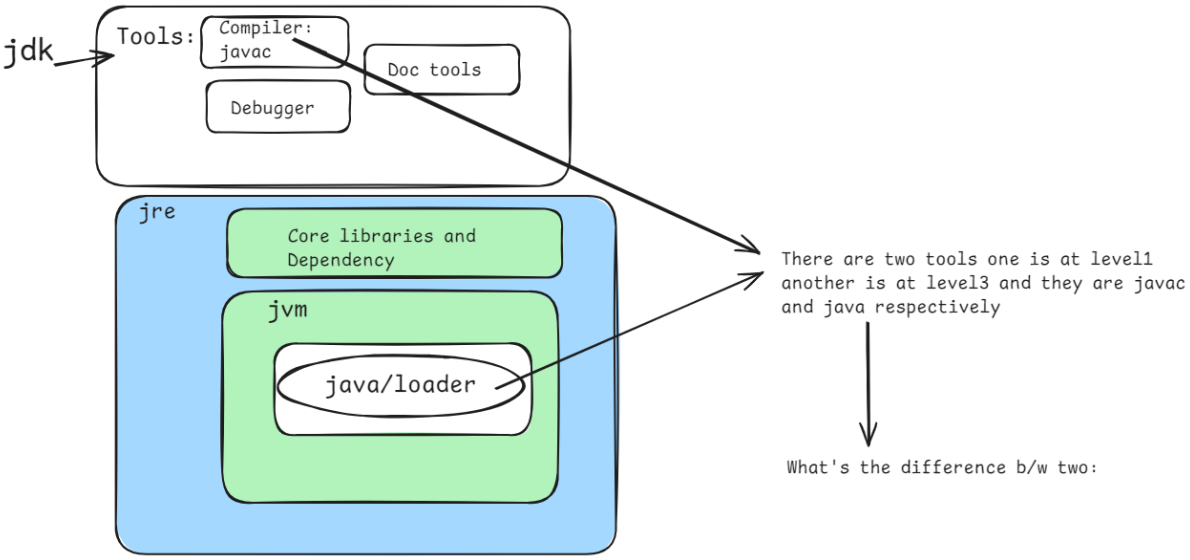


Explanation of Components

1. **JDK**:
   - The top-level package that includes all tools and components necessary for Java development.
2. **Tools**:
   - **Compiler** (`javac`): Translates Java source code into bytecode.
   - **Debugger** (`jdb`): Helps in debugging Java applications.
   - **Documentation Tools**: Generate API documentation from comments in the source code.
3. **JRE (Java Runtime Environment)**:
   - Contains the components needed to run Java applications, including:
   - **JVM (Java Virtual Machine)**: Executes Java bytecode.
   - **Core Libraries**: Standard libraries that provide the basic functionality for Java programs.

Summary

The JDK is an essential part of Java development, providing all necessary tools and libraries to compile, run, and debug Java applications. If you need any more details or a different format, feel free to ask!



`javac` and `java` are two essential commands in the Java Development Kit (JDK), but they serve different purposes:

`javac`

- **Purpose**: This command is the Java Compiler.
- **Function**: It compiles Java source code (`.java` files) into Java bytecode (`.class` files). This bytecode is what the Java Virtual Machine (JVM) executes.
- **Usage**: You typically use `javac` when you're developing Java applications and need to convert your source code into bytecode.
- **Example**:

  bash Copy code
  ```bash
  javac MyProgram.java
  ```

  After running this command, if there are no errors, it generates `MyProgram.class`, which contains the bytecode.

`java`

- **Purpose**: This command is the Java Application Launcher.
- **Function**: It runs Java applications by invoking the JVM to execute the compiled bytecode (`.class` files).
- **Usage**: You use `java` when you want to run a compiled Java program.
- **Example**:

  bash Copy code
  ```bash
  java MyProgram
  ```

  This command runs the `MyProgram` class, which should contain a `main` method as the entry point.

Summary of Differences

Feature	`javac`	`java`
Type	Compiler	Application launcher
Input	Java source files (`.java`)	Compiled bytecode (`.class`)
Output	Compiled bytecode (`.class`)	Executes the Java application
Purpose	To compile code	To run code

Example Workflow

1. **Compile the Code:**

bash Copy code

```bash
javac MyProgram.java
```

2. **Run the Compiled Program:**

bash Copy code

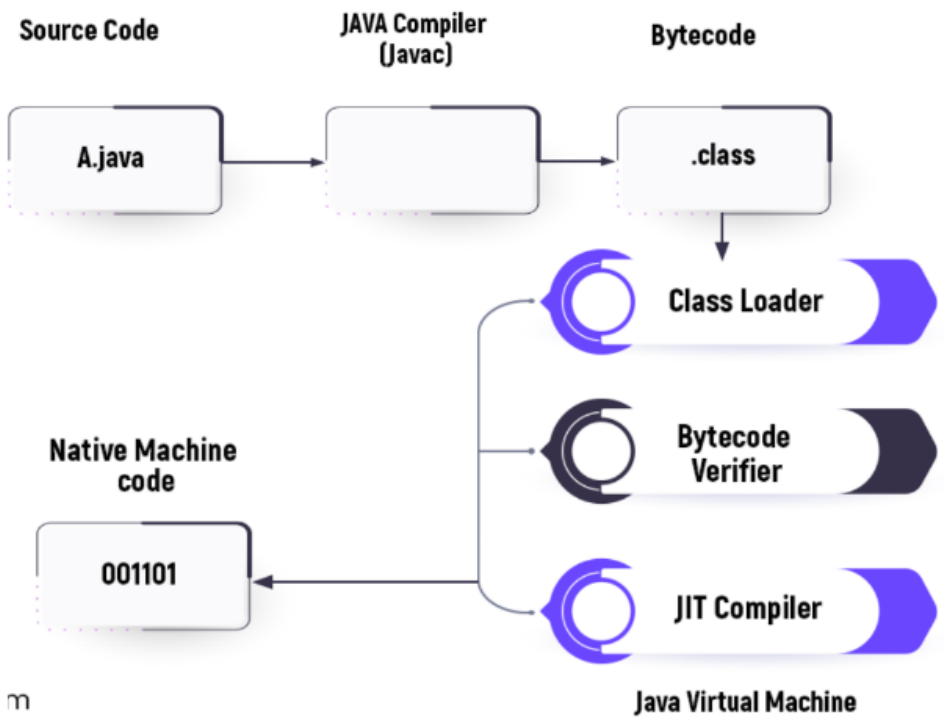
```bash
java MyProgram
```

This distinction is crucial for understanding the Java development process. If you have any further questions or need clarification, feel free to ask!

Final work flow of Java compilation process

lua Copy code

```lua
+----------------------+
|     Java Source      |
|      (MyProgram.java) |
+----------------------+
            |
            v
+----------------------+
|     Java Compiler    |
|        (javac)      |
+----------------------+
            |
            v
+----------------------+
|     Bytecode         |
|      (MyProgram.class)|
+----------------------+
            |
            v
+----------------------+
|     JVM              |
| +------------------+ |
| |  Class Loader    | |
| +------------------+ |
| |   Execution Engine | |
| +------------------+ |
+----------------------+
```


Source Code
JAVA Compiler
(Javac)
Bytecode
A.java
.class
Class Loader
Bytecode
Verifier
Native Machine
code
001101
JIT Compiler
Java Virtual Machine

What is a class ?

->Suppose we have a main.java file when compiled the main.java file will change into main.class file which is a byte code but what is a .class file or a class

->so a class is a user define data type:
suppose if we have to declare a int varibale syntax is:int num1;
so the syntax is :

```
<datatype> variable_name=value;
```

Similarly the syntax to use class is:

```
<class_name> instance_name=new class_name();
```

now why is it i said that class are user define data type well as we know that java is a static programming language and in order to declare variable we need to first declare the datatype it belongs too as it's necessary to allocate that perticular amount of memory to that variable

Similarly if we have a class:

```
class HelloWorld(){
    String greeting;
    public void greet()
    {
        system.out.println(greeting);
    }
}

public static void main(String[] args)
{
    HelloWorld greetObject=new HelloWorld();
    greetObject.greeting="Hello World!";
    greetObject.greet();
}
```

This is a user define datatype which will have its own property and methods

Now here we are initiating the datatype and for that we need to use a "new" keyword:

since its a user define data type Java compiler does not know how much memory should be allocated to it so it has to go in a dynamic memory also know as Heap for that we use new keyword

What is Class?

->A class is user define schema/object or a datatype which shares common property/attribute and behaviour(mehthod)

->a class is not a real world entity.It is just a template or blueprint or more specifically its prototype for genrating object

->Just defining class does not occupy memory for class to occupy memory we will have to initiate it using (new) keyword and load it into the memory

*A class in java can contain

->A data member(variable inside class)
->member function/method
->interface and nested class

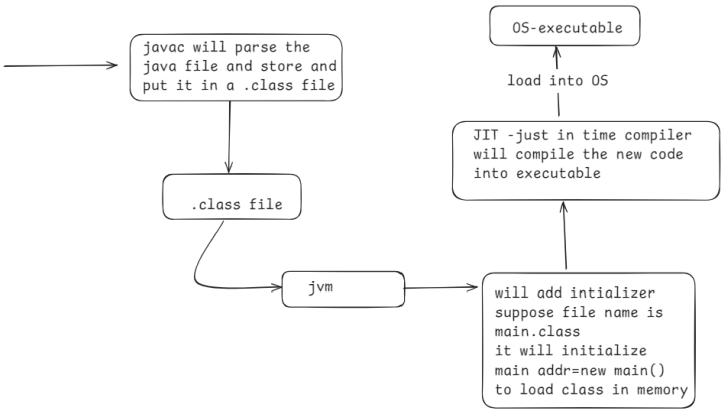
Syntax to declare a class

```
access_modifier class class_name{
    data_member;
    methods();
    constructor;
    interface;
    nested_class;
}
```

Similarly if we have a class:

```
class HelloWorld(){
    String greeting;
    public void greet()
    {
        system.out.println(greeting);
    }
}

public static void main(String[] args)
{
    HelloWorld greetObject=new
HelloWorld();
    greetObject.greeting="Hello World!";
    greetObject.greet();
}
```



Now Java Does not allow us to see the byte code directly for memory safety purpose,so its hard to see the class schema in byte code ie .class but javascript allow us to and the concept of class is same and in both js and java

```java
class HelloWorld {
    // Your program begins with a call to main().
    // Prints "Hello, World" to the terminal window.
    public static void main(String args[])
    {
        System.out.println("Hello, World");
    }
}
```

Replicate it in js

```javascript
class Main{
  main()
  {
    console.log("Hello World!");
  }
}
```

```javascript
console.log(new Main());
```

byte code

```
▼ Main {}
  ▼ constructor: ƒ Main()
      length: 0
      name: "Main"
      ▶ prototype: {}
      ▶ [[Prototype]]: ƒ ()
  ▼ main: ƒ main()
      length: 0
      name: "main"
      ▶ [[Prototype]]: ƒ ()
  ▶ [[Prototype]]: {}
```

```
//javac --- "class Main{main(){console.log("Hello");}}" and assign it to a file of .class

//class loader--will load the .class file and add initiator to load into memory
let addr1=new Main();//this new keyword will load the file into heap that is dynamic memory and will store it in a file Main.class

//and when we will run java Main -->this will trigger Main.class into JVM and unwrao the Main class and will look for varibles and function
//in the class file and apply the instance on it

addr1.main();
```

Output

```
Console ×

Hello World!
```