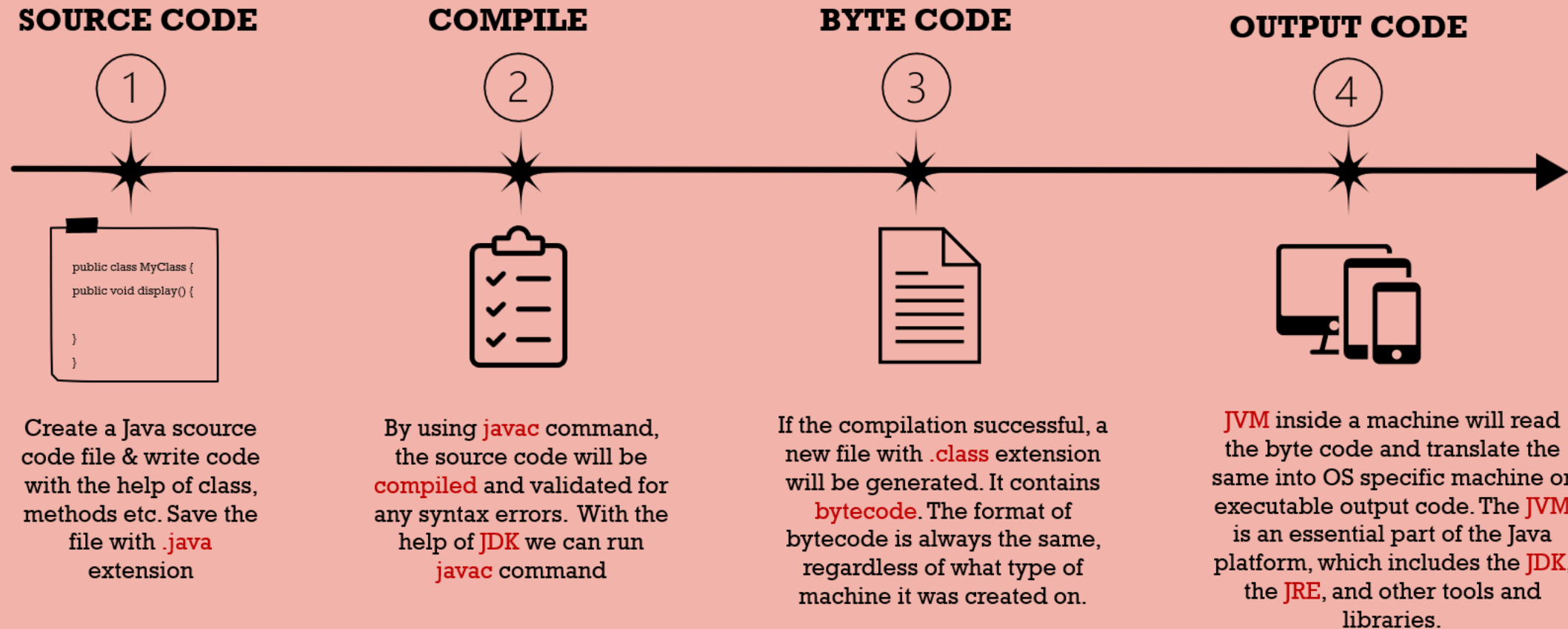
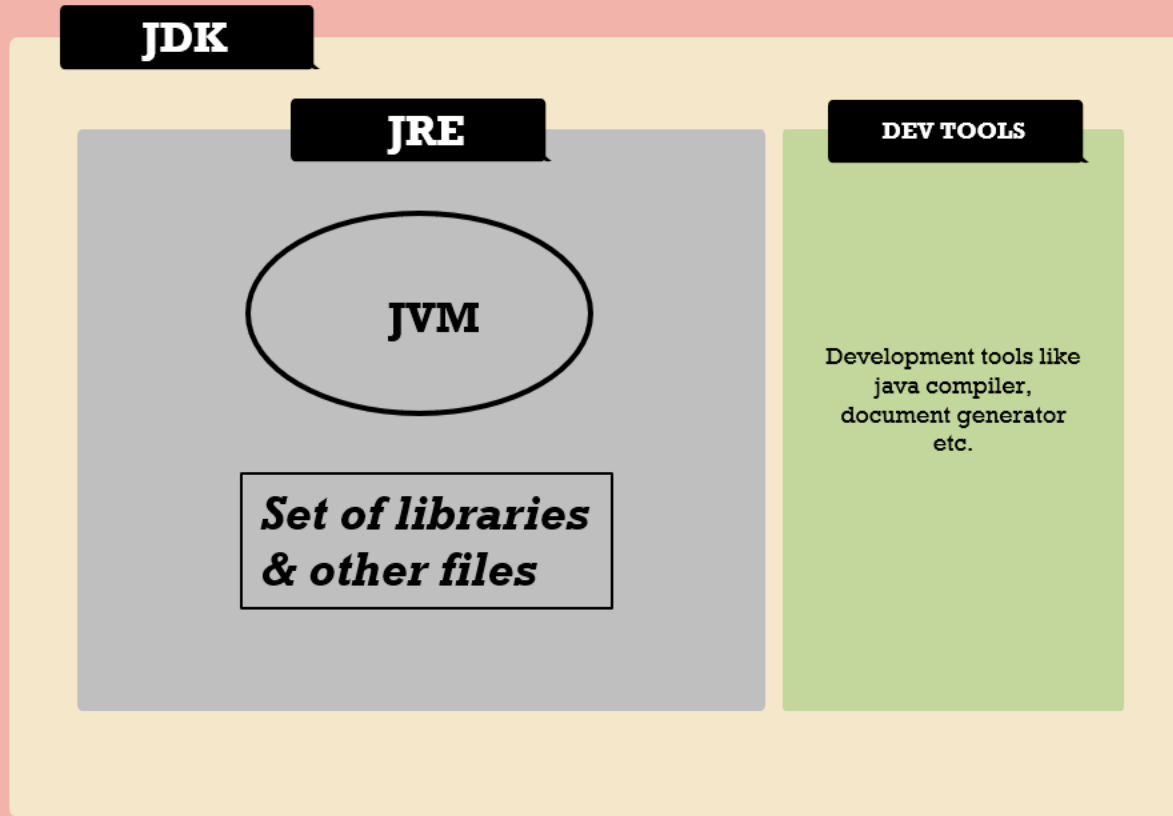


# The LifeCycle of a Java Program



# What is JDK, JRE, JVM ?



## Java Development Kit (JDK)

A collection of programs (tools and utilities), including the 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.

## Java Compiler

A program that can read a text written in the Java programming language and translate it into bytecode that can be interpreted by the Java Virtual Machine (JVM) into binary code that is executable by a computer

## Java Runtime Environment (JRE)

The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files. Usually developers only need JDK & all other people install JRE alone

## Java Virtual Machine (JVM)

A program that reads bytecode of the compiled Java program and interprets it into OS specific binary code that is executable by a computer.



$\text{JDK} = \text{JRE}_{(\text{JVM} + \text{LIBRARIES})} + \text{DEV TOOLS}$

$\text{JRE} = \text{JVM} + \text{LIBRARIES}$

From Java 9, the JRE is no longer offered. Only the JDK is offered. This is to reduce the java installation size by removing duplicate files.

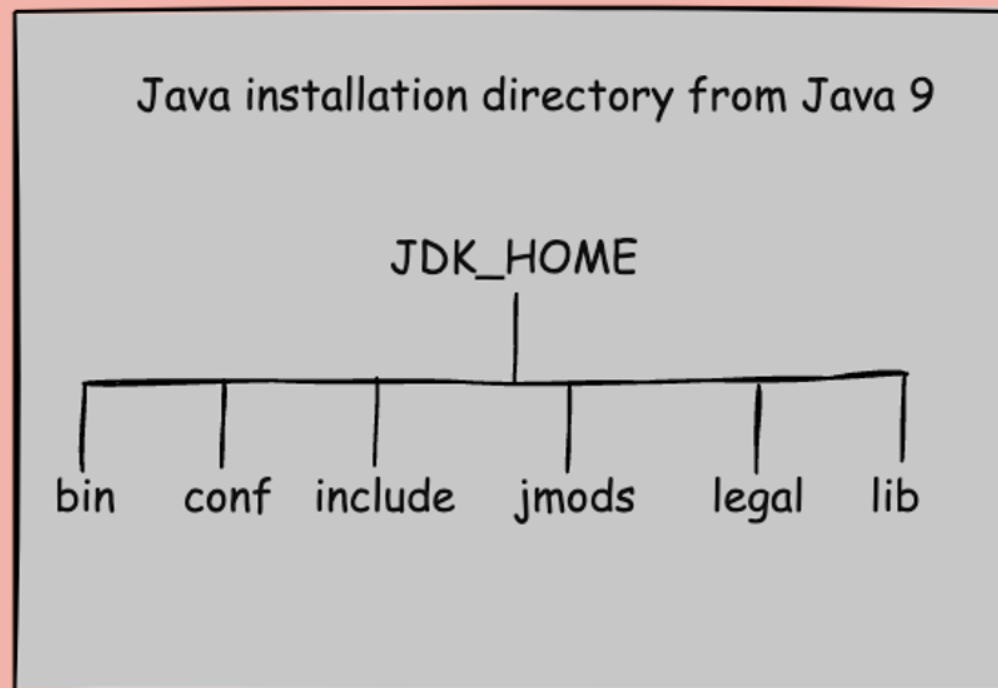
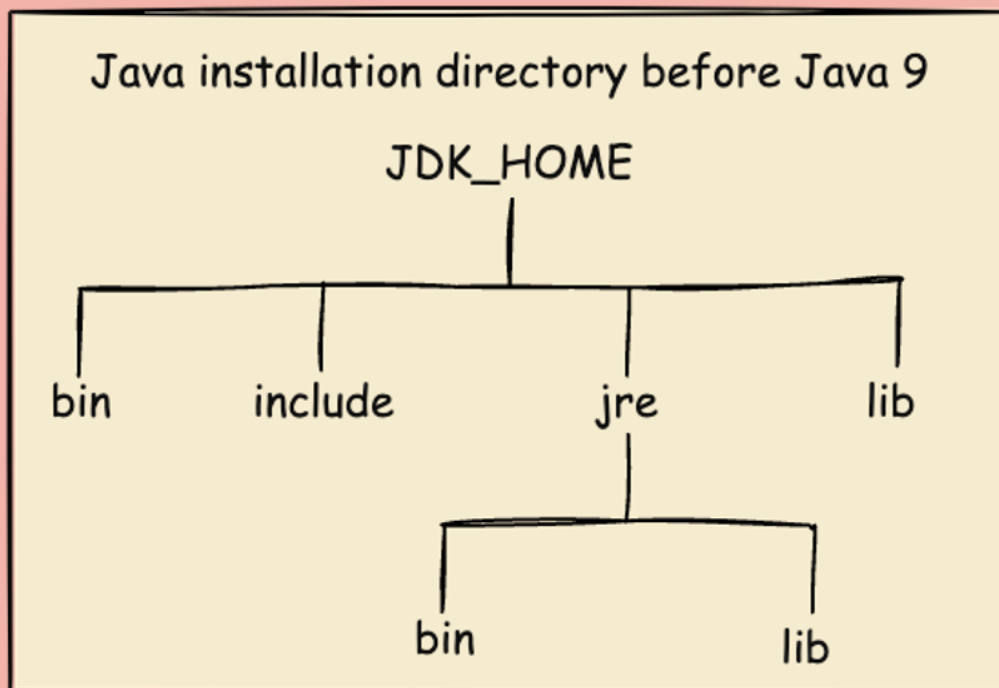
# What is JDK, JRE, JVM ?

Before JDK 9, the Java Development Kit (JDK) build system generated two distinct types of runtime images: a Java Runtime Environment (JRE) and a Java Development Kit (JDK). The JRE constituted a comprehensive implementation of the Java SE platform, while the JDK included both an embedded JRE and additional development tools and libraries. Users were given the flexibility to install either the standalone JRE or the JDK, which inherently included an embedded JRE.

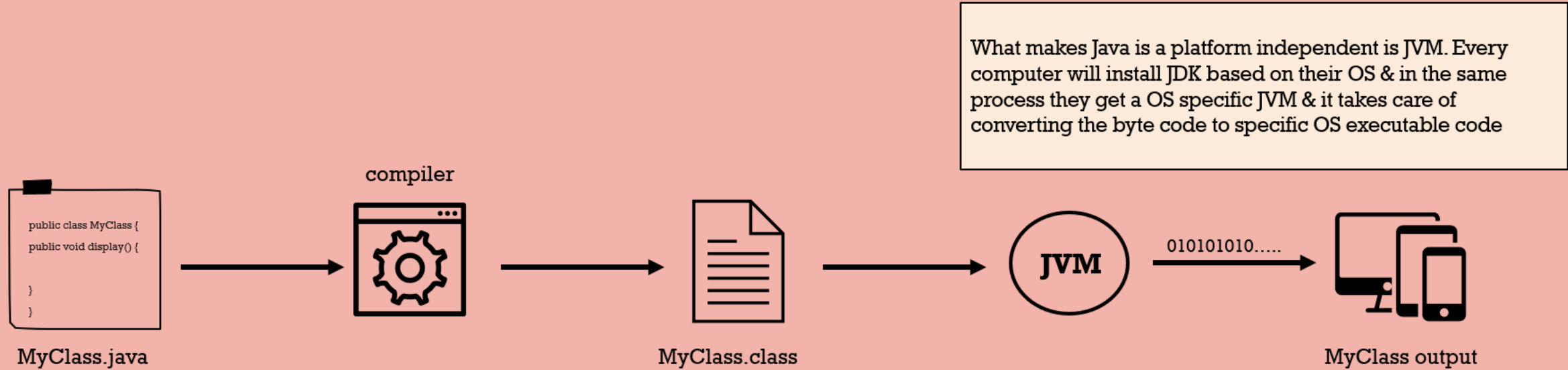
With the advent of Java 9, a substantial reorganization took place in the directory structure of the JDK. Notably, the traditional dichotomy between a JDK and a JRE was eliminated. This restructuring led to a flattened directory hierarchy for the JDK, thereby streamlining the organization of its components. The distinctions between a JDK with an embedded JRE and a standalone JRE were removed, simplifying the deployment and usage of Java runtime environments.

If needed, Organizations or Developers can build their own custom JRE using **jlink** option introduced in Java 9.

Below are the JDK folder structure differences from before Java 9 and after Java 9,



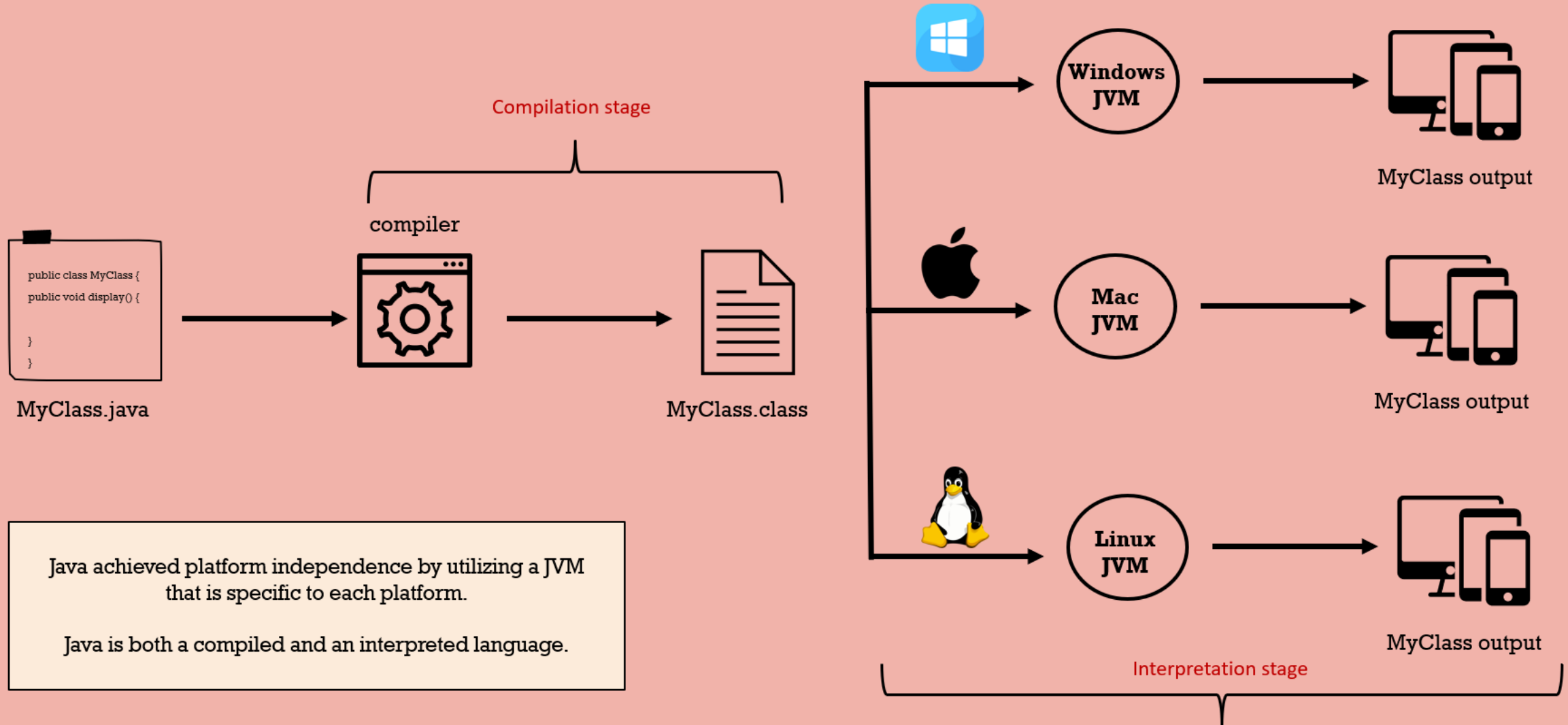
# The LifeCycle of a Java Program



Java is a platform independent language & it follows the principle “Write Once, Run Anywhere (WORA)”. It means the same compiled Java code developed by developer can be executed in different machines of various OS like windows, Linux, Mac etc.

Usually the developers responsibility is to write a valid source code, compile & test the program. So the java source code & compiled .class file will be same in all the machines.

# The LifeCycle of a Java Program





## What happens in compilation stage ?

Java source code undergoes a transformation into basic binary instructions. Nevertheless, while C or C++ source code is translated into native instructions tailored for a specific processor model, Java source code is compiled into a standardized format—comprising instructions intended for the virtual machine, commonly referred to as bytecode.



## What happens in interpretation stage ?

When you run a Java program, the JVM reads the bytecode instructions line by line and interprets them. The JVM's interpreter component translates each bytecode instruction into machine-specific code or actions, which are executed by the underlying hardware. This allows Java programs to be executed on different computer architectures without modification, as long as there is a JVM available for that platform.



## Dynamic or JIT Compilation

Java goes a step further to optimize performance. It incorporates a feature known as "dynamic" or "just-in-time" (JIT) compilation. With JIT compilation, the JVM can translate bytecode into native machine code on-the-fly as the program runs. JIT compilation is a significant performance enhancement. Instead of interpreting bytecode line by line, the JVM translates frequently executed portions of bytecode into native machine code. This native code can run much faster than interpreted bytecode because it's specific to the host system's architecture. This dynamic compilation happens at runtime, so only the parts of the code that are actively used get compiled to native code, reducing overhead.



Imagine you have a big book of instructions, but you can't read it directly. Instead, you have to interpret it line by line, telling yourself what to do next. This is what the Java Virtual Machine (JVM) does when it runs a Java program.

However, if you read the book many times, you can start to remember what to do without having to interpret the instructions every time. This is what the JIT compiler does. It compiles the Java bytecode into machine code that the CPU can understand directly.

The JIT compiler doesn't compile all of the bytecode at once. Instead, it compiles the methods that are used most often. This means that the first time you run a Java program, it may start slowly. But as you use the program more and more, the JIT compiler will compile more of the bytecode, and the program will run faster.



Here is a simple analogy that may help you understand JIT compilation:

Imagine you are a child, and your mother gives you a list of chores to do. You could read the list and do each chore one at a time. But if you are smart, you will start to group the chores together so that you can do them more efficiently. For example, you might group together all of the chores that require you to go to the kitchen, like setting the table and putting away the dishes.

The JIT compiler does something similar. It groups together the Java bytecode that is used most often and compiles it into machine code. This means that the CPU can execute the code more efficiently.

JIT compilation is one of the reasons why Java programs can run so fast, even though they are not compiled into machine code before they are executed.



## Is Java language platform dependent or independent ?

Java is a platform-independent language, meaning that code written in Java on a Windows operating system can be executed on a Mac or Linux operating system, and the resulting output will be the same.



## Is Java software platform dependent or independent ?

Java software is platform dependent due to the platform-dependent nature of the JVM, which is separately available for each operating system.



## Is JVM installed automatically with OS ?

No, JVM is not installed automatically with the operating system. It needs to be separately installed on the system in order to run Java programs.



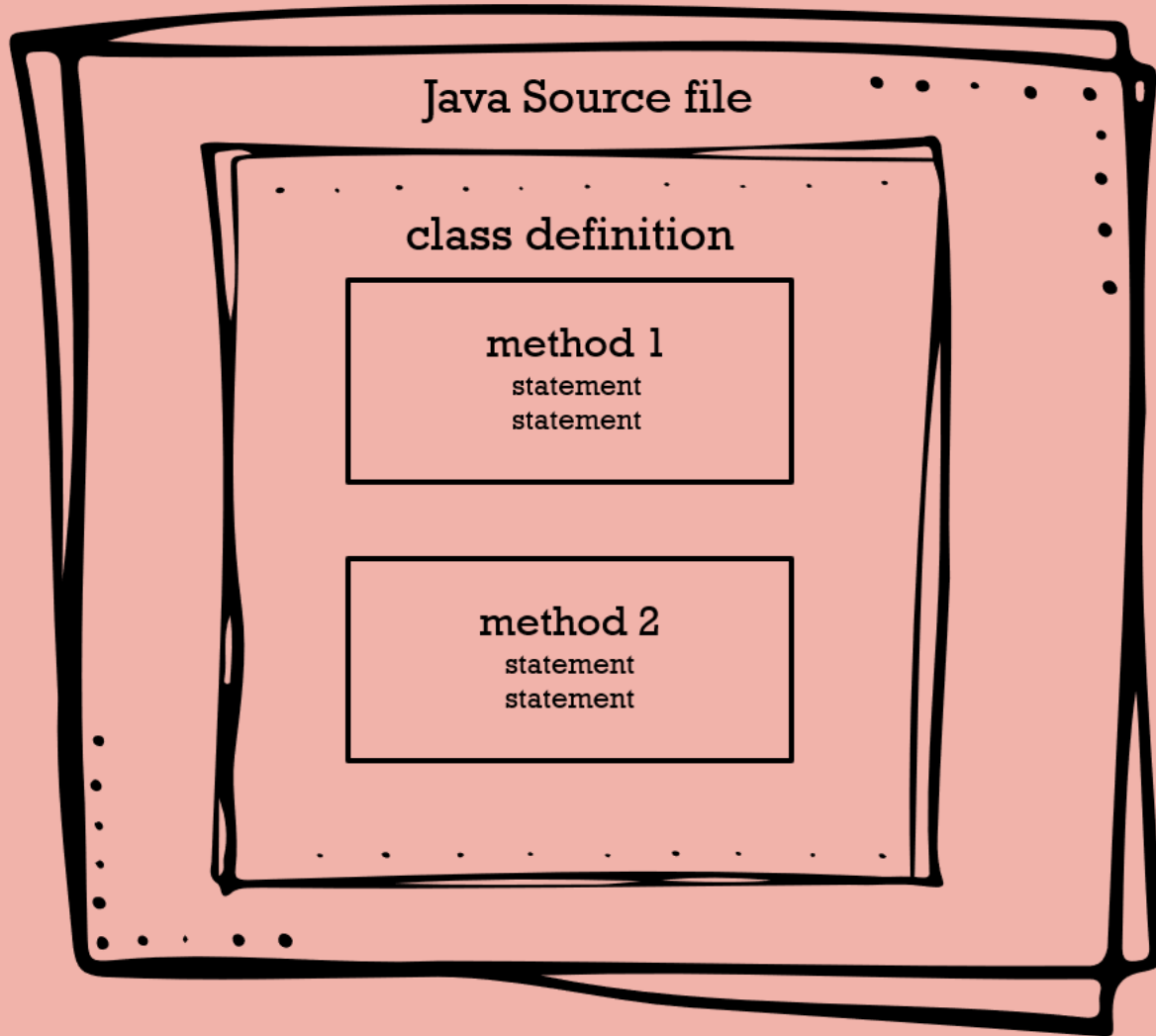
## What are important components of Java installation ?

JDK (Java Development Kit) - It is a software development kit that provides all the tools necessary to develop, compile, and run Java programs. It includes the Java compiler, Java runtime environment (JRE), and other development tools such as debuggers and documentation.

JVM (Java Virtual Machine) - It is an abstract machine that provides a runtime environment in which Java programs can run. JVM interprets the compiled Java bytecode and provides a layer of abstraction between the code and the underlying operating system. JVM is platform-dependent and must be installed separately on each operating system.

JRE (Java Runtime Environment) - It is a subset of the JDK that includes only the JVM and other components required to run Java applications. By default, JRE is no more available as a separate option from JDK 9.





✦ In a java source file, define a class.

✦ Inside a class, define methods.

✦ Inside a method, write java statements.



## Java program code structure

### What goes in a source file?

A source code file (with the .java extension) typically holds one class definition. The class represents a piece of your program. The class must go within a pair of curly braces.

```
public class Vehicle  
{  
  
}
```

→ **Vehicle.java  
class**

### What goes in a class?

A class has one or more methods. In the Vehicle class, the move method will hold instructions on how the Vehicle should move. Your methods must be declared inside a class & should also have pair of curly braces.

```
public class Vehicle  
{  
    void move() {  
  
    }  
}
```

→ **move method  
inside a class**

### What goes in a method?

Within the curly braces of a method, we can write instructions on how that method should be performed. Method code is basically a set of java statements

```
public class Vehicle  
{  
    void move() {  
        statement1;  
        statement2;  
    }  
}
```

→ **statements**

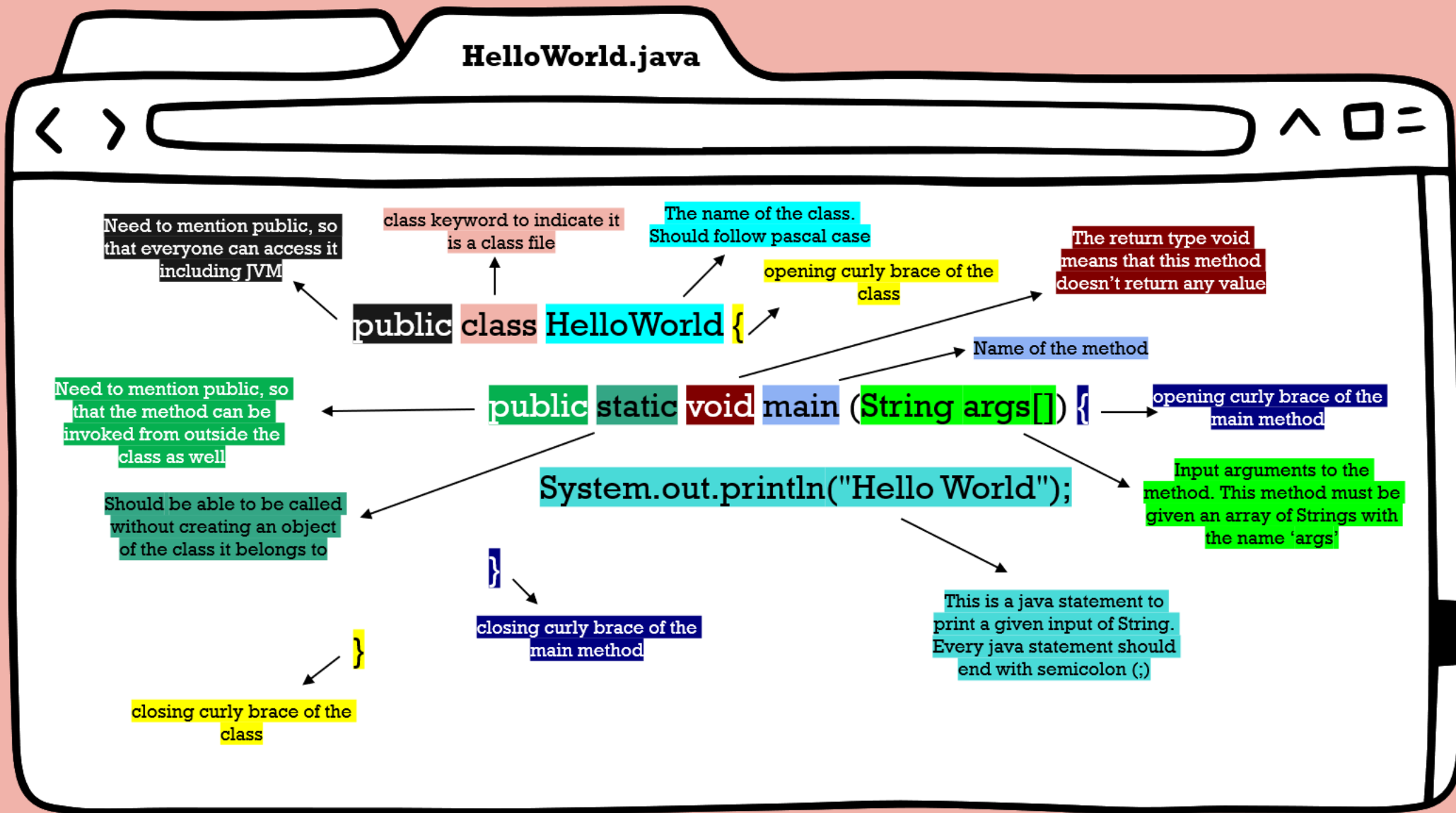
### How JVM knows where to start ?

Upon startup, the JVM scans for the class specified in the command line. Following this, it seeks a particular method that is written in a prescribed manner, as shown below. The code enclosed within the curly braces { } of the main method is then executed by the JVM. It is necessary for every Java application to contain at least one class with a main method (not one main per class; just one main per application)

```
public static void main (String[] args)  
{  
  
    // your code  
}
```

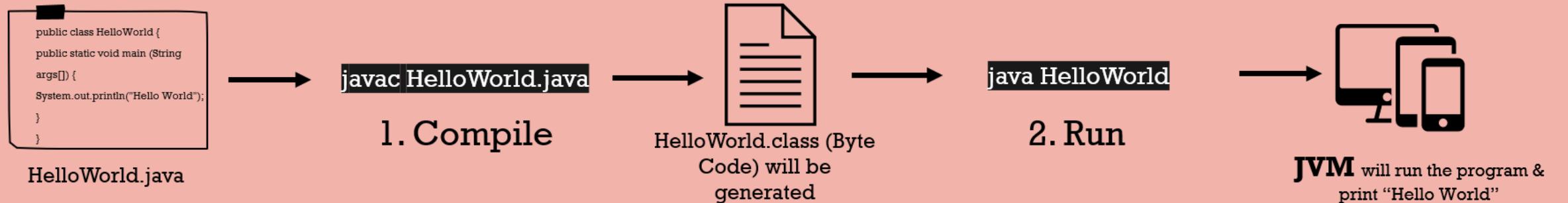
# Let's write first Java class

eazy  
bytes



# Commands to compile & run Java program

Compiling & running a java program means telling the Java Virtual Machine (JVM) to “Load the class, then start executing its main() method. Keep running ’til all the code in main is finished.” For the same we need to run few commands with the help of a window/mas/linux OS terminal. Offcourse they are more automatic/advance ways using the products like IDE which we are going to learn in coming sections. For now let’s try to learn the basics.



**javac:** This reads a .java file, compiles it, and creates .class file post validating the java syntaxes

```
Terminal  Shell  Edit  View  Window  Help  
eazybytes@Eazys-MBP Java % javac HelloWorld.java  
eazybytes@Eazys-MBP Java % java HelloWorld  
Hello World  
eazybytes@Eazys-MBP Java %
```

**java:** This executes a .class file with the help of JVM

IDEs (Integrated Development Environments) are software applications that provide developers with a comprehensive set of tools to write, test, and debug code in one place. Here are some reasons why IDEs like IntelliJ are essential for developers:

## **Increased productivity**

IDEs offer developers a range of features such as code completion, refactoring, debugging, and testing, which save a lot of time and effort. With an IDE, developers can write, test, and debug their code much faster, leading to increased productivity.

## **Integration with tools and frameworks**

IDEs provide integration with popular tools and frameworks, such as Git, Maven, and Spring, making it easier for developers to work with these technologies.

## **Better code quality**

IDEs provide developers with features like syntax highlighting, code formatting, and error highlighting, which help identify and fix errors and inconsistencies in the code. This results in code that is easier to read and maintain, leading to better overall code quality.

## **Collaboration**

IDEs provide collaboration features that allow multiple developers to work on the same codebase simultaneously, making it easier to manage code changes and track progress.



**Various famous IDEs  
available for Developers**



IntelliJ IDEA



Eclipse



Visual Studio code



NetBeans

# Why IntelliJ IDEA ?



1

Undoubtedly, IntelliJ IDEA is the preferred IDE for software developers. Its design prioritizes efficiency and intelligence, providing a seamless workflow experience for designing, implementing, building, deploying, testing, debugging & refactoring. With a vast array of features and a wide selection of plugins that can be integrated into the editor, IntelliJ IDEA offers developers a comprehensive & powerful toolset.

2

One of the things that stands out to me about IntelliJ IDEA is its ability to provide helpful and intelligent suggestions. These suggestions go beyond just technology integrations and productivity shortcuts, as they offer guidance on best practices for naming conventions and instructions on migrating to new Java features. I often highlight this feature in conversations with students because of its simplicity and usefulness.

3

There are two versions of IntelliJ IDEA available: a **paid version** and a **free community edition**. While the paid version is widely regarded as the top Java IDE, the community edition is also highly capable, and that is the version we will be using in this course. To use an analogy, the community edition is akin to a Netflix HD subscription, whereas the paid version would be more like a Netflix 4K subscription.

4

To install IntelliJ IDEA, visit the following URL and download the appropriate installer for your operating system. Once downloaded, run the installer and accept the default settings.

<https://www.jetbrains.com/idea/>