CORE JAVA

By : Kalpesh Chauhan

INTRODUCTION

WHAT IS JAVA

Java is a **programming language** and a **platform**.

Java is a high level, robust, object-oriented and secure programming language.

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 is Developed by **James Gosling and team.**

Java is just a small, simple, safe, object-oriented, interpreted or dynamically optimized, byte-coded, architecture-neutral, garbage collected, multithreaded programming language with a strongly typed exception-handling mechanism for writing distributed, dynamically extensible programs.

- Sun cofounder, Bill Joy

APPLICATION

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

TYPES OF JAVA APPLICATIONS

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

JAVA PLATFORMS / EDITIONS

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform which is mainly used to develop web and enterprise applications. It is built on the top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform which is mainly used to develop mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a light-weight user interface API.

HISTORY OF JAVA

HISTORY OF JAVA

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as **Green Team**), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted and Dynamic".

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given the significant points that describe the history of Java.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team.**

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling, and file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

7) **Why had they chosen java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in(January 23, 1996).

JAVA VERSION HISTORY

Many java versions have been released till now. The current stable release of Java is Java SE 10.

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)

- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- Java SE 8 (18th March 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th March 2018)

FEATURES OF JAVA

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java *buzzwords*.

A list of most important features of Java language is given below.

Simple

Object-Oriented

Portable

Platform independent

Secured

Robust

Architecture neutral

Interpreted

High Performance

Multithreaded

Distributed

Dynamic

FEATURES EXPLAINED

SIMPLE

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

Java syntax is based on C++ (so easier for programmers to learn it after C++).

Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.

There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

OBJECT-ORIENTED

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

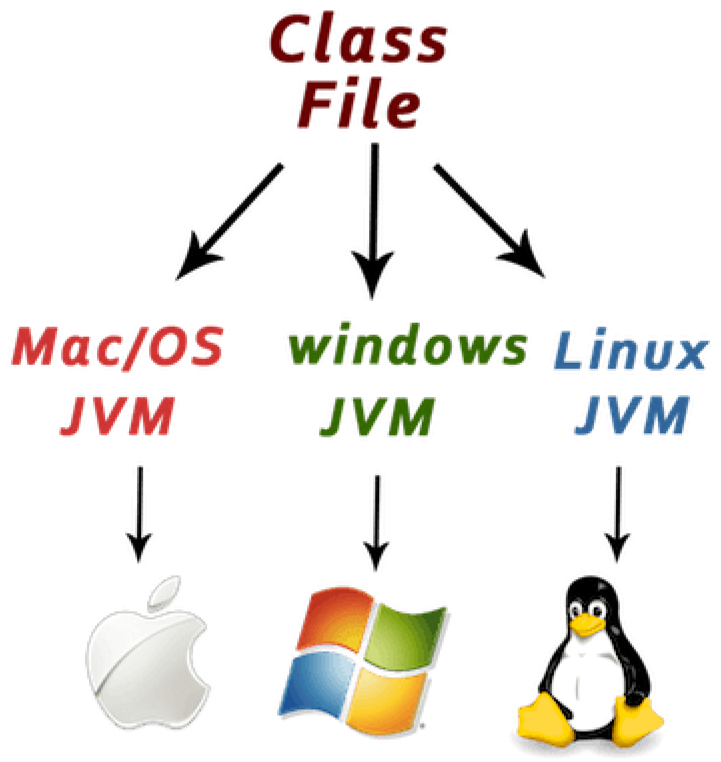
- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

PLATFORM INDEPENDENT

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:


Class File
Mac/OS JVM
windows JVM
Linux JVM

- Runtime Environment
- API(Application Programming Interface)

Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

SECURED

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

No explicit pointer

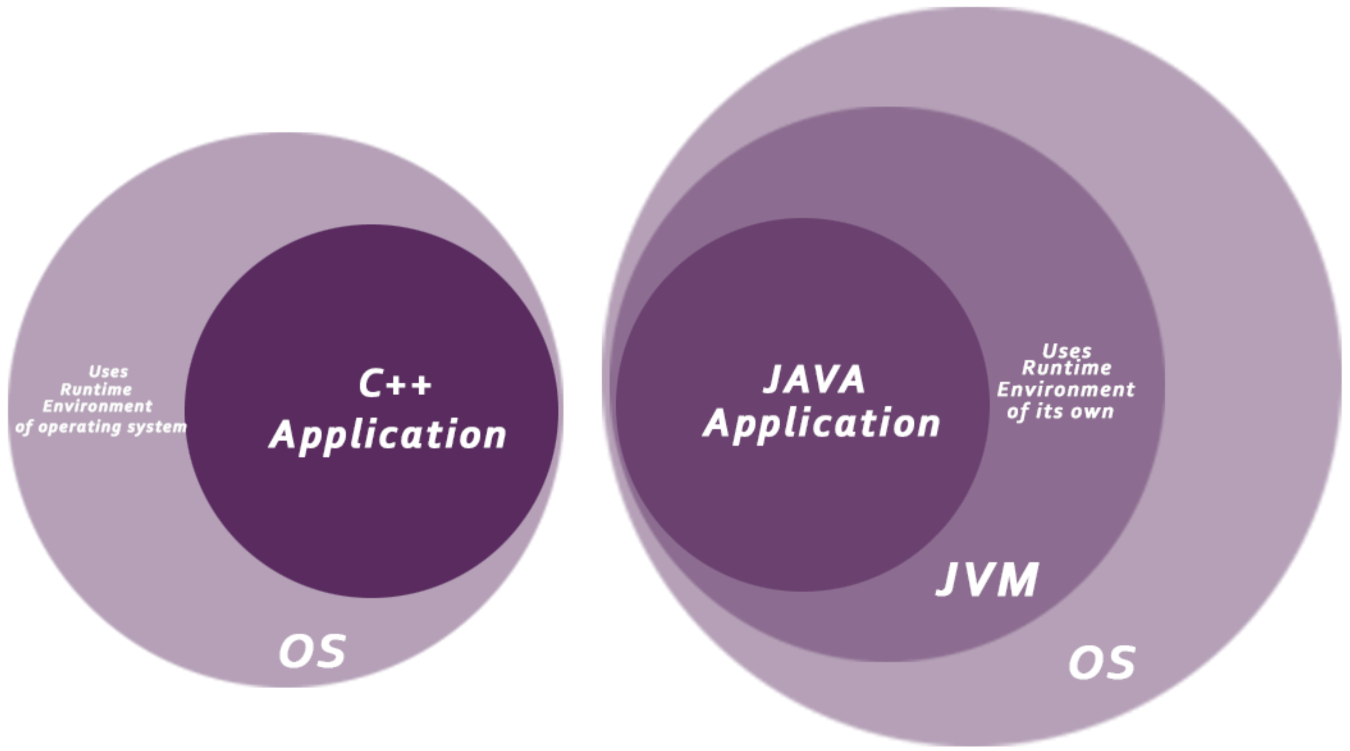
Java Programs run inside a virtual machine sandbox

Classloader: Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier: It checks the code fragments for illegal code that can violate access right to objects.

Security Manager: It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.


Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

ROBUST

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

ARCHITECTURE-NEUTRAL

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

PORTABLE

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

HIGH-PERFORMANCE

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

DISTRIBUTED

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

MULTI-THREADED

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

DYNAMIC

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

FIRST JAVA PROGRAM

HELLO WORLD EXAMPLE

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains the main method. Let's understand the requirement first.

THE REQUIREMENT FOR JAVA HELLO WORLD EXAMPLE

For executing any java program, you need to

- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory.
- Create the java program
- Compile and run the java program

CREATING HELLO WORLD EXAMPLE

Let's create the hello java program:

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

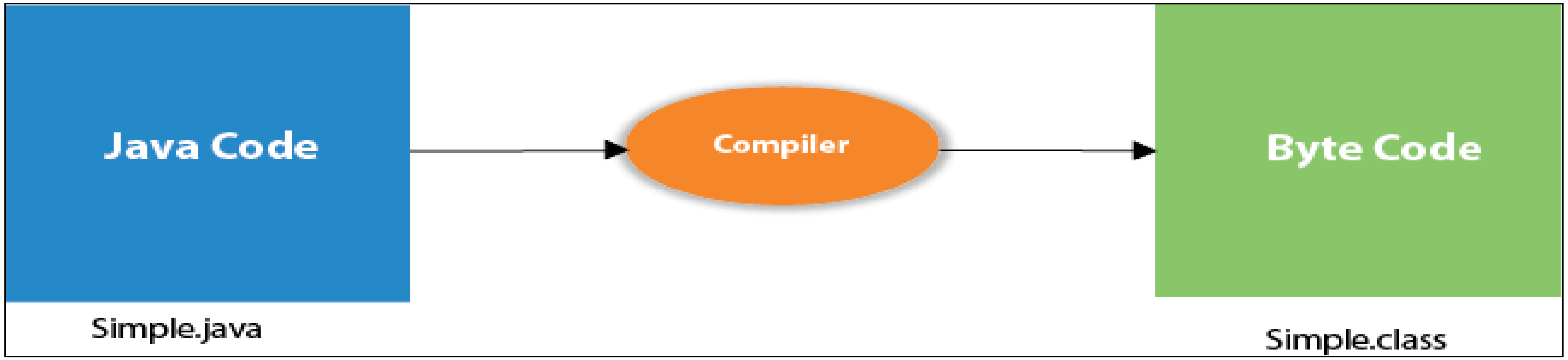
Save above file as Simple.java and run following commands.

COMPILE AND RUN

To compile	javac Example.java
To run	java Example
Output	Hello Java

COMPILATION FLOW:

When we compile Java program using javac tool, java compiler converts the source code into byte code.



PARAMETERS USED IN FIRST JAVA PROGRAM

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

class keyword is used to declare a class in java.

public keyword is an access modifier which represents visibility. It means it is visible to all.

static is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.

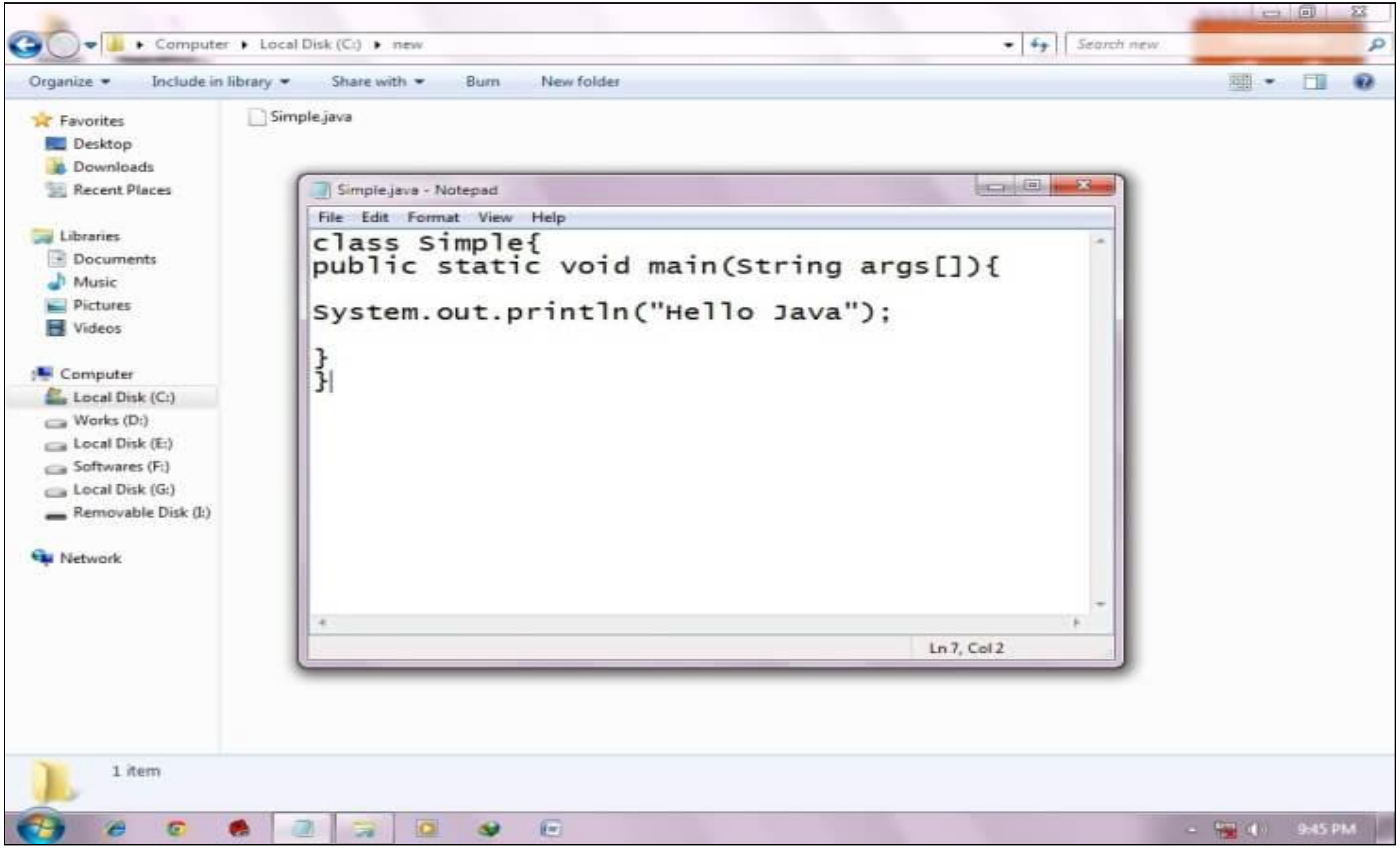
void is the return type of the method. It means it doesn't return any value.

main represents the starting point of the program.

String[] args is used for command line argument. We will learn it later.

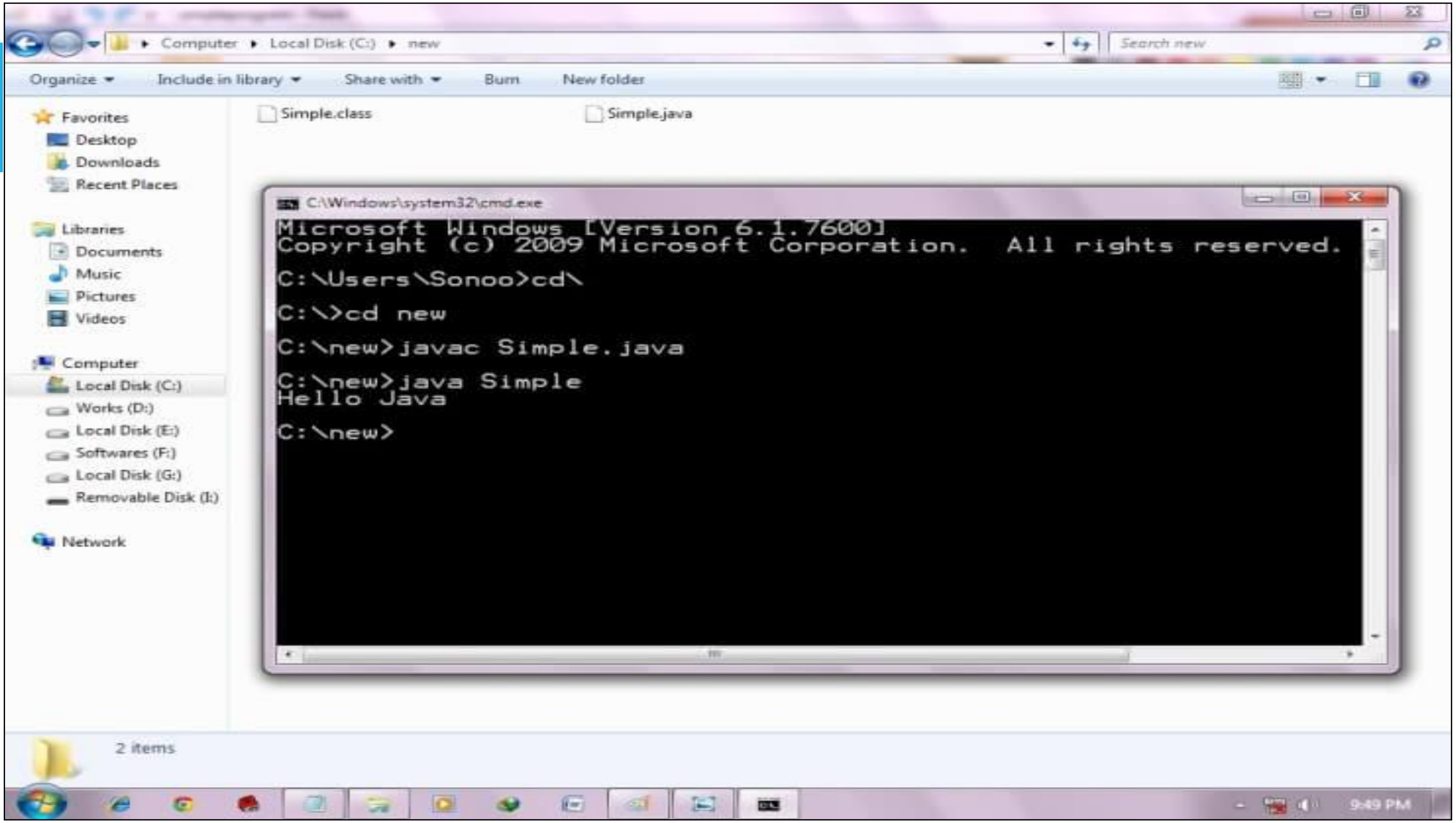
System.out.println() is used print statement. We will learn about the internal working of System.out.println statement later.

To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> notepad** and write a simple program as displayed below:



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

System.out.println("Hello Java");

}
}
```

As displayed in the above diagram, write the simple program of java in notepad and saved it as Simple.java. To compile and run this program, you need to open the command prompt by **start menu -> All Programs -> Accessories -> command prompt**.

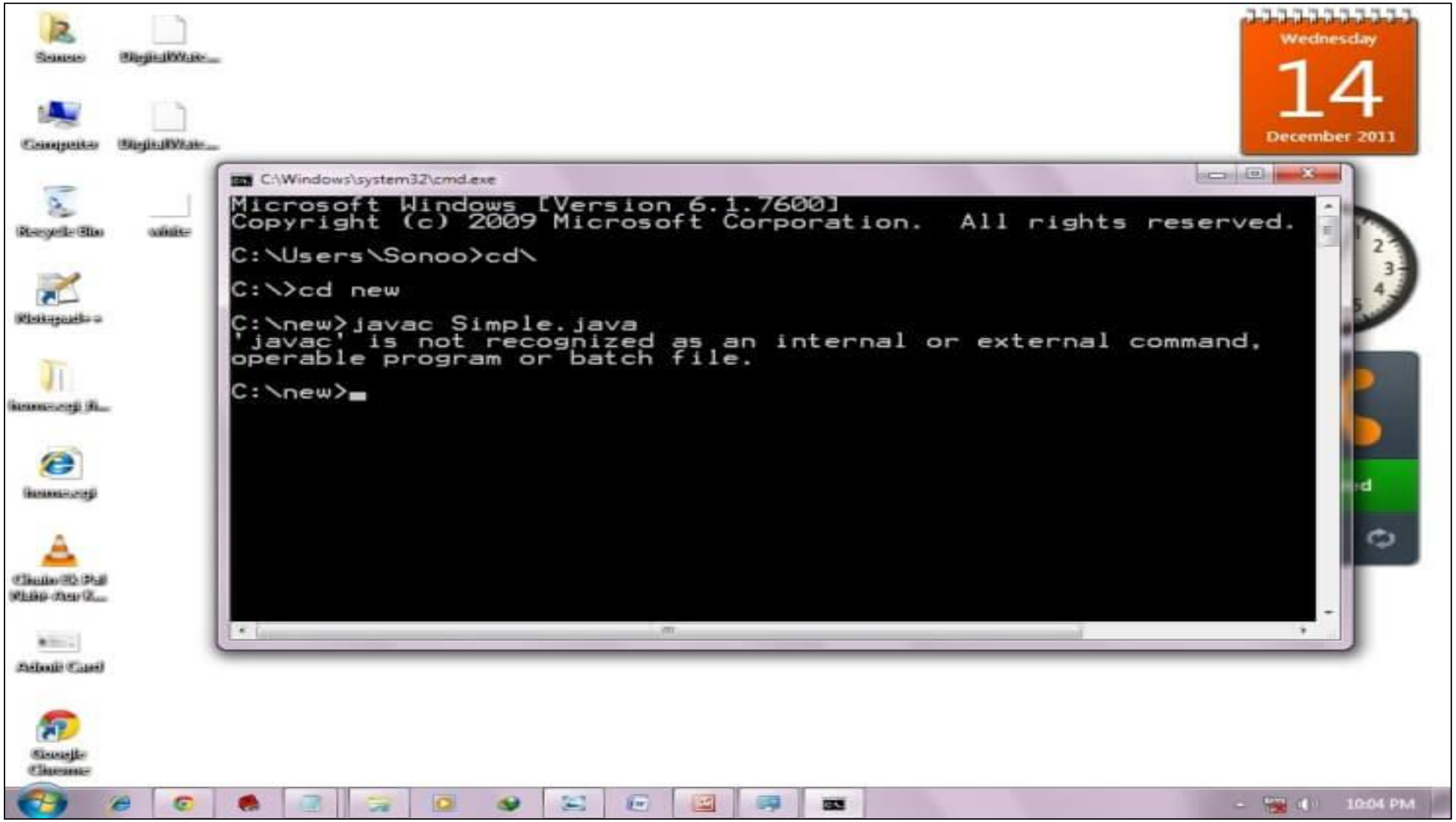


```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sonoo>cd\

C:\>cd new

C:\new>javac Simple.java

C:\new>java Simple
Hello Java

C:\new>
```

JAVAC NOT RECOGNIZED COMMAND ERROR

Resolving an error "javac is not recognized as an internal or external command"?

If there occurs a problem like displayed in the below figure, you need to set path. Since DOS doesn't know javac or java, we need to set path. The path is not required in such a case if you save your program inside the JDK/bin directory. However, it is an excellent approach to set the path



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sonoo>cd\

C:\>cd new

C:\new>javac Simple.java
'javac' is not recognized as an internal or external command,
operable program or batch file.

C:\new>
```

CLASSPATH

How to set path in Java

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

There are two ways to set the path in Java:

- Temporary
- Permanent

How to set the Temporary Path of JDK in Windows

To set the temporary path of JDK, you need to follow the following steps:

- Open the command prompt
- Copy the path of the JDK/bin directory
- Write in command prompt: set path=copied_path

For Example:

```
set path=C:\Program Files\Java\jdk1.6.0_23\bin
```

How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> press ok -> press ok -> press ok

WHAT IS DIFFERENCE BETWEEN JVM, JDK AND JRE

Difference between JDK and JRE and JVM is one of the popular interview questions. You might also be asked to explain JDK vs JRE vs JVM.

JDK is for development purpose whereas **JRE is** for running the java programs. **JDK and JRE** both contains **JVM** so that we can run our java program. **JVM is** the heart of java programming language and provides platform independence.

THE JDK

Java Development Kit is the core component of Java Environment and provides all the tools, executables and binaries required to compile, debug and execute a Java Program. JDK is a platform-specific software and that’s why we have separate installers for Windows, Mac, and Unix systems. We can say that JDK is the superset of JRE since it contains JRE with Java compiler, debugger, and core classes. The current version of JDK is 11 also known as Java 11.

THE JVM

JVM is the heart of Java programming language. When we run a program, JVM is responsible for converting Byte code to the machine specific code. JVM is also platform dependent and provides core java functions like memory management, garbage collection, security etc. JVM is customizable and we can use java options to customize it, for example allocating minimum and maximum memory to JVM. JVM is called *virtual* because it provides an interface that does not depend on the underlying operating system and machine hardware. This independence from hardware and the operating system is what makes java program write-once-run-anywhere.

THE JRE

JRE is the implementation of JVM, it provides a platform to execute java programs. JRE consists of JVM and java binaries and other classes to execute any program successfully. JRE doesn’t contain any development tools like java compiler, debugger etc. If you want to execute any java program, you should have JRE installed but we don’t need JDK for running any java program.

JDK VS JRE VS JVM

Let’s look at some of the important difference between JDK, JRE, and JVM.

1. JDK is for development purpose whereas JRE is for running the java programs.
2. JDK and JRE both contains JVM so that we can run our java program.
3. JVM is the heart of java programming language and provides platform independence.

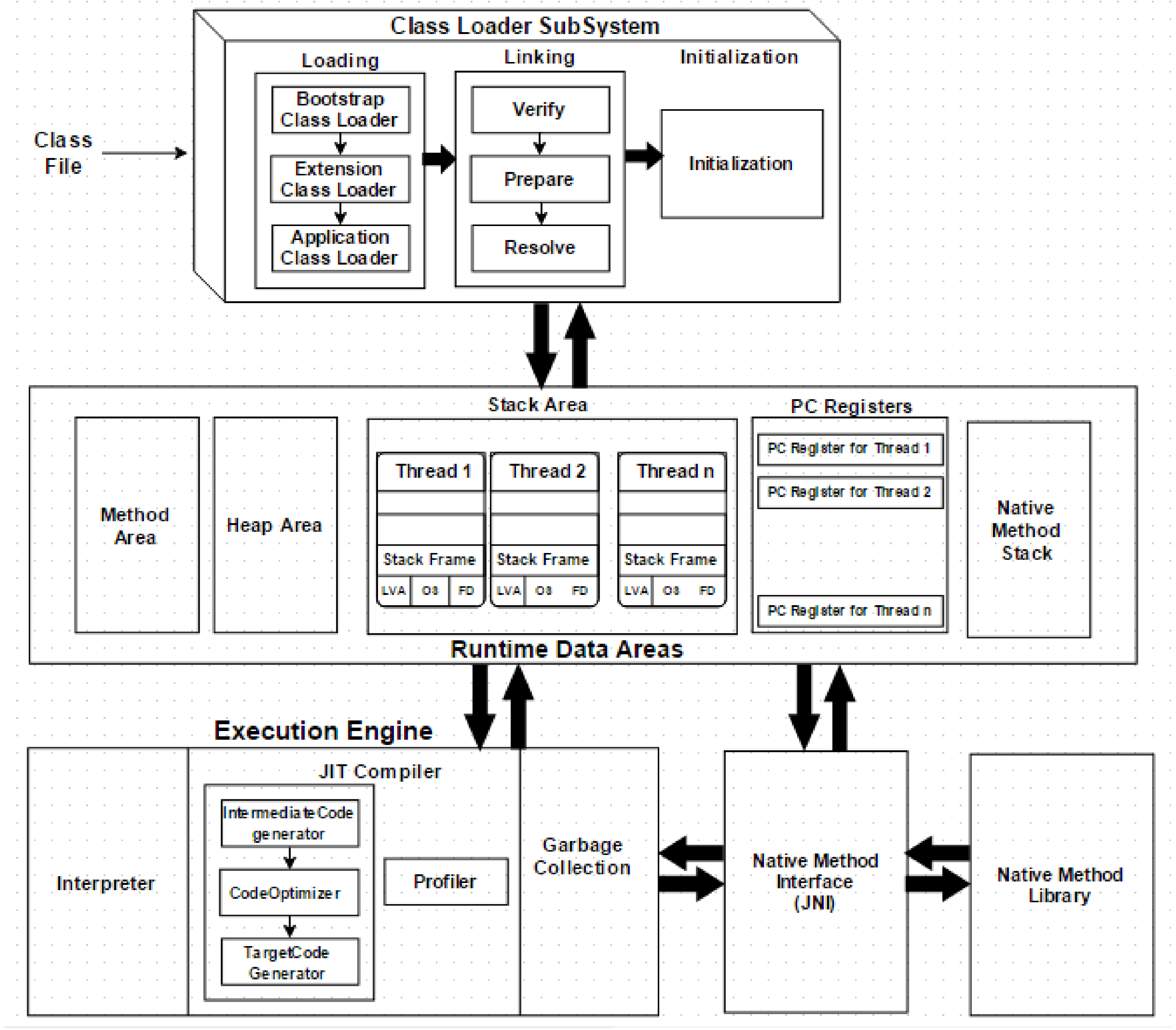
JUST-IN-TIME COMPILER (JIT)

Sometimes we heard this term and being it a part of JVM it confuses us. JIT is part of JVM that optimizes byte code to machine specific language compilation by compiling similar byte codes at the same time, hence reducing overall time taken for the compilation of byte code to machine specific language.

JVM ARCHITECTURE

Every Java developer knows that bytecode will be executed by **JRE** (Java Runtime Environment). But many don't know the fact that JRE is the implementation of **Java Virtual Machine** (JVM), which analyzes the bytecode, interprets the code, and executes it. It is very important as a developer that we should know the Architecture of the JVM, as it enables us to write code more efficiently. In this article, we will learn more deeply about the JVM architecture in Java and the different components of the JVM.

A **Virtual Machine** is a software implementation of a physical machine. Java was developed with the concept of **WORA** (***Write Once Run Anywhere***), which runs on a **VM**. The **compiler** compiles the Java file into a Java **.class** file, then that .class file is input into the JVM, which Loads and executes the class file. Below is a diagram of the Architecture of the JVM.


Class File
Class Loader SubSystem
Loading
Bootstrap Class Loader
Extension Class Loader
Application Class Loader
Linking
Verify
Prepare
Resolve
Initialization
Initialization
Stack Area
PC Registers
Method Area
Heap Area
Thread 1
Thread 2
Thread n
Stack Frame
LVA
OS
FD
PC Register for Thread 1
PC Register for Thread 2
PC Register for Thread n
Native Method Stack
Runtime Data Areas
Execution Engine
Interpreter
JIT Compiler
Intermediate Code generator
CodeOptimizer
TargetCode Generator
Profiler
Garbage Collection
Native Method Interface (JNI)
Native Method Library

HOW DOES THE JVM WORK?

As shown in the above architecture diagram, the JVM is divided into three main subsystems:

1. Class Loader Subsystem
2. Runtime Data Area
3. Execution Engine

CLASS LOADER SUBSYSTEM

Java's dynamic class loading functionality is handled by the class loader subsystem. It loads, links. and initializes the class file when it refers to a class for the first time at runtime, not compile time.

LOADING

Classes will be loaded by this component. BootStrap class Loader, Extension class Loader, and Application class Loader are the three class loader which will help in achieving it.

Boot Strap ClassLoader – Responsible for loading classes from the bootstrap classpath, nothing but **rt.jar.** Highest priority will be given to this loader.

Extension ClassLoader – Responsible for loading classes which are inside the ext folder **(jre\lib).**

Application ClassLoader –Responsible for loading Application Level Classpath, path mentioned Environment Variable etc.

The above Class Loaders will follow Delegation Hierarchy Algorithm while loading the class files.

LINKING

Verify – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error.

Prepare – For all static variables memory will be allocated and assigned with default values.

Resolve – All symbolic memory references are replaced with the original references from Method Area.

INITIALIZATION

This is the final phase of Class Loading, here all static variables will be assigned with the original values, and the static block will be executed.

RUNTIME DATA AREA

The Runtime Data Area is divided into 5 major components:

Method Area – All the class level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.

Heap Area – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.

Stack Area – For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called as Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three subentities:

1. **Local Variable Array** – Related to the method how many local variables are involved and the corresponding values will be stored here.
2. **Operand stack** – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.
3. **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data.

PC Registers – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.

Native Method stacks – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

EXECUTION ENGINE

The bytecode which is assigned to the **Runtime Data Area** will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.

Interpreter – The interpreter interprets the bytecode faster, but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.

JIT Compiler – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.

1. **Intermediate Code generator** – Produces intermediate code
2. **Code Optimizer** – Responsible for optimizing the intermediate code generated above
3. **Target Code Generator** – Responsible for Generating Machine Code or Native Code
4. **Profiler** – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.

Garbage Collector : Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.

Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.

Native Method Libraries: This is a collection of the Native Libraries which is required for the Execution Engine.

CONTINUE IN NEXT UNIT.....