Dr. Ambedkar Institute of Technology, Bengaluru

Department of Computer Science & Engg.

UG Programme

Subject Name: JAVA Programming
Subject Code: 18CS52

PRESENTATION BY

Dr. SMITHA SHEKAR B

ASSOCIATE PROFESSOR

DEPT. OF C.S.E

DR. AMBEDKAR INSTITUTE OF TECHNOLOGY

BANGALORE-560056

The History and Evolution of Java

Chapter 1

Introduction to Java

- Java and Java applications
- Java Development Kit(JDK)
- Java is interpreted, Byte code, JVM
- Object-Oriented Programming
- Simple Java programs
- Data types and other tokens: Boolean variables, int ,long, char operators, arrays, whitespaces, literals, assigning values

Continues….

- Creating and Destroying Objects
- Access Specifies, Operators and Expressions : Arithmetic Operators, Bitwise operators, Relational operators, The assignment operator, The ? Operator, Operator precedence
- Logical expression
- Type casting
- Strings
- Control Statements
- Selection Statements
- Iteration Statements
- Jump Statements

Classes, Inheritance, Exception, Applets

- Classes
- Classes in Java
- Super classes
- Constructors
- Creating Instances of class
- Inner classes
- Inheritance: Simple, Multiple and Multilevel Inheritance
- Overriding, Overloading
- Exception Handling : Exception handling in java,
- The Applet Class : Two types of Applets, Applet basics, Applet Architecture, An Appletskeleton, Simple Applet display methods, Requesting repainting, Using the Status Window, The HTML APPLET tag, Passing parameters to Applets, getDocumentbase() and getCodebase(), AppletContext and showDocument(), The AudioClip Interface, The AppletStub Interface, Output to the Console

Multi Threaded Programming , Event Handling

- Multi Threaded Programming
- What are threads?
- How to make the classes threadable
- Extending threads
- Implementing runnable
- Synchronization
- Changing state of the thread
  - Producer-Consumer problems
- Event Handling : Two Event handling Mechanisms
- The delegation event model
- Event classes
- Sources of Events
- Event listener interfaces
- Using the delegation event model
- Adapter classes Inner classes

Swings

- Swings: The origins of Swing, Two key Swing features, Components and Containers, The Swing Packages, A simple Swing Application, Create a Swing Applet
- Jlabel and ImageIcone
- JTextField
- The Swing Buttons
- Jtabbedpane
- JScrollPane
- Jlist
- JComboBox
- JTable

Java2 Enterprise Edition Overview, Database Access

▪Overview of J2EE and J2SE

▪The concept of JDBC

▪JDBC Driver Types

▪JDBC Packages

▪A Brief Overview of the JDBC process

▪Database Connection
▪Associating the JDBC /ODBC Bridge with the Database
▪Statement Objects
▪ResultSet

▪Transaction Processing
▪Metadata
▪Data Types
▪Exceptions

Servlets

- Background
- The life Cycle of a Servlet
- Using Tomcat for Servlet Development
- A simple Servlet
- The Servlet API
- The javax.servlet Package
- Reading Servlet Parameter
- The javax.servlet.http Package
- Handling HTTP Requests and Respones
- Using Cookies
- Session Tracking

JSP, RMI

- Java Server Pages(JSP)
- JSP, JSP Tags

- Tomcat
- Request String
- User Sessions
- Cookies
- Session Objects

- Java Remote Method Invocation

- Remote Method Invocation Concept : Server side and Client side

Enterprise Java Beans

- Enterprise Java Beans
- Deployment Descriptors
- Session Java Bean
- Request String
- Entity Java Bean
- Message-Driven Bean
- The JAR file

What Is Java?

- History
- Characteristics of Java
- JDK Versions
- JDK Editions
- Java IDE Tools

History

- James Gosling and Sun Microsystems
- Oak
- Java, May 20, 1995, Sun World
- HotJava
    - The first Java-enabled Web browser
- JDK Evolutions
- J2SE, J2ME, and J2EE

There were five primary goals in the creation of the Java language.

- 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.

Characteristics of Java

- Java is simple
- Java is object-oriented
- Java is distributed
- Java is interpreted
- Java is robust
- Java is secure

- Java is architecture-neutral
- Java is portable
- Java’s performance
- Java is multithreaded
- Java is dynamic

JDK Versions

- As of March 2019, Java 8 is supported; and both Java 8 and 11 as Long Term Support (LTS) versions. Major release versions of Java, along with their release dates:
- JDK 1.0 (January 23, 1996)
- JDK 1.1 (February 19, 1997)
- J2SE 1.2 (December 8, 1998)
- J2SE 1.3 (May 8, 2000)
- J2SE 1.4 (February 6, 2002)
- J2SE 5.0 (September 30, 2004)
- Java SE 6 (December 11, 2006)
- Java SE 7 (July 28, 2011)
- Java SE 8 (March 18, 2014)
- Java SE 9 (September 21, 2017)
- Java SE 10 (March 20, 2018)
- Java SE 11 (September 25, 2018)
- Java SE 12 (March 19, 2019)

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets and Java Server Pages.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

This book uses J2SE to introduce Java programming.

Java IDE Tools

- Forte by Sun MicroSystems
- Borland JBuilder
- Microsoft Visual J++
- WebGain Café
- IBM Visual Age for Java
- Eclipse

- Java is a blend of the best elements of its rich heritage combined with the innovative concepts required by its unique mission.

- Java is first and foremost a programming language.

Java applications are called WORA (Write Once Run Anywhere).

This means a programmer can develop Java code on one system and can expect it to run on any other Java enabled system without any adjustment.
This is all possible because of JVM.

Java’s Lineage

- Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages.

- From C, Java derives its syntax. Many of Java’s object oriented features were influenced by C++.

- In fact, several of Java’s defining characteristics come from—or are responses to—its predecessors.

Java History

- Computer language innovation and development occurs for two fundamental reasons:

    1) to adapt to changing environments and uses

    2) to implement improvements in the art of programming

- The development of Java was driven by both in equal measures.
- Many Java features are inherited from the earlier languages:

    B C C++ Java

Before Java: C

- Designed by Dennis Ritchie in 1970s.
- Before C: BASIC, COBOL, FORTRAN, PASCAL
- C- structured, efficient, high-level language that could replace assembly code when creating systems programs.
- Designed, implemented and tested by programmers.

Before Java: C++

- Designed by Bjarne Stroustrup in 1979.
- Response to the increased complexity of programs and respective improvements in the programming paradigms and methods:
    1) assembler languages
    2) high-level languages
    3) structured programming
    4) object-oriented programming (OOP)
- OOP – methodology that helps organize complex programs through the use of inheritance, encapsulation and polymorphism.
- C++ extends C by adding object-oriented features.

Java: History

- In 1990, Sun Microsystems started a project called Green.
- Objective: to develop software for consumer electronics.
- Project was assigned to James Gosling, a veteran of classic network software design. Others included Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan.
- The team started writing programs in C++ for embedding into
  - toasters
  - washing machines
  - VCR’s
- Aim was to make these appliances more “intelligent”.

Java: History (contd.)

- Robustness is essential. Users have come to expect that Windows may crash or that a program running under Windows may crash. ("This program has performed an illegal operation and will be shut down")
- However, users do not expect toasters to crash, or washing machines to crash.
- A design for consumer electronics has to be *robust*.
- Replacing pointers by references, and automating memory management was the proposed solution.

Java: History (contd.)

- Team built a new programming language called Oak, which avoided potentially dangerous constructs in C++, such as pointers, pointer arithmetic, operator overloading etc.

- So, the software and programming language had to be *architecture neutral*.

Java: History (contd)

- It was soon realized that these design goals of consumer electronics perfectly suited an ideal programming language for the Internet and WWW, which should be:
  - ❖ object-oriented (& support GUI)
  - ❖ – robust
  - ❖ – architecture neutral

Java History

- Designed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems in 1991.
- The original motivation is not Internet: platform-independent software embedded in consumer electronics devices.
- Java as an “Internet version of C++”? No.
- Java was not designed to replace C++, but to solve a different set of problems.

How Java Changed the Internet

- The Internet helped Java to the forefront of programming, and Java, in turn, had a profound effect on the Internet.

- Simplifying web programming in general

- Java innovated a new type of networked program called the applet that changed the way the online world thought about content.

- Java also addressed some of the thorniest issues associated with the Internet: portability and security.

Java Applets

- An *applet* is a special kind of Java program
    - Designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser.
- An applet is downloaded on demand, without further interaction with the user.
    - If the user clicks a link that contains an applet, the applet will be automatically downloaded and run in the browser.
- In essence, the applet allows some functionality to be moved from the server to the client.

- By contrast, the applet is a dynamic, self-executing program.
  - Such a program is an active agent on the client computer, yet it is initiated by the server.

Java's Magic: The Bytecode

The key that allows Java to solve both the security and the portability problems

- Is that the output of a Java compiler is not executable code. Rather, it is bytecode.
- *Bytecode* is a highly optimized set of instructions
- Designed to be executed by the Java run-time system, which is called the *Java Virtual Machine (JVM)*. In essence, the original JVM was designed as an *interpreter for bytecode*.

However, the fact that a Java program is executed by the JVM helps solve the major problems associated with web-based programs. Here is why.

Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments

- because only the JVM needs to be implemented for each platform.
- once the run-time package exists for a given system, any Java program can run on it.

Remember, although the details of the JVM will differ from platform to platform, all understand the same Java bytecode.

- Because bytecode has been highly optimized, the use of bytecode enables the JVM to execute programs much faster than you might expect.

- Although Java was designed as an interpreted language, there is nothing about Java that prevents on-the-fly compilation of bytecode into native code in order to boost performance.

- For this reason, the HotSpot technology was introduced not long after Java's initial release.
    - HotSpot provides a Just-In-Time (JIT) compiler for bytecode.

- When a JIT compiler is part of the JVM, selected portions of bytecode are compiled into executable code in real time, on a piece-by-piece, demand basis.

- It is important to understand that it is not practical to compile an entire Java program into executable code all at once,
    - because Java performs various run-time checks that can be done only at run time.

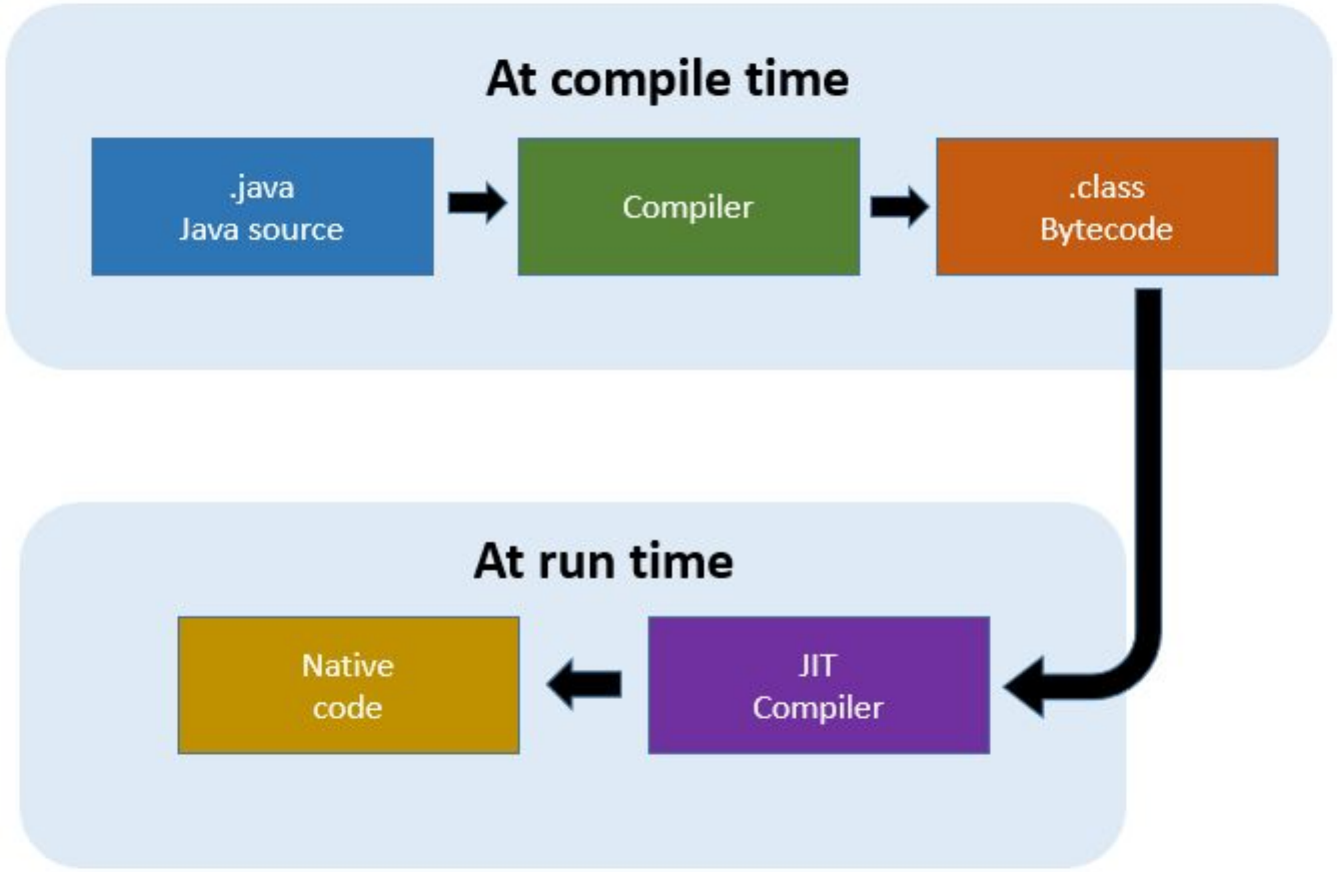
- Instead, a JIT compiler compiles code as it is needed, during execution.

Note

Furthermore, not all sequences of bytecode are compiled—only those that will benefit from compilation. The remaining code is simply interpreted.

However, the just-in-time approach still yields a significant performance boost.

Even when dynamic compilation is applied to bytecode, the portability and safety features still apply, because the JVM is still in charge of the execution environment.


At compile time
.java
Java source
Compiler
.class
Bytecode
At run time
Native
code
JIT
Compiler

Servlets: Java on the Server Side

As useful as applets can be, they are just one half of the client/server equation. Not long after the initial release of Java, it became obvious that

- Java would also be useful on the server side. The result was the *servlet*.

- A servlet is a small program that executes on the server.

- Just as applets dynamically extend the functionality of a web browser, servlets dynamically extend the functionality of a web server.
- Thus, with the advent of the servlet, Java spanned both sides of the client/server connection.

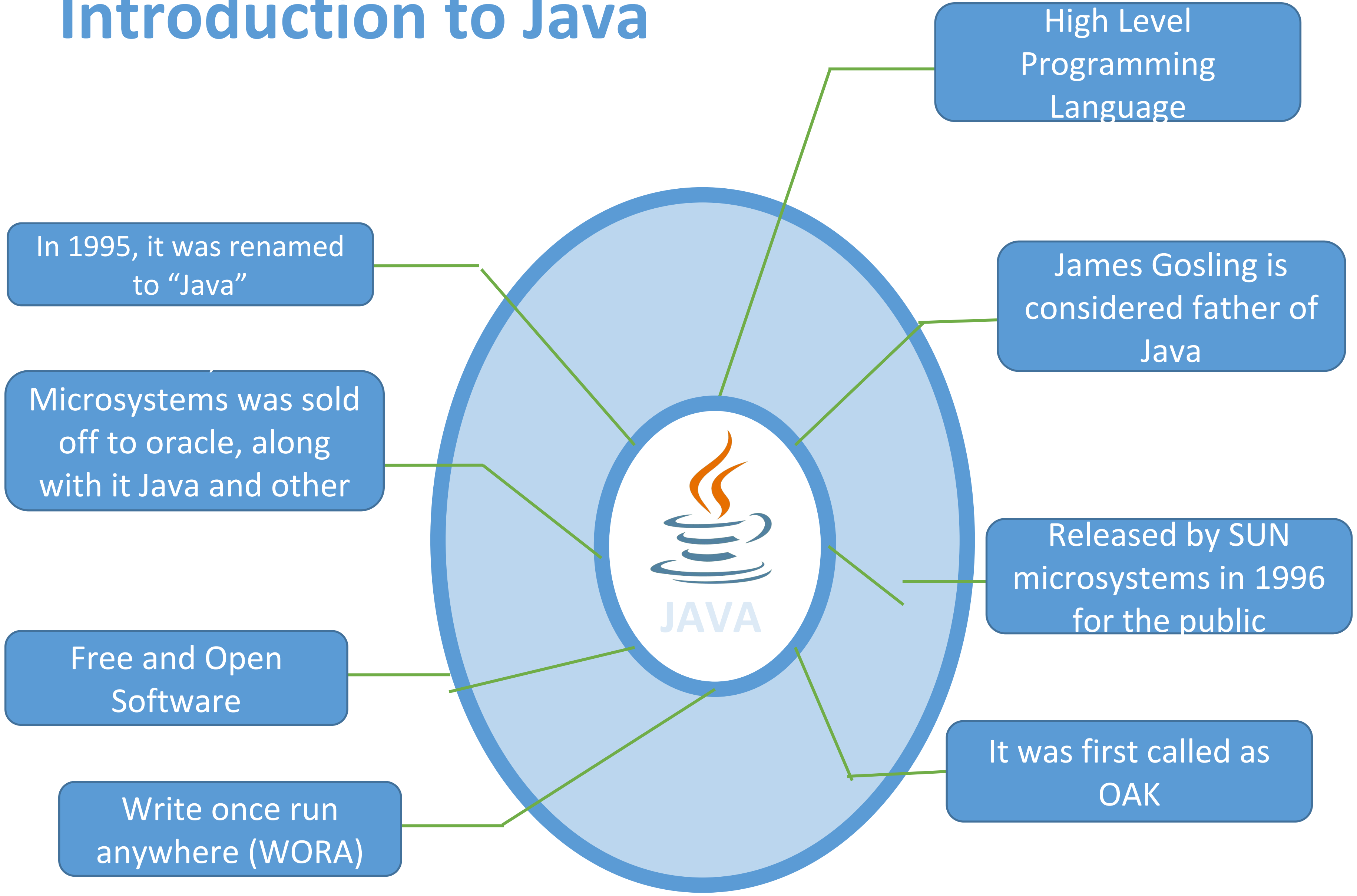
- Servlets (like all Java programs) are compiled into bytecode and executed by the JVM, they are highly portable.
  - Thus, the same servlet can be used in a variety of different server environments.
  - The only requirements are that the server support the JVM and a servlet container.

programming

- JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation.
- It was developed by James Gosling and Patrick Naughton.

- It is a simple programming language. Writing, compiling and debugging a program is easy in java.
  - It helps to create modular programs and reusable code.

Introduction to Java



High Level Programming Language

James Gosling is considered father of Java

Released by SUN microsystems in 1996 for the public

It was first called as OAK

In 1995, it was renamed to “Java”

Microsystems was sold off to oracle, along with it Java and other

Free and Open Software

Write once run anywhere (WORA)

Java terminology

- Java Virtual Machine (JVM)
- Bytecode
- Java Development Kit(JDK)
- Java Runtime Environment(JRE)

Java terminology

- **Java Virtual Machine (JVM)**
  This is generally referred as **JVM**.

- **Phases** are as follows: we write the program, then we compile the program and at last we run the program.
  1) Writing of the program is of course done by java programmer like you and me.
  2) Compilation of program is done by javac compiler, javac is the primary java compiler included in java development kit (JDK). It takes java program as input and generates java bytecode as output.
  3) In third phase, JVM executes the bytecode generated by compiler. This is called program run phase.

So, now that we understood that the primary function of JVM is to execute the bytecode produced by compiler. **Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems**. That is why we call java as platform independent language.

A Java virtual machine is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode.

The JVM is detailed by a specification that formally describes what is required in a JVM implementation.

NOTE:

Java Virtual Machine, or JVM, loads, **verifies and executes Java bytecode**.

It is known as the interpreter or the core of Java programming language because it executes Java programming.

- **bytecode**

  --javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM.

  - The bytecode is saved in a .class file by compiler.

- **Java Development Kit(JDK)**

  --As the name suggests this is complete java development kit that includes JRE (Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc.

  --In order to create, compile and run Java program you would need JDK installed on your computer.

- **Java Runtime Environment(JRE)**

  --JRE is a part of JDK which means that JDK includes JRE.

  --When you have JRE installed on your system, you can run a java program however you won't be able to compile it.

  --JRE includes JVM, browser plugins and applets support.

  --When you only need to run a java program on your computer, you would only need JRE.

Main Features of JAVA

❖ **Java is a platform independent language**

- Compiler(javac) converts source code (.java file) to the byte code(.class file).

- JVM executes the bytecode produced by compiler.
  - This byte code can run on any platform such as Windows, Linux, Mac OS etc. Which means a program that is compiled on windows can run on Linux and vice-versa.

- Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems.
  - That is why we call java as platform independent language.

- **Java is an Object Oriented language**

Object oriented programming is a way of organizing programs as collection of objects, each of which represents an instance of a class.

4 main concepts of Object Oriented programming are:
Abstraction
Encapsulation
Inheritance
Polymorphism

- **Simple**

Java is considered as one of simple language

- because it does not have complex features like Operator overloading, **Multiple inheritance**, pointers and Explicit memory allocation.

❖ **Robust Language**

- Robust means reliable.
- Java programming language is developed in a way that puts a lot of emphasis on early checking for possible errors, that’s why java compiler is able to detect errors that are not easy to detect in other programming languages.
- The main features of java that makes it robust are garbage collection, Exception Handling and memory allocation.

❖ **Secure**

- We don’t have pointers and we cannot access out of bound arrays (you get ArrayIndexOutOfBoundsException if you try to do so) in java.
- That’s why several security flaws like stack corruption or buffer overflow is impossible to exploit in Java.

- ❖ **Java is distributed**
  - Using java programming language we can create distributed applications. RMI(Remote Method Invocation) and EJB(Enterprise Java Beans) are used for creating distributed applications in java.
  - In simple words: The java programs can be distributed on more than one systems that are connected to each other using internet connection.
  - Objects on one JVM (java virtual machine) can execute procedures on a remote JVM.
- ❖ **Multithreading**
  - Java supports **multithreading**.
  - Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilisation of CPU.
- ❖ **Portable**
  - Java code that is written on one machine can run on another machine.
  - The platform independent byte code can be carried to any platform for execution that makes java code portable

What are the three main components of the Java language?

The three main components of Java language are JVM, JDK and JRE which stands for Java Virtual Machine, Java Development Kit and Java Runtime Environment respectively.

Each components work separately.

JVM (Java Virtual Machine)

It is an abstract machine. It is a specification that provides run-time environment in which java bytecode can be executed.

It follows three notations:

Specification: It is a document that describes the implementation of the Java virtual machine. It is provided by Sun and other companies.

Implementation: It is a program that meets the requirements of JVM specification.

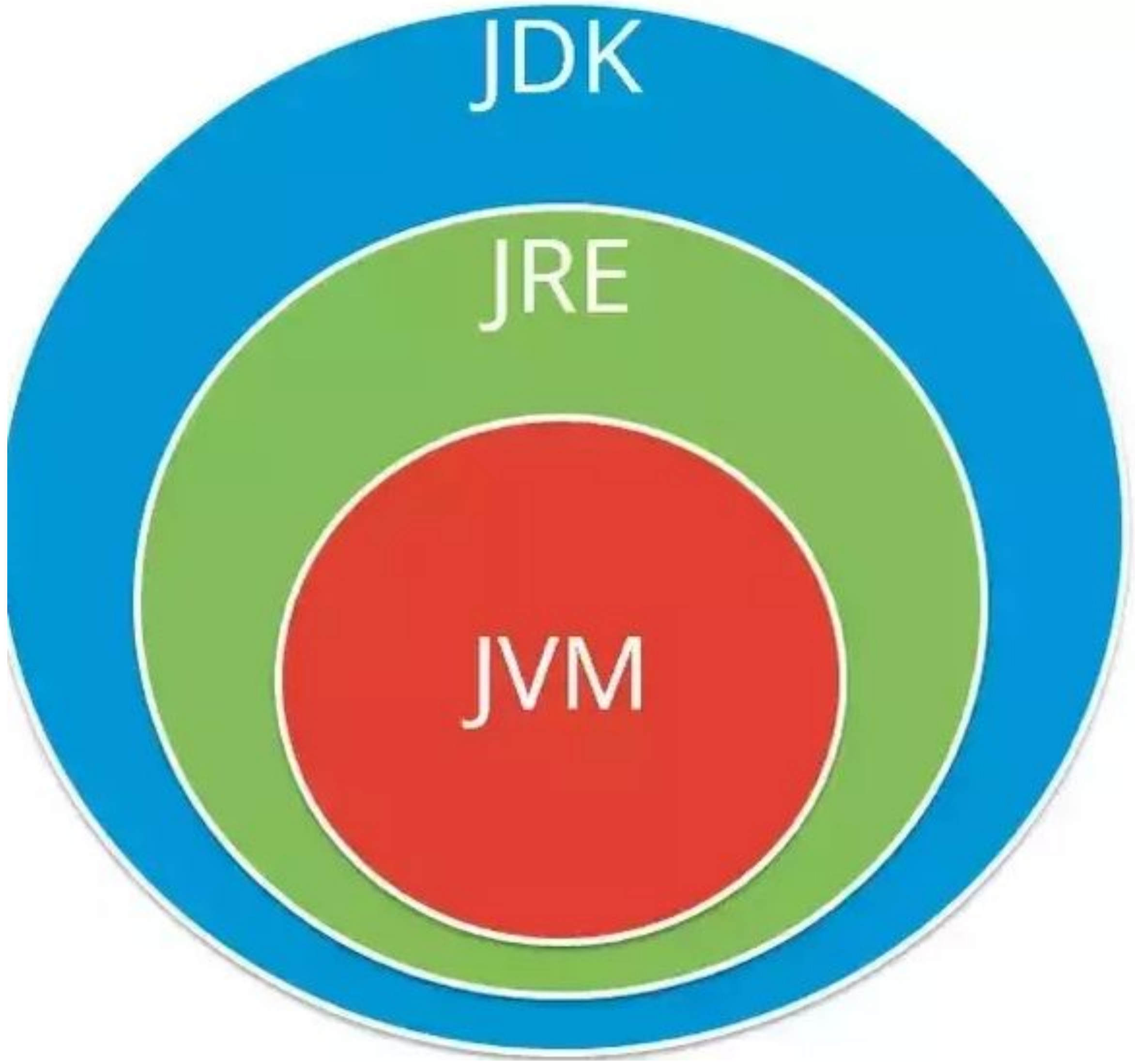
Runtime Instance: An instance of JVM is created whenever you write a java command on the command prompt and run the class.

JRE (Java Runtime Environment)

JRE refers to a runtime environment in which java bytecode can be executed. It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime. So JRE is a software package that contains what is required to run a Java program. Basically, it’s an implementation of the JVM which physically exists.

JDK(Java Development Kit)

It is the tool necessary to compile, document and package Java programs. The JDK completely includes JRE which contains tools for Java programmers. The Java Development Kit is provided free of charge. Along with JRE, it includes an interpreter/loader, a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development. In short, it contains JRE + development tools.

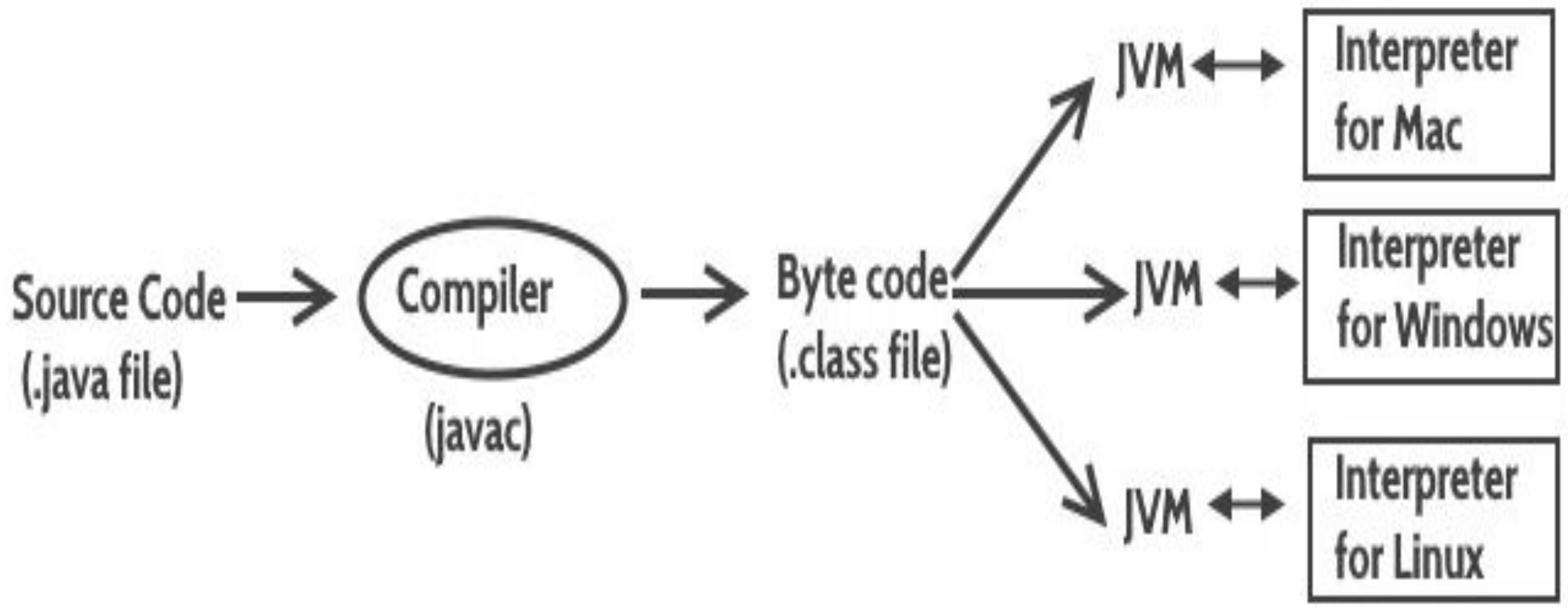

JDK
JRE
JVM

Java Virtual Machine (JVM), Difference JDK, JRE & JVM – Core Java

Java is a high level programming language. A program written in high level language cannot be run on any machine directly. First, it needs to be translated into that particular machine language. The **javac compiler** does this thing, it takes java program (.java file containing source code) and translates it into machine code (referred as byte code or .class file).

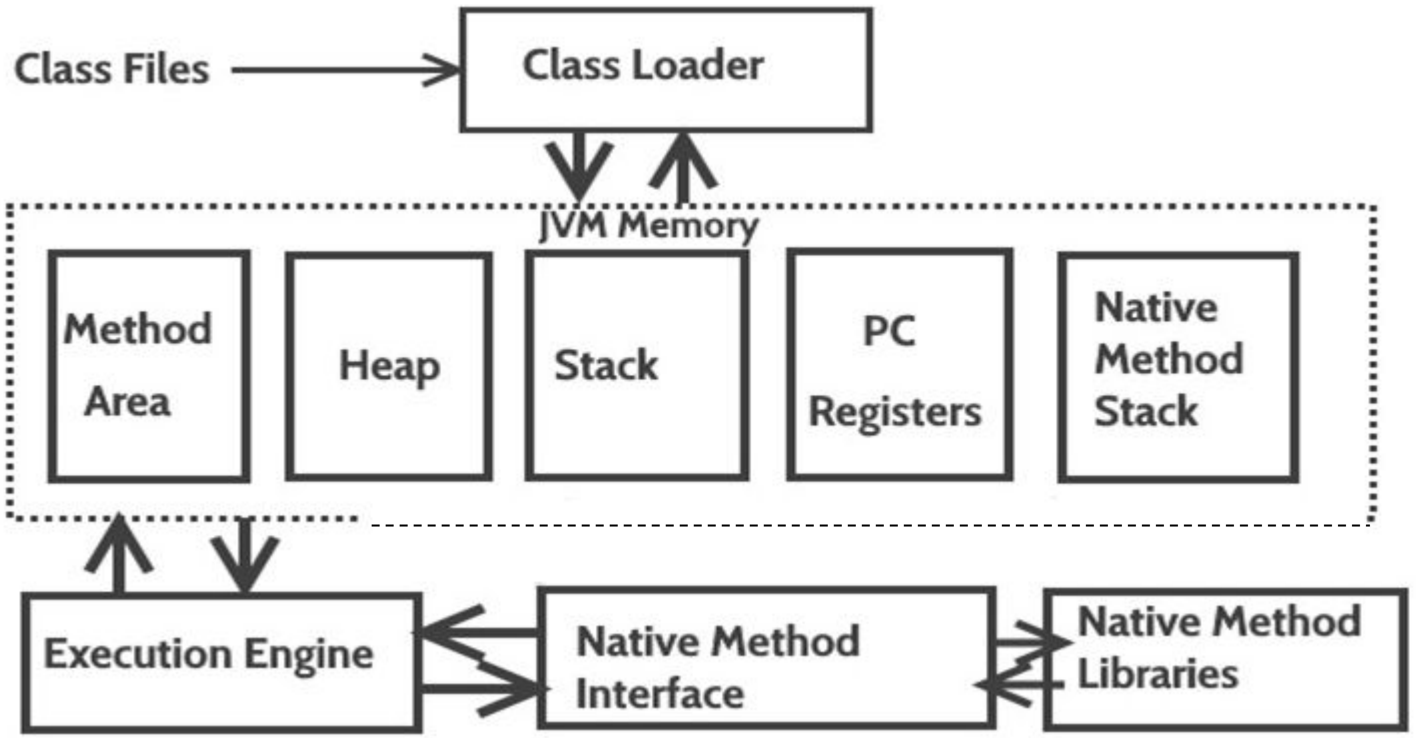
Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the **machine language for JVM is byte code**. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output. **JVM is the one that makes java platform independent**.

So, now we understood that the primary function of JVM is to execute the byte code produced by compiler. **Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems.** Which means that the byte code generated on Windows can be run on Mac OS and vice versa. That is why we call java as platform independent language. The same thing can be seen in the diagram below



So to summarise everything: The Java Virtual machine (JVM) is the virtual machine that runs on actual machine (your computer) and executes Java byte code. The JVM doesn’t understand Java source code, that’s why we need to have javac compiler that compiles *.java files to obtain *.class files that contain the byte codes understood by the JVM. JVM makes java portable (write once, run anywhere). Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems.

JVM Architecture



https://www.youtube.com/watch?v=G1ubVOl9IBw&feature=youtu.be

Lets see how JVM works:

Class Loader: The class loader reads the .class file and save the byte code in the **method area**.

Method Area: There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.

Heap: Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each .class file.

Stack: Stack is a also a part of JVM memory but unlike Heap, it is used for storing temporary variables.

PC Registers: This keeps the track of which instruction has been executed and which one is going to be executed. Since instructions are executed by threads, each thread has a separate PC register.

Native Method stack: A native method can access the runtime data areas of the virtual machine.

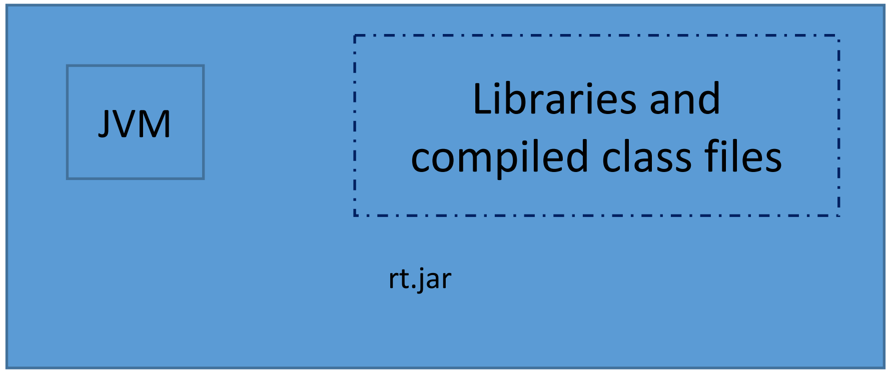
Native Method interface: It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.

Garbage collection: A class instance is explicitly created by the java code and after use it is automatically destroyed by garbage collection for memory management.

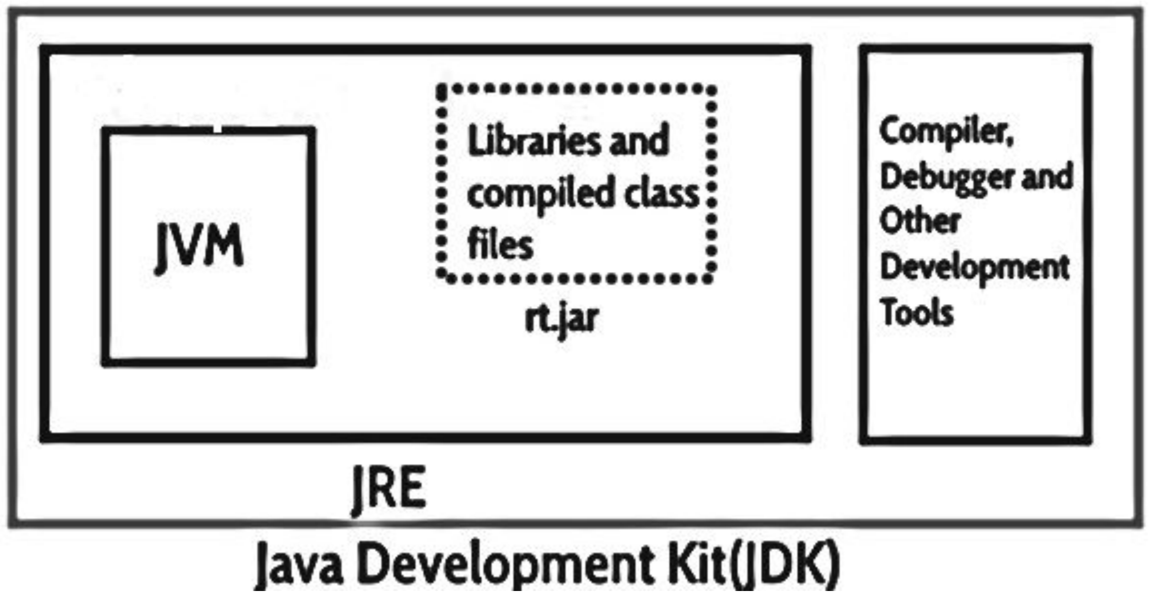
JVM Vs JRE Vs JDK

JRE: JRE is the environment within which the java virtual machine runs. JRE contains Java virtual Machine(JVM), class libraries, and other files excluding development tools such as compiler and debugger. Which means you can run the code in JRE but you can't develop and compile the code in JRE.

JVM: JVM runs the program by using class, libraries and files provided by JRE.

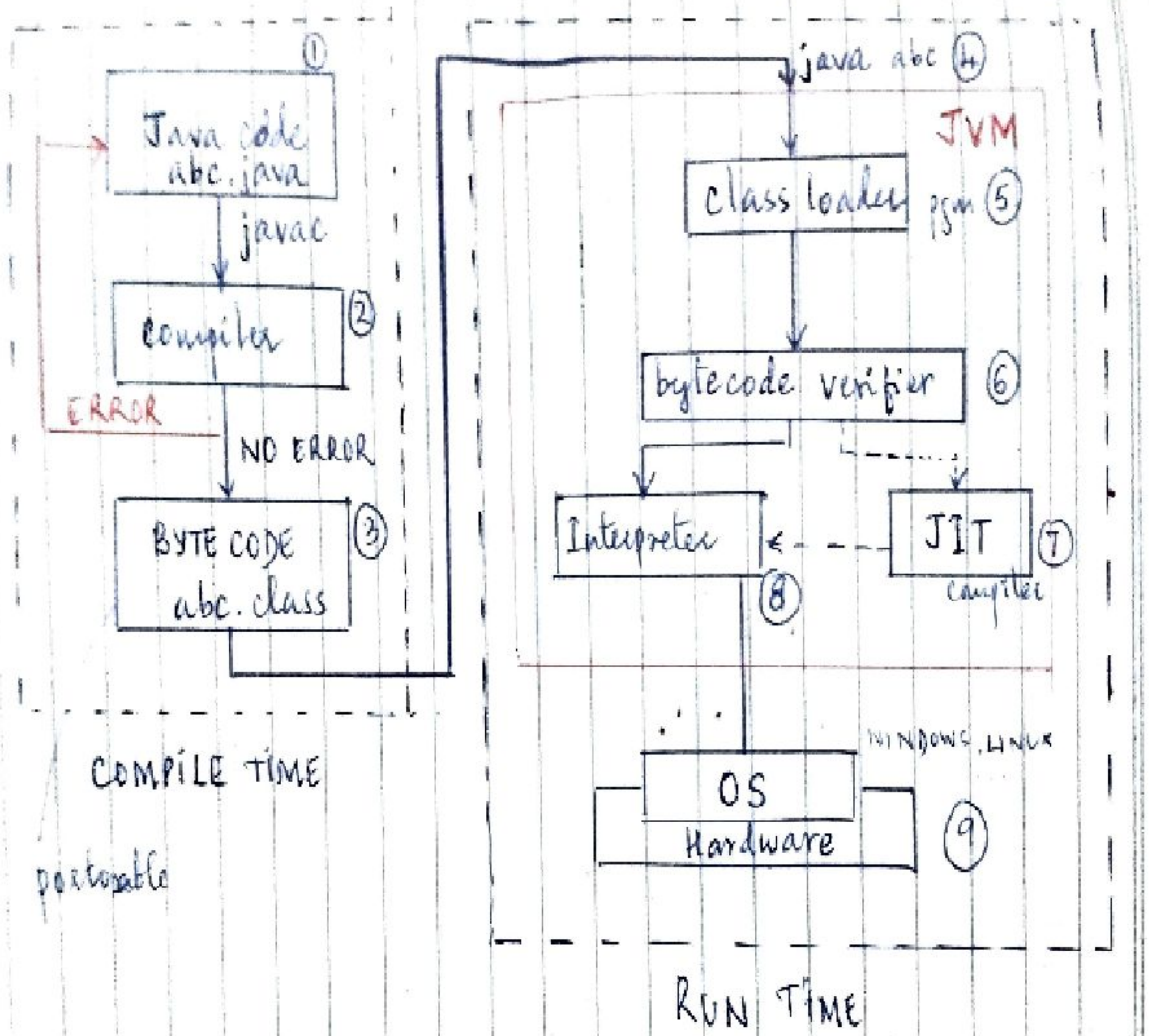


JDK: JDK is a superset of JRE, it contains everything that JRE has along with development tools such as compiler, debugger etc.


JVM
Libraries and compiled class files
rt.jar
Compiler, Debugger and Other Development Tools
JRE
Java Development Kit(JDK)

Execution Process of a JAVA pgm.

* COMPILED & INTERPRETED.



An overview 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.

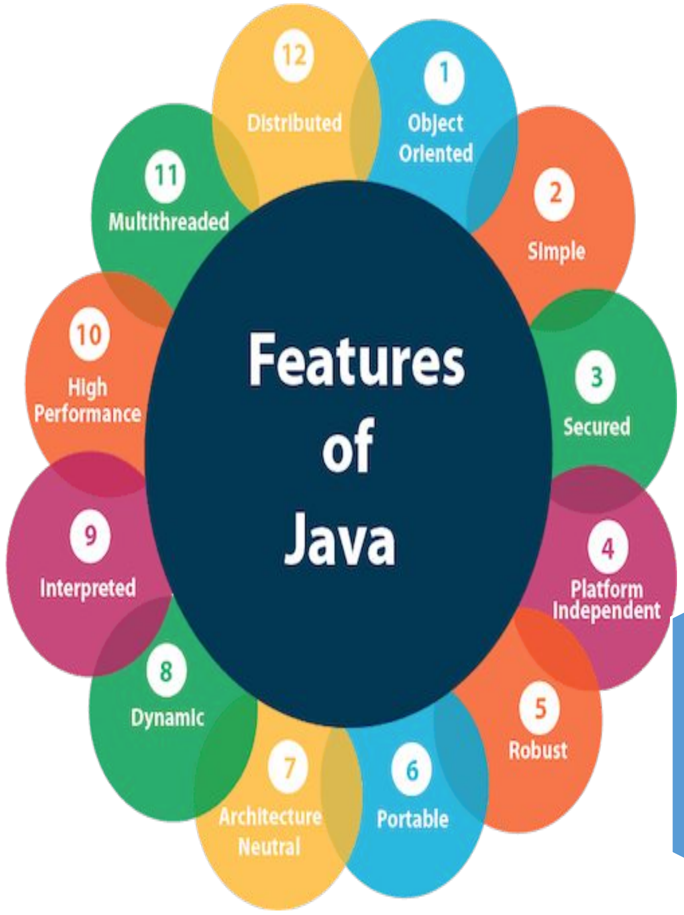
The features of Java are also known as java *buzzwords*.

The Java Buzzwords

- The key considerations were summed up by the Java team in the following list of buzzwords:
    - ❖ Simple
    - ❖ Secure
    - ❖ Portable
    - ❖ Object-oriented
    - ❖ Robust
    - ❖ Multithreaded
    - ❖ Architecture-neutral
    - ❖ Interpreted
    - ❖ High performance
    - ❖ Distributed
    - ❖ Dynamic

Features of Java

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



- Simple
- Object-Oriented
- Portable
- Architecture neutral
- Platform independent
- Secured
- Robust
- Interpreted
- High Performance
- Multithreaded
- Distributed
- Dynamic

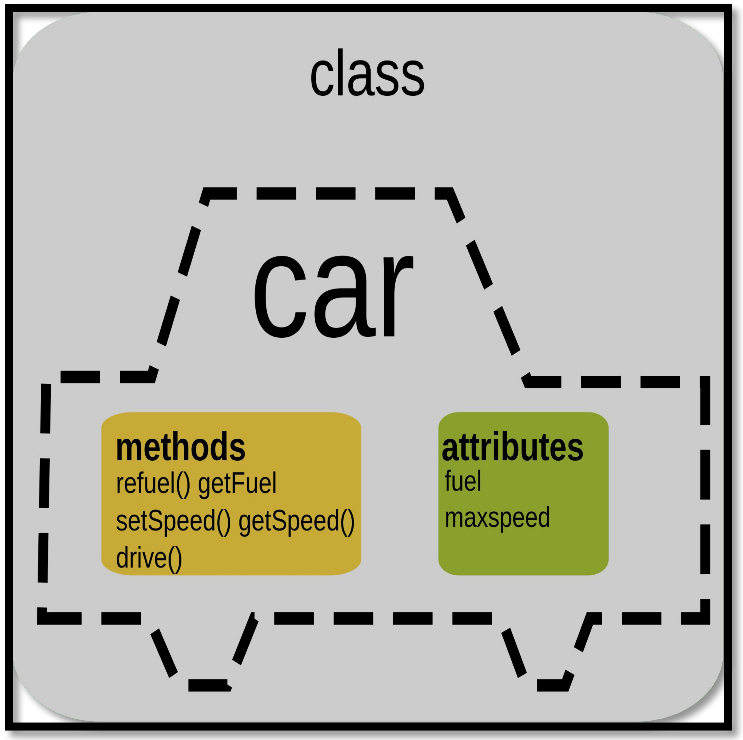
Simple

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.

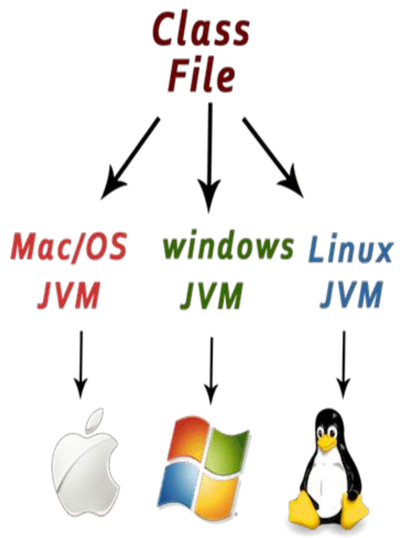
Object-Oriented

- 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:
  1. Object
  2. Class
  3. Inheritance
  4. Polymorphism
  5. Abstraction
  6. Encapsulation



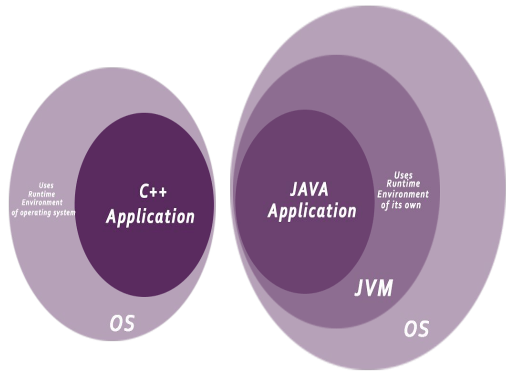
Platform Independent

- 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.



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

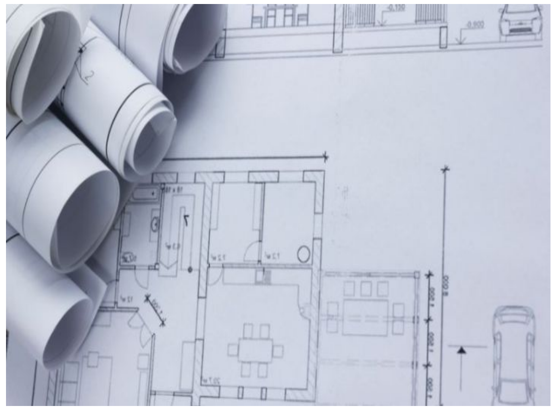
Robust

- 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

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.

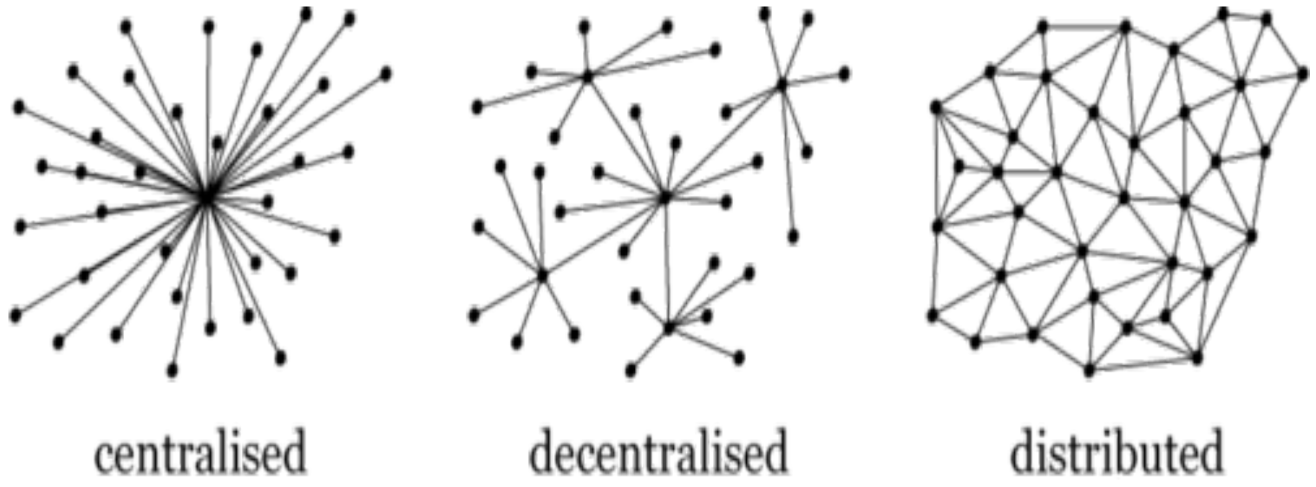
High-performance

- 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

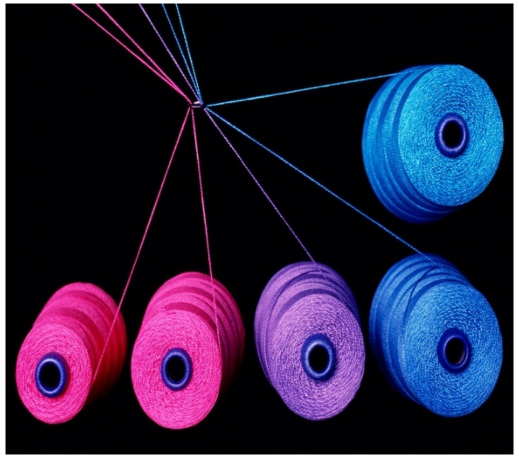
- Java is distributed because it facilitates users to create distributed applications in Java.

- 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).

- **simple** – Java is designed to be easy for the professional programmer to learn and use.
- **object-oriented:** a clean, usable, pragmatic approach to objects, not restricted by the need for compatibility with other languages.
- **Robust:** restricts the programmer to find the mistakes early, performs compile-time (strong typing) and run-time (exception-handling) checks, manages memory automatically.
- **Multithreaded:** supports multi-threaded programming for writing program that perform concurrent computations
- **Architecture-neutral:** Java Virtual Machine provides a platform independent environment for the execution of Java byte code
- **Interpreted and high-performance:** Java programs are compiled into an intermediate representation – byte code:

  a) can be later interpreted by any JVM

  b) can be also translated into the native machine code for

- **Distributed:** Java handles TCP/IP protocols, accessing a resource through its URL much like accessing a local file.
- **Dynamic:** substantial amounts of run-time type information to verify and resolve access to objects at run-time.
- **Secure:** programs are confined to the Java execution environment and cannot access other parts of the computer.
- **Portability:** Many types of computers and operating systems are in use throughout the world—and many are connected to the Internet.
- For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed. The same mechanism that helps ensure security also helps create portability.
- Indeed, Java's solution to these two problems is both elegant and efficient.

The Evolution of Java

Java 1.0

Java 2.0

With Java 2, Sun repackaged the Java product as J2SE (Java 2 Platform Standard Edition), and the version numbers began to be applied to that product.

Java 2 added support for a number of new features, such as Swing and the Collections ,Framework, and it enhanced the Java Virtual Machine and various programming tools.

Java SE 7

The newest release of Java is called KiJava SE 7, with the Java Developer’s t being called JDK 7, and an internal version number of 1.7. Java SE 7 is the first major release of Java since Sun Microsystems was acquired by Oracle (a process that began in April 2009 and that was completed in January 2010). Java SE 7 contains many new features, including significant additions to the language and the API libraries. Upgrades to the Java run-time system that support non-Java languages are also included, but it is the language and library additions that are of most interest to Java programmers.

A Culture of Innovation

- Java has been at the center of a culture of innovation.

- Its original release redefined programming for the Internet.
- The Java Virtual Machine (JVM) and bytecode changed the way we think about security and portability. The applet (and then the servlet) made the Web come alive.
- The Java Community Process (JCP) redefined the way that new ideas are assimilated into the language. Because Java is used for Android programming, Java is part of the smartphone revolution. The world of Java has never stood still for very long.
- Java SE 7 is the latest release in Java’s ongoing, dynamic history.

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform independent.

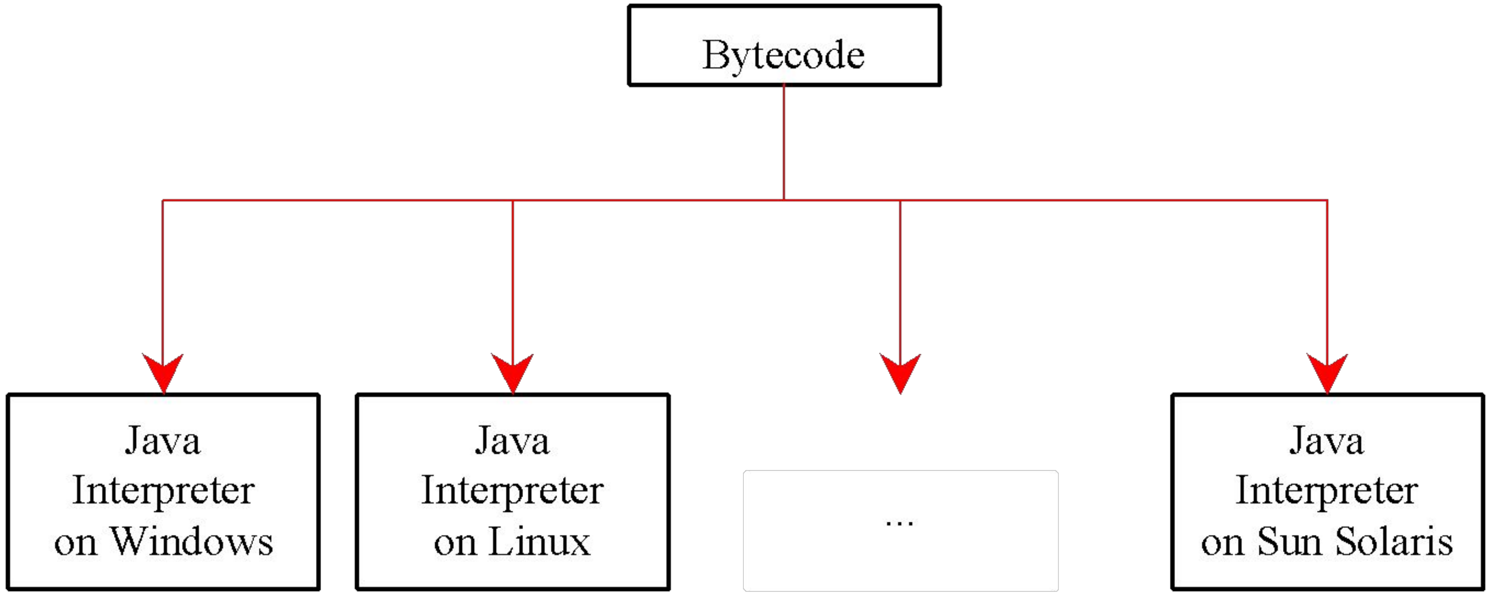
Comparison Index	C++	Java
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object

Getting Started with Java Programming

- A Simple Java Application
- Compiling Programs
- Executing Applications

Executing Applications

- On command line
  - `java classname`


Bytecode
Java Interpreter on Windows
Java Interpreter on Linux
...
Java Interpreter on Sun Solaris

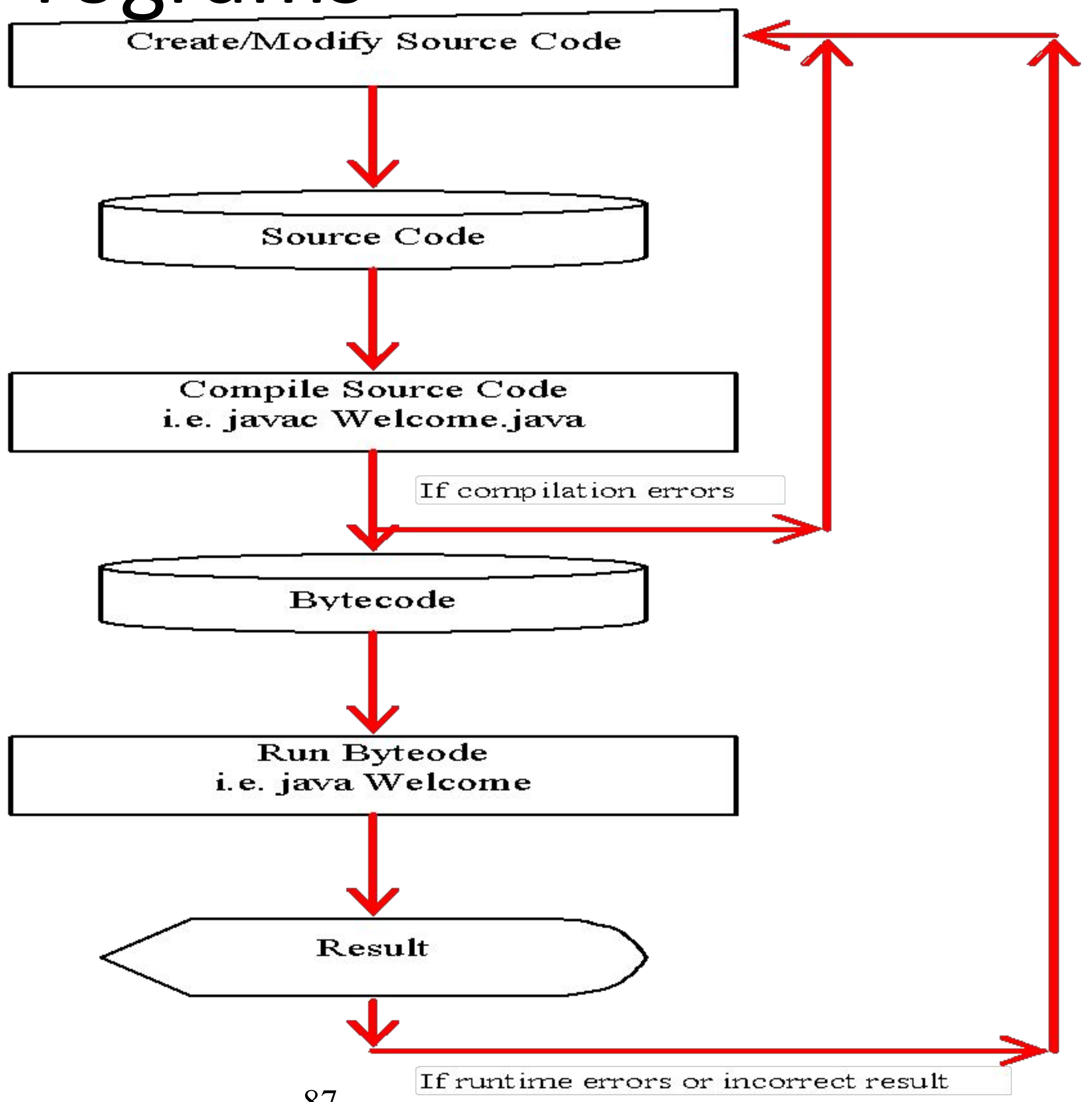
A Simple Application

Example

```
//This application program prints Welcome to Java!

package chapter1;

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

        System.out.println("Welcome to Java!");
    }
  }
```

Creating and Compiling Programs

- On command line
  - `javac file.java`

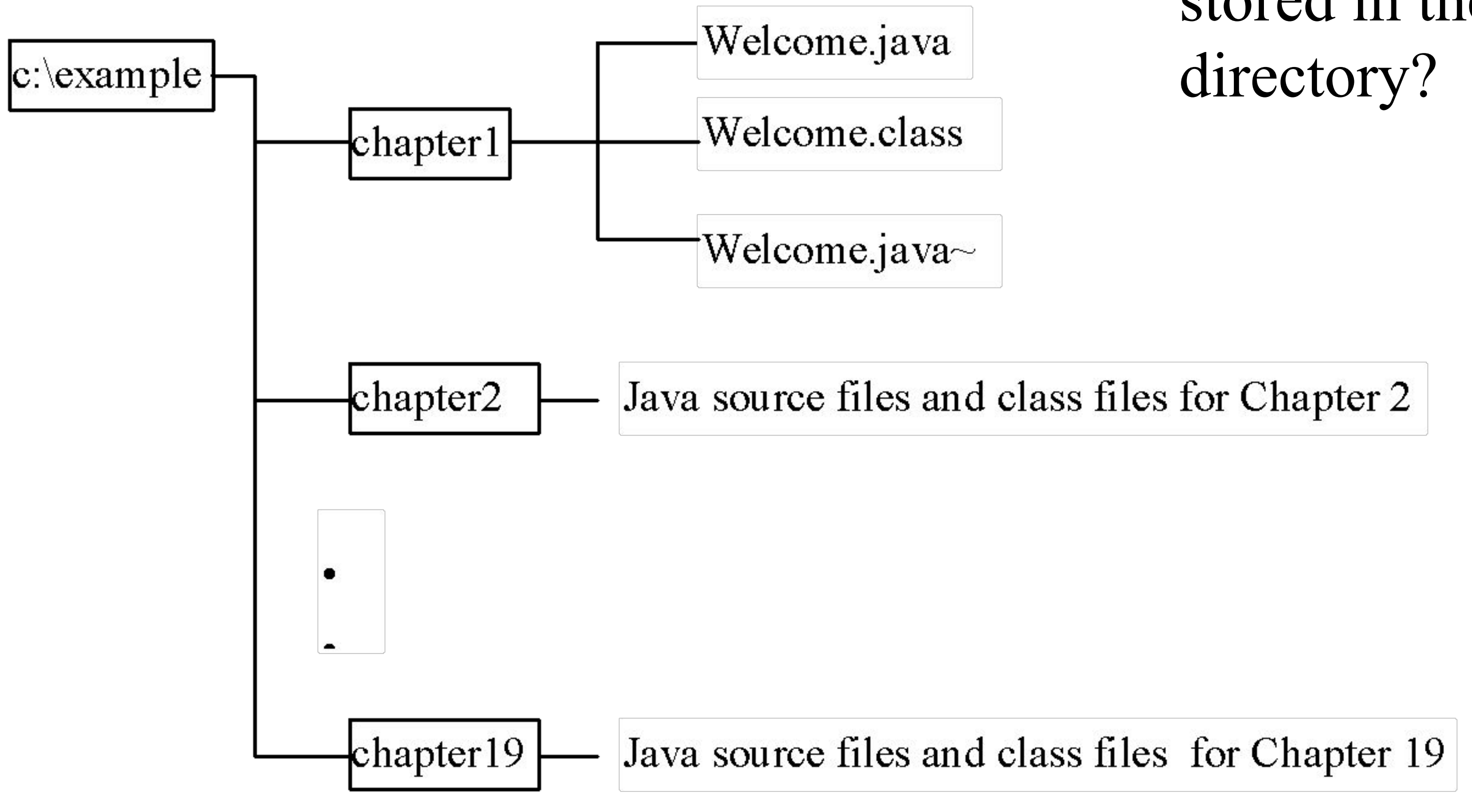


Create/Modify Source Code
Source Code
Compile Source Code i.e. javac Welcome.java
If compilation errors
Bytecode
Run Byteode i.e. java Welcome
Result
If runtime errors or incorrect result

Example

```
javac Welcome.java

java Welcome

output:...
```

Compiling and Running a Program


c:\example
chapter1
Welcome.java
Welcome.class
Welcome.java~
chapter2
Java source files and class files for Chapter 2
chapter19
Java source files and class files for Chapter 19

Where are the files stored in the directory?

How to Compile and Run your First Java Program

Simple Java Program

```
public class FirstJavaProgram {
public static void main(String[] args){
System.out.println("This is my first program in java");
}//End of main
}//End of FirstJavaProgram Class
```

Output: This is my first program in java

A First Simple Program

```
/*
This is a simple Java program.
Call this file "Example.java".
*/
class Example {
// Your program begins with a call to main().
public static void main(String args[]) {
System.out.println("This is a simple Java program.");
}
}
```

Entering the Program

For most computer languages, the name of the file that holds the source code to a program is immaterial. However, this is not the case with Java.

- The first thing that you must learn about Java is that the name you give to a source file is very important.
- For this example, the name of the source file should be **Example.java**.
- In Java, a source file is officially called a *compilation unit*. It is a text file that contains (among other things) one or more class definitions. (For now, we will be using source files that contain only one class.)
- The Java compiler requires that a source file use the **.java** filename extension. As you can see by looking at the program, the name of the class defined by the program is also **Example**. This is not a coincidence.
- In Java, all code must reside inside a class.

- By convention, the name of the main class should match the name of the file that holds the program. You should also make sure that the capitalization of the filename matches the class name. The reason for this is that Java is case-sensitive.

- At this point, the convention that filenames correspond to class names may seem arbitrary. However, this convention makes it easier to maintain and organize your programs.

Compiling the Program

To compile the **Example** program, execute the compiler, **javac**, specifying the name of the source file on the command line, as shown here:

- C:\>javac Example.java
- The **javac** compiler creates a file called **Example.class** that contains the bytecode version of the program.
- The Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute. Thus, the output of **javac** is not code that can be directly executed.
- To actually run the program, you must use the Java application launcher called **java**.
- To do so, pass the class name **Example** as a command-line argument, as shown here:

```
C:\>java Example
```

- When the program is run, the following output is displayed:

```
This is a simple Java program.
```

- When Java source code is compiled, each individual class is put into its own output file named after the class and using the **.class** extension. This is why it is a good idea to give your Java source files the same name as the class they contain—the name of the source file will match the name of the **.class** file.

- When you execute **java** as just shown, you are actually specifying the name of the class that you want to execute. It will automatically search for a file by that name that has the **.class** extension. If it finds the file, it will execute the code contained in the specified class.

How to compile and run the above program

Prerequisite: You need to have java installed on your system.

Step 1: Open a text editor, like Notepad on windows and TextEdit on Mac. Copy the above program and paste it in the text editor.

For the sake of simplicity, I will only use text editor and command prompt (or terminal)

You can also use IDE like Eclipse to run the java program

Step 2: Save the file as **FirstJavaProgram.java**. You may be wondering why we have named the file as FirstJavaProgram, the thing is that we should always name the file same as the public classname. In our program, the public class name is FirstJavaProgram, that’s why our file name should be **FirstJavaProgram.java**.

Step 3: In this step, we will compile the program. For this, open **command prompt (cmd) on Windows**, if you are **Mac OS then open Terminal**. To compile the program, type the following command and hit enter.

```
javac FirstJavaProgram.java
```

You may get this error when you try to compile the program: “**javac’ is not recognized as an internal or external command, operable program or batch file**“. This error occurs when the java path is not set in your system

If you get this error then you first need to set the path before compilation.

Set Path in Windows

Open command prompt (cmd), go to the place where you have installed java on your system and locate the bin directory, copy the complete path and write it in the command like this.

```
set path=C:\Program Files\Java\jdk1.8.0_121\bin
```

Note: Your jdk version may be different. Since I have java version 1.8.0_121 installed on my system, I mentioned the same while setting up the path.
The steps above are for setting up the path temporary which means when you close the command prompt or terminal, the path settings will be lost and you will have to set the path again next time you use it.

permanent path setup guide

Step 4: After compilation the .java file gets translated into the .class file(byte code). Now we can run the program. To run the program, type the following command and hit enter:

```
java FirstJavaProgram
```

Note that you should not append the .java extension to the file name while running the program.

Closer look to the First Java Program

Now that we have understood how to run a java program, let have a closer look at the program we have written above.

```
public class FirstJavaProgram {
```

This is the first line of our java program. Every java application must have at least one class definition that consists of class **keyword** followed by class name. When I say keyword, it means that it should not be changed, we should use it as it is. However the class name can be anything.

I have made the class public by using public access modifier, all you need to know now that **a java file can have any number of classes** but it can have only one public class and the file name should be same as public class name.

```java
public static void main(String[] args) {
```
This is our next line in the program, lets break it down to understand it:

public: This makes the main method public that means that we can call the method from outside the class.

static: We do not need to create object for static methods to run. They can run itself.

void: It does not return anything.

main: It is the method name. This is the entry point method from which the JVM can run your program.

(String[] args): Used for command line arguments that are passed as strings.

```java
System.out.println("This is my first program in java");
```

This method prints the contents inside the double quotes into the console and inserts a newline after.