CST 205
OOP USING JAVA
MODULE 1-PART 2

SMITHA JACOB,AP,CSE

SJCET,PALA

MODULE 1 Introduction:

- Approaches to Software Design
  - Functional Oriented Design,
  - Object Oriented Design,
- Case Study of Automated Fire Alarm System.
- Basic Object Oriented concepts,
- Object Modeling Using Unified Modeling Language (UML) UML diagrams, Use case model, Class diagram, Interaction diagram, Activity diagram, State chart diagram.

MODULE 1

Introduction to Java –

Java programming Environment and Runtime Environment,

Development Platforms -Standard, Enterprise.

Java Virtual Machine (JVM),

Java compiler, Bytecode, Java applet, Java Buzzwords,

Java program structure, Comments, Garbage Collection, Lexical Issues.

Introduction to Java

Java overview

- When the chronicle of computer languages is written, *B led to C, C evolved into C++, and C++ set the stage for Java.* (**B** is a programming language that was developed at Bell Labs)
- Java is related to C++, which is a direct descendent of C.
- Java is inherited from these two languages.
- From C, Java derives its syntax.
- Many of Java's object-oriented features were influenced by C++.

Creation of Java

- Java was conceived by James Gosling, Arthur Vanhoff,Andy Bechotolshem at Sun Microsystems, Inc. in 1991
- Patrick Naughton, Chris Warth, and Mike Sheridan and many more individuals behind the project.
- Took 18 months to develop the first working version.
- Initially called "Oak", but was renamed "Java" in 1995.
- The initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995

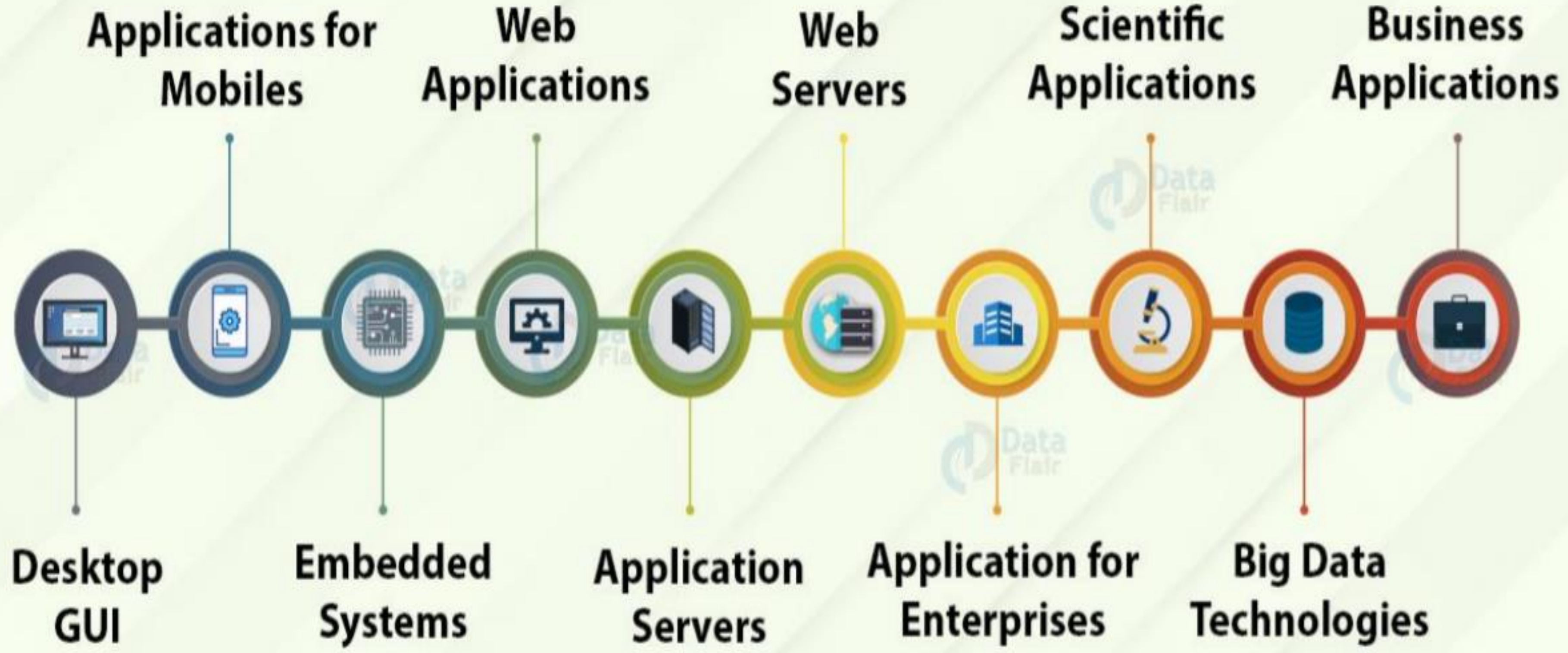
Green Project,Oak,Java

- Sun Microsystems funded an internal research project “Green”
- Result: A programming language called “Oak”
- Newly created language – Oak
- https://www.youtube.com/watch?v=1CsTH9S79qI
- Problem: There was already a programming language called Oak.
- The “Green” team met at a local coffee shop to come up with another name...Java!

Green Team

Source: https://tech-insider.org/

Applications of Java



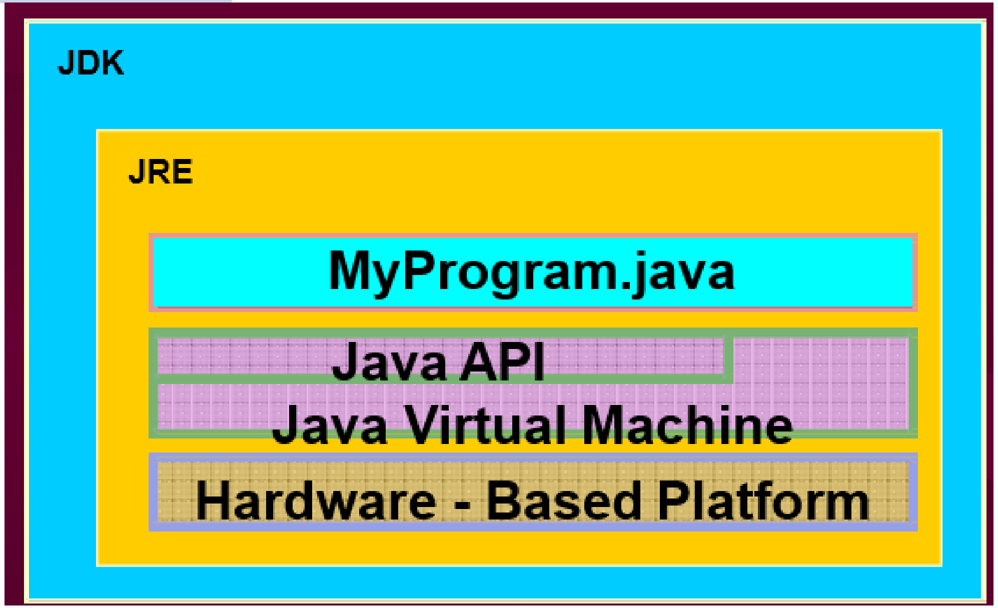
Java programming Environment and Runtime Environment

Java programming Environment and Runtime Environment

- Java is a simple,portable,distributed, concurrent, class-based, object-oriented programming language with a programming and runtime environment, consisting of:
- A programming language
- An API specification
- A virtual machine specification

Java Development Kit

- JDK is a set of Java tools for developing Java programs
- Consists of Java API, Java Compiler, and JVM
- API's are prewritten code, organized into packages of similar topics

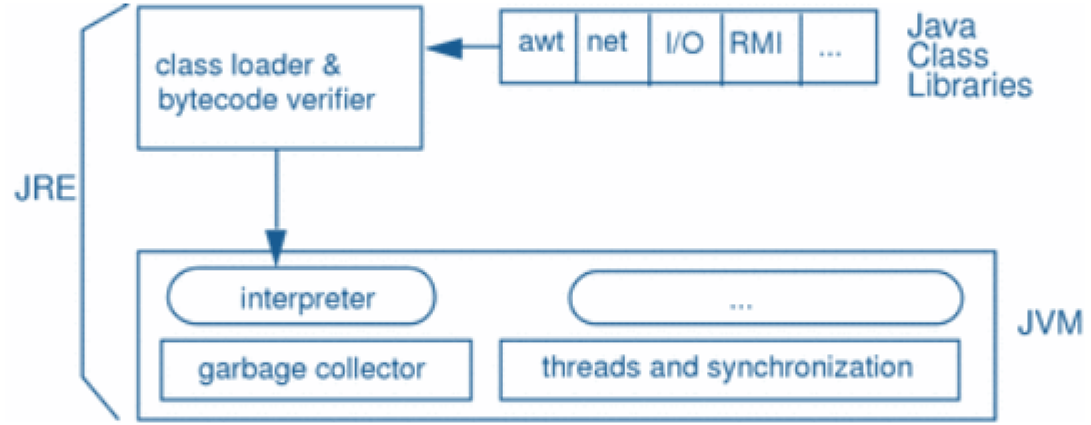


JRE(Java Runtime Environment) Components

- The JRE is the software environment in which programs compiled for a typical JVM implementation can run.
- The runtime system includes:
  - Code necessary to run Java programs,
  - dynamically link native methods, manage memory, and handle exceptions
  - Implementation of the JVM

JRE Components

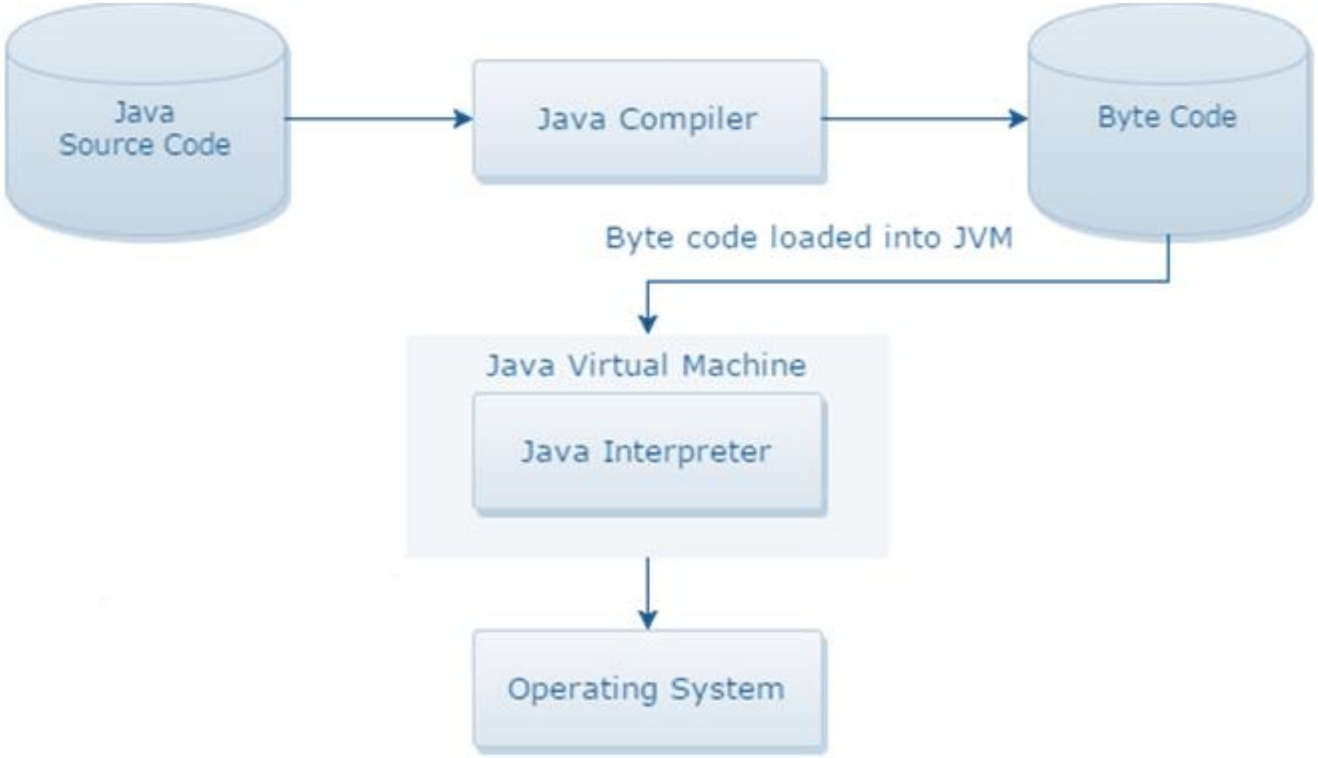
- The following figure shows the JRE and its components, including a typical JVM implementation's various modules and its functional position with respect to the JRE and class libraries



JVM Components

- JVM is the main component of Java architecture and it is the part of the JRE (Java Runtime Environment).
- Program of JVM is written in "C Programming Language" and JVM is Operating System dependent.
- The Java compiler, javac, outputs bytecodes and puts them into a .class file. The JVM then interprets these bytecodes, which can then be executed by any JVM implementation, thus providing Java's cross-platform portability.

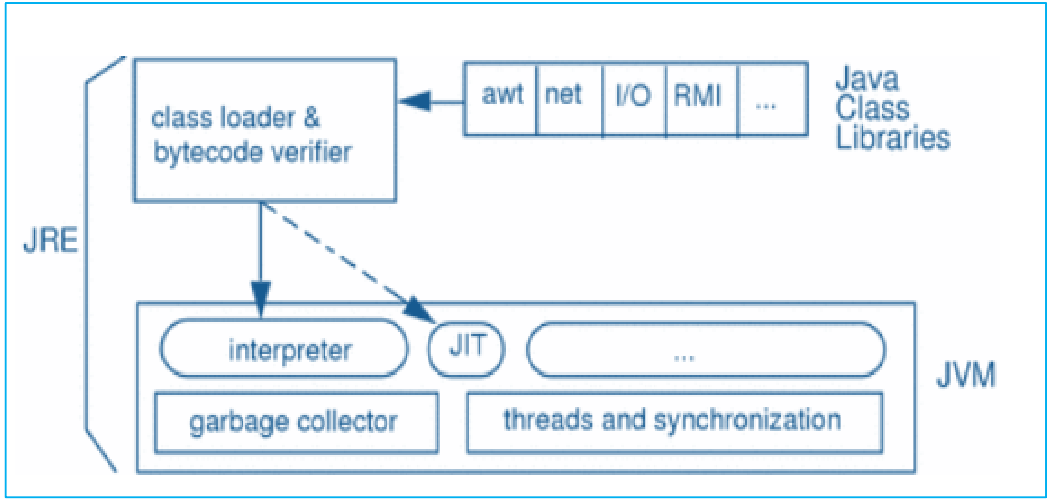
DIAGRAM OF JVM

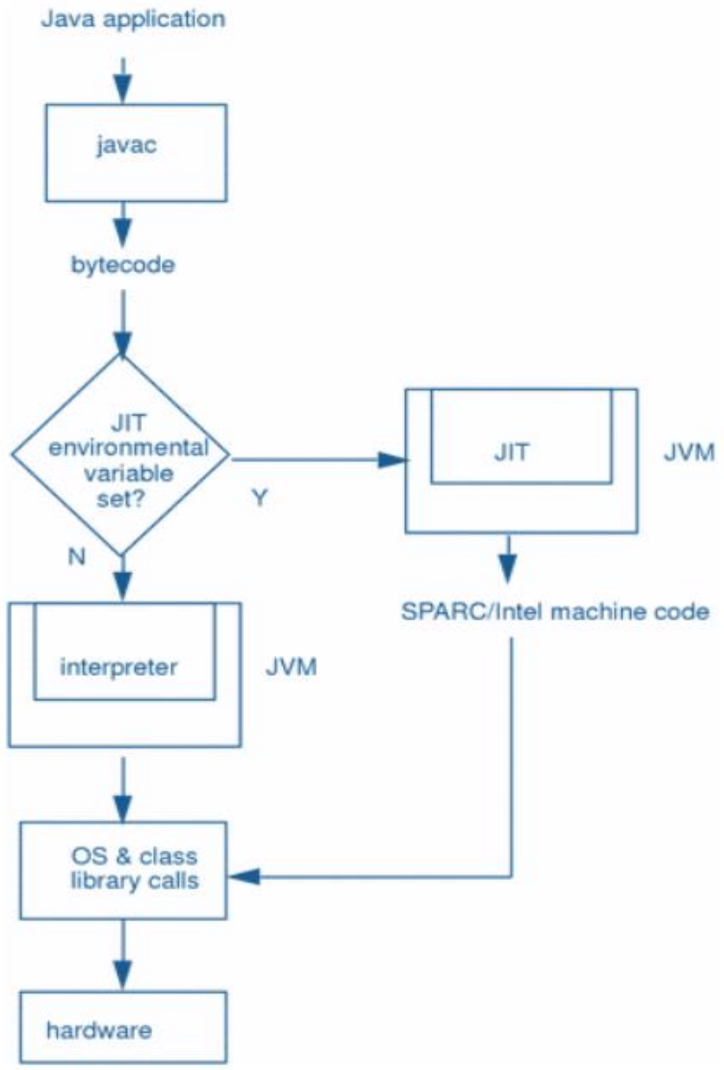


Sun Just-In-Time (JIT) Compiler

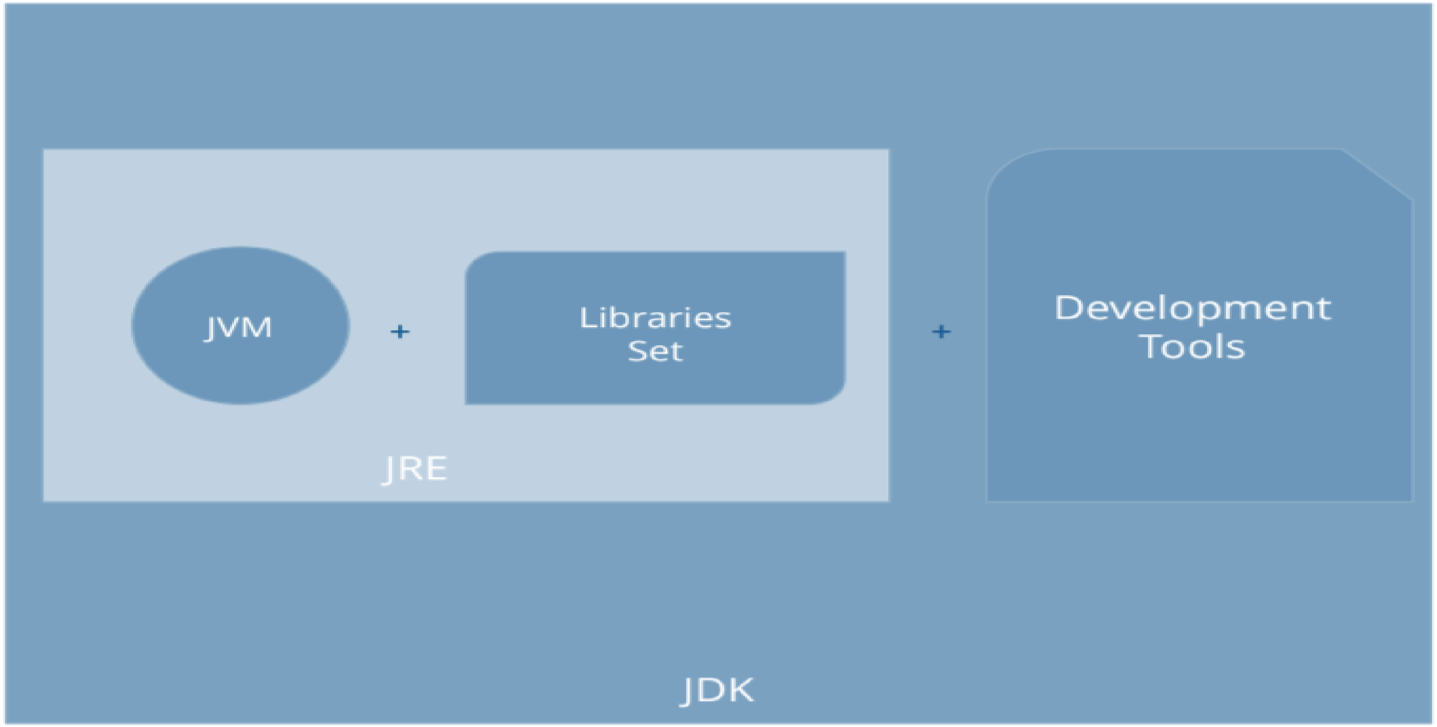
- The Sun Java JIT compiler, an integral part of the JVM, can accelerate execution performance many times over previous levels.
- The execution engine of JVM consists of three parts namely interpreter,JIT Compiler,Garbage collector
- The JIT compiler then compiles the bytecodes into native code for the platform(os+processor) on which it is running.
- Native code will have the knowledge of the underlying processor

Sun Just-In-Time (JIT) Compiler



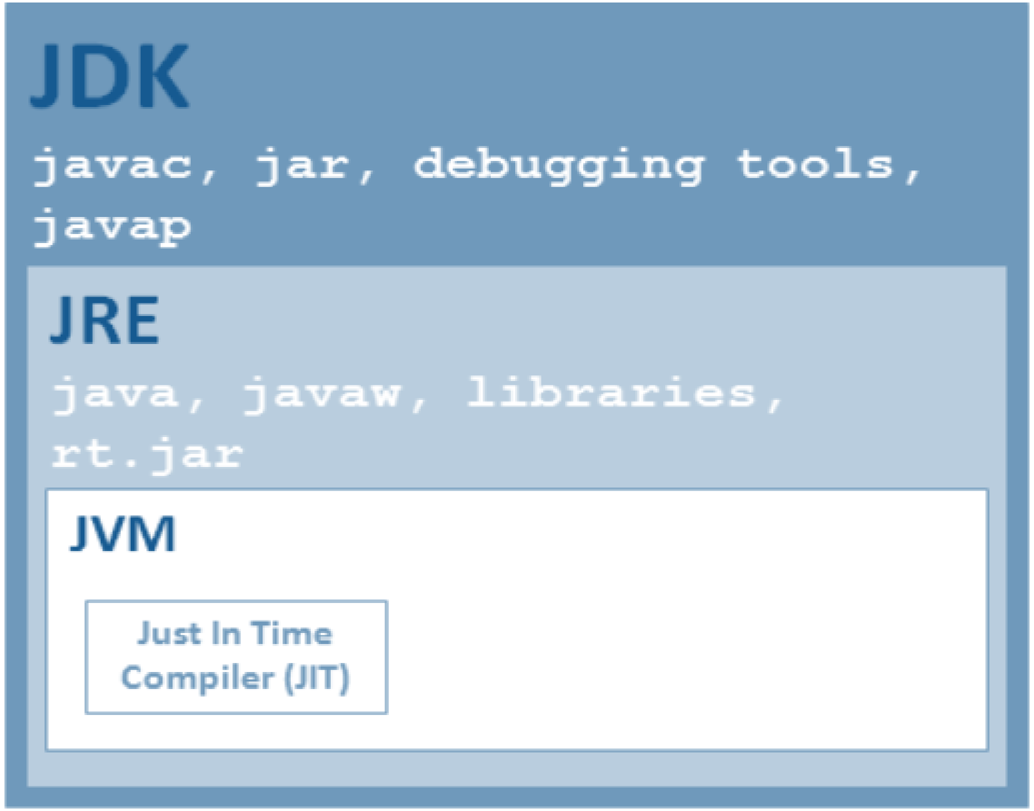


JVM,JRE,JDK



Java Development Kit

JIT,JVM,JRE,JDK


JDK
javac, jar, debugging tools, javap
JRE
java, javaw, libraries, rt.jar
JVM
Just In Time Compiler (JIT)

Development Platforms

History of Java

- In 1993 the World Wide Web appeared on the Internet and transformed the text-based interface to a graphical rich environment.
- The team developed Web applets (time programs) that could run on all types of computers connected to the Internet.
- Many companies such as Netscape and Microsoft announced their support for Java
- JDK 1.0 released in(January 23, 1996)
- By 1996 - Java established itself it self as both
  - "the language for Internet programming"
  - a general purpose OO language

3 Java Platform Editions

- Java 2 Platform, Standard Edition (Java SE)-formerly known as (**J2SE**).
  - ▻Core Java Platform targeting applications running on workstations
  - ▻The development of J2SE was led by Sun and progressed following the Java Community Process (JCP) to include input from a variety of constituents.
- Java 2 Platform, Enterprise Edition(Java EE) (formerly known as J2EE)
  - ▻Component-based approach to developing distributed, multi-tier enterprise applications
- Java 2 Platform, Micro Edition(Java ME) (formerly known as J2ME)
  - ▻Targeted at small, stand-alone or connectable consumer and embedded devices

3 Java Platform Editions

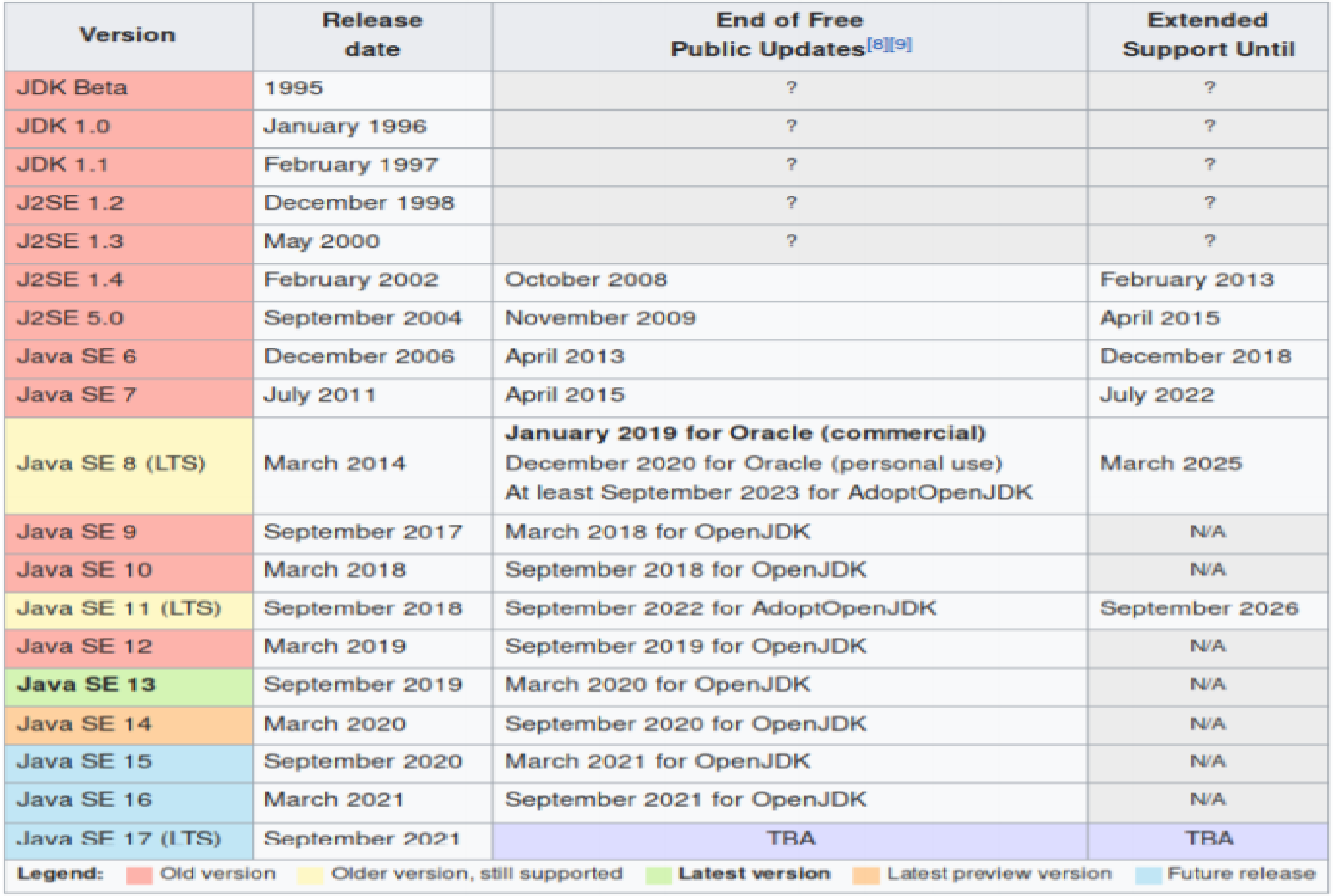
- The platform was known as Java 2 Platform, Standard Edition or J2SE from version 1.2, until the name was changed to Java Platform, Standard Edition or Java SE in V 1.5.
- The "2" was originally intended to emphasize the major changes introduced in version 1.2, but was removed in version 1.6.
- The naming convention has been changed several times over the Java version history.
- Java Platform, Enterprise Edition (Java EE) is a related specification that includes all the classes in Java SE, plus a number that are more useful programs that run on servers
- Java Platform, Micro Edition (Java ME) is a related specification intended to provide a certified collection of Java APIs for the development of software for devices such as cell phones, PDAs and set-top boxes.

3 Java Platform Editions

- Starting with J2SE 1.4 (Merlin), Java SE has been developed under the Java Community Process(JCP), which produces descriptions of proposed and final specifications for the Java platform called Java Specification Requests (JSR).
- JSR 59 was the umbrella specification for J2SE 1.4 and
- JSR 176 specified J2SE 5.0 (Tiger).
- Java SE 6 (Mustang) was released under JSR 270.

JRE and JDK in J2SE

- Oracle provides two principal software products in the Java Platform, Standard Edition (Java SE) family:
- **Java SE Runtime Environment (JRE)**
- The JRE provides the libraries, Java virtual machine, and other components necessary for you to *run* applets and applications written in the Java programming language
- **Java SE Development Kit (JDK)**
- The JDK includes the JRE plus command-line development tools such as compilers and debuggers that are necessary or useful for *developing* applets and applications.



Version	Release date	End of Free Public Updates[8][9]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** December 2020 for Oracle (personal use) At least September 2023 for AdoptOpenJDK	March 2025
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	September 2022 for AdoptOpenJDK	September 2026
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	TBA	TBA

Legend: 🟥 Old version 🟨 Older version, still supported 🟩 **Latest version** 🟧 Latest preview version 🟦 Future release

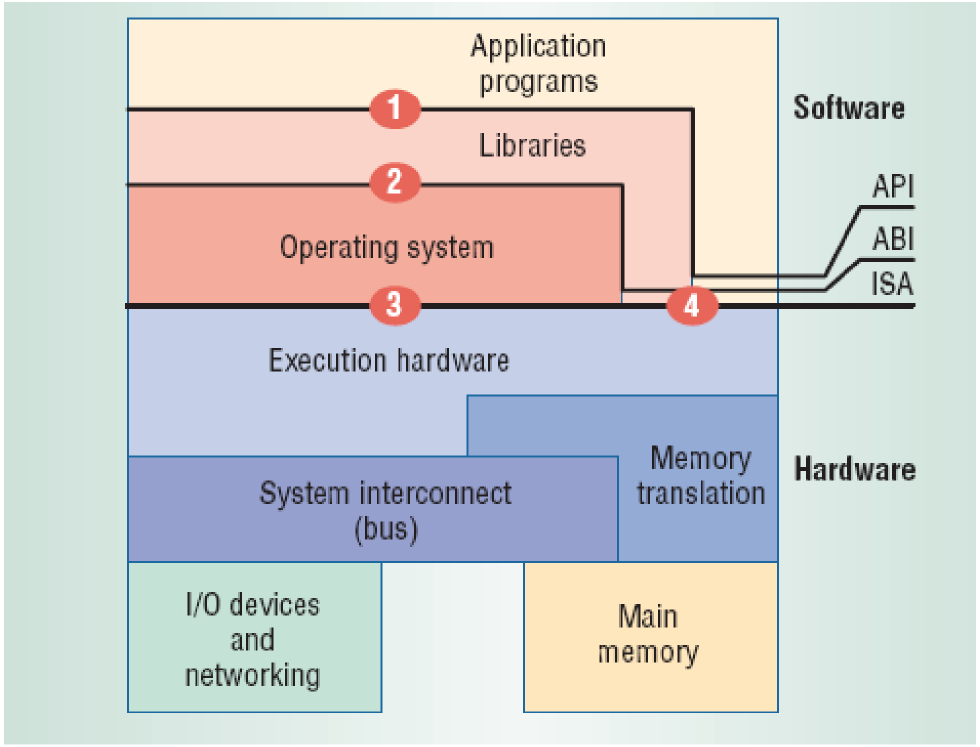
Java Virtual Machine

(JVM)

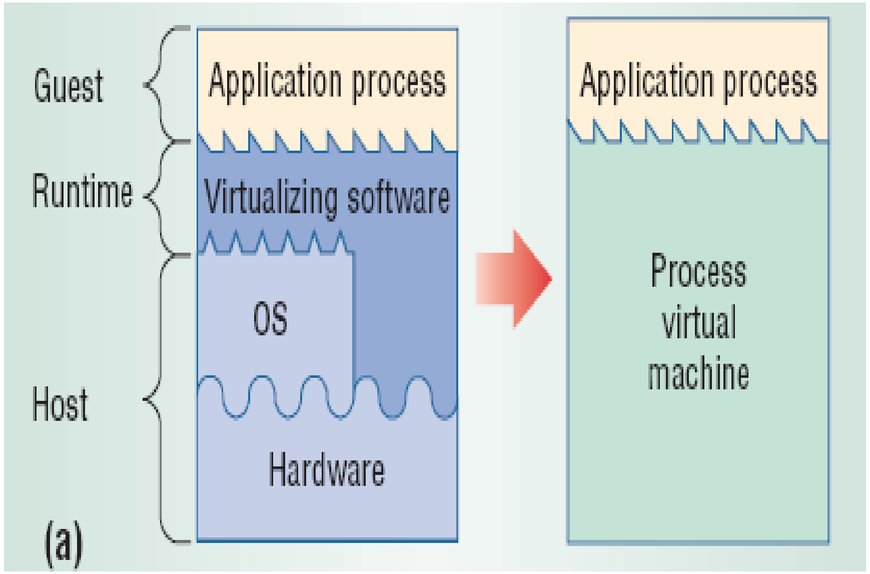
VIRTUAL MACHINE

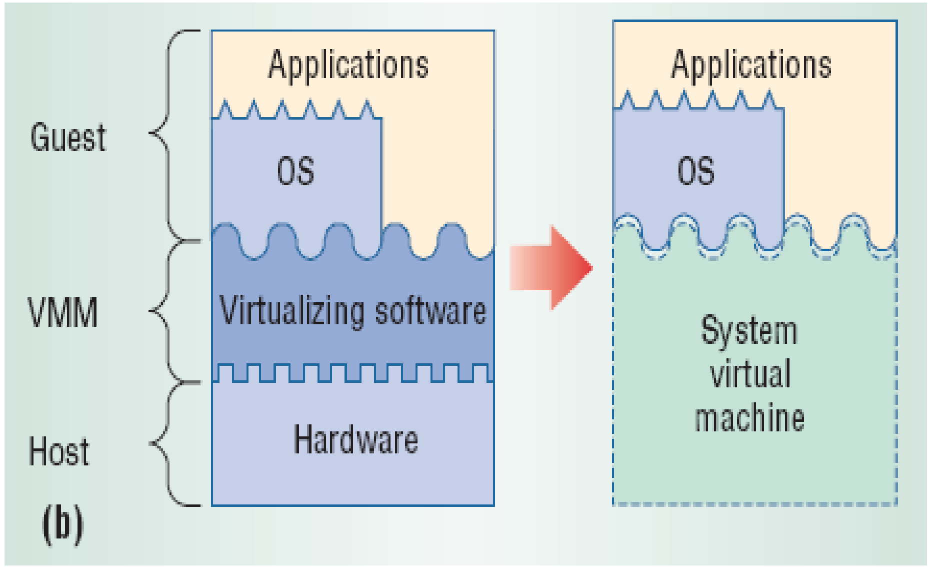
- A virtual machine (VM) provides a complete Programming environment
- Virtual machines are "an efficient, isolated duplicate of a real machine"
- Two types :
  - process VM and
  - system VM
- JVM is an example of process VM

COMPUTER SYSTEM



VIRTUAL MACHINE

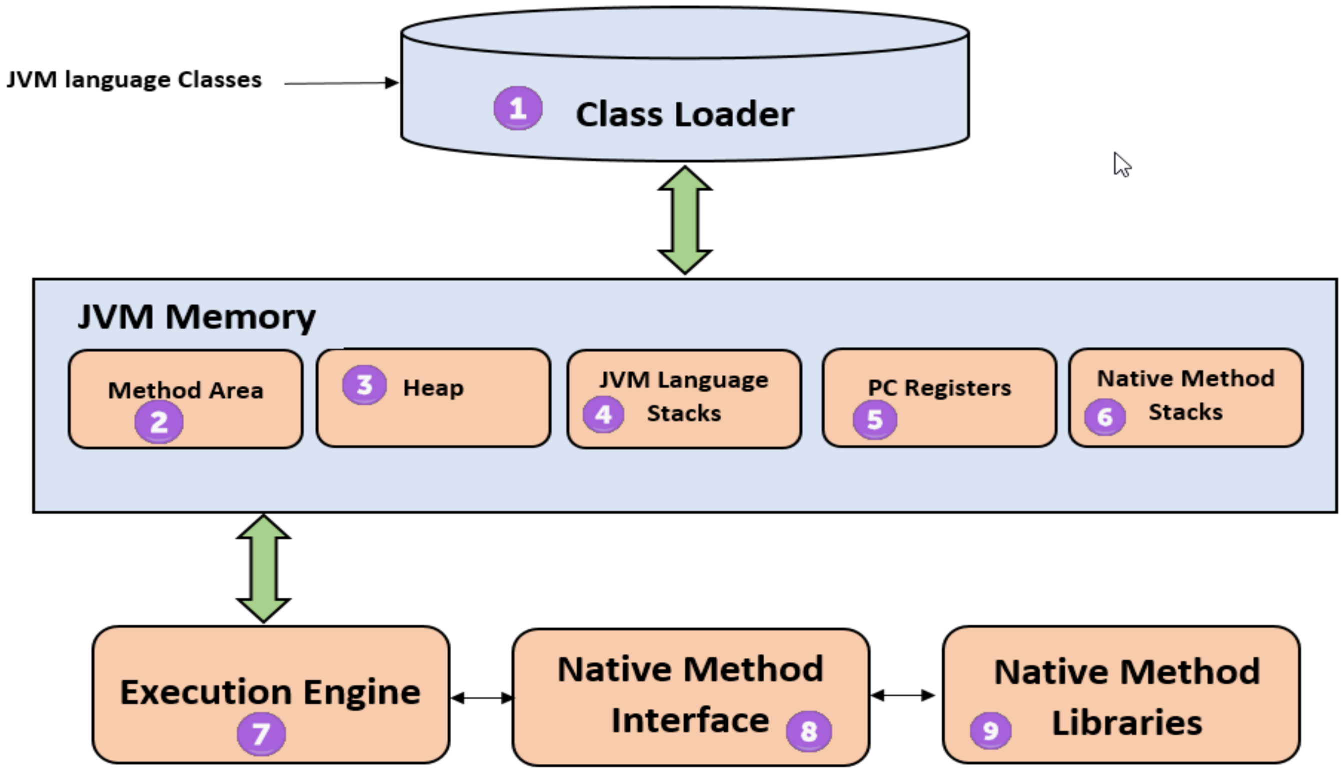




Java Virtual Machine (JVM)

- JVM is a process Virtual machine , consists of an engine that drives the java code.
- Java compiler javac transforms a .java source file into a .class file that is written in Java bytecode, which is the machine language for an imaginary machine known as the Java Virtual Machine.
- Java is called platform independent because of Java Virtual Machine.
- As different computers with the different operating system have their own JVM, when we submit a .class file to any operating system, **JVM interprets the bytecode into machine level language.**

JVM ARCHITECTURE



JVM ARCHITECTURE : Class Loader

- subsystem used for loading class files.
- It performs three major functions
  - Loading
  - Linking
  - Initialization.

JVM ARCHITECTURE : Class Loader

Loading :

- The Class loader reads the .class file, generate the corresponding binary data and save it in method area.
- For each .class file, JVM stores following information in method area.
  - Fully qualified name of the loaded class and its immediate parent class.
  - Whether .class file is related to Class or Interface
  - Modifier, Variables and Method information etc.

JVM ARCHITECTURE : Class Loader

Loading :

- After loading .class file, JVM creates an object of type Class to represent this file in the heap memory.
- this object is of type Class predefined in java.lang package.
- This Class object can be used by the programmer for getting class level information like name of class, parent name, methods and variable information etc.
- To get this object reference we can use getClass() method of Object class.

JVM ARCHITECTURE : Class Loader

- **Linking** : Performs verification, preparation

- Verification : It ensures the correctness of .class file i.e. it check whether this file is properly formatted and generated by valid compiler or not. If verification fails, we get run-time exception java.lang.VerifyError.
- Preparation : JVM allocates memory for class variables and initializing the memory to default values.

JVM ARCHITECTURE : Class Loader

Initialization :

- In this phase, all static variables are assigned with their values defined in the code and static block(if any).
- This is executed from top to bottom in a class and from parent to child in class hierarchy.

JVM ARCHITECTURE : Method Area

- All class level information like
  - class name
  - immediate parent class name
  - methods and variables information etc. are stored, including static variables.
- There is only one method area per JVM, and it is a shared resource.

JVM ARCHITECTURE : Heap area

- Information of all objects is stored in heap area.
- There is also one Heap Area per JVM.
- It is also a shared resource.

JVM ARCHITECTURE : Stack area

- For every thread, JVM create one run-time stack which is stored here.
- Every block of this stack is called activation record/stack frame which store methods calls.
- All local variables of that method are stored in their corresponding frame.
- After a thread terminate, it's run-time stack will be destroyed by JVM.
- It is not a shared resource.

JVM ARCHITECTURE : PC Registers

- Stores the address of next instruction to be executed ,of a thread.
- Each thread has separate PC Registers.

JVM ARCHITECTURE : Native Method Stacks

- Native method stacks **hold the instruction of native code** depends on the native library.
- It is written in another language instead of Java.

JVM ARCHITECTURE : Execution Engine

- Execution engine execute the .class (bytecode).
- It reads the byte-code line by line, use data and information present in various memory area and execute instructions.
- It can be classified in three parts

▻ **Interpreter**

▻ It interprets the bytecode line by line and then executes.

▻ The disadvantage here is that when one method is called multiple times, every time interpretation is required.

JVM ARCHITECTURE : Execution Engine

- **Just-In-Time Compiler(JIT)**
- It is used to increase efficiency of interpreter.
- It compiles the entire bytecode and changes it to native code so whenever interpreter see repeated method calls, JIT provide direct native code for that part so re-interpretation is not required, thus efficiency is improved.
- **Garbage Collector**
- It destroy un-referenced objects.

JVM ARCHITECTURE : Java Native Interface (JNI)

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

JVM ARCHITECTURE : Native Method Libraries

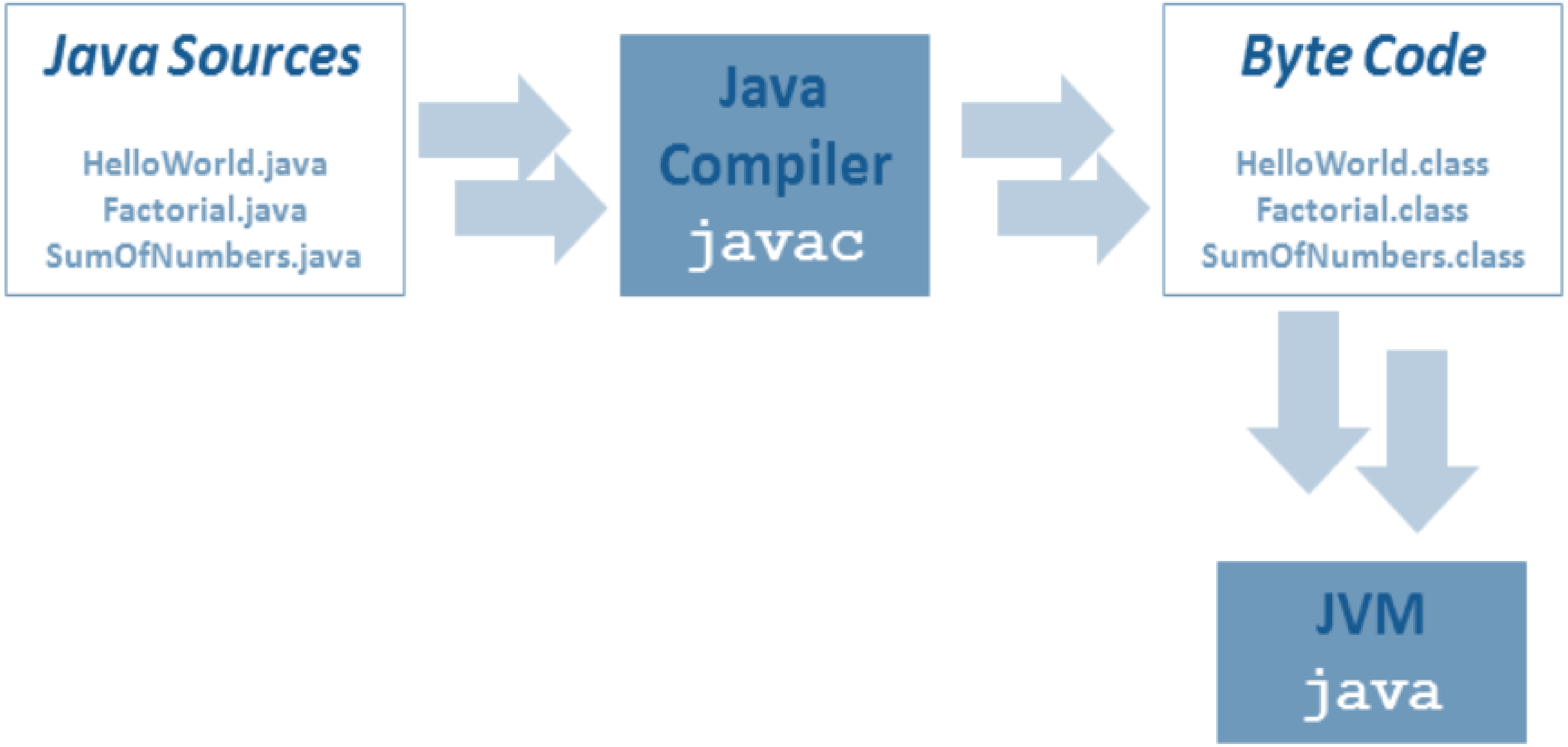
- Native library is a **library that contains native code**
- NATIVE CODE is code that has been compiled for a specific hardware architecture or operating system
- Native library generally means a **non Java library** that's used by the system written in C/C++(Ex DLL or lib)
- It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.
- Java can load these native libraries through JNI

Java Compiler & Bytecode

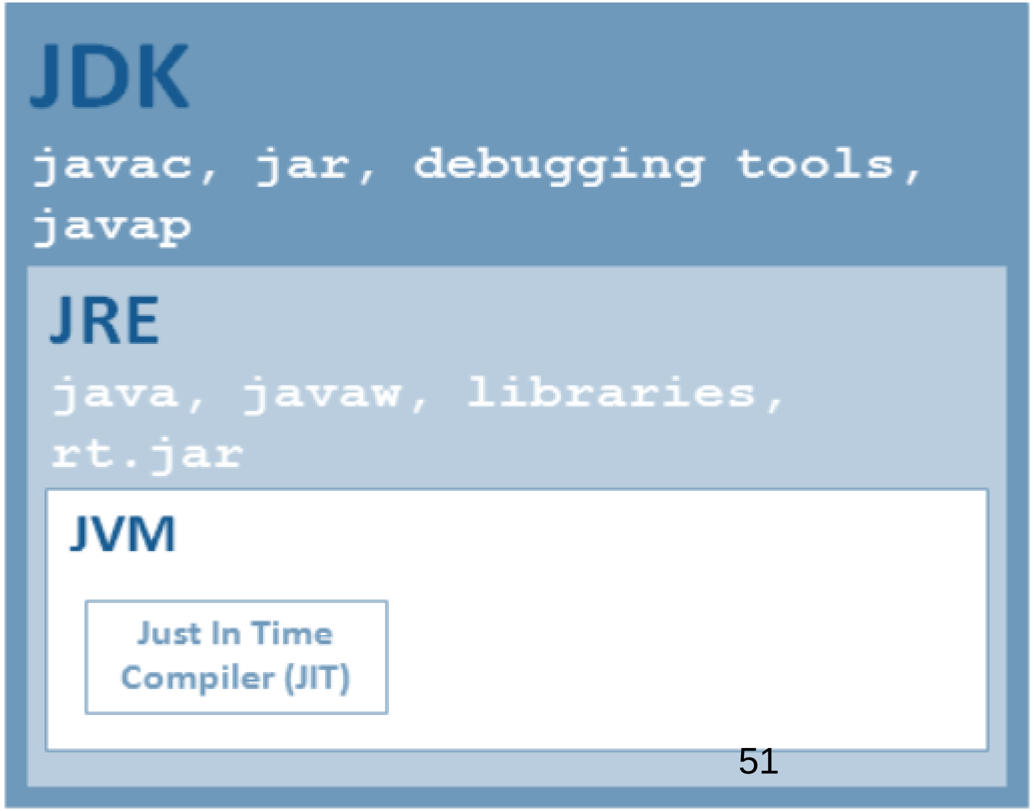
Java Compiler

- A Java compiler is a compiler for the programming language Java.
- The most common form of output from a Java compiler is Java class files containing platform-neutral Java bytecode,
- There are also compilers that output optimized native machine code for a particular hardware/operating system combination.
- The Java virtual machine (JVM) loads the class files and either interprets the bytecode or just-in-time compiler compiles it to machine code and then possibly optimizes it using dynamic compilation.

Java compiler



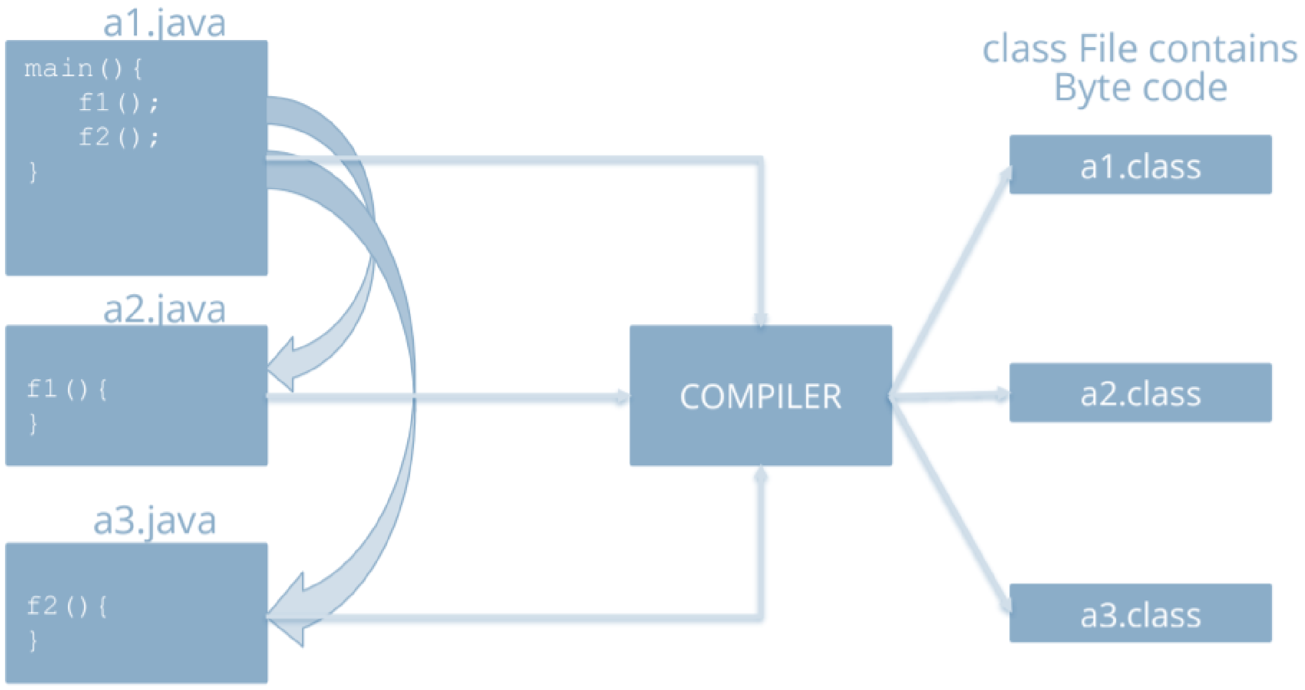
- Java is distributed in two packages - JDK and JRE.
- When JDK is installed it also contains JRE, JVM and JIT apart from the compiler,and debugging tools.
- When JRE is installed it contains the JVM and JIT and the class libraries.


JDK
javac, jar, debugging tools, javap
JRE
java, javaw, libraries, rt.jar
JVM
Just In Time Compiler (JIT)

Bytecode

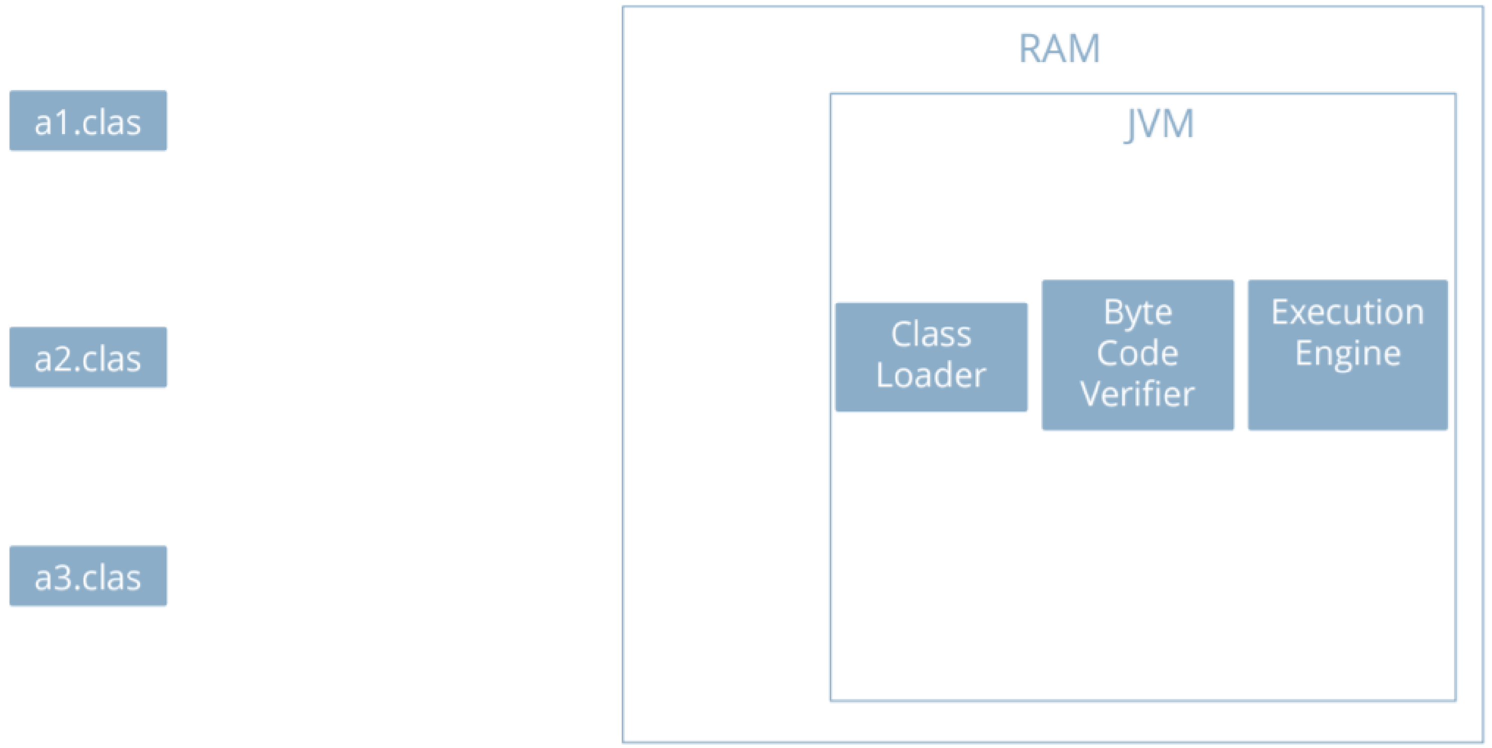
- Output from the compiler are called as bytecodes.
- Bytecode is a highly optimized set of instructions designed to be executed by the JVM.
- original JVM was designed as an interpreter for bytecode.The execution of bytecode by the JVM is the easiest way to create truly portable programs.
- The main difference between the machine code and the bytecode is that the machine code is a set of instructions in machine language or binary which can be directly executed by the CPU.
- While the bytecode is a non-runnable code generated by compiling a source code that relies on an interpreter to get executed.

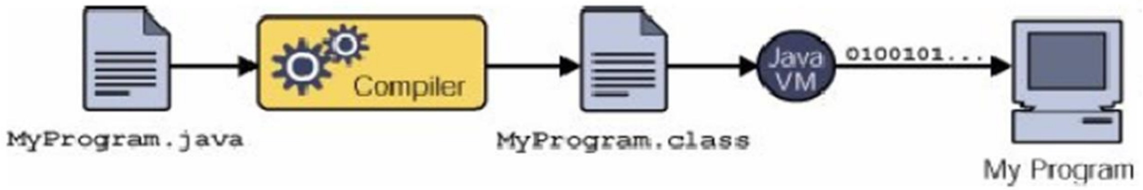
Execution of java

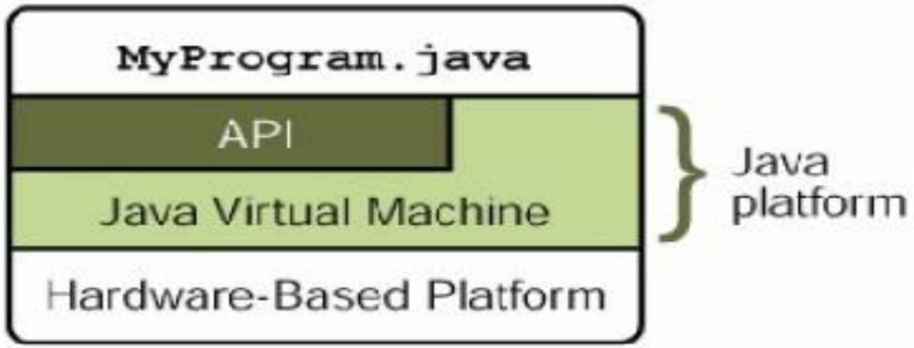


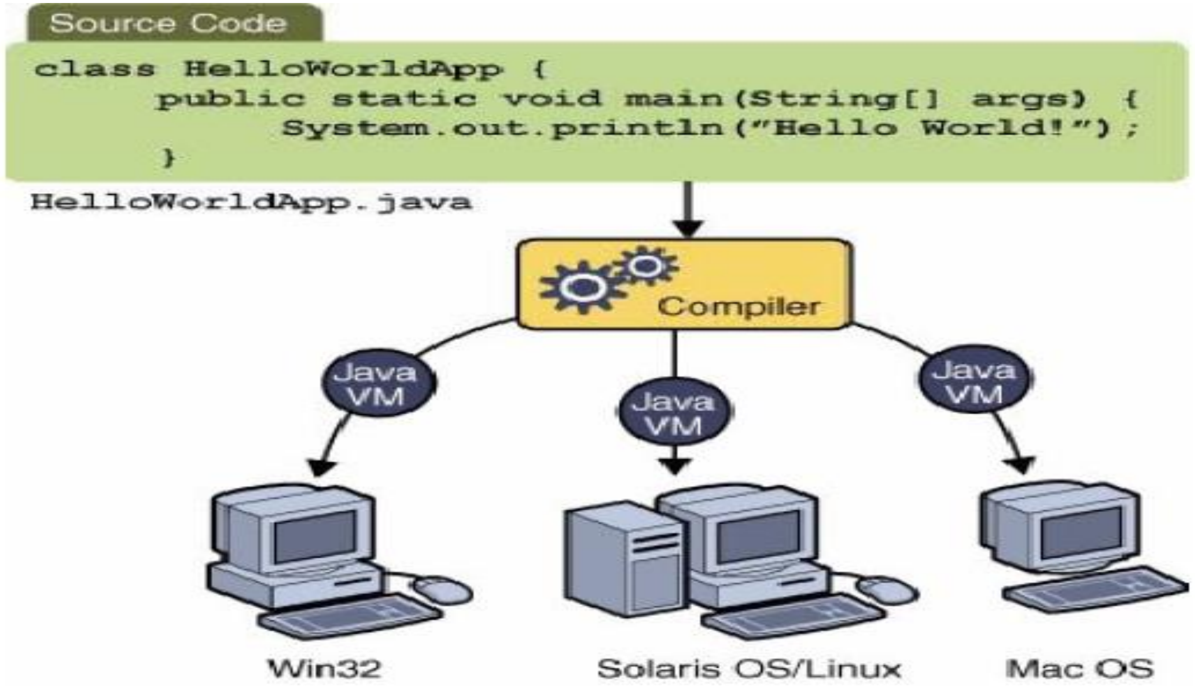
~~No Linking Process~~

Execution of java program




MyProgram.java
Compiler
MyProgram.class
Java VM
0100101...
My Program


MyProgram.java
API
Java Virtual Machine
Hardware-Based Platform
Java platform



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

HelloWorldApp.java

Java Applets

Java Applet

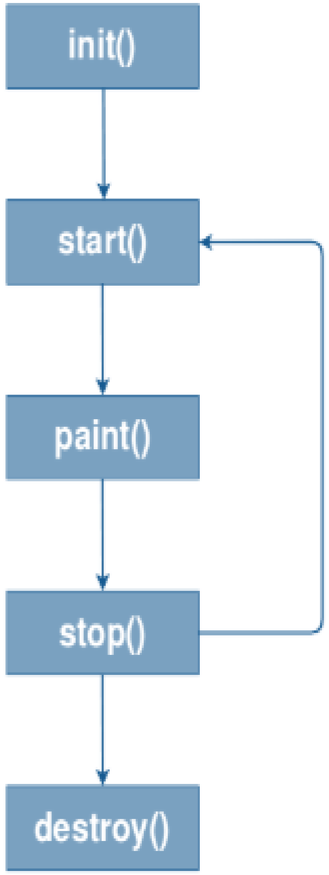
- An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side.
- An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.
- All applets are sub-classes of *java.applet.Applet* class.
- Applets are not stand-alone programs. they run within either a web browser or an applet viewer(JDK provides a standard applet viewer tool called applet viewer).
- Execution of an applet does not begin at main() method.
- Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

Applets

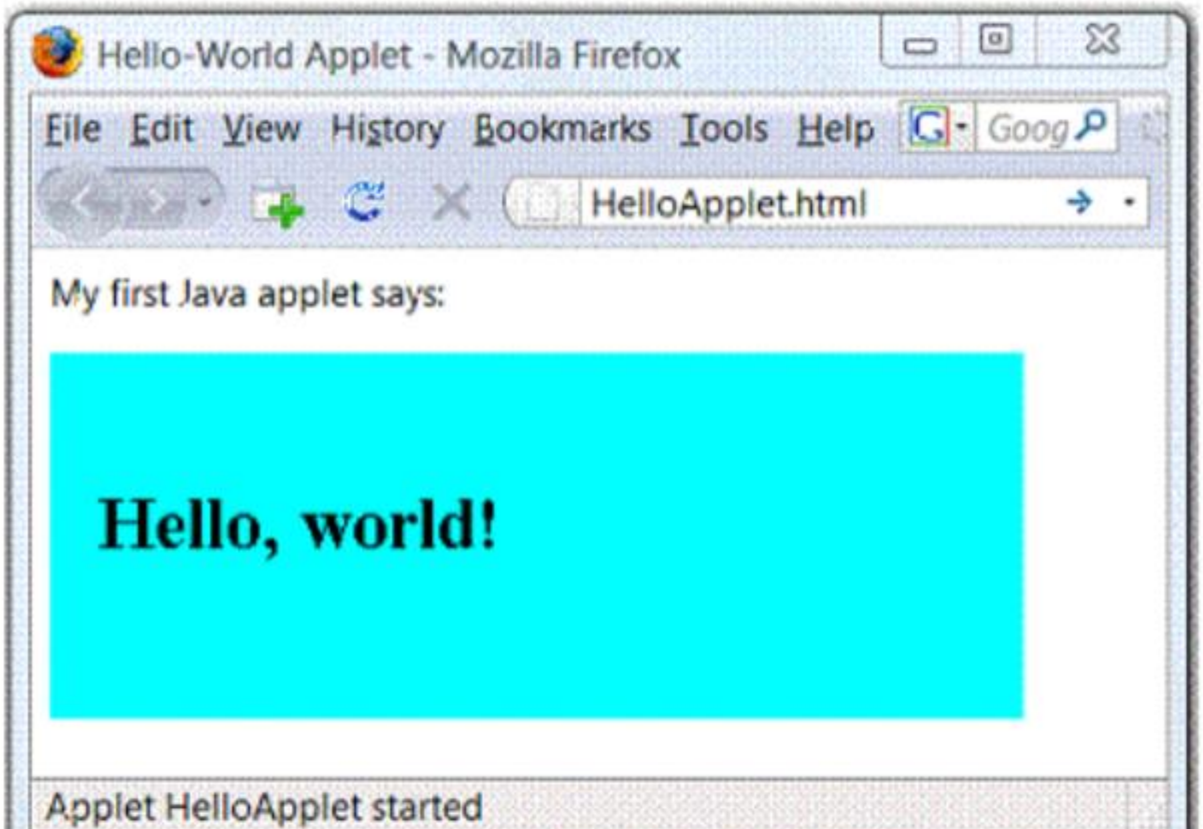
- It runs inside the browser and works at the client side.
- Applets cannot read from or write to the files on the local computer.
- They cannot communicate with other servers on the network.
- They cannot run any programs on the local computer.
- They are restricted from using libraries from other programming languages.

Life cycle of Applet

- When an applet begins, the following methods are called, in this sequence:
- init( )
- start( )
- paint( )
- When an applet is terminated, the following sequence of method calls takes place:
- stop( )
- destroy( )



Hello world output using Applet


Hello-World Applet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Goog
HelloApplet.html

My first Java applet says:

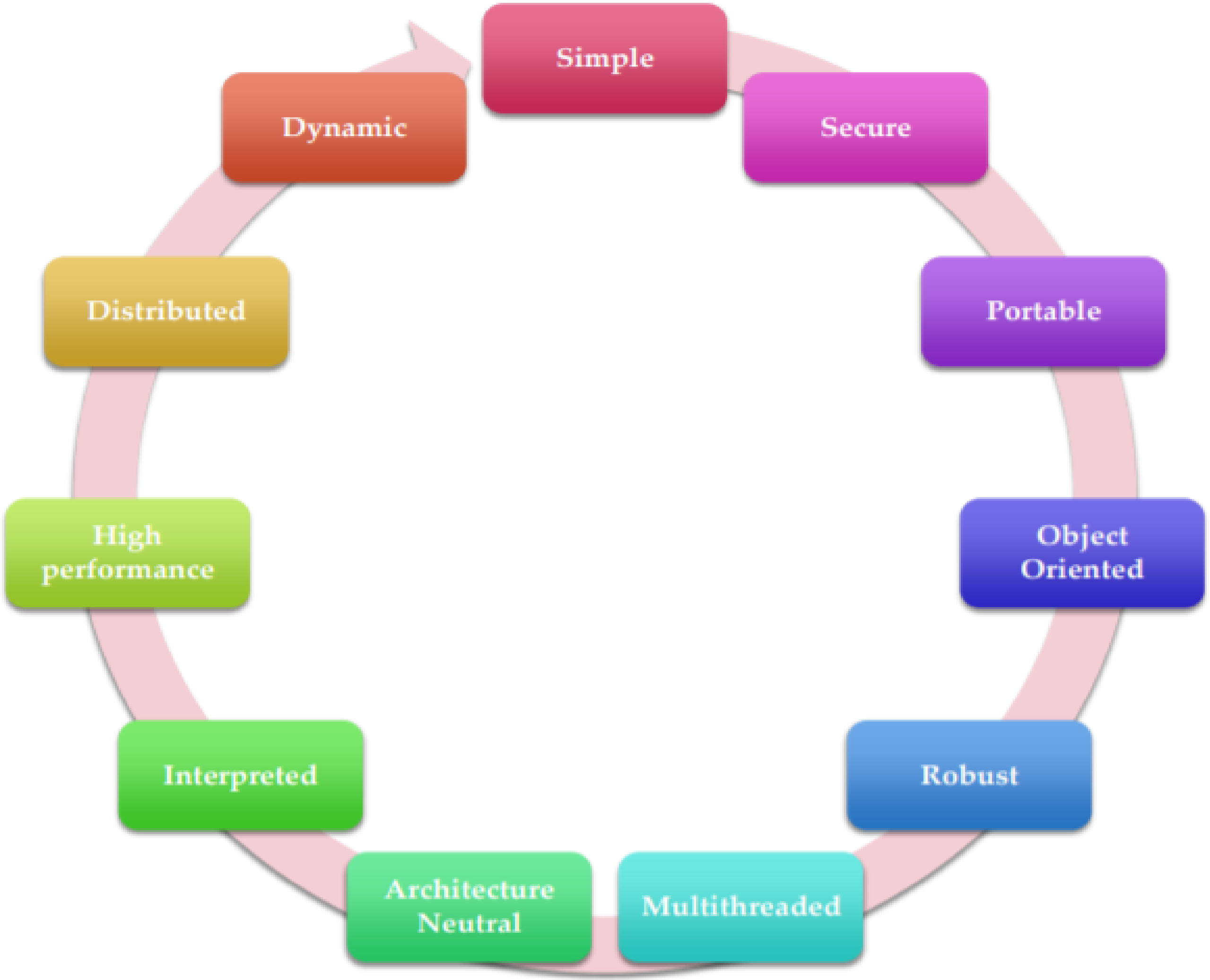
Hello, world!

Applet HelloApplet started

Hello world output using Applet

```
/ A Hello World Applet
// Save file as HelloWorld.java

import java.applet.Applet;
import java.awt.Graphics;

// HelloWorld class extends Applet
public class HelloWorld extends Applet
{
    // Overriding paint() method

    public void paint(Graphics g)
    {
        g.drawString("Hello World", 20, 20);
    }

}
```

Java Buzzwords


Simple
Secure
Portable
Object Oriented
Robust
Multithreaded
Architecture Neutral
Interpreted
High performance
Distributed
Dynamic

Simple

- Java is very easy to learn and understand. its syntax is simple and clean
- It is designed to be easy for the professional programmer to learn and use effectively
- Java syntax is based on C& C++ (so easier for programmers to learn it after C++).
- Moving to java requires very less effort for an experienced C++ programmer.
- ▻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.

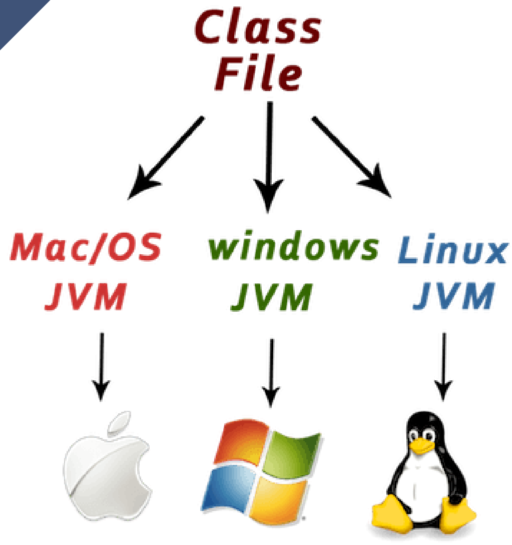
Object-oriented

- Java is an object-oriented programming language.
- To learn java easily , understand the basic concepts of oops.
- 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.

Platform Independent

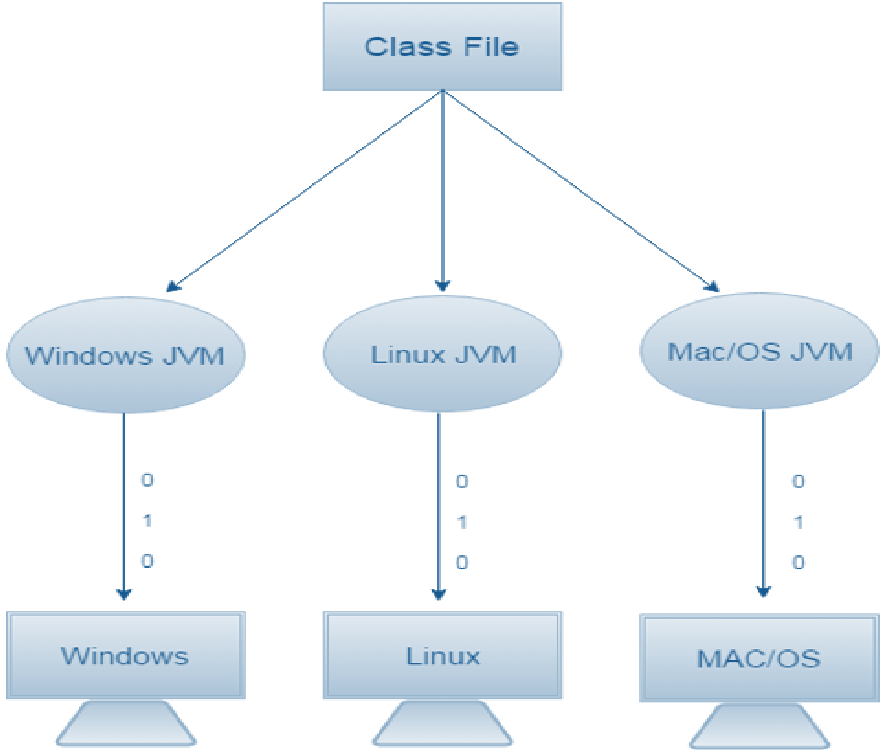
- A platform is the hardware or software environment in which a program runs.
- two types of platforms software-based and hardware-based.
- Java provides software-based platform.
- Java code can be run on multiple platforms with the help of JVM

 e.g. Windows, Linux, Sun Solaris, Mac/OS etc.

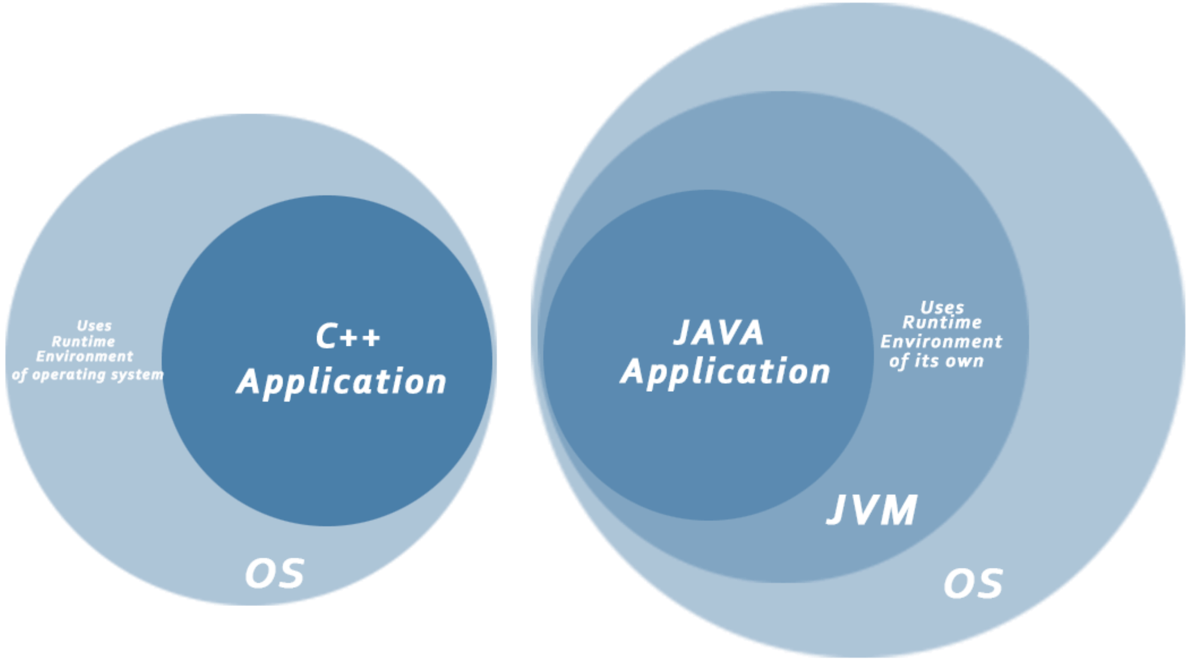


Platform Independent

- 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


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

Secured

- When you download a program , malicious code can cause damage, by gaining unauthorized access to our system resources.
- Java applet prevents such attacks and achieves the protection by confining an applet to the java execution enviornment.

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
- 4 bytes of memory for 64-bit architecture.
- However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Java Features Cont..

Portable

▻Java is portable because it facilitates you to carry the Java bytecode to any platform.

▻It doesn't require to add separate implementation.

High-performance

▻Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code

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 remote machine** on the internet.

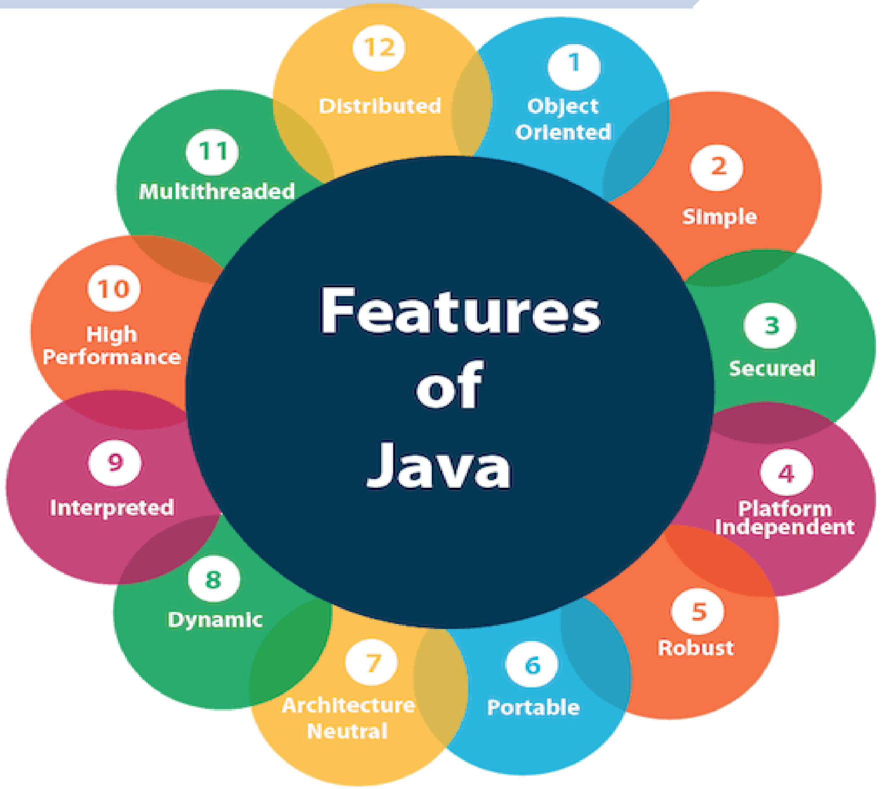
Multi-threaded

- A thread is like a separate program, **executing concurrently**.
- It is a **small java program having single flow of control**.
- 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 (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).

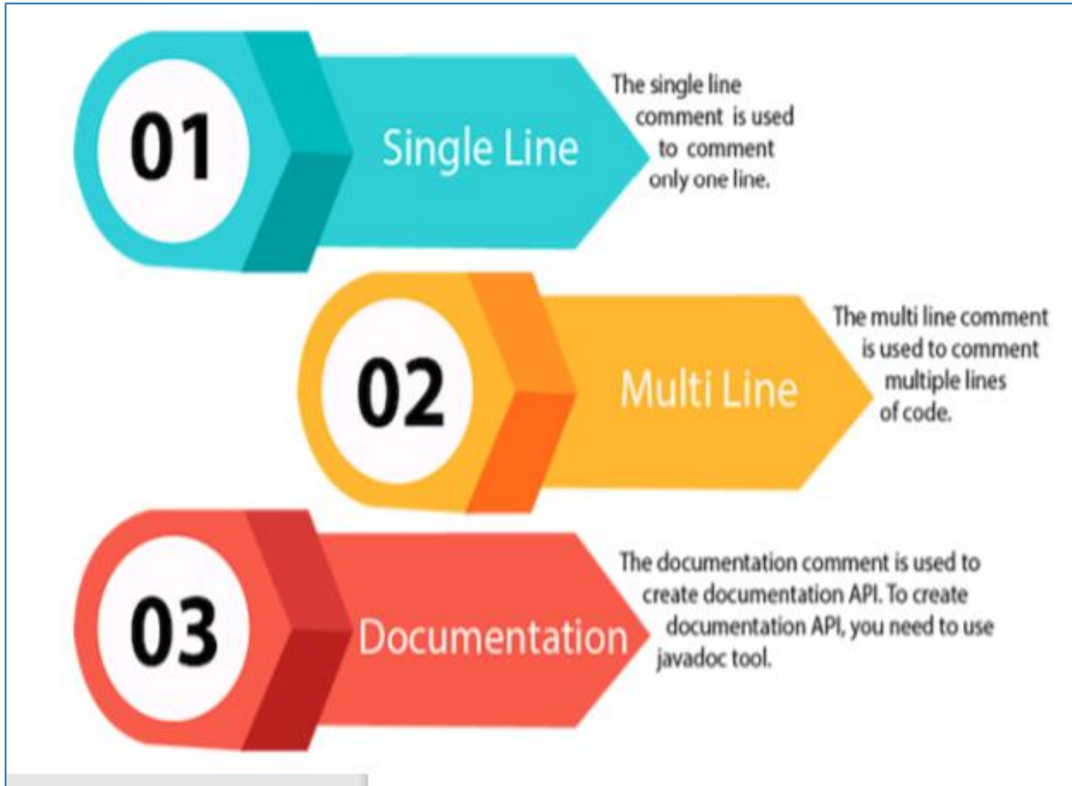
Java Buzzwords-Features of Java



Java Comments

Types of comments

- // Single line
- The compiler ignores everything from // to the end of the line
- /* Multi-line */
- The compiler ignores everything from /* to */.
- /** documentation */
- This indicates a documentation comment

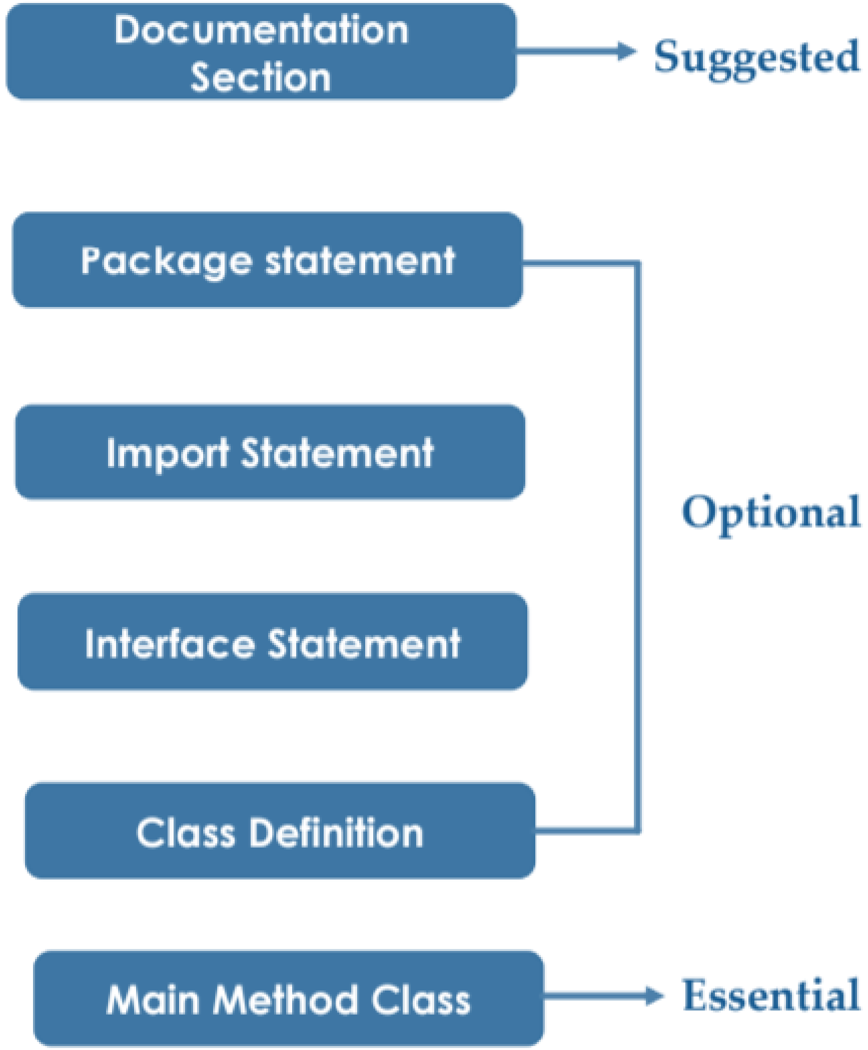

01
Single Line
The single line comment is used to comment only one line.
02
Multi Line
The multi line comment is used to comment multiple lines of code.
03
Documentation
The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool.

Java Program Structure

Program structure

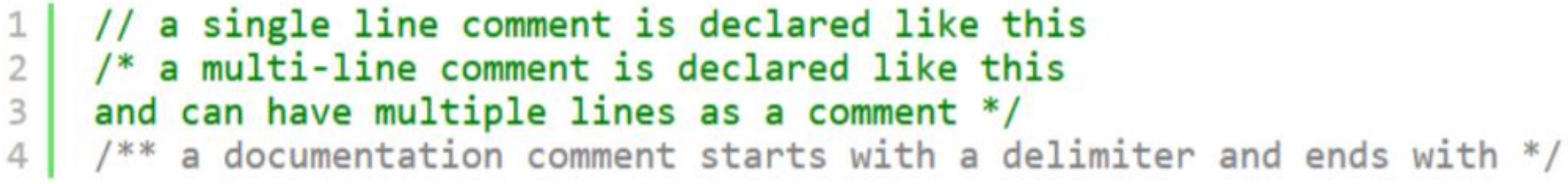
A typical structure of a Java program contains the following elements.

- Package declaration
- Import statements
- Comments
- Class definition
- Class variables, Local variables
- Methods/Behaviors



Documentation Section

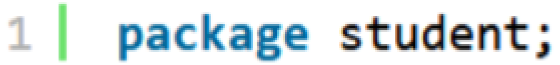
- You can write a comment in this section.
- Comments are beneficial for the programmer because they help them understand the code.
- These are optional, but we suggest you use them because they are useful to understand the operation of the program, so you must write comments within the program.



```
// a single line comment is declared like this
/* a multi-line comment is declared like this
and can have multiple lines as a comment */
/** a documentation comment starts with a delimiter and ends with */
```

Package Statement

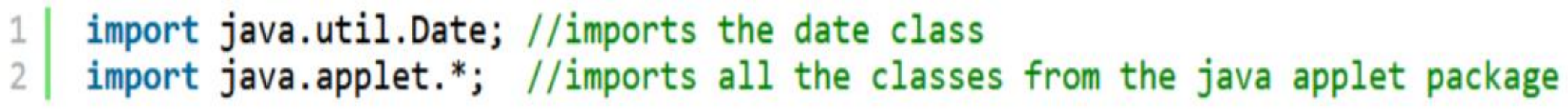
- You can create a package with any name.
- A package is a group of classes that are defined by a name.
- to declare many related classes then declare it within a package.
- It is an optional part of the program, i.e., if you do not want to declare any package, then there will be no problem with it, and you will not get any errors.
- Package is a keyword that tells the compiler that package has been created.
- It is declared as: *package package-name;*



```
package student;
```

Import Statement

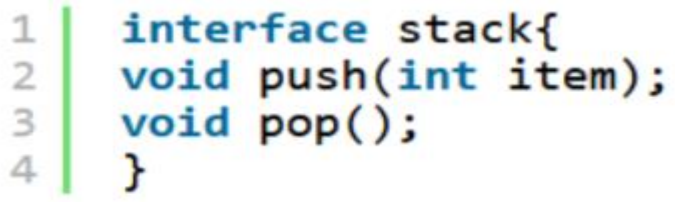
- There can be classes written in other **folders/packages** of our working java project and also there are many classes written by individuals, companies, etc which can be useful in our program.
- To use them in a class, we need to **import** the class that we intend to use.
- Many classes can be imported in a single program and hence multiple import statements can be written.
- E.x. *import java.io.*;*



```
import java.util.Date; //imports the date class
import java.applet.*;  //imports all the classes from the java applet package
```

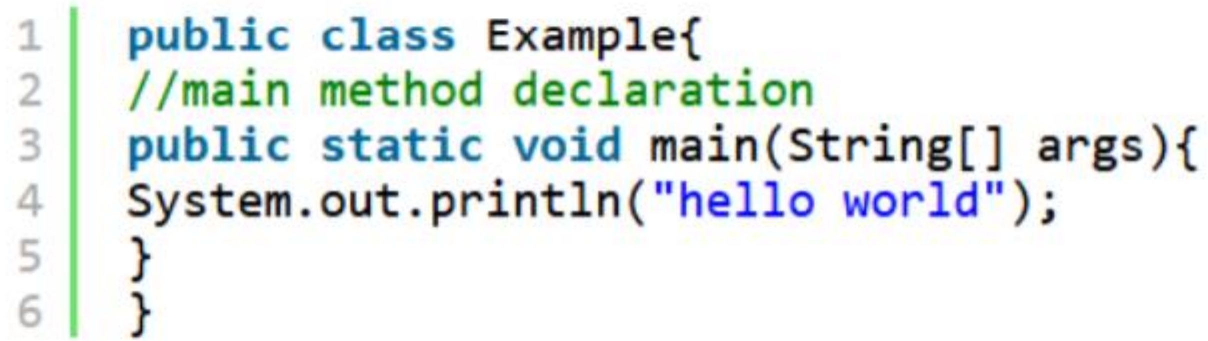
Interface Statement

- Interfaces are like a class that includes a group of method declarations.
- This section is used to specify an interface in Java.
- It is an optional section which is mainly used to implement multiple inheritance
- An interface is a lot similar to a class in Java but it contains only abstract methods and final variables
- An interface cannot be instantiated but it can be implemented by classes or extended by other interfaces.



```
interface stack{
void push(int item);
void pop();
}
```

Class Defintion



```java
public class Example{
//main method declaration
public static void main(String[] args){
System.out.println("hello world");
}
}
```

- A Java program may contain several class definitions.
- Classes are the main and essential elements of any Java program.
- A name should be given to a **class** in a java file and class name starts with a capital letter, and the word public means it is accessible from any other classes..
- This name is used while creating an **object of a class**, in other classes/programs.
- A class is a collection of variables and methods that operate on the fields.
- Every program in Java will have at least one class with the main method.

MAIN METHOD CLASS

- Since every Java stand-alone program requires the main method as the starting point of the program.
- This class is essentially a part of Java program.
- A simple Java program contains only this part of the program.

SAMPLE CODE OF JAVA “HELLO JAVA” PROGRAM

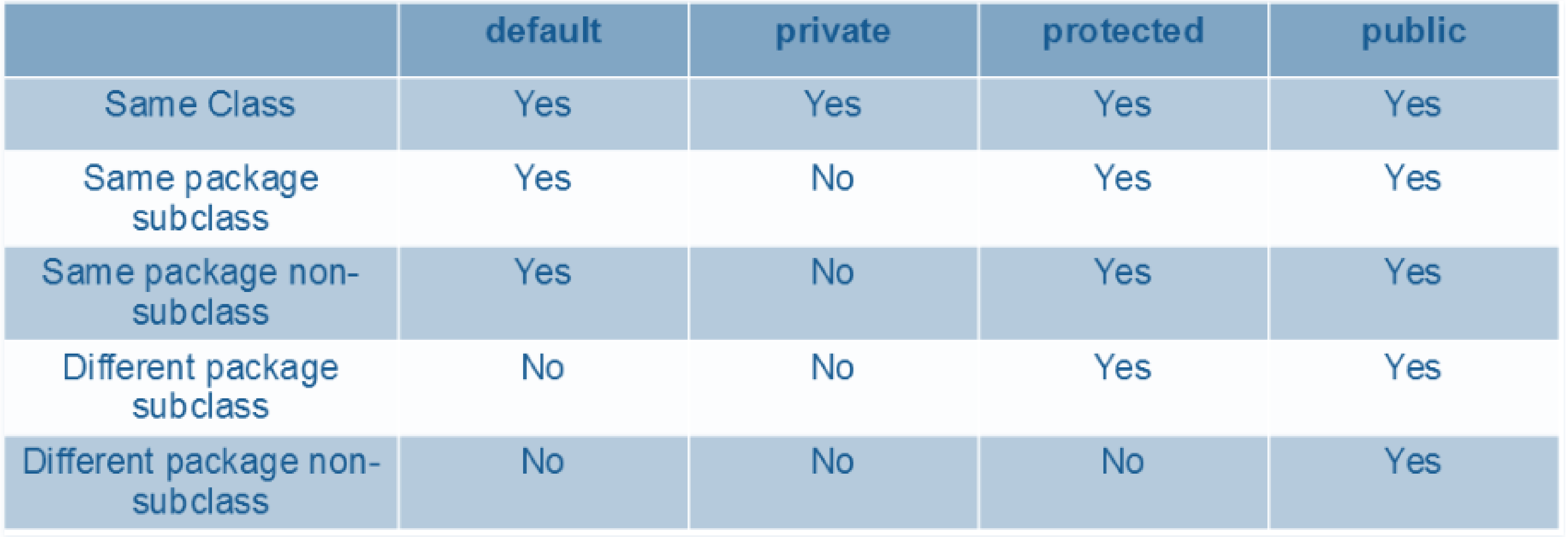
```
public class Hello
{
            /* Author: SJCET Palai
           Date: 4-09-2020
           Description: Display "Hello Java" on the screen */
                    public static void main(String[] args)
                     {
                              System.out.println("Hello Java");
                    }
}
```

- Class can be created by the keyword ‘class’ followed by classname .
- Class name may be preceded by access specifier,that allows the programmer to control the accessibility of the class members .
- By default it is default-within the class & its package.
- only public , abstract and final keywords allowed.
- **static** keyword allows the main() to be called by the java interpreter before any objects are made.

Access Modifiers in Java

- There are four types of access modifiers available in java:
- Default – No keyword required-within the package& outside the class
- Private –accessed within the class
- Protected-accessed within the immediate descendants.
- Public-accessed anywhere

Access Modifiers in Java



	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

public static void main()

- When the main method is declared public, it means that it can also be used by code outside of its class.
- The word static used when we want to access a method without creating its object, as we call the main method, before creating any class objects.
- The word void indicates that a method does not return a value. main( ) is declared as void because it does not return a value.
- main is a method; this is a starting point of a Java program.
- main method code has been moved to some spaces left.

▻It is called indentation which used to make a program easier to read and understand.

String args[] & System.out

- main() has only one parameter String args[] declares a string array parameter of names args,which stores character strings.
- args receives any command line arguments present when the program is executed.
- println() displays any string and terminates the line.
- but print () displays a string only.
- System is a predefined class that provides access to the system
- out is the object of PrintStream class, println() is the method of PrintStream class.

SAMPLE CODE OF JAVA “HELLO JAVA” PROGRAM

- name the java source file as Hello.java
- In Java, a source file is officially called a *compilation unit. It is a text file that contains one* or more class definitions.
- The Java compiler requires that a source file use the **.java filename**
- name of the class defined by the program is also Hello**.**
- **By** convention, the name of that class should match the name of the file that holds the program.(make sure that the capitalization of the filename matches the class name)

SAMPLE CODE OF JAVA “HELLO JAVA” PROGRAM

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

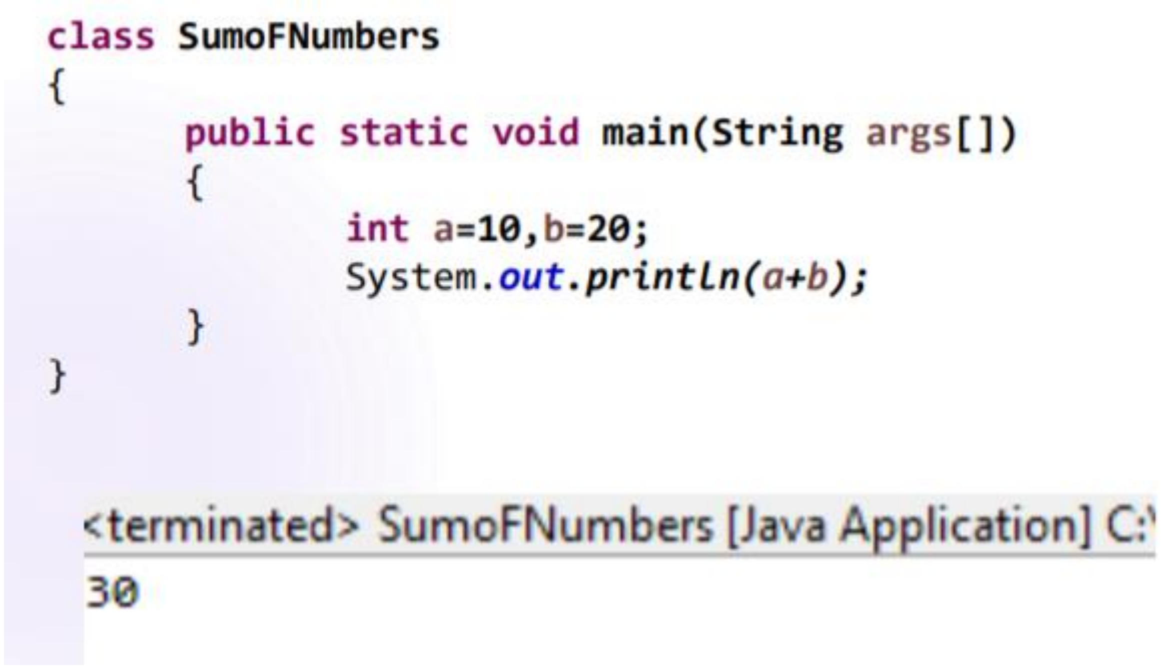
```
C:/>javac Hello.java
```

- The javac compiler creates a file called Hello.class that contains the bytecode version of the program. Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute.
- To run the program, you must use the Java interpreter, called **java.**
- To do so, pass the class name **Example as a command-line argument, as shown here:** `C:/>java Hello`
- When the program is run, the following output is displayed: **Hello Java**

Valid java main method signature

1. **public static void** main(String[] args)
2. **public static void** main(String []args)
3. **public static void** main(String args[])
4. **static public void** main(String[] args)
5. **public static final void** main(String[] args)

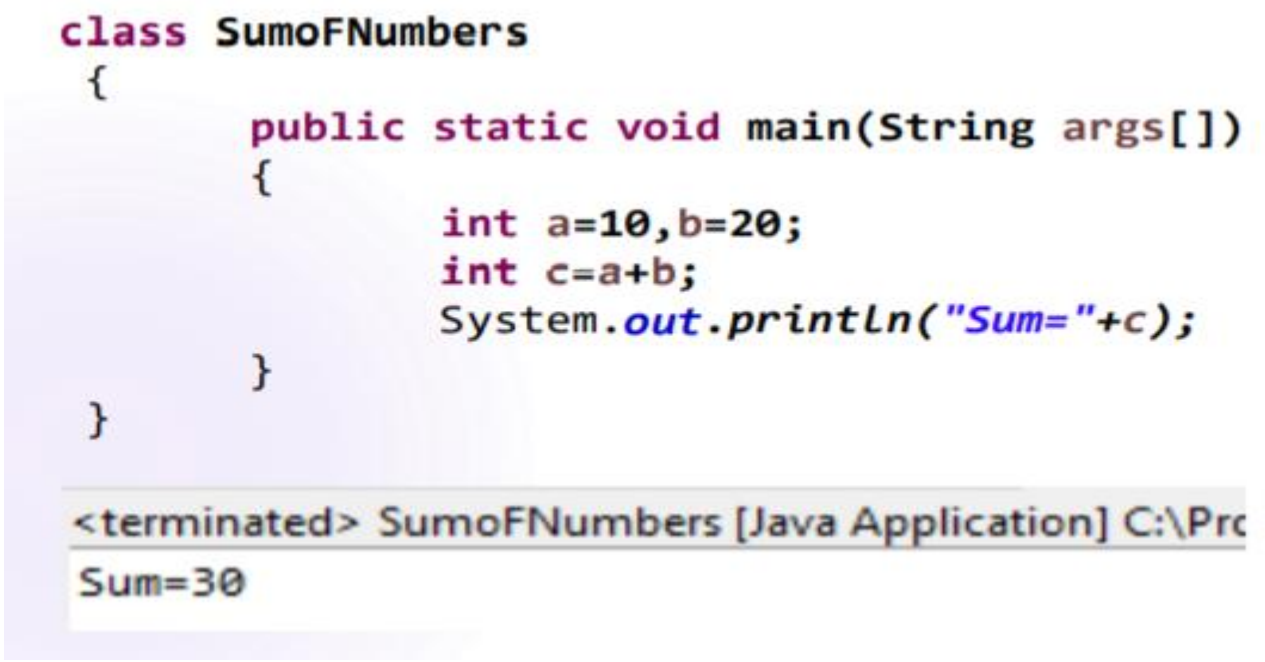
//pgm-sum of two numbers



```
class SumoFNumbers
{
        public static void main(String args[])
        {
                int a=10,b=20;
                System.out.println(a+b);
        }
}
```

```
<terminated> SumoFNumbers [Java Application] C:\
30
```

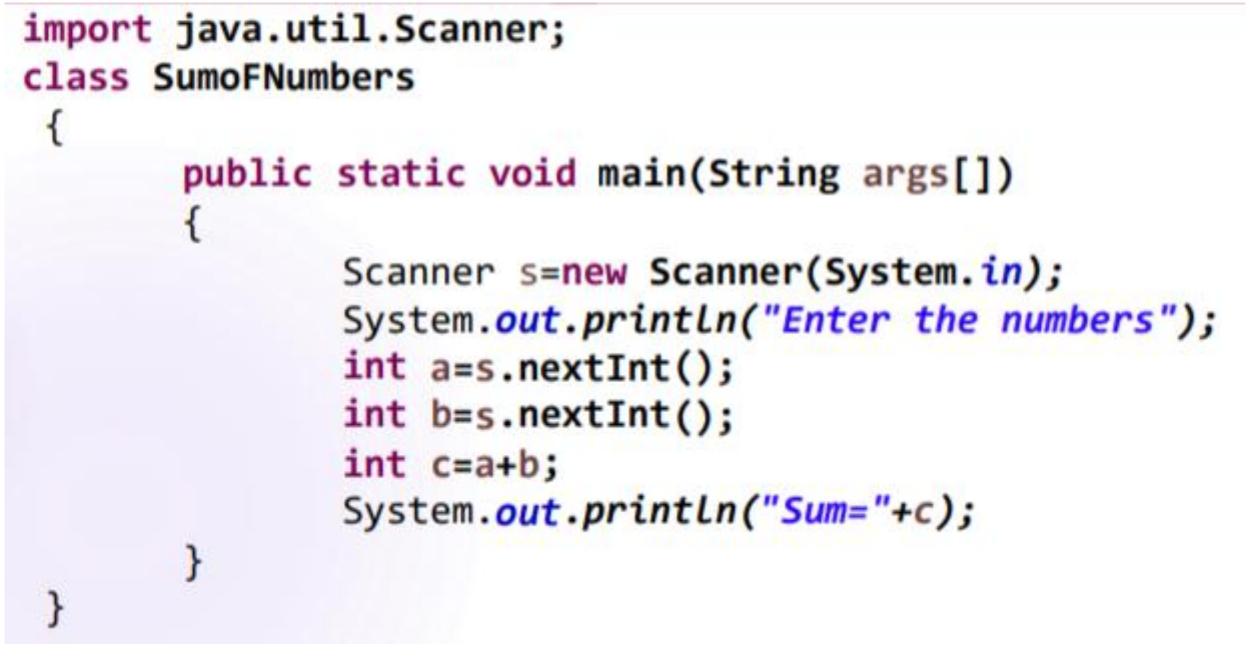
//pgm-sum of two numbers



```
class SumoFNumbers
 {
        public static void main(String args[])
        {
                int a=10,b=20;
                int c=a+b;
                System.out.println("Sum="+c);
        }
 }
```

```
<terminated> SumoFNumbers [Java Application] C:\Prc
Sum=30
```

Sum of two numbers by taking user i/p



```
import java.util.Scanner;
class SumoFNumbers
 {
        public static void main(String args[])
        {
                Scanner s=new Scanner(System.in);
                System.out.println("Enter the numbers");
                int a=s.nextInt();
                int b=s.nextInt();
                int c=a+b;
                System.out.println("Sum="+c);
        }
 }
```

Garbage Collection

Garbage Collection

- Java garbage collection is an automatic process.
- The programmer does not need to explicitly mark objects to be deleted.
- The garbage collection implementation lives in the JVM
- When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program.
- Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

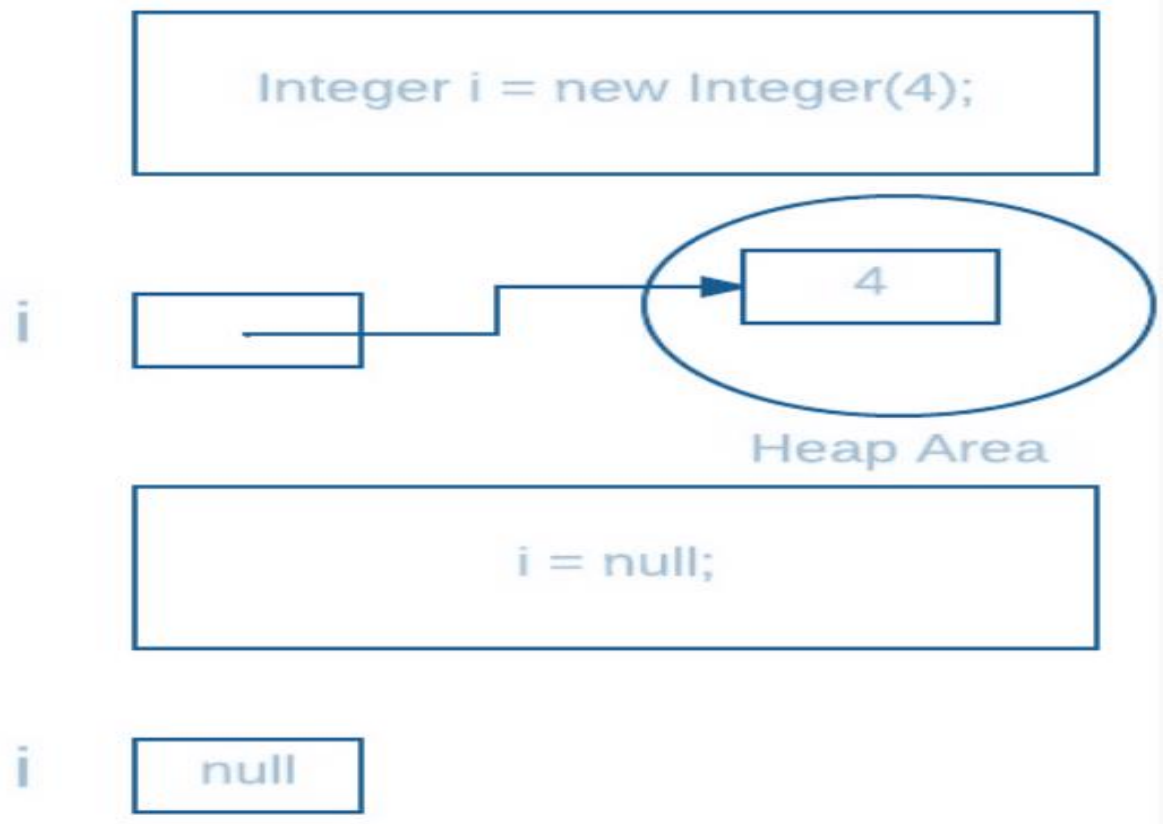
Garbage Collection

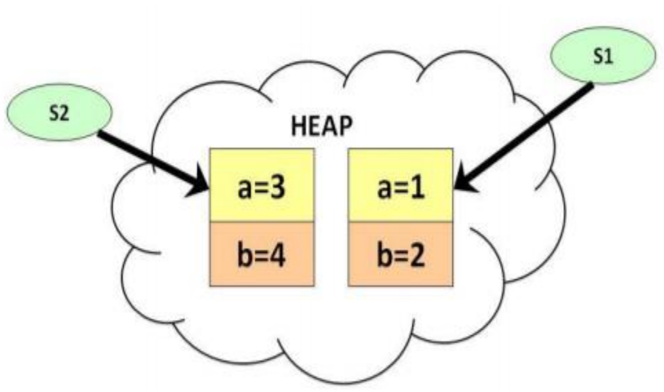
- When JVM starts up, it creates a heap area which is known as runtime data area.
- This is where all the objects (instances of class) are stored.
- Since this area is limited, it is required to manage this area efficiently by removing the objects that are no longer in use.
- The process of removing unused objects from heap memory is known as Garbage collection and this is a part of memory management in Java.
- java perform garbage collection when the object is no longer reachable

Garbage Collection

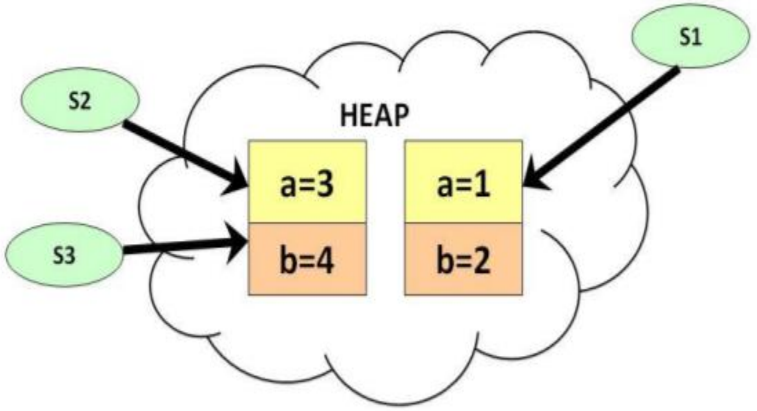
- The biggest benefit of Java garbage collection is that it automatically handles the deletion of unused objects or objects that are out of reach to free up vital memory resources.
- Main objective of Garbage Collector is to free heap memory by destroying **unreachable objects**.
- **Unreachable objects :** An object is said to be unreachable iff it doesn’t contain any reference to it.
- **Eligibility for garbage collection :** An object is said to be eligible for GC(garbage collection) iff it is unreachable

```
Integer i = new Integer(4);
// the new Integer object is reachable  via the reference in 'i'
i = null;
// the Integer object is no longer reachable.
```

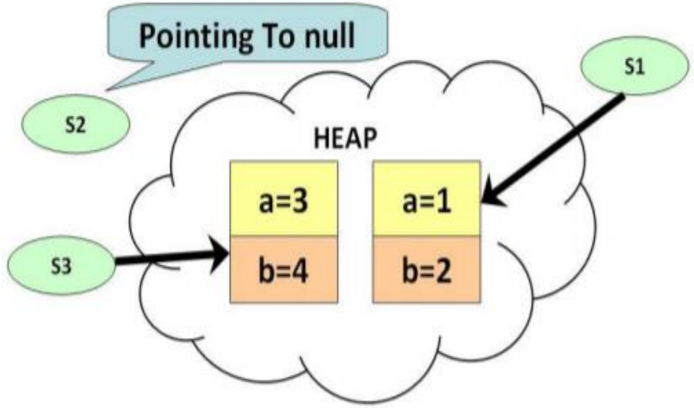




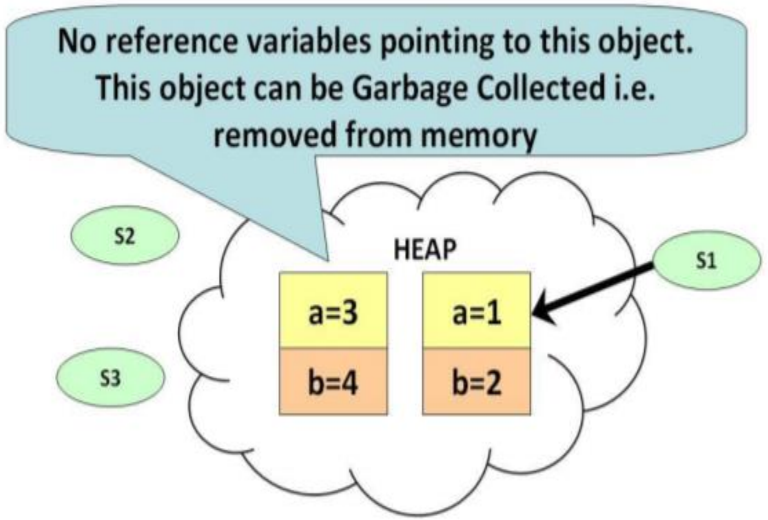
a) Two objects and two reference variables are created.



b) Two reference variables are pointing to the same object



c) s2 becomes null, but s3 is still pointing to the object and is not eligible for java garbage collection


No reference variables pointing to this object. This object can be Garbage Collected i.e. removed from memory
S2
S3
S1
HEAP
a=3
b=4
a=1
b=2

Lexical Issues

Lexical Issues

1 – Whitespaces & Comments

2 - Identifiers

3 - Keywords

4 - Literals

5 - Operators and Miscellaneous Separators

Lexical Issues

- Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.
- White Spaces

► Java is a free form language.

► This means that you do not need to follow any special indentation rules.

► In java , white spaces is a space , tab or new line.

Tokens

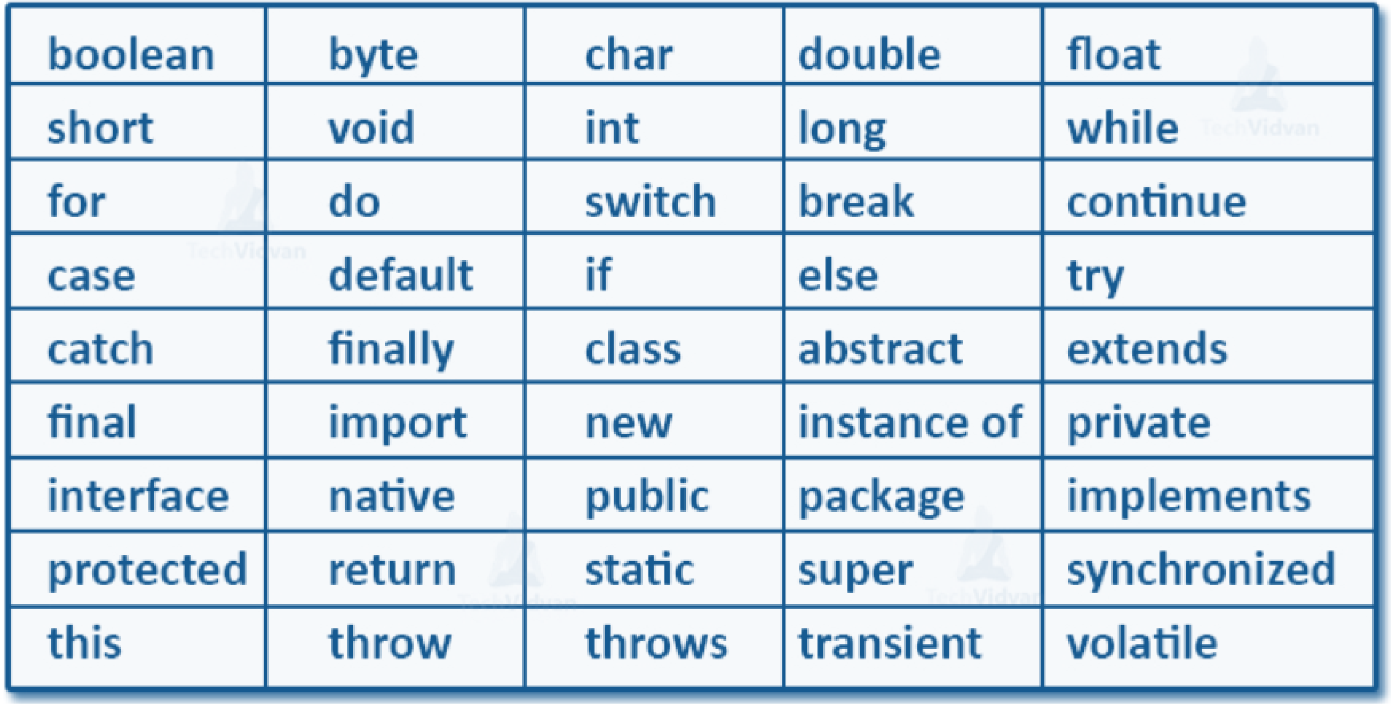
- Java programs are written using the Unicode character set or some character set that is converted to Unicode before being compiled.
- Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages(character holds 2 byte).
- During compilation, the characters in Java source code are reduced to a series of tokens.
- The Java compiler recognizes five kinds of tokens:
  - identifiers, keywords, literals, operators, and miscellaneous separators.
  - Comments and white space such as blanks, tabs, line feeds, and are not tokens, but they often are used to separate tokens.

2 Identifiers

- Identifiers are used for class names , method names and variable names.
- The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0- 9]), ‘$‘(dollar sign) and ‘_’ (underscore).
- Identifiers should not start with digits([0-9]).
- Java identifiers are case-sensitive.
- Reserved Words can’t be used as an identifier There are 53 reserved words in Java.

3.Keywords

The following identifiers are reserved for use as keywords.



boolean	byte	char	double	float
short	void	int	long	while
for	do	switch	break	continue
case	default	if	else	try
catch	finally	class	abstract	extends
final	import	new	instance of	private
interface	native	public	package	implements
protected	return	static	super	synchronized
this	throw	throws	transient	volatile

4.Literals

Literals are the basic representation of any integer, floating point, boolean, character, or string value.

- 4.1 - Integer Literals
- 4.2 - Floating Point Literals
- 4.3 - Boolean Literals
- 4.4 - Character Literals
- 4.5 - String Literals

4.1 Integer Literals

- Integers can be expressed in decimal (base 10), hexadecimal (base 16), or octal (base 8) format.
- A decimal integer literal consists of a sequence of digits without a leading 0 (zero).
- A leading 0 (zero) on an integer literal means it is in octal; Octal integers can include only the digits 0-7.
- a leading 0x (or 0X) means hexadecimal. Hexadecimal integers can include digits (0-9) and the letters a-f and A-F.

- Integer literals are of type int unless they are larger than 32-bits, in which case they are of type long.
- A literal can be forced to be long by appending an L or l to its value.
- The following are all legal integer literals:
- 2, 2L , 0777 , 0xDeadBeef

4.2 Floating Point Literals

A floating point literal can have the following parts:

- a decimal integer,
- a decimal point ("."),
- a fraction (another decimal number),
- an exponent, and a type suffix.
- The exponent part is an e or E followed by an integer, which can be signed.
- A floating point literal must have at least one digit, plus either a decimal point or e/E.

Some examples of floating point literals are:

3.1415 3.1E12 .1e12 2E12

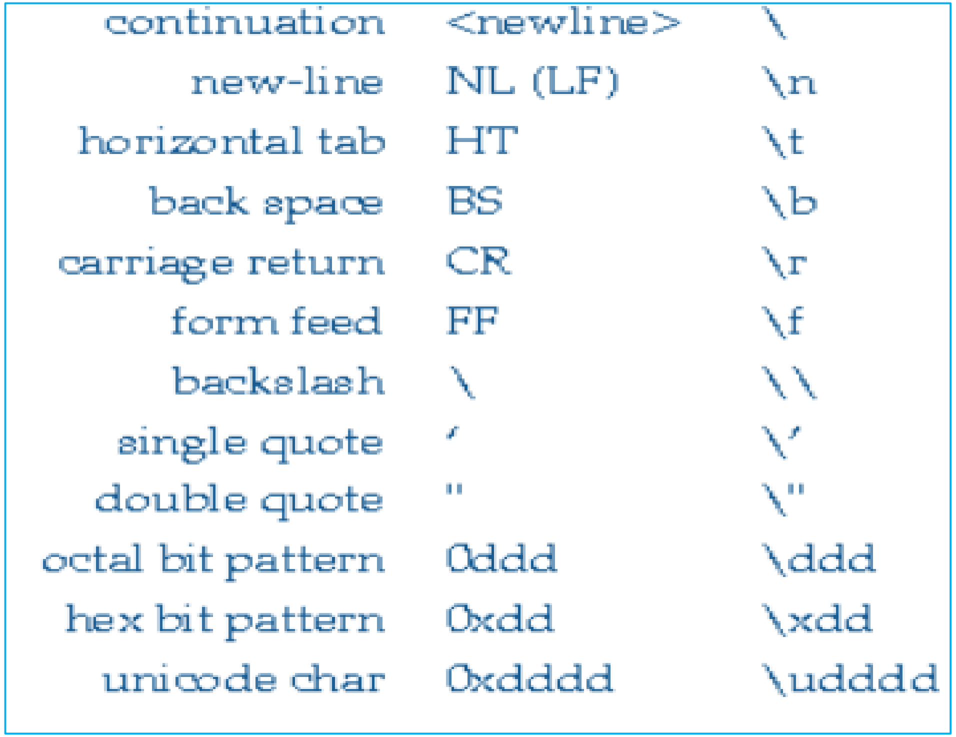
- The Java language has two floating point types:
- float (IEEE 754 single precision) and
- double (IEEE 754 double precision).
- You specify the type of a floating point literal as follows:
- 2.0d or 2.0D   double
- 2.0f or 2.0F or 2.0 float

4.3 BOOLEANS Literals

- Java has a primitive type, called **Boolean** and the type Boolean has two literal values: true and false.
- It can have only one of two possible values, **true** or **false**.
- This is the type returned by all relational operators, as in the case of **a < b**.
- Example: **boolean** b;

4.4 Character Literals

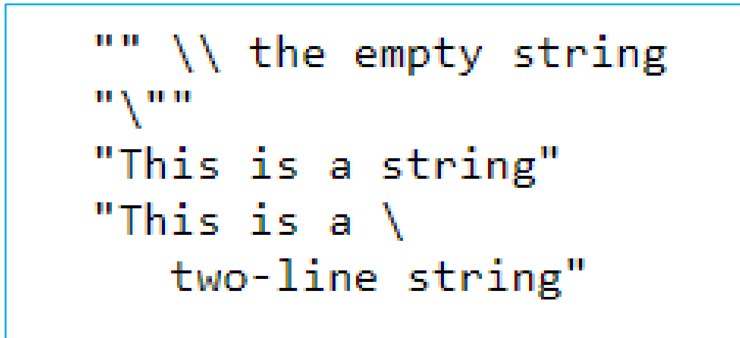
- A character literal is a character (or group of characters representing a single character) enclosed in single quotes.
- Characters have type char and are drawn from the Unicode character set
- The following escape sequences allow for the representation of some non-graphic characters as well as the single quote, “’" and the backslash “\", in Java code:



continuation	<newline>	\
new-line	NL (LF)	\n
horizontal tab	HT	\t
back space	BS	\b
carriage return	CR	\r
form feed	FF	\f
backslash	\	\\
single quote	'	\'
double quote	"	\"
octal bit pattern	0ddd	\ddd
hex bit pattern	0xdd	\xdd
unicode char	0xdddd	\udddd

4.5 String Literals

- A string literal is zero or more characters enclosed in double quotes.
- Each string literal is implemented as a String object (not as an array of characters).
- For example, "abc" creates an new instance of class String.
- The following are all legal string literals:



```
"" \\ the empty string
"\""
"This is a string"
"This is a \
   two-line string"
```

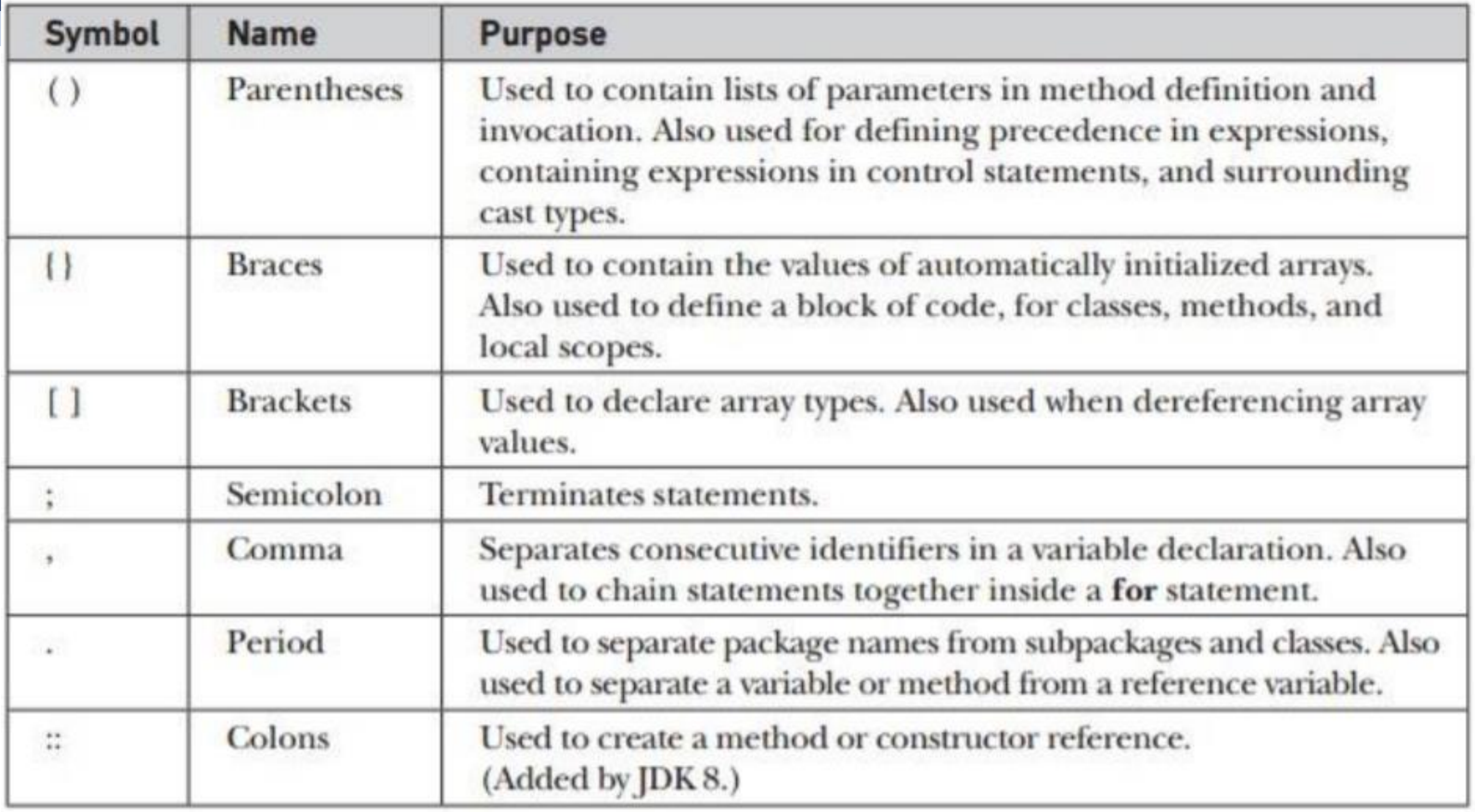
5 Operators and Miscellaneous Separators

- The following characters are used in source code as operators or separators:
  - + - ! % ^ & * | ~ / > <
  - ( ) { } [ ] ; ? : , . =
- In addition, the following character combinations are used as operators:
  - ++ -- == <= >= != << >>
  - >>> += - = *= /= &=|=
  - ^= %= <<= >>= >>>= || &&

OPERATORS

- Operators are special symbols used for: mathematical functions, assignment statements, logical comparisons
- Operators can be divided into the following four groups: arithmetic, bitwise, relational, and logical
- Expressions: can be combinations of variables and operators that result in a value.

seperators



Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference. (Added by JDK 8.)

Thank you

By

PROF.SMITHA JACOB,AP,CSE,SJCET,PALAI