

COURSE INTRODUCTION TO JAVA

B.Tech (CSE)
January, 2025

Nihar Ranjan Roy
Associate Professor,
VIPS School of Engineering & Technology
nihar.roy@vips.edu

VIPS

योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

The Java Programming Language

According to sun Microsystems java is a

- Simple
- Object Oriented
- Distributed
- Multithreaded
- Dynamic,
- Architecture Neutral
- Portable
- High performance,
- Robust and
- Secure

programming language

Brief History of Java-Inception

Java, was developed by **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** at **Sun Microsystems** in 1991. Initially, it was part of the **Green Project**, which aimed to create a platform-independent language for interactive television systems.

Key Milestones:

1. 1991 – The Birth of Java:

- Originally named **Oak** after an oak tree outside James Gosling's office.
- Intended for embedded systems and consumer electronics.

2. 1994 – Transition to the Web:

- Recognizing the potential of the internet, the project shifted focus to developing web-based applications.
- Oak was renamed **Java**, inspired by the coffee consumed by the developers.

3. 1995 – Public Release:

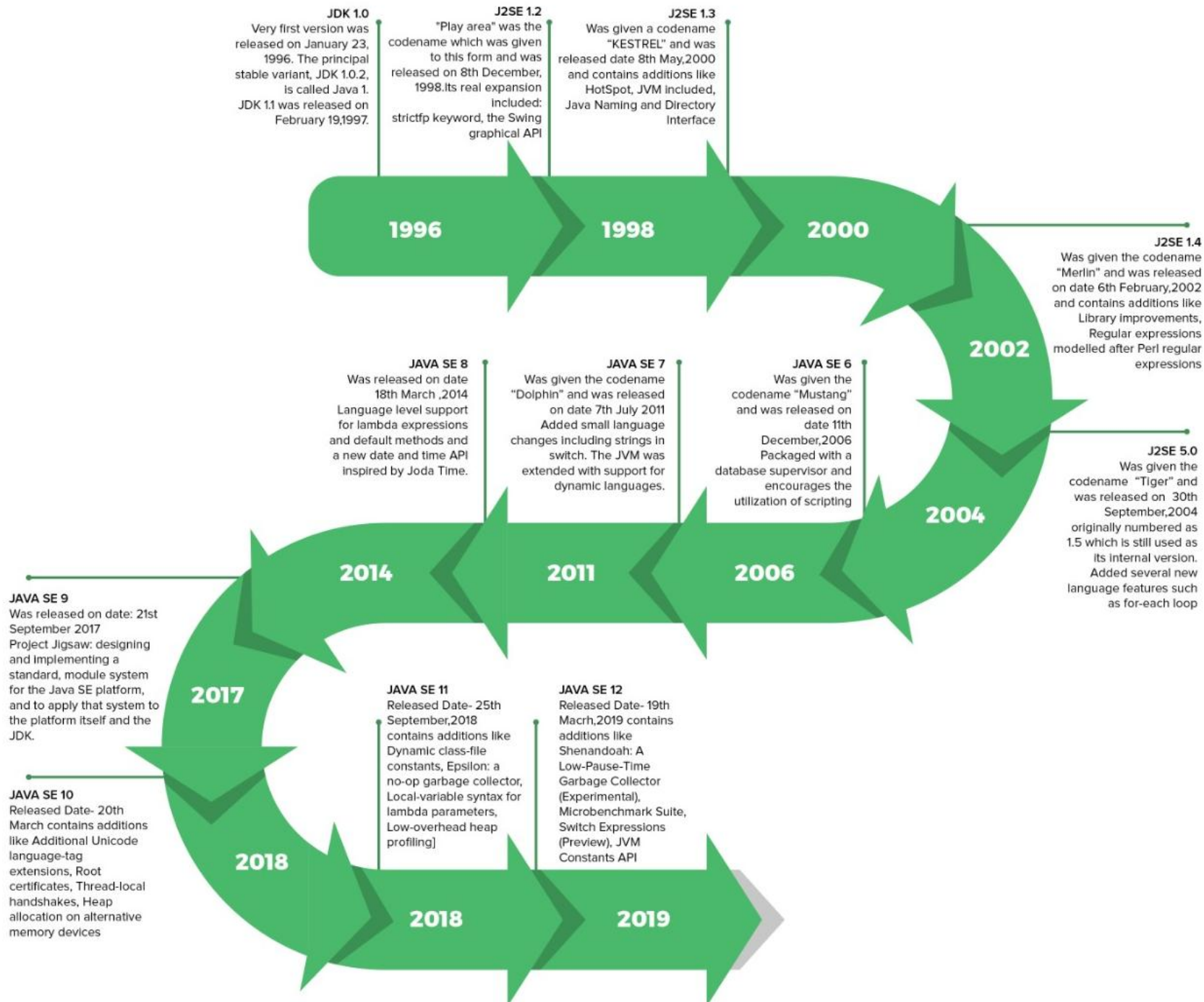
- Java was officially launched with the slogan "**Write Once, Run Anywhere (WORA)**," emphasizing platform independence.
- The first version of Java, **Java 1.0**, was released.

4. 1997 – Standardization:

- Java was standardized by the **Java Community Process (JCP)** to maintain consistency across platforms.

5. Acquisition by Oracle:

- In 2010, **Oracle Corporation** acquired Sun Microsystems, taking over the development and stewardship of Java.



Version	Release Date	Major changes
JDK Beta	1995	
JDK 1.0	Jan-96	The Very first version was released on January 23, 1996. The principal stable variant, JDK 1.0.2, is called Java 1.
JDK 1.1	Feb-97	Was released on February 19, 1997. There were many additions in JDK 1.1 as compared to version 1.0 such as
		A broad retooling of the AWT occasion show
		Inner classes added to the language
		JavaBeans
		JDBC
J2SE 1.2	Dec-98	RMI
		“Play area” was the codename which was given to this form and was released on 8th
		December 1998. Its real expansion included: strictfp keyword
		the Swing graphical API was coordinated into the centre classes
		Sun’s JVM was outfitted with a JIT compiler out of the blue
		Java module
		Java IDL, an IDL usage for CORBA interoperability
		Collections system

J2SE 1.3	May-00	Codename- “KESTREL” Release Date- 8th May 2000 Additions:
		HotSpot JVM included
		Java Naming and Directory Interface
		JPDA
		JavaSound
		Synthetic proxy classes
J2SE 1.4	Feb-02	Codename- “Merlin” Release Date- 6th February 2002 Additions: Library improvements
		Regular expressions modelled after Perl regular expressions
		The image I/O API for reading and writing images in formats like JPEG and PNG
		Integrated XML parser and XSLT processor (JAXP) (specified in JSR 5 and JSR 63)
		Preferences API (java.util.prefs)
		Public Support and security updates for this version ended in October 2008.
J2SE 5.0	Sep-04	Codename- “Tiger” Release Date- “30th September 2004” Originally numbered as 1.5 which is still used as its internal version. Added several new language features such as:
		for-each loop
		Generics
		Autoboxing
		Var-args

JAVA SE 6	Dec-06	Codename- “Mustang” Released Date- 11th December 2006 Packaged with a database supervisor and encourages the utilization of scripting languages with the JVM. Replaced the name J2SE with java SE and dropped the .0 from the version number. Additions:
		Upgrade of JAXB to version 2.0: Including integration of a StAX parser.
		Support for pluggable annotations (JSR 269).
		JDBC 4.0 support (JSR 221)
JAVA SE 7	Jul-11	Codename- “Dolphin” Release Date- 7th July 2011 Added small language changes including strings in the switch. The JVM was extended with support for dynamic languages. Additions:
		Compressed 64-bit pointers.
		Binary Integer Literals.
		Upstream updates to XML and Unicode.
JAVA SE 8	Mar-14	Released Date- 18th March 2014 Language level support for lambda expressions and default methods and a new date and time API inspired by Joda Time.
JAVA SE 9	Sep-17	Release Date: 21st September 2017 Project Jigsaw: designing and implementing a standard, a module system for the Java SE platform, and to apply that system to the platform itself and the JDK.

JAVA SE 10	Mar-18	Released Date- 20th March Addition:
		Additional Unicode language-tag extensions
		Root certificates
		Thread-local handshakes
		Heap allocation on alternative memory devices
		Remove the native-header generation tool – javah.
		Consolidate the JDK forest into a single repository.
JAVA SE 11	Sep-18	Released Date- 25th September, 2018 Additions-
		Dynamic class-file constants
		Epsilon: a no-op garbage collector
		The local-variable syntax for lambda parameters
		Low-overhead heap profiling
		HTTP client (standard)
		Transport Layer Security (TLS) 1.3
		Flight recorder
JAVA SE 12	Mar-19	Released Date- 19th March 2019 Additions-
		Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)
		Microbenchmark Suite
		Switch Expressions (Preview)
		JVM Constants API
		One AArch64 Port, Not Two
		Default CDS Archives

JAVA SE 13	Sep-19	Released Date – 17th September 2019
		Additions-Text Blocks (Multiline strings).
		Switch Expressions.
		Enhanced Thread-local handshakes.
JAVA SE 14	Mar-20	Released Date – 17th March 2020
		Additions-Records (new class type for data modeling).
		Pattern Matching for instanceof.
		Helpful NullPointerExceptions.
JAVA SE 15	Sep-20	Released Date – 15th September 2020
		Additions-Sealed Classes.
		Hidden Classes.
		Foreign Function and Memory API (Incubator).
JAVA SE 16	Mar-21	Released Date – 16th March 2021
		Additions-Records (preview feature).
		Pattern Matching for switch (preview feature).
		Unix Domain Socket Channel (Incubator).
JAVA SE 17	Sep-21	Released Date – 14th September 2021
		Sealed Classes (finalized).
		Pattern Matching for instanceof (finalized).
		Strong encapsulation of JDK internals by default.
		New macOS rendering pipeline.

Version	Type	Class file format version ^[7]	Release date	End of public updates (free)	End of extended support (paid)
Java SE 18	LTS	62	22nd March 2022	September 2022	—
Java SE 19		63	20th September 2022	March 2023	—
Java SE 20		64	21st March 2023	September 2023	—
Java SE 21		65	19th September 2023	September 2028 for Oracle ^[4] September 2028 for Microsoft Build of OpenJDK ^[16] December 2029 for Red Hat ^[10] December 2029 for Eclipse Temurin ^[14] October 2030 for Amazon Corretto ^[15] September 2031 for Azul ^[9]	September 2031 for Oracle ^[4] March 2032 for BellSoft Liberica ^[12]
Java SE 22		66	19th March 2024	September 2024	—
Java SE 23	LTS	67	17th September 2024	March 2025 for Oracle September 2032 for Azul ^[9]	—
Java SE 24		68	March 2025	September 2025	—
Java SE 25		69	September 2025	September 2030 for Oracle ^[4]	September 2033 for Oracle ^[4] March 2034 for BellSoft Liberica ^[12]

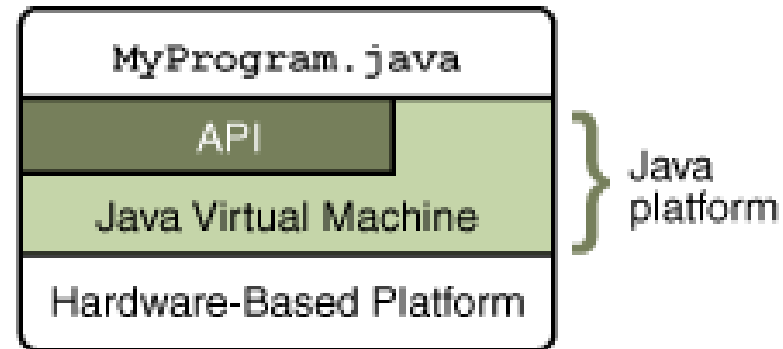
What is a java Platform?

A *platform* is the hardware or software environment in which a program runs.

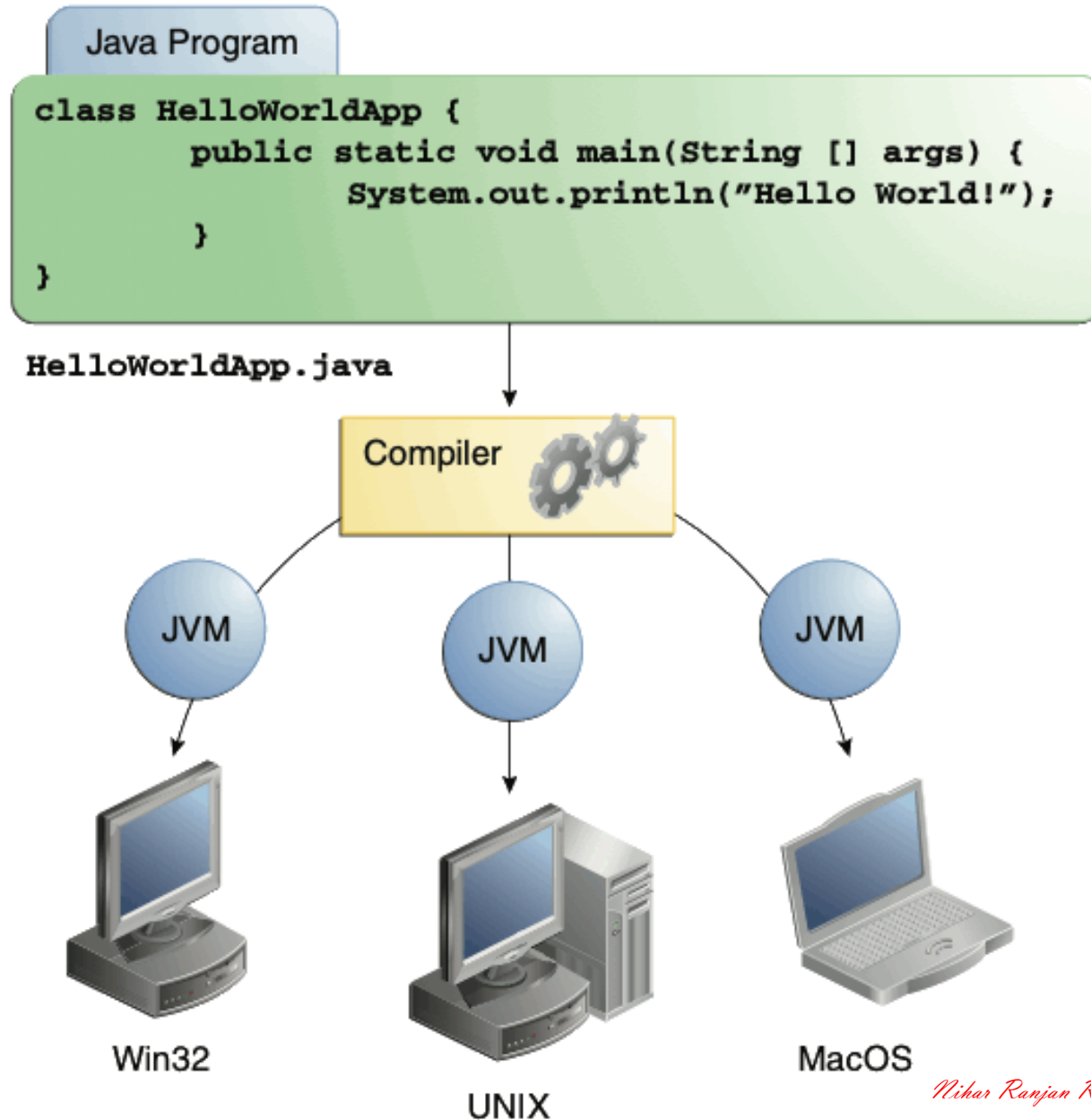
Example of most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS

The Java platform has two components:

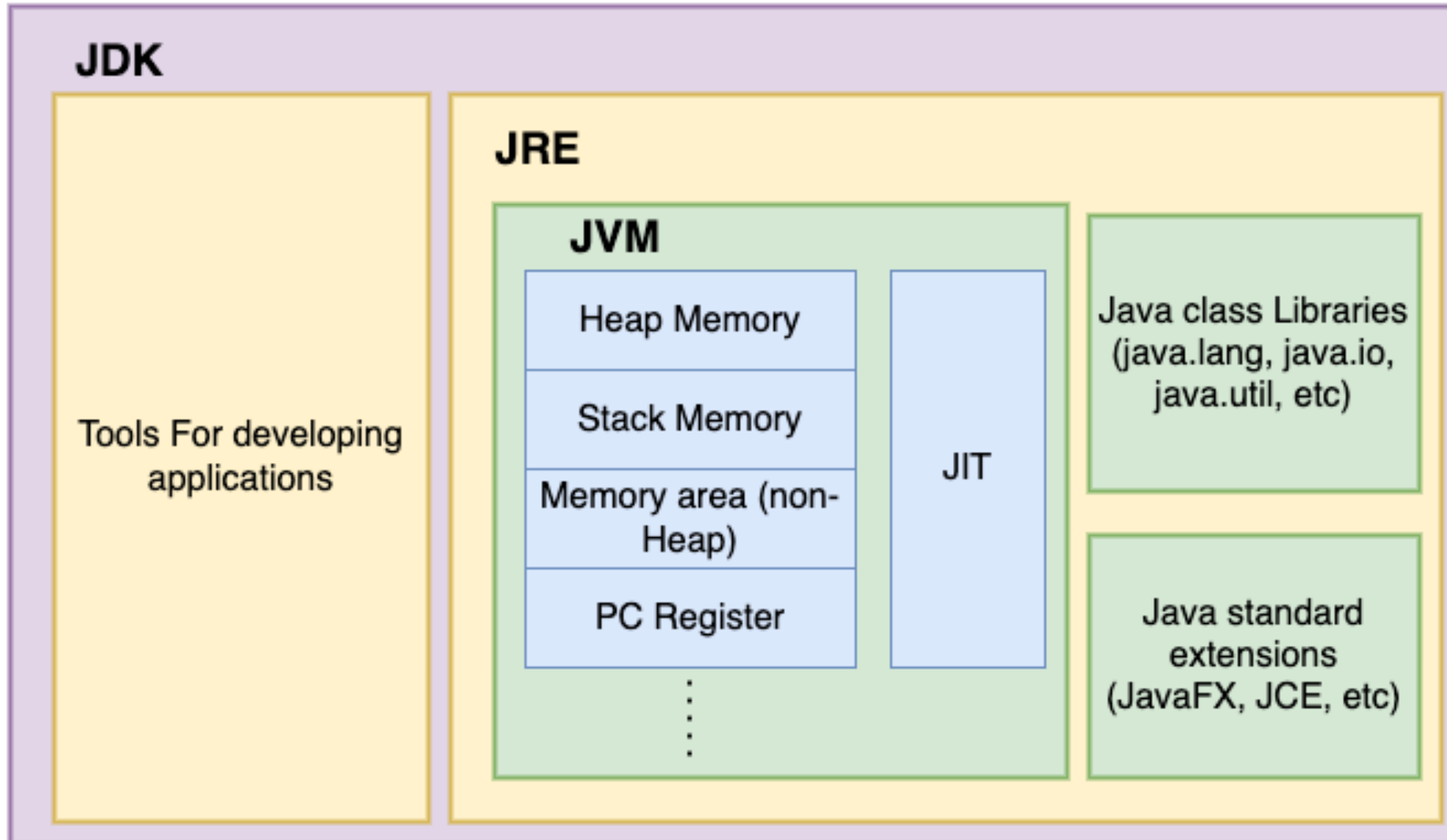
- ❖ The *Java Virtual Machine*
- ❖ The *Java Application Programming Interface (API)*



Java Virtual machine (JVM)



Platform Independent



Different Editions of Java

Key Differences Among the Editions:

Feature	Java SE	Java EE (Jakarta EE)	Java ME	Java Card
Focus	General-purpose	Enterprise systems	Mobile/embedded devices	Secure smart cards
API Scope	Core libraries	SE + enterprise APIs	Limited SE + device APIs	Highly limited APIs
Device Compatibility	Desktop, servers, etc.	Servers	Small devices	Smart cards
Scalability	Medium	Large-scale systems	Small-scale systems	Minimal resources

How to write a java program?

- ❖ Check that JDK is installed
- ❖ Write your program using any text Editor
- ❖ Compile
- ❖ Execute

From where to Download

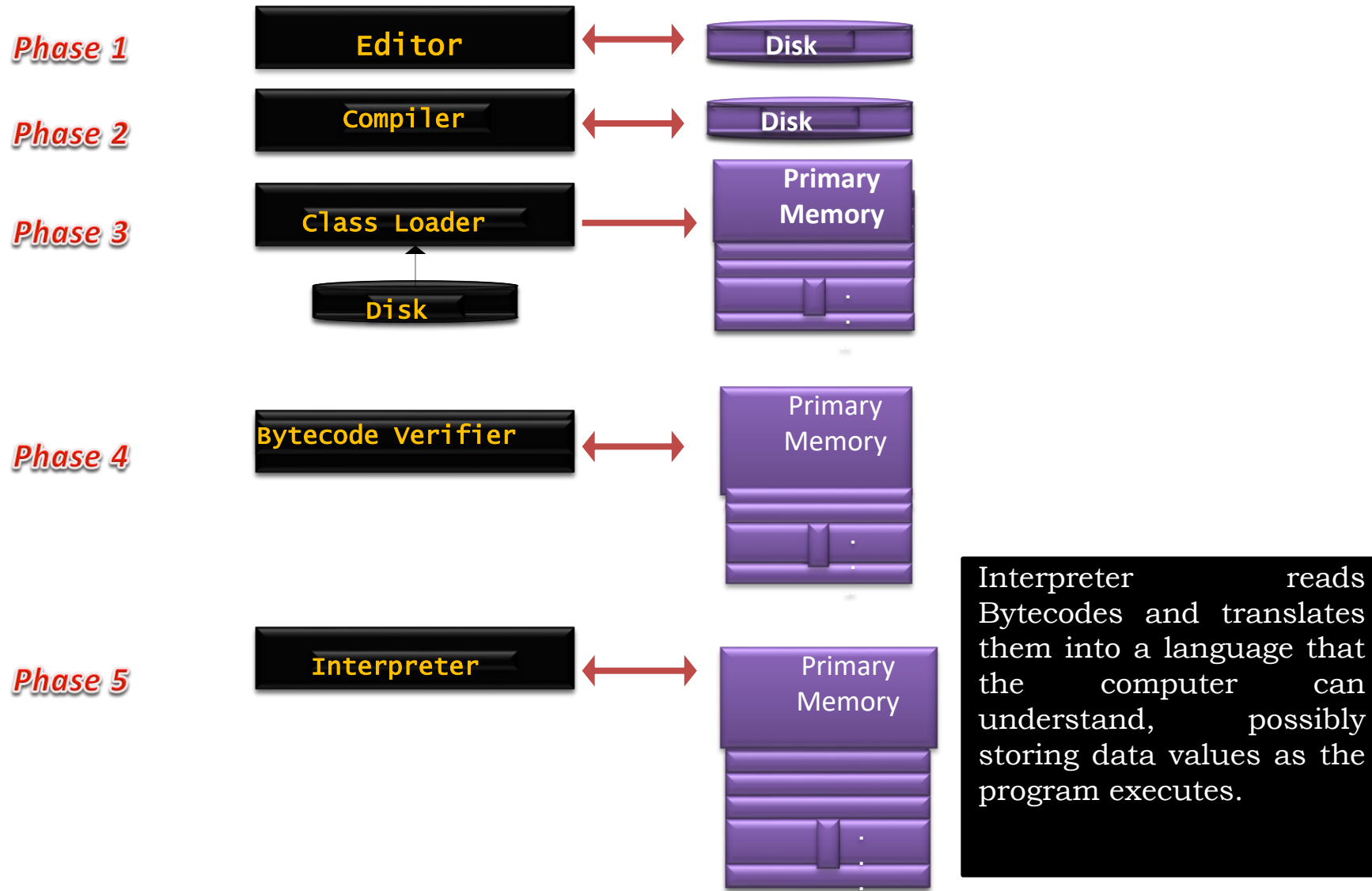
- Download Webpage

<https://www.oracle.com/in/java/technologies/downloads/>

- Installation Instructions

<https://docs.oracle.com/en/java/javase/23/install/installation-jdk-microsoft-windows-platforms.html#GUID-E3C75F92-D3B2-421D-A9BE-933C15F7CD1B>

Phases of a java program execution



Hello World Program

```
/* The HelloWorldApp class implements an application
   that simply prints "Hello World!" to standard output. */
//File Name HelloWorldApp.java
class HelloWorldApp
{   public static void main(String[] args)
    {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

```
C:\ Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\localuser>cd C:\java

C:\java>dir
Volume in drive C has no label.
Volume Serial Number is 242E-E457

Directory of C:\java

11/20/2005  08:43 PM    <DIR>          .
11/20/2005  08:43 PM    <DIR>          ..
11/20/2005  08:43 PM                284 HelloWorldApp.java
                1 File(s)                284 bytes
                2 Dir(s)      1,918,476,288 bytes free

C:\java>_
```

Next compile it

javac HelloWorldApp.java

```
C:\ Command Prompt

C:\java>dir
Volume in drive C has no label.
Volume Serial Number is 242E-E457

Directory of C:\java

11/21/2005  12:36 PM    <DIR>          .
11/21/2005  12:36 PM    <DIR>          ..
11/21/2005  12:36 PM                432 HelloWorldApp.class
11/20/2005  08:43 PM                284 HelloWorldApp.java
                2 File(s)              716 bytes
                2 Dir(s)      1,479,315,456 bytes free

C:\java>
```

Next Run it

java HelloWorldApp

```
C:\ Command Prompt

C:\java>java HelloWorldApp
Hello World!

C:\java>
```

Nihar Ranjan Roy

Dissection of HelloWorld Program

```
/* The HelloWorldApp class implements an application
   that simply prints "Hello World!" to standard output. */

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

Command Line arguments

```
class CmdArgs
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        System.out.println("Number of arguments"+args.length);
```

```
    }
```

```
}
```



A screenshot of a Windows command prompt window. The title bar shows 'C:\WINNT\system32\cmd.exe'. The command prompt shows the following text:

```
C:\>java CmdArgs one two three
Number of arguments3
C:\>
```

Problem

Write a program for adding two integers, the two numbers should be provided as command line arguments

{hint:

inorder to convert a string into integer we have

```
int i=Integer.parseInt(str)}
```

```
class AddInt
{
public static void main(String args[])
{
    if(args.length!=2)
        System.out.println("Improper arguments");
    else
    {
        int i=Integer.parseInt(args[0]);
        int j=Integer.parseInt(args[1]);
        System.out.println("Sum of " + i + " and " + j + " is " +
        (i+j));
    }
}
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINNT\system32\cmd.exe'. The command prompt shows the following sequence of commands and output:

```
C:\java>java AddInt 23 34
Sum of 23 and 34 is 57
C:\java>_
```


Problem

Given the following,

```
1. public class X {  
2.         public static void main(String [] args) {  
3.                 String names [] = new String[5];  
4.                 for (int x=0; x < args.length; x++)  
5.                     names[x] = args[x];  
6.                 System.out.println(names[2]);  
7.             }  
8. }
```

and the command line invocation is java X a b

what is the result? (Choose one.)

A. names

B. Null

C. Compilation fails

D. An exception is thrown at runtime

E. b

What are packages in java?

Predefined, related classes grouped by directories on disk

All in directory java or javax, or subdirectories

Referred to collectively as the Java class library or the Java applications programming interface (Java API)

import - locates classes needed to compile program

Example

```
import javax.swing.JOptionPane;  
import javax.swing.*;
```

Taking GUI input

Class JOptionPane

Package

`javax.swing.JOptionPane;`

Contains methods that display a dialog box

- static method `showMessageDialog`
 - First argument - null ()
 - Second argument - string to display
- static method `showInputDialog`
 - Argument of type String

Use

```
JOptionPane.showMessageDialog(null, "Hello");
```

```
String str=JOptionPane.showInputDialog("Enter ur age");
```

Terminate the program with

```
System.exit(0);
```

Problem

DO THE LAST PROBLEM AGAIN BUT THIS TIME TAKE THE USER INPUT AT RUN TIME USING JOPTIONPANE.

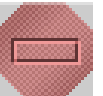





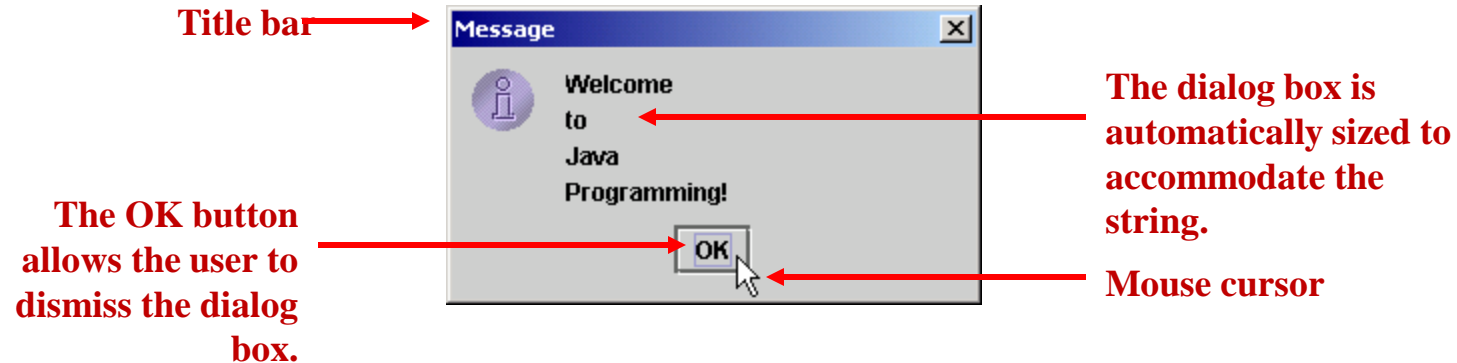
```
import javax.swing.JOptionPane;  
class AddIntJOpt  
{  
public static void main(String args[])  
{  
int i=Integer.parseInt(JOptionPane.showInputDialog("Pls enter the first no"));  
int j=Integer.parseInt(JOptionPane.showInputDialog("Pls enter the Second no"));  
JOptionPane.showMessageDialog(null,"Sum is"+(i+j));  
System.exit(0);  
}  
}
```

More on showMessageDialog

- Show an error dialog that displays the message

```
JOptionPane.showMessageDialog(null, "msg", "Title", JOptionPane.ERROR_MESSAGE);
```

MESSAGE DIALOG TYPE	ICON
JOptionPane.ERROR_MESSAGE	
JOptionPane.INFORMATION_MESSAGE	
JOptionPane.WARNING_MESSAGE	
JOptionPane.QUESTION_MESSAGE	
JOptionPane.PLAIN_MESSAGE	no icon



Problem

Write a program in which we have a **method** that accepts two integers and prints their sum.

Restriction: object creation not allowed

```
class StaticAdd

{public static void main(String args[])

{int i=10,j=20;

addInt( i,j);           //call to method

}

public void addInt(int i, int j)

{ //displaying sum

System.out.println("Sum is "+(i+j));

}}
```



A screenshot of a Windows command prompt window titled "C:\WINNT\system32\cmd.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt shows the following text:

```
C:\java>javac StaticAdd.java
StaticAdd.java:8: non-static method addInt(int,int) cannot be referenced from a
static context
addInt(i,j);
^
1 error
C:\java>
```

The error message indicates that the non-static method `addInt` is being referenced from a static context (the `main` method). The cursor is positioned at the end of the line `addInt(i,j);` on line 8.


```
class StaticAdd
{public static void main(String args[])
{int i=10,j=20;
addInt(i,j);           //call to method
}
public static void addInt(int i, int j)
{ //displaying sum
System.out.println("Sum is "+(i+j));
}
}
```

Input via Scanner Class

The **Scanner** class in Java, part of the `java.util` package, is used to read input from various sources, such as the keyboard, files, or other input streams. It provides methods to parse and read different types of input like strings, integers, doubles, and more.

Steps to Use the Scanner Class:

1. Import the Scanner Class

Include `import java.util.Scanner;` at the beginning of your program.

2. Create a Scanner Object

Instantiate a Scanner object by linking it to an input source, such as `System.in` for keyboard input.

3. Read Input

Use appropriate methods provided by the Scanner class to read input based on the data type.

4. Close the Scanner

It's a good practice to close the Scanner after usage to release resources.

Example of Scanner Class

```
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {
        // Step 1: Create a Scanner object
        Scanner scanner = new Scanner(System.in);
        // Step 2: Prompt the user and read input
        System.out.println("Enter your name:");
        String name = scanner.nextLine(); // Reads a line of text
        System.out.println("Enter your age:");
        int age = scanner.nextInt(); // Reads an integer
        System.out.println("Enter your height (in cm):");
        double height = scanner.nextDouble(); // Reads a double value
        // Step 3: Display the inputs
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Height: " + height + " cm");
        // Step 4: Close the Scanner
        scanner.close();
    }
}
```

Common Scanner methods

Method	Description
<code>nextLine()</code>	Reads a full line of text.
<code>next()</code>	Reads a single word (up to the next space).
<code>nextInt()</code>	Reads an integer.
<code>nextDouble()</code>	Reads a double.
<code>nextBoolean()</code>	Reads a boolean.
<code>hasNext()</code>	Checks if there is more input available.

Caution

```
import java.util.Scanner;
public class ScannerTest
{
    public static void main(String args[])
    {
        Scanner scan=new Scanner(System.in);
        System.out.println("input an integer");
        int no=scan.nextInt();
        System.out.println("Input number is "+no);
        String s=scan.nextLine();
        System.out.println("String is "+s);
        s=scan.nextLine();
        System.out.println("String is "+s);
    }
}
```

C:\WINDOWS\system32\cmd.exe

input an integer

23

Input number is 23

String is

nihar ranjan

String is nihar ranjan

Press any key to continue . . .

Caution

Avoid Mixing `nextLine()` with Other Methods: When using `nextLine()` after methods like `nextInt()` or `nextDouble()`, a newline character (`\n`) from the previous input might be left in the buffer. Use `scanner.nextLine()` once to consume it.

```
System.out.println("Enter your age:");  
int age = scanner.nextInt(); // Reads the integer  
scanner.nextLine(); // Clears the buffer  
System.out.println("Enter your name:");  
String name = scanner.nextLine(); // Now reads the full name correctly
```