

# **Object Oriented Programming using Java**

## **Concept of Java Programming**

Dr. Partha Pratim Sarangi

## **Books**

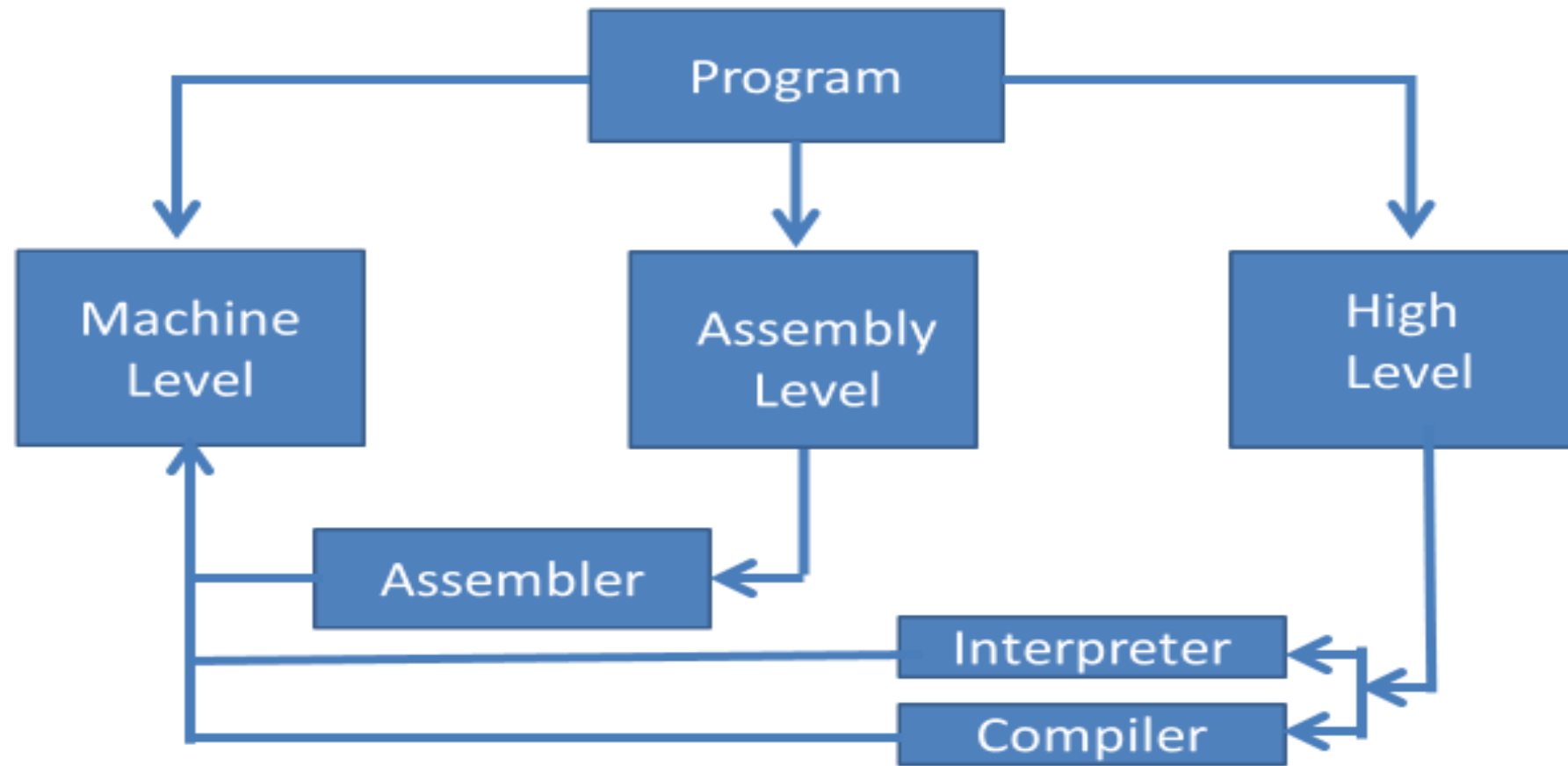
- **The Complete Reference Java 2 (10<sup>th</sup> Edition)**  
**Hebert Schildt, Tata Mc Graw Hill**
  - **PROGRAMMING WITH JAVA (6th Edition)**  
**E Balaguruswamy**
-

# Introduction

Various Programming approaches since the invention of computer:

- Machine level Programming
  - Assembly level Programming
  - Structured Programming
  - Object Oriented Programming
  - Scripting Programming
-

## Type of Programming Languages



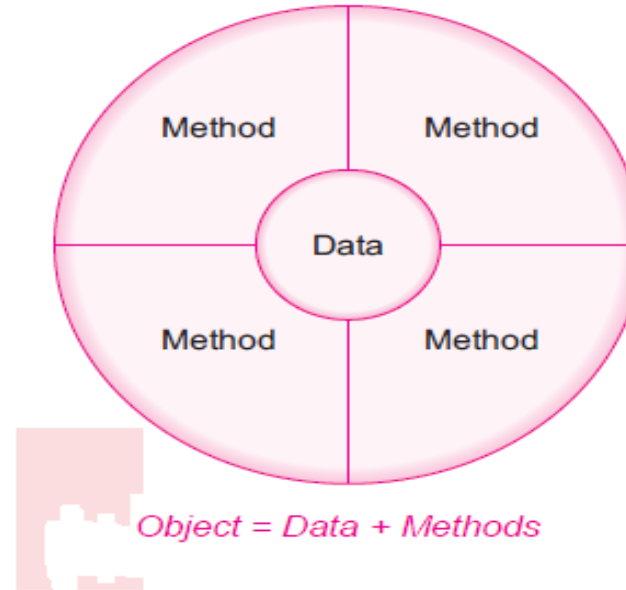
# Object Oriented Paradigm

Objective: Eliminate flaws experienced in procedural approach.

OOP:

- Treats data as critical element in program development.
  - Does not allow it to flow freely in the system.
  - Ties data closely to the functions that operate on it.
  - Protects data from unintentional modification by other functions.
  - Allows to decompose problem into number of entities called objects.
-

# Object Oriented Paradigm



***Object-oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area of both data and functions that can be used as templates for creating copies of such modules on demand***

---

# Object Oriented Paradigm

## Features:

- Emphasis on data than procedure.
  - Programs are divided into what is known as objects.
  - Data structures are designed such that they characterize the objects.
  - Methods that operate on the data of an object are tied together in data structure.
  - Data is hidden and cannot be accessed by external functions.
  - Objects may communicate with each other through methods
  - New data and methods can be added whenever necessary.
  - Follows **bottom up approach in program design**.
-

# Basic Concepts of Object Oriented Programming

Object and Classes

Data abstraction and Encapsulation

Inheritance

Polymorphism

Compile Time and Runtime Mechanisms

Dynamic binding

Message Communication

---



# History of Java

- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of Sun engineers called Green Team.
  - The main objectives of that language were the software should be compatible with all the existing hardware at the same time occupy as little memory space as possible.
  - Java was originally designed for small, embedded systems in electronic appliances like set-up boxes, but it was too advanced technology for the digital television industry at that time.
  - After that, it was called **Oak** and was developed as a part of the Green project. Java team members initiated this project to develop a language for micro devices.
  - Later, Java technology was incorporated by Netscape and well suited for networking.
-

## How Java is named Java ?

- Java was called Oak as it is symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.
  - The team wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.
  - In 1995, Oak was renamed as Java
    - Java is an island of Indonesia where first coffee was produced (called Java coffee)
  - The primary motivation for Java was the need for a platform-independent (ie. Architecture-Neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.
  - JDK (Java Development Kit) 1.0 released in January 23, 1996.
-

# What is Java?

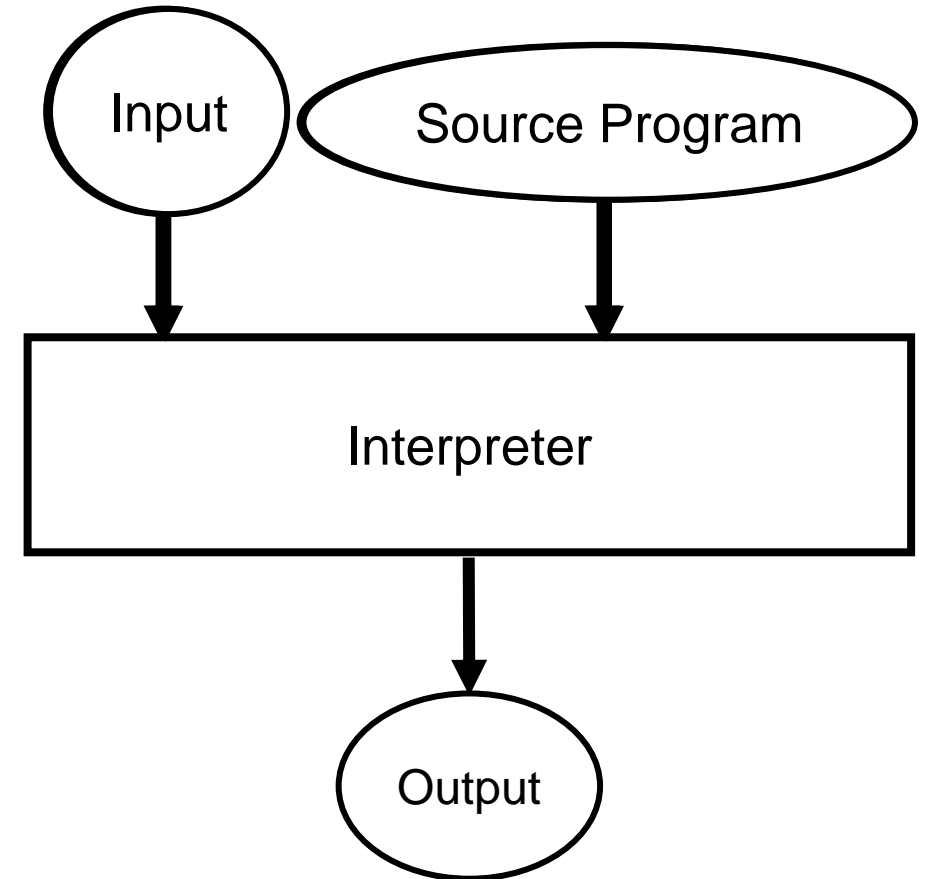
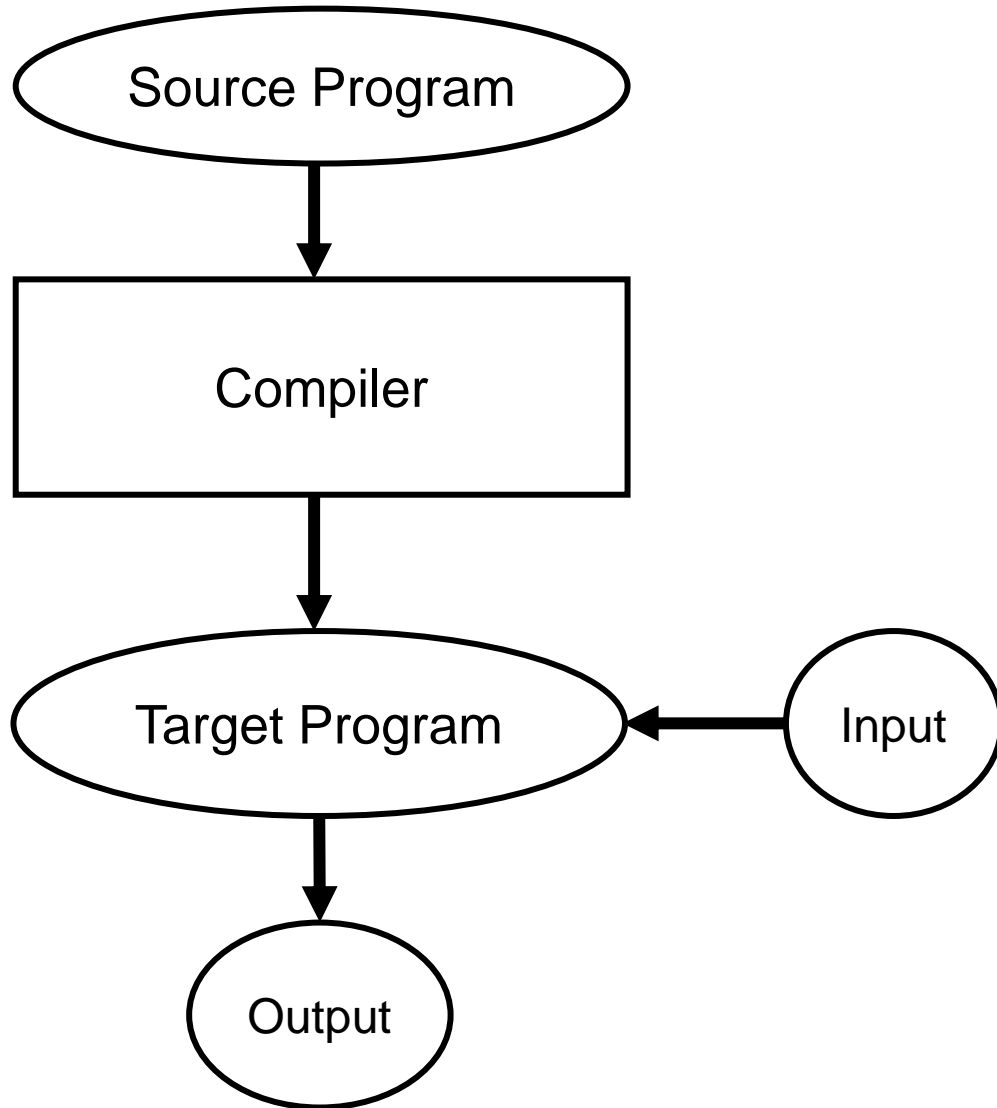
**Java is simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language.**

## Java Features

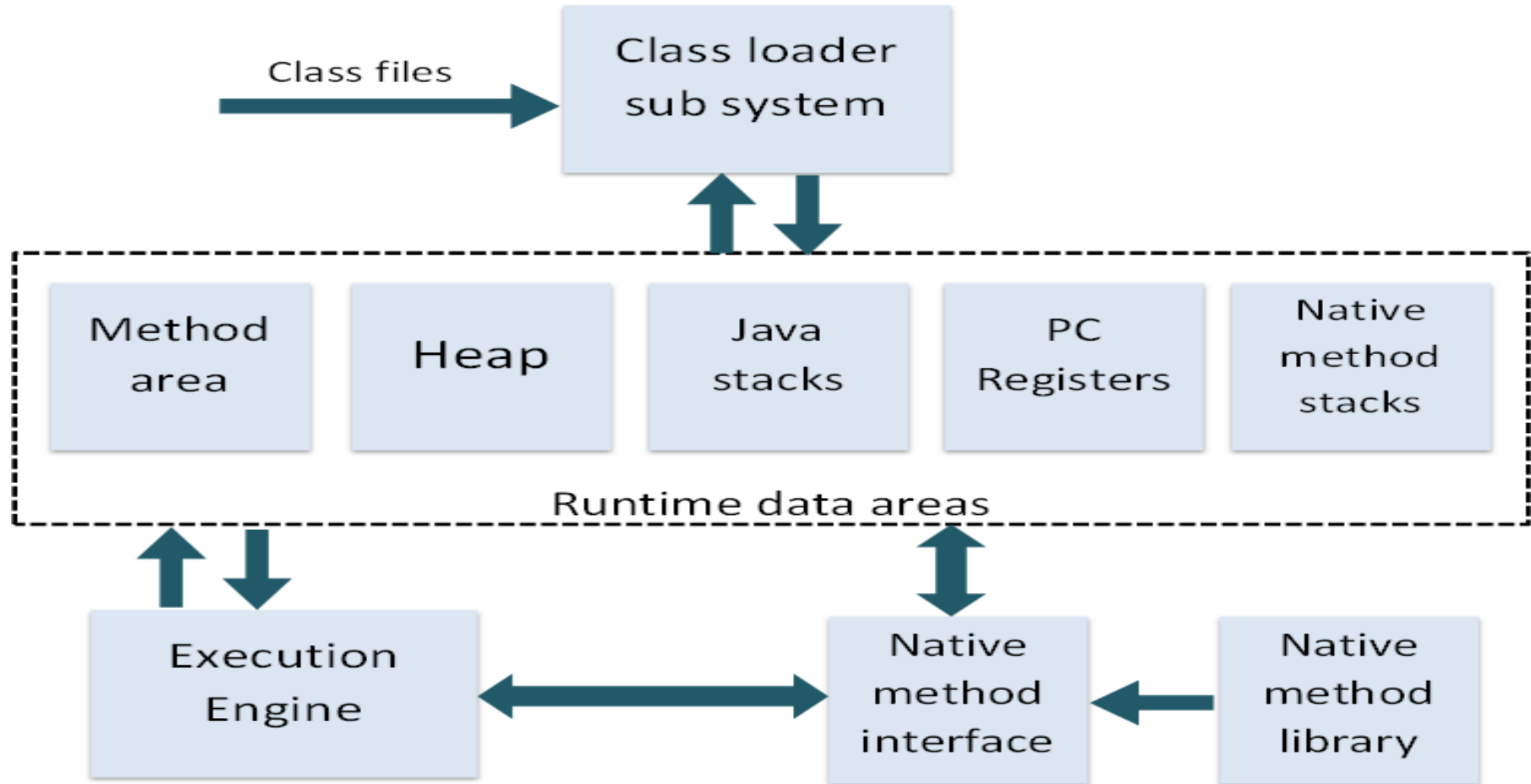
### Compiled and Interpreted

- Java compiler translates source code into byte code instructions.
  - Byte code is not machine instructions.
  - Machine instructions are generated by Java interpreter.
  - Machine code is directly executed by the machine that is running the Java Program.
  - Java is both compiled and interpreted.
-

# Compiler and Interpreter



# JVM Architecture



# Java Features

## Platform-Independent and Portable

- Java Programs can be easily moved from one computer system to the other.
- Change or upgrade in OS, processor or system resources will not force any change in Java Programs.
- Java interconnects different kinds of systems worldwide.

Ensures portability in two ways:

- Generates byte code instructions that can be implemented in any machine.
  - Primitive data types are machine independent.
-

# Java Features

## Object Oriented

- Everything in Java is an object.
  - All the program code and data resides within objects and classes.
  - Comes with extensive set of classes arranged in packages.
  - These classes can be used in our programs by inheritance.
  - Object model in Java is simple and easy to extend.
-

# Java Features

## **Robust and Secure**

- Provides many safeguards to ensure reliable code.
  - Follows strict compile time and run time checking for data types.
  - Designed as garbage-collected language.
  - Relieves all memory management problems.
  - Concept of exception handling captures serious errors and eliminates risks of crashing the system.
  - Verify memory access and ensure no viruses are communicated with the applet.
  - Absence of pointers ensure that access to memory location cannot be provided without proper authorization.
-



# Java Features

- **Distributed**
  - Designed as a distributed language for creating applications on networks.
  - It can share data and programs.
  - Remote objects on the internet can be accessed as easily as the can do in a local system.
  - Multiple programmers at multiple remote locations can collaborate an work together on a single project.
-

# Java Features

- **Simple, Small and Familiar**
  - It is small and simple language.
  - Redundant and unreliable features of C and C++ are not part of Java.
  - Java looks familiar because it is modelled on C and C++ languages.
  - Java code looks like a C++ code.
-

# Advantages of Java

- Java is a true object-oriented language. Almost everything in Java is an object.
  - All program code and data reside within objects and classes. The object model in Java is simple and easy to extend.
  - Portable: Java programs can be moved from one computer system to another anywhere and anytime.
  - Platform-Independent
  - Write once, run anywhere: As java programs are compiled into machine independent byte codes, they run consistently on any java platform.
  - Write robust and reliable programs
  - Build an application on almost any platform and run that application on any other supported platform without having to recompile your code.
  - Distribute your applications over a network in a secure fashion.
-

# First Java Program

//file name must be FirstJavaPgm.java

```
class FirstJavaPgm
```

```
{
```

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

```
    {
```

```
        System.out.println("Welcome to the world to Java programming");
```

```
    }
```

```
}
```

- Save the file as FirstJavaPgm.java
- Compile the program using javac (javac compiler JDK tool)

```
C:\javac FirstJavaPgm.java
```

- Run the program using java (java interpreter JDK tool)

```
C:\java FirstJavaPgm
```

---

## Description of the above program:

The output is:

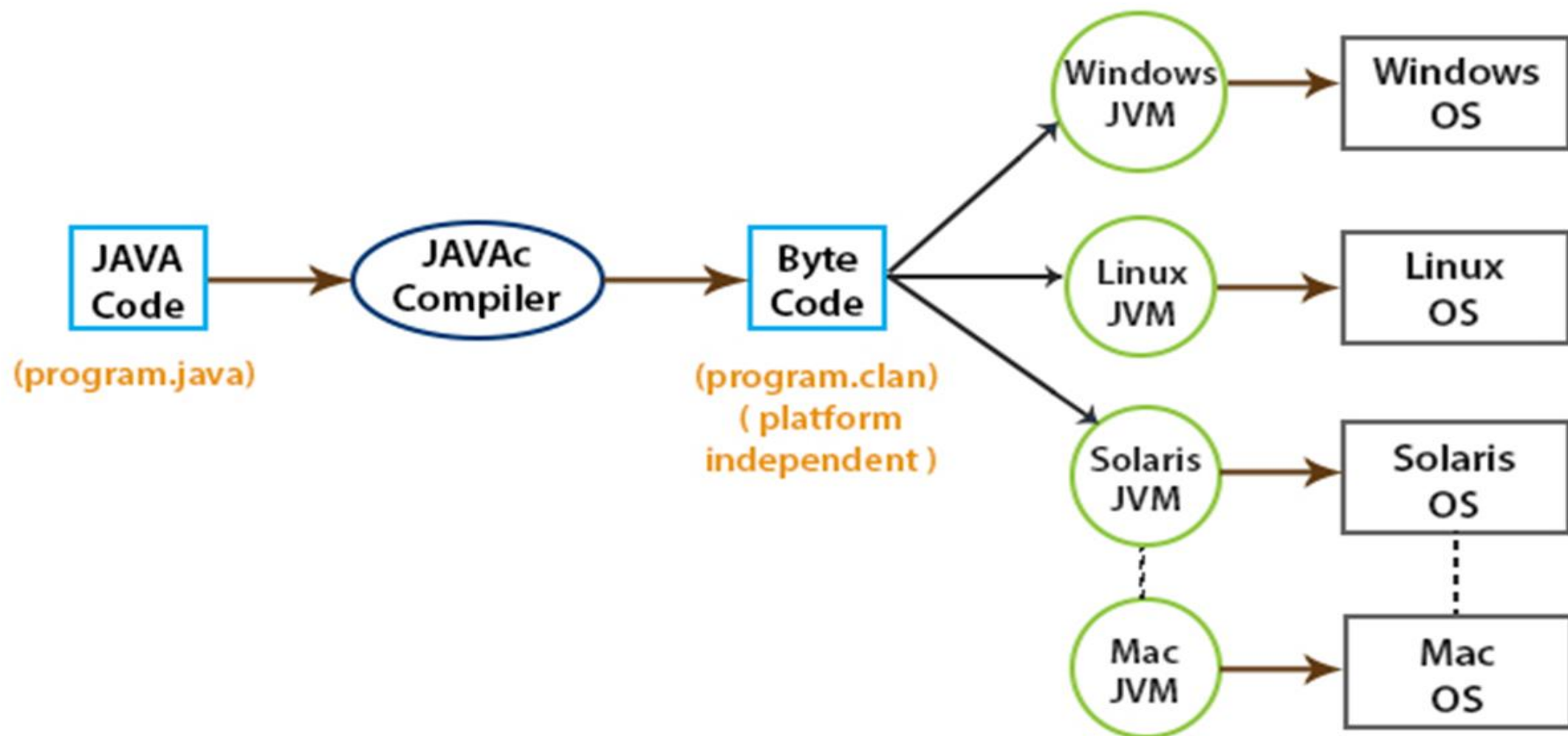
Welcome to the world of java programming

- The first line “//file name must be FirstJavaPgm” is a comment line. The compiler ignores the line. Comment line can be indicated by
    - //      Single line comment
    - /\*               \*/      Multiline Comment
  - The second line class FirstJavaPgm
  - Declares a class, which is an object-oriented construct. “FirstJavaPgm” is user defined class name. A java program may contain multiple class definitions. Classes are the primary and essential element of a Java program. These classes are used to map the objects of real world problems.
-

- Opening and Closing Brace:
  - Every class definition in Java begins with an opening brace “{” and ends with a matching closing brace “}” appearing in the last line in the example. This indicates the beginning and closing of any block.
  - Main function declaration line:
  - `public static void main(String args[ ])`
  - The main method must be declared as public, since it must be called from the outside of its class by JVM. If main is public then only it will be visible to JVM as beginning of execution point of the program.
  - The keyword static allows main() to be called without having to instantiate a particular instance of the class. This is necessary since main() is called by the JVM before any objects are made.
  - The key word void tells the compiler that main() does not return any value.
-

- The argument to `main()` is an array of string objects. Whether the command line arguments are used in this program or not, but they have to be there because they hold the arguments invoked on the command line. As `main` is the beginning point of the java program, whenever you execute JVM searches for the `main` method, which is public, static, with return type `void` and a `String` array as an argument. If anything is missing the JVM raises an error.
  - The Output line: `System.out.println("Welcome to the world of Java programming");`
  - This is similar to the `printf()` statement of C. Since Java is a true Object oriented programming language every method is a part of an object. The `println` method is a member of the `PrintStream` class which is referred by a static data member **out** of the `System` class.
  - The `System` is one of the core classes in Java and belongs to the `java.lang` package and all of its data members and methods contained in this class are static in nature. The purpose of the `System` class is to provide access to system resources.
-

# Platform Independence in Java

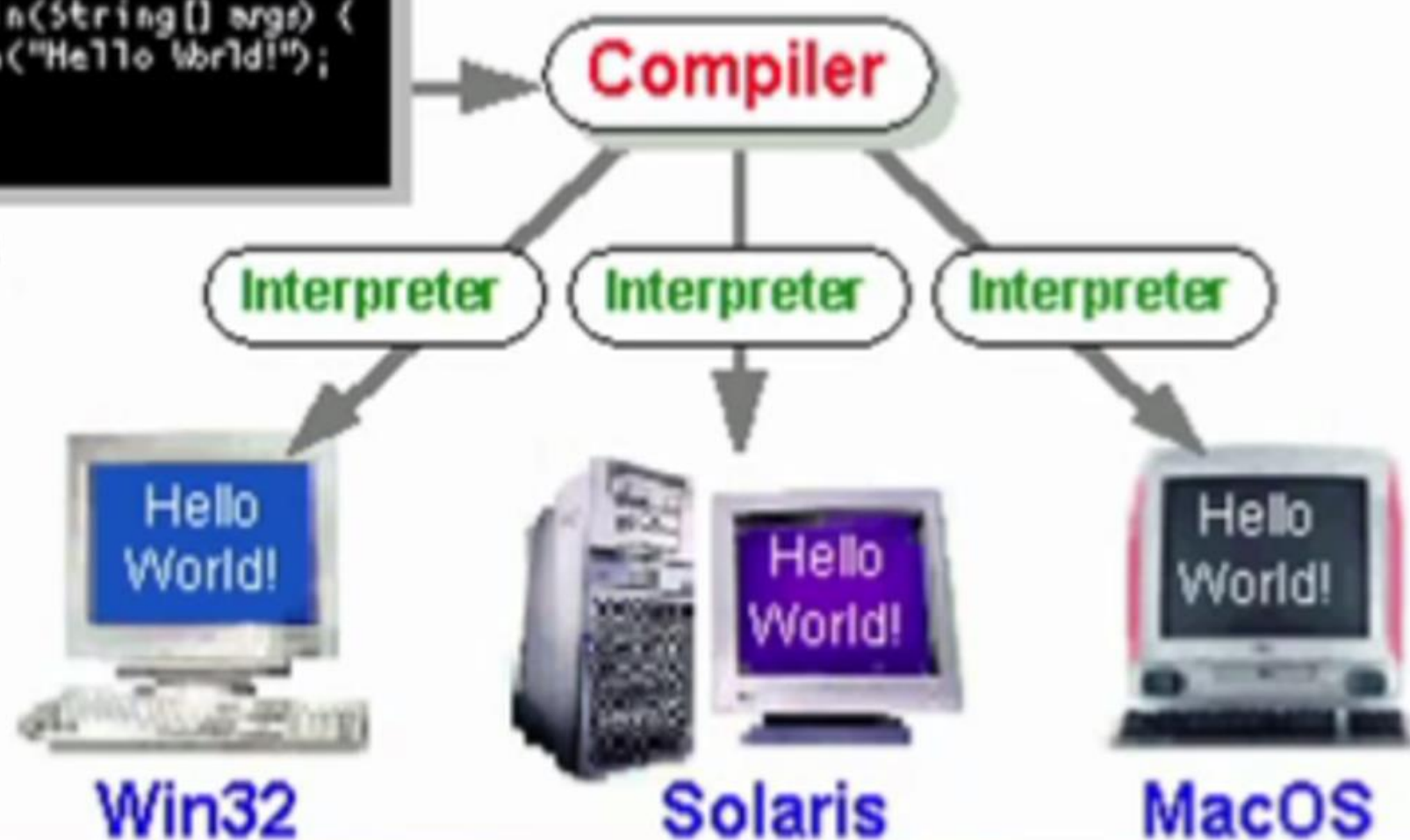




## Java Program

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

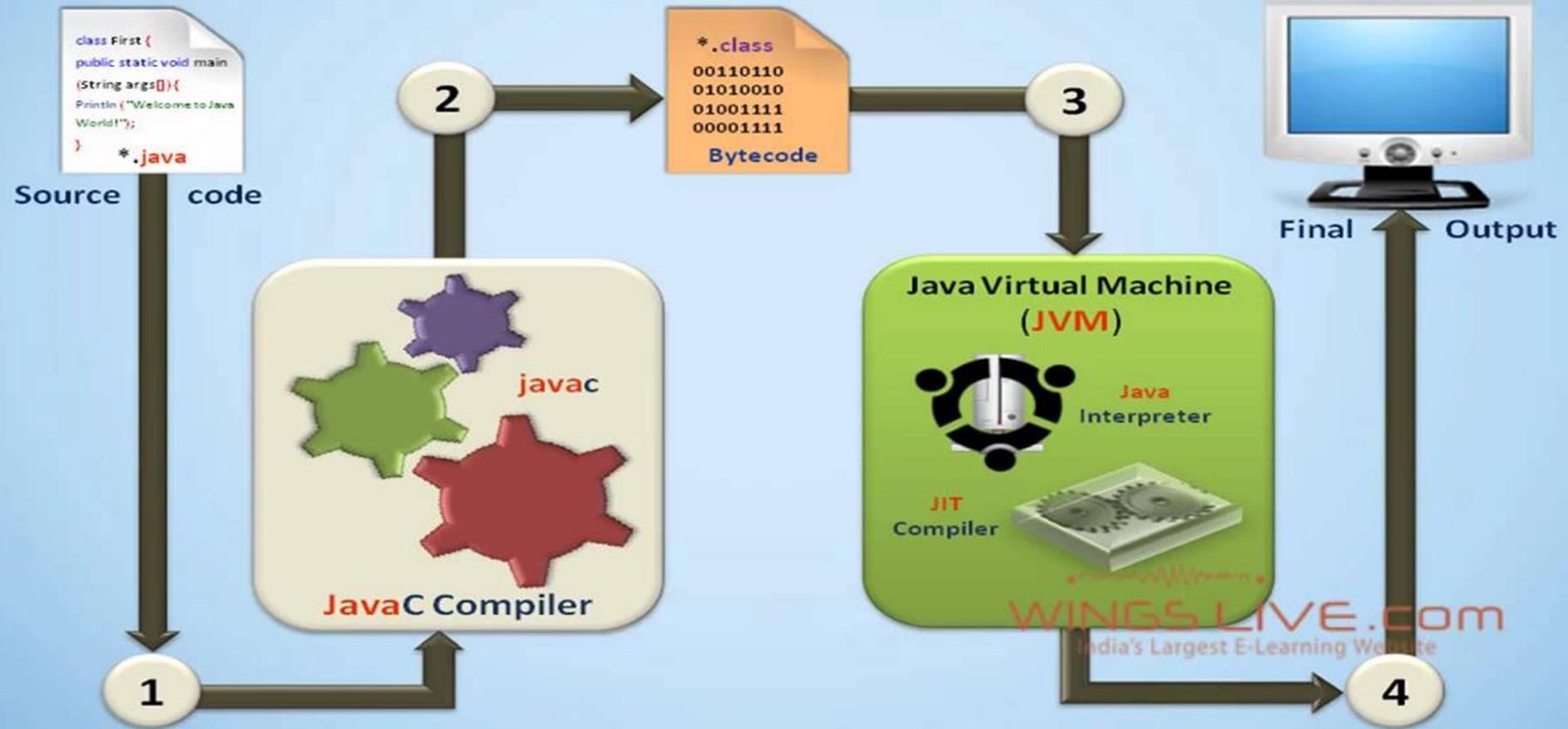
HelloWorldApp.java





# Java is a Platform Independent Language

## Execution of Java Bytecode



# C Versus Java

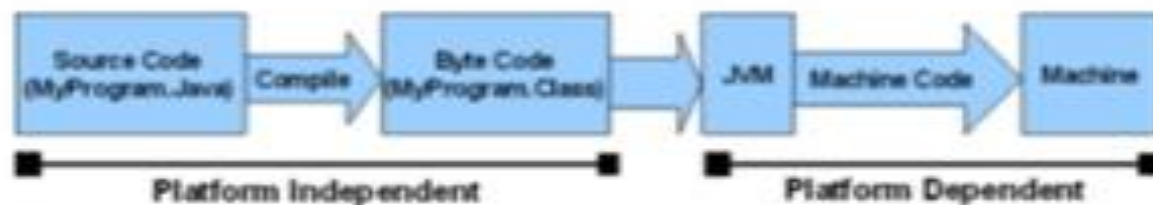
Aspects	C	Java
Paradigms	Procedural	Object-oriented
Platform Dependency	Dependent	Independent
Datatypes : union, structure	Supported	Not supported
Pre-processor directives	Supported (#include, #define)	Not supported
Header files	Supported	Use packages (import)
Storage class	Supported	Not supported

Contd...

Aspects	C	Java
Inheritance	No inheritance	Supported (Simple inheritance)
Pointers	Supported	No Pointers
Code translation	Compiled	Interpreted
Multi-threading and Interfaces	Not supported	Supported
Exception Handling	No exception handling	Supported
Database Connectivity	Not supported	Supported

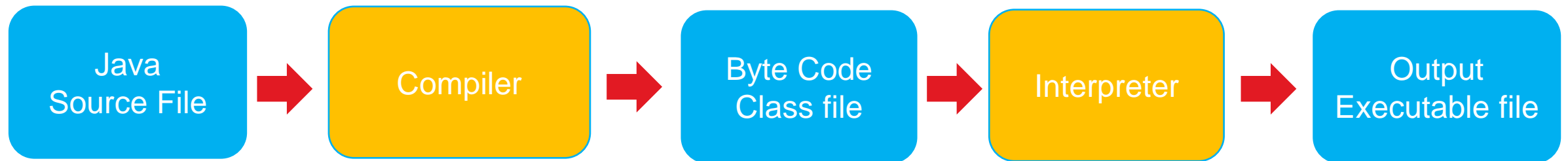
## Note

- JVM and JRE both are platform dependent.
- Only Java Bytecode (.class file) generated after compilation is platform independent in a sense that it is only a binary code and of course it has nothing to do with platform and JVM reads it with respective operating system.



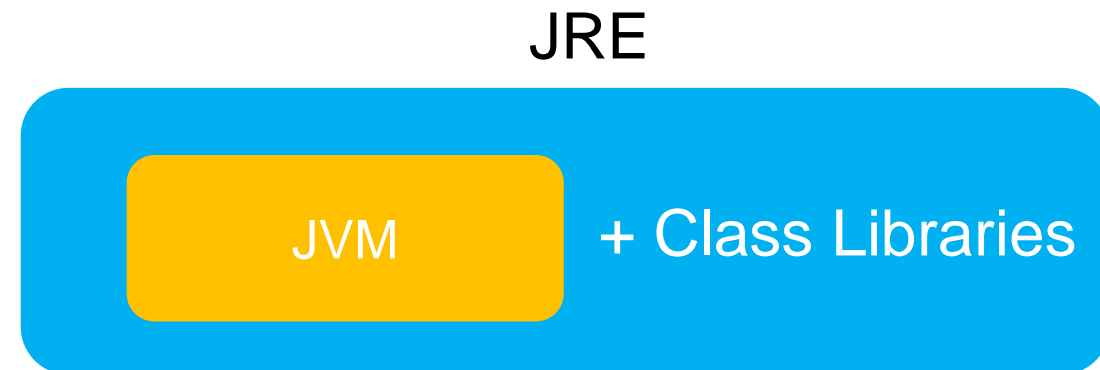
# Java Virtual Machine (JVM)

- JVM is responsible to converting byte code to the machine specific code.
- JVM provides a machine interface that does not depend on the underlying operating system and machine hardware architecture. This independence from hardware and operating system is what makes java program write-once run-anywhere.
- JVM is platform dependent. Many different form of JVM exist that depends on operating systems.
- JVM provides core java functions like memory management, garbage collection, security, etc.



## Java Runtime Environment (JRE)

- JRE provides the minimum requirements for executing a Java application.
- Java Runtime Environment contains JVM, class libraries and other supporting files.
- JRE is a superset of JVM.
- If you want to run any java program, you need to have JRE installed in the system.
- It contains many development tools like compiler, debugger, etc.





## Java Development Kit (JDK)

- Java Development Kit (JDK) contains JRE along with various development tools like Java libraries, Java source compiler, Java debugger, bundling and deployment tools like: java, javac, Javadoc, javap, javah, appletviewer, etc.
- You need JDK, if at all you want to write your own programs, and to compile them. For running java programs, JRE is sufficient.

JDK

JRE

+ Compiler + Debugger + Javadoc + Appletviewer + ...



# Combined View

JDK

JRE

JVM

+

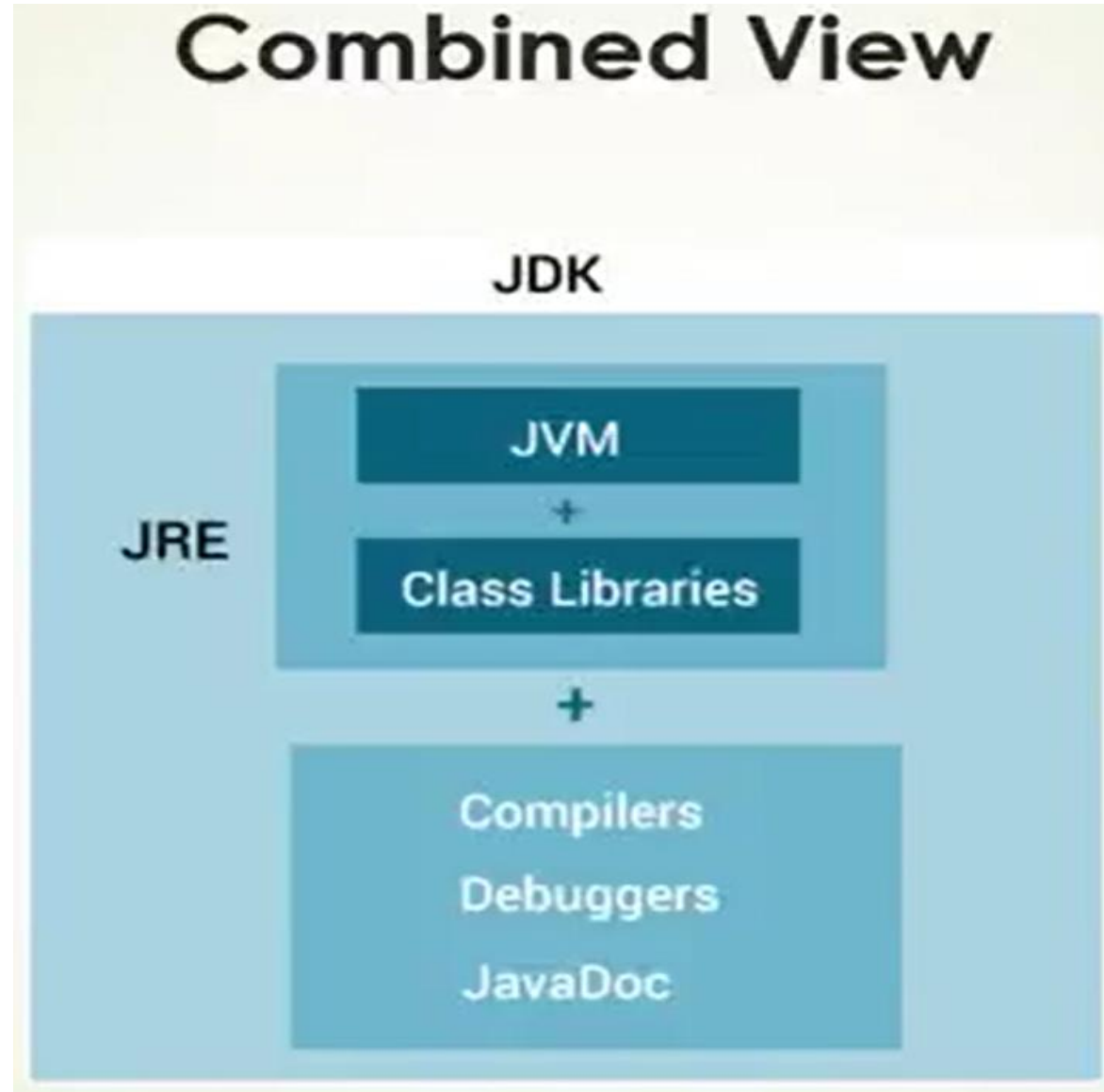
Class Libraries

+

Compilers

Debuggers

JavaDoc



## JDK, JRE & JVM

- **JVM** (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment
- **JRE** (Java Runtime Environment) is used to provide runtime environment. It is the implementation of JVM. It contains set of libraries + other files that JVM uses at runtime
- **JDK** (Java Development Kit) contains JRE + Development tools

- Java Development Kit (JDK) is composed of three basic components as follows:
    - Java Compiler
    - Java Virtual Machine (JVM)
    - Java Application Programming Interface (API)
  - Application programming interface (API) is a collection of prewritten packages, classes, and Interfaces with their respective methods, fields and constructors.
-

- To develop and run the Java application, the required environment is JDK (Java Development Kit). That means JDK provides an environment which is used to develop and execute the Java program.
  - At the client side, just to run the application, the required environment is JRE (Java Runtime Environment).
  - Then JVM (Java Virtual Machine) executes the program. JVM is an interpreter which reads and executes line by line statements in a program.
-



## Tools in the JDK

- JDK contains the necessary facilities for the compilation of Java programs into intermediate **byte code** and their interpretation. It consists of various tools that can be used by the programmer to develop java programs:
  - **javac**: it takes as input a java source code file and produces a class file that contains the byte code
  - **java**: it takes as input, a class file containing the byte code and runs the program by interpreting it
  - **jdb**: it is a debugger that assists the developer in detecting errors
  - **javadoc**: it takes a java source code file as input and produces documentation in html for it
  - **javah**: it takes a java source code file as input and produces header files for use with native methods
  - **javap**: it takes the byte code file as input and produces a file that gives a description of the source code file from which the byte code file was created after compilation. It is a java disassembler
  - **appletviewer**: it is a tool that permits the user to execute java applets without using any java-compatible browser

# Java Technologies

- Java technology is both a programming language and a platform
- Different technologies depending on the target applications:
  - Java Platform, Standard Edition (**Java SE**)
  - Java Platform, Enterprise Edition (**Java EE**)
  - Java Platform, Micro Edition (**Java ME**)
  - Smart Card Applications - JavaCard
- Each edition puts together a large collections of packages offering functionality needed and relevant to a given application
- The Java Virtual Machine remains essentially the same



## Identifiers in Java

- Identifiers in Java are symbolic names used to identify various elements of a program.
- They can be a class name, variable name, method name, package name, constant name, variable name and many more.
- Reserve words can not be used as an identifier.
- An identifier must begin with a letter or an underscore.
- Identifiers can be as long as you want but names that are too long usually are too cumbersome.

## Java Reserved Keywords

- Java reserved keywords are predefined words, which are reserved for any functionality or meaning.
  - We can not use those keywords as identifier in our program.
-

## Some Illegal Identifiers

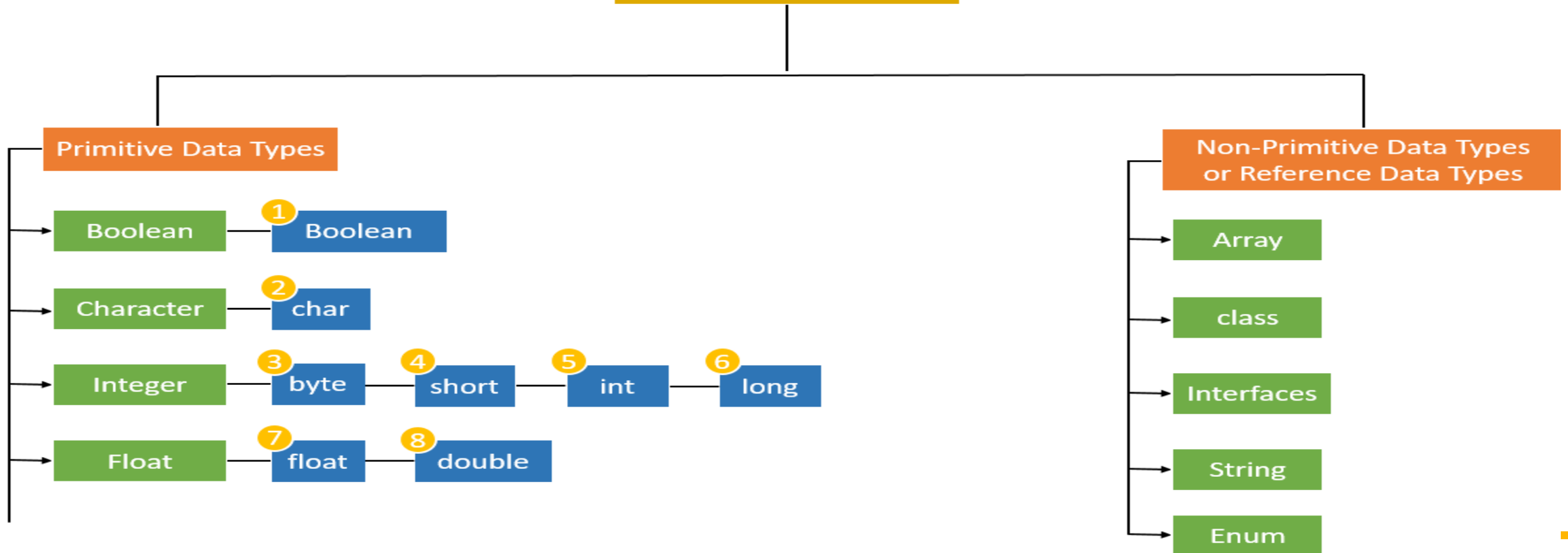
<u>Illegal Identifier</u>	<u>Reason</u>	<u>Suggested Identifier</u>
<code>my age</code>	Blanks are not allowed	<code>myAge</code>
<code>2times</code>	Cannot begin with a number	<code>times2</code> or <code>twoTimes</code>
<code>four*five</code>	<code>*</code> is not allowed	<code>fourTimesFive</code>
<code>time&amp;ahalf</code>	<code>&amp;</code> is not allowed	<code>timeAndAHalf</code>



# Data Types in Java

- Data types in Java are divided into 2 categories:
  - Primitive data types
  - Non-primitive data types

## Data Types In Java



## (Primitive) Data types in JAVA

Domain	Java type
Integer	<code>byte, short, int, long</code>
Fractional numbers	<code>float, double</code>
Boolean	<code>boolean</code>
Characters	<code>char</code>

# Numeric types in JAVA

Type	range	size in bits
byte	-128 to 127	8
short	-32768 to 32767	16
int	$-2 \times 10^9$ to $2 \times 10^9$	32
long	$-9 \times 10^{18}$ to $9 \times 10^{18}$	64
float	$\pm 3.4 \times 10^{38}$ and as small as $\pm 1.4 \times 10^{-45}$	32
double	$\pm 1.7 \times 10^{308}$ and more precision than float	64

# Java Variables

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.
- Variable is a name of memory location. There are three types of variables in java: local, instance and static.
- There are three types of variables in Java:
  - Local variable
  - Instance variable
  - Class variable
- **Local variable:** A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class can not use this variable.
- A local variable cannot be defined with "static" keyword.
- Initialization of the local variable is mandatory.

- **Instance Variable:** A variable declared inside the class but outside the body of the method, is called an instance variable. It is also called non-static variables.
  - These variables are created when an object of the class is instantiated and destroyed when the object is destroyed.
  - It is called an instance variable because its value is instance-specific and is not shared among instances.
  - Initialization of an instance variable is not mandatory. Its default value is 0.
  - **Class variable:** A variable that is declared as static is called a class variable. It is also called static variable. It cannot be local.
  - For static variable only single copy is created and share it among all the instances of the class.
  - Memory allocation for class variables happens only once when the class is loaded in the memory.
-

# Non-primitive Data Types in Java

- In Java, non-primitive data types are known as reference types. In other words, a variable of class type is called reference data type.
  - These are the datatypes which refer to objects. Hence they are called reference types.
  - Variables of primitive data types hold “**data or value**” and variables of reference types hold “**reference or address**” of dynamically created objects.
  - The objects are created in heap memory and whose addresses are stored in the stack memory.
  - Reference data types are primarily classes, arrays, strings or interfaces.
  - Primitive data types can be initialized with default values, if which are not initialized by us. Similarly, reference data types are initialized with “**null**” before instantiation of objects. Null simply indicates the absence of “reference or address”.
-

## **Wrapper classes in Java**

- The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.
- When an object to a wrapper class is created, it contains a field and in this field, we can store primitive data types.
- In other words, we can wrap a primitive value into a wrapper class object.

### **Need of Wrapper Classes**

- They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
  - The classes in java.util package handles only objects and hence wrapper classes help in this case also.
  - Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
  - An object is needed to support synchronization in multithreading.
-

## Primitive Data types and their Corresponding Wrapper class

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Autoboxing is the automatic conversion of a primitive value into an object of corresponding wrapper class. Unboxing is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing.



- Example of **Autoboxing** – conversion of int to Integer, long to Long, double to Double etc.
- Example of **Unboxing**– conversion of Integer to int, Long to long, Double to double, etc.
- There are two ways to convert primitive types to objects: using wrapper class (i) **Constructor** and (ii) **Static factory method**.
- For example:
  - by constructor of a wrapper class: `Integer a = new Integer(10);`
  - by using static factory method: `Integer b = Integer.valueOf(10);`
- We should always use the static factory method over the constructor to convert the primitive values to objects. As the compiler does some performance optimization which is not possible if we use a constructor.
- How to convert object to primitive type in Java:
  - `Double a = Double.valueOf(5.5);`      `Integer x = Integer.valueOf(10);`
    - `double b = a.doubleValue();`      `int y = x.intValue();`
- Both autoboxing and unboxing are done internally by compiler using `valueOf()` and `datatypeValue()` methods.

# Operation defined on Data types for Integer

Arithmetic operators		
Operator	Meaning	Result
+	addition	Integer
—	subtraction	“
*	multiplication	“
/	<b>integer division</b>	“
%	<b>mod</b>	“

19/4 : 4

19%4 : 3

# Operation defined on Data types for Integer

Relational operators		
<	less than	Boolean
<=	less than or equal	“
==	equal	“
!=	not equal	“

`2 < 3 : true`

`3 == 3 : true`

In a similar fashion `>`, `>=` are defined.

# Operation defined for fractional numbers

Same as that of integer data type except :

/ is the same as the usual division operator.

% is the remainder by usual division.

$19.0/4.0 : 4.75$

$19.0\%4.0 : 3.0$

---

# Operation defined on Data type for Boolean

Logical operators		
Operator	Meaning	Result
!	NOT	boolean
&, &&	AND	boolean
,	OR	boolean
Relational operators		
==	Equal	boolean
!=	Not equal	boolean

# Evaluation of Expressions

$2 + 3 * 4$  is equal to **14**

$96/4/2$  is equal to **12**

Is  $3/2 * 60 * 60$  equal to  $*60 * 60 * 3/2$  ? : **NO**

---

# Evaluation of Expressions

An important tool :

**It is always better to use parentheses in writing any expression.**

However, if the expression is not fully parenthesized, then the following rules are followed

- terms in parentheses are evaluated first
- the operators of **higher precedence** are evaluated before the operators of lower precedence.
- two consecutive operators have **same precedence**, they are evaluated from left to right. (called left associative).

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$  are left associative.

---

## Scope of a variable

Within the block in which it is declared  
and after its declaration.

---



## Example : Scope of variable

```
1.class scope
2.{    public static void main(String args[])
3.    {
4.        int i;
5.        i = 100;
6.        System.out.println("value of i here is "+i);
7.        {
8.            int j;
9.            j=55;
10.           i = i*j;
11.           System.out.println(i);
12.           System.out.println(j);
13.        }
14.        System.out.println(j);
15.    }
16.}
```

The above code will give compilation error at line 14 because no j exists at this line.

The scope of i is from line 5 to 14, scope of j is from line 9 to 12 only.

---

## Rules of Local variables

- Local variables are defined in a method or block of code.
  - They are named local because they can be accessed within a method or a block.
  - If local variable is not initialized, it don't take default value. Hence, it must be initialized before using it, otherwise it will give a compilation error.
  - Local variable scope is within the method where it is declared. It cannot be seen outside the method.
  - In C/C++, an uninitialized local variable takes a garbage value. However, it will give a compilation error and there is no such thing called a garbage value in Java.
-

# Motivation for If statement

Find the minimum of two or more numbers.

```
class if_example
{
    public static void main(String args [ ])
    {
        int i, j, max;
        i=100; j=79;
        min = i;
        if(j<i)
            min=j;
        System.out.println("minimum of "+i+" and "+j+" is "+min);
    }
}
```

---

## IF-ELSE statement

```
statement_a;  
if(condition)  $\Rightarrow$  statement_b;  
else        statement_c;  
statement_d;
```

**If condition is true statement\_b is executed  
otherwise statement\_c is executed.**

---

## Continued...

```
class IF_Else_example1
{
    public static void main(String args[])
    {
        int i,j;
        i = 100; j = 0;
        if(j==0) System.out.println("Division by z e r o error")
        else System.out.println(i+" divided by "+j+" is "+(i/j));
        i = i+j;
    }
}
```

---

Continued...

```
if(condition)
{
    .
    statements
    .
}
else
{
    .
    statements
    .
}
```


# Motivation for LOOPS in a Program

Many computational problems which require performing similar or same tasks a number of times.

## **Examples :**

- Print a statement 100 times.
  - Print all odd integers upto 1000.
  - Print all prime integers upto 10000.
-

# While Loop

```
statement_A;  
while ( condition )  
{  
      
}  
statement_B;
```



## Continued...


How to print a statement 10 times.

Observations :

1. The condition must change during an iteration.
2. The condition must be true for first 10 iteration and then become false.

```
class print10
{
    public static void main(String args[])
    {
        int counter; counter = 1;
        while(counter <=10)
        {
            System.out.println("`Welcome !'");
            counter = counter + 1;
        }
    }
}
```

# For Loop

```
stmt_A;  
⇒ for ( stmt_1 ; condition ; stmt_2 )  
  {  
      
  }  
stmt_B;
```

## Nested for loop

- What is the output of the following nested for loops

```
for (int i = 1; i <= 6; i++) {  
    for (int j = 1; j <= i; j++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

Output:

```
*  
**  
***  
****  
*****  
*****
```

---