GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Accredited with A+ Grade by NAAC

12-B Status from UGC

BCA III SEMESTER
BCAC0023
PROGRAMMING IN JAVA
MODULE 1
INTRODUCTION TO JAVA

Presented by:

Jayati Krishna Goswami

Assistant Professor, Dept. Of CEA

GLA University, Mathura

Introduction to Java

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

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

Platform

- Any hardware or software environment in which a program runs, is known as a platform.

- Since Java has a runtime environment (JRE) and API, it is called a platform.

Application (Where Java is used)

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

Cont...

- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web Applications such as irctc.co.in, javatpoint.com, etc.
- Enterprise Applications such as banking applications.

Cont…

- Mobile
- Embedded System
- Smart Card

Cont…

- Robotics

- Games, etc.

Types of Java Applications

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

1) Standalone Application

- Standalone applications are also known as desktop applications or window-based applications.

Cont...

- These are traditional software that we need to install on every machine.
- **Examples of standalone application** are Media player, antivirus, etc.
- AWT and Swing are used in Java for creating standalone applications.

Cont…

2) Web Application

- An application that runs on the server side and creates a dynamic page is called a web application.

- Currently, Servlet, JSP, Struts, Spring, Hibernate etc. technologies are used for creating web applications in Java.

Cont...

3) Enterprise Application

- An application that is distributed in nature, such as banking applications, etc. is called enterprise application.

- It has advantages of the high-level security, load balancing etc.

- In Java, EJB is used for creating enterprise applications.

Cont...

4) Mobile Application

- An application which is created for mobile devices is called a mobile application.

- Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

- There are Following platforms or editions of Java:

1) **Java SE (Java Standard Edition)**

    - It is a Java programming platform.

    - It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc.

Cont…

- It includes core topics like **OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.**

Cont…

2) Java EE (Java Enterprise Edition)

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

Cont…

3) Java ME (Java Micro Edition)

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

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**.

- Initially designed for small, embedded systems in electronic appliances like set-top boxes.

Cont...

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

- After that, it was called **Oak.**

Cont…

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

- In 1995, Oak was renamed as **”Java”**.

Cont…

- **Why java?** Java is an island of Indonesia where the first coffee was produced (called java coffee).

- Java name was chosen by James Gosling while having coffee near his office.

Cont…

- Java was developed by James Gosling, who is known as the father of Java

Java Version History

- Many java versions have been released till now.

- The current stable release of Java is Java SE 24

Cont...

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)

Cont...

- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)

Cont…

- Java SE 7 (28th July 2011)
- Java SE 8 (18th Mar 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th Mar 2018)

Features of Java

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

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

Cont...

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent

Cont...

5. Secured

6. Robust

7. Architecture neutral

1. Simple

- According to Sun, Java language is a simple programming language because:

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

Cont...

- 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.

2. Object-oriented

- Java is an object-oriented programming language.

- Everything in Java is an object and it follow the Object-oriented programming (OOPs) concept.

Cont...

- Basic concepts of OOPs are:

  A. Class

  B. Object

  C. Inheritance

Cont...

D. Polymorphism

E. Abstraction

F. Encapsulation

Cont…

A. Class

- A class is a group of objects which have common properties.

- It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

Cont…

- A class in Java can contain:
  - **Fields**
  - **Methods**
  - **Constructors**
  - **Blocks**
  - **Nested class and interface**

Cont...

B. Object

- An object is an instance of a class.
- An object in Java is the physical because when we are creating a object then memory will be allocated for object
- Object will be created with the help of new keyword.

Cont...

C. Inheritance

- **Inheritance in Java** is a mechanism in which one class inherit the property of another class.

D. Polymorphism

- **Polymorphism in Java** is a concept by which we can perform a single action in different ways.

Cont...

- Polymorphism is derived from 2 Greek words: poly and morphs.

- The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Cont…

E. Abstraction

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Cont...

F. Encapsulation

- **Encapsulation in Java** is a process of wrapping code (Function/ method) and data together into a single unit,

- for example, a capsule which is mixed of several medicines.

Capsule

3. Portable

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

4. Platform independent

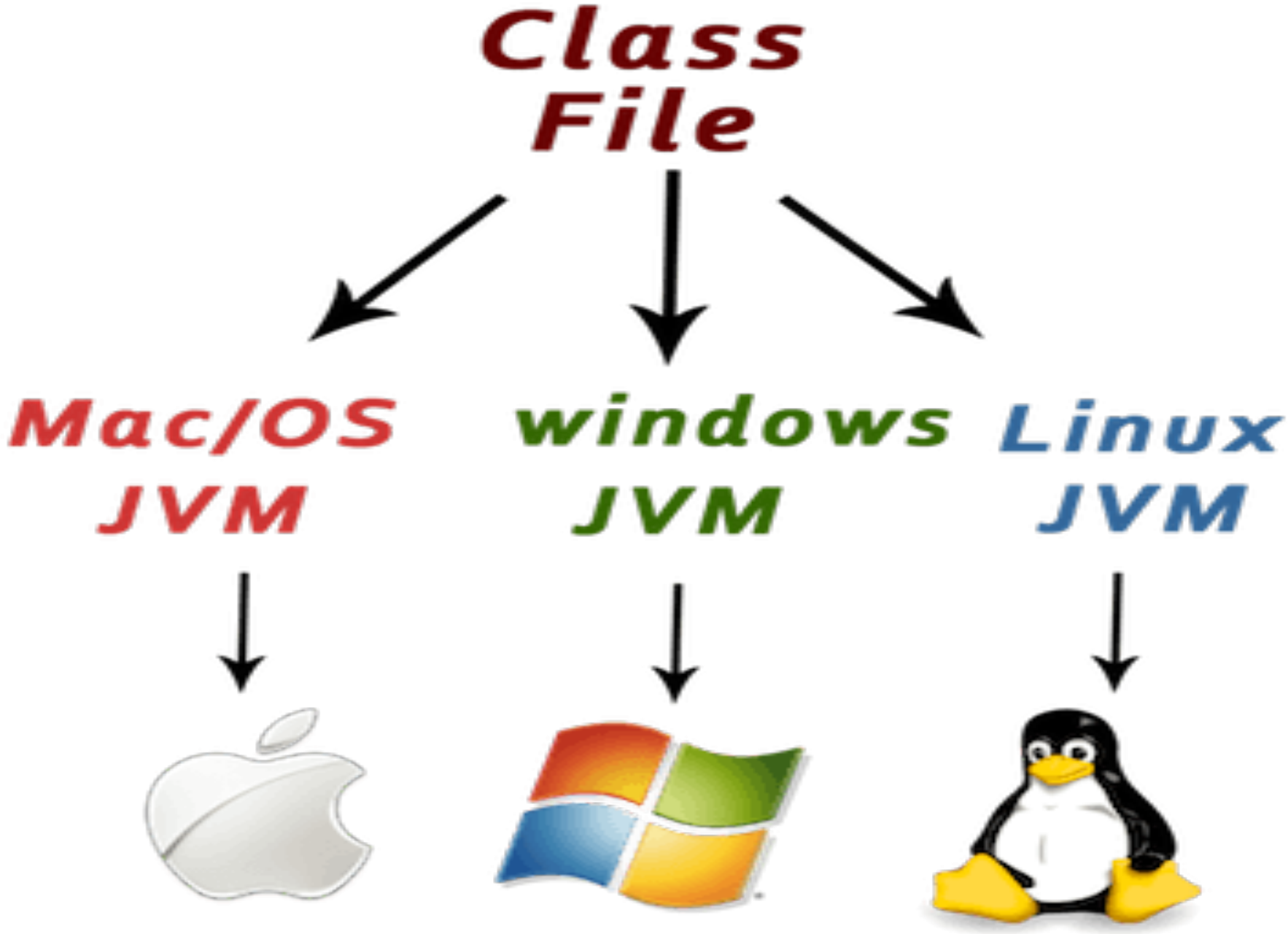
- Java is platform independent because Java code can be run on multiple platforms,

- for example, Windows, Linux, Sun Solaris, Mac/OS, etc.

Cont...

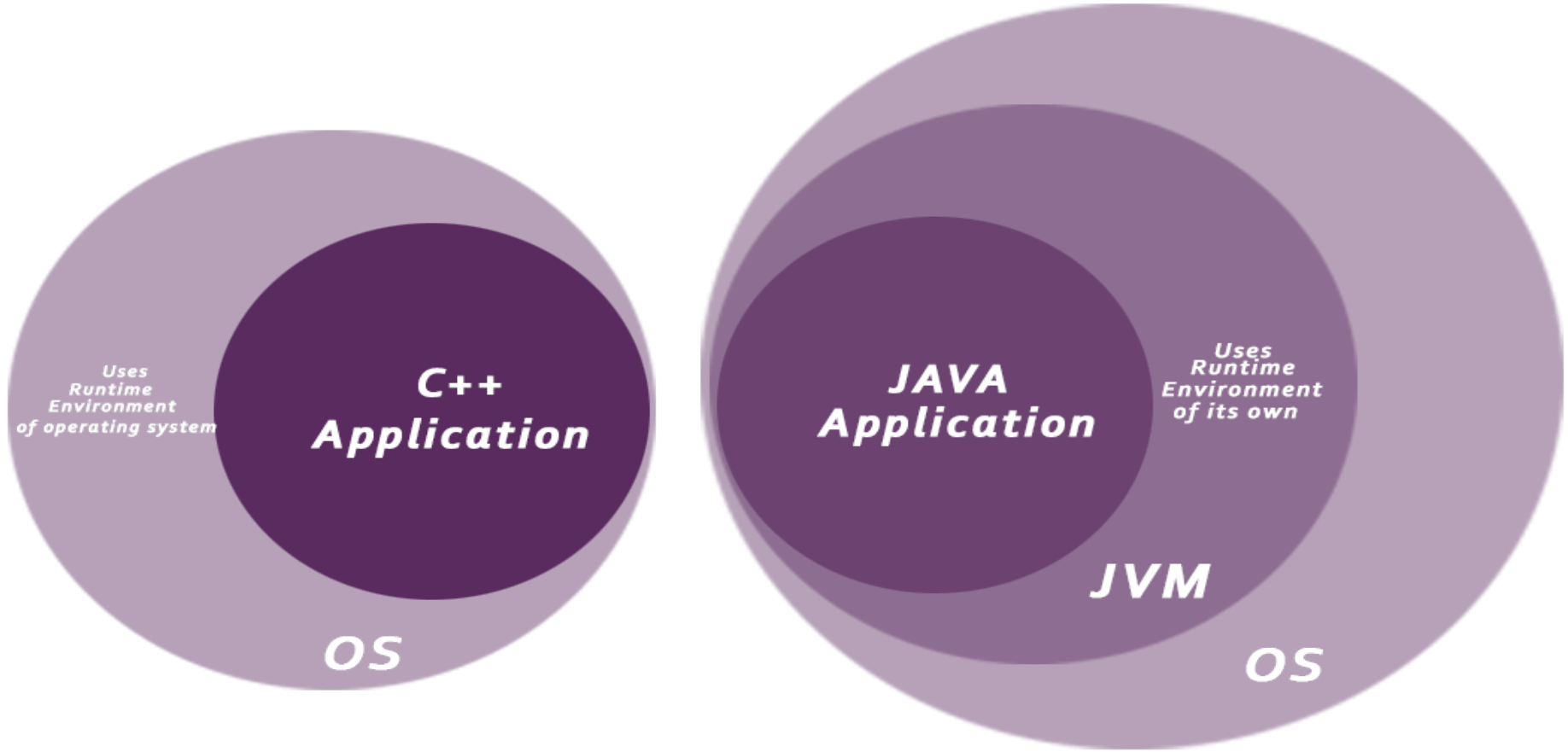
- Java code is compiled by the compiler and converted into bytecode.

- This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).


Class
File
Mac/OS
JVM
windows
JVM
Linux
JVM

5. Secured

- Java is secured because:
  - **No explicit pointer**
  - **Java Programs run inside a virtual machine sandbox**


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

6. Robust

- Robust simply means strong. Java is robust because:
  - It uses strong memory management.
  - There is a lack of pointers that avoids security problems.

Cont…

- 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.

- All these points make Java robust.

7. Architecture-neutral

- Java is architecture neutral because there are no implementation dependent features,

- for example, the size of primitive types is fixed.

Cont...

- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture.

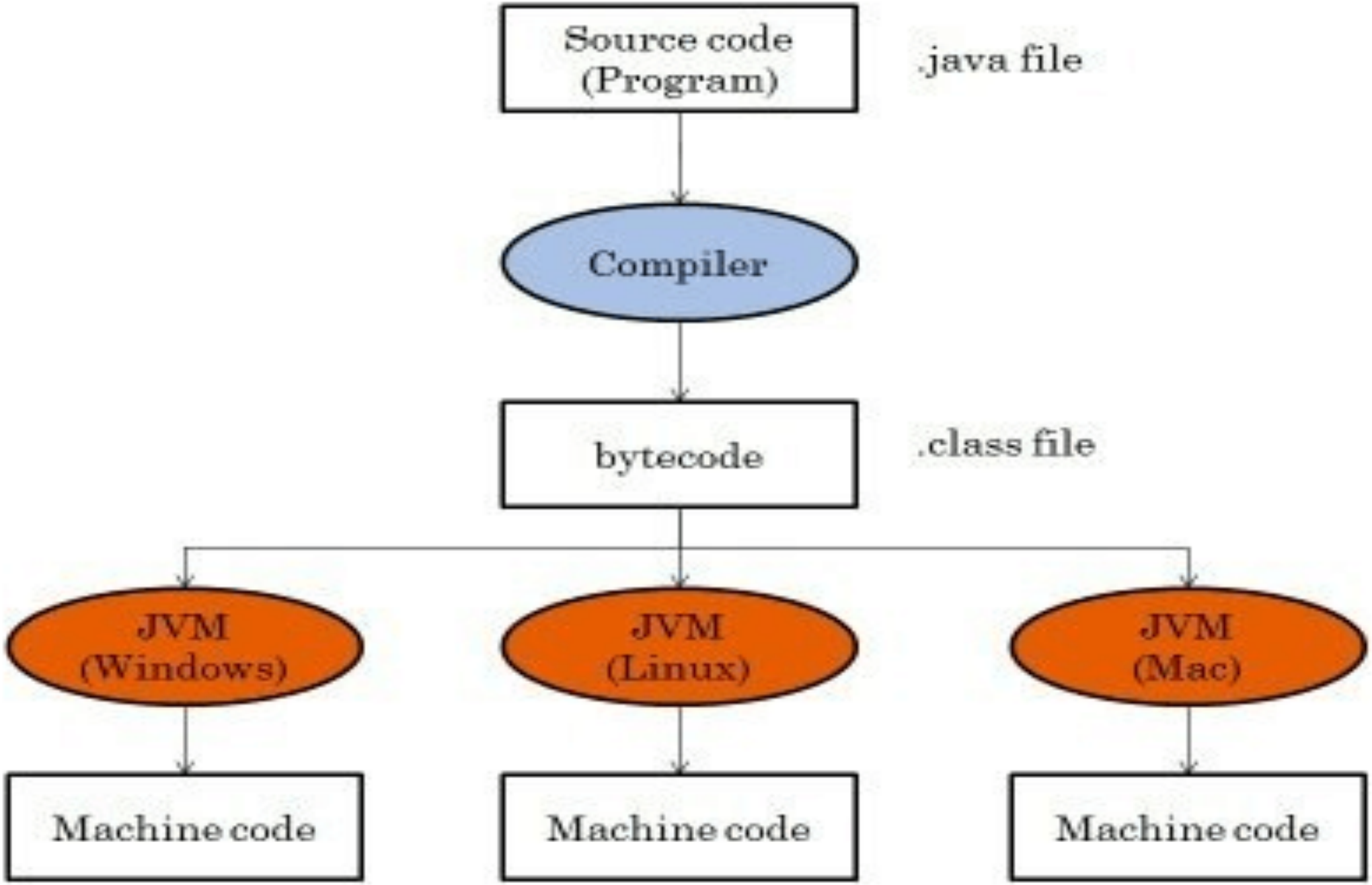
C++ vs Java

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.

Java Bytecode

- 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).


Source code
(Program)
.java file
Compiler
bytecode
.class file
JVM
(Windows)
JVM
(Linux)
JVM
(Mac)
Machine code
Machine code
Machine code

JVM (Java Virtual Machine)

- JVM (Java Virtual Machine) is an abstract machine.
- It is called a virtual machine because it doesn't physically exist.
- It is a specification that provides a runtime environment in which Java bytecode can be executed.

Cont...

- JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other.

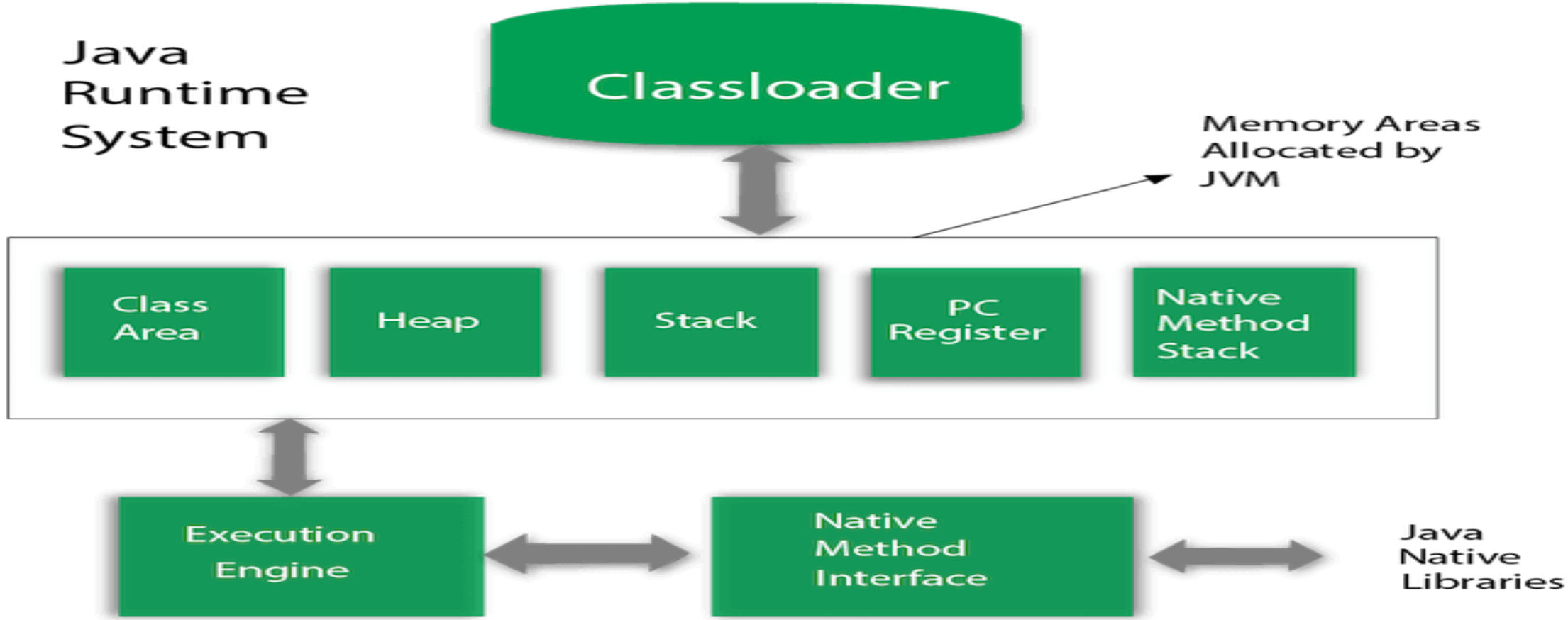
- However, Java is platform independent.

Cont...

- The JVM performs the following main tasks:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

JVM Architecture
(Internal architecture of JVM)

- It contains class loader, memory area, execution engine etc.



1) Classloader

- Classloader is a subsystem of JVM which is used to load class files.

- Whenever we run the java program, it is loaded first by the classloader.

2) Class(Method) Area

- Class(Method) Area stores per-class structures such as the runtime constant pool, method data, the code for methods.

Cont...

3) Heap

- It is the runtime data area in which objects are allocated.

4) Stack

- It holds local variables.

Cont...

5) Program Counter Register

- PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

- It contains all the native methods used in the application.

Cont...

7) Execution Engine

- It contains:
  - **A virtual processor**
  - **Interpreter:** Read bytecode stream then execute the instructions.
  - **Just-In-Time(JIT) compiler:** It is used to improve the performance.

Cont...

8) Java Native Interface

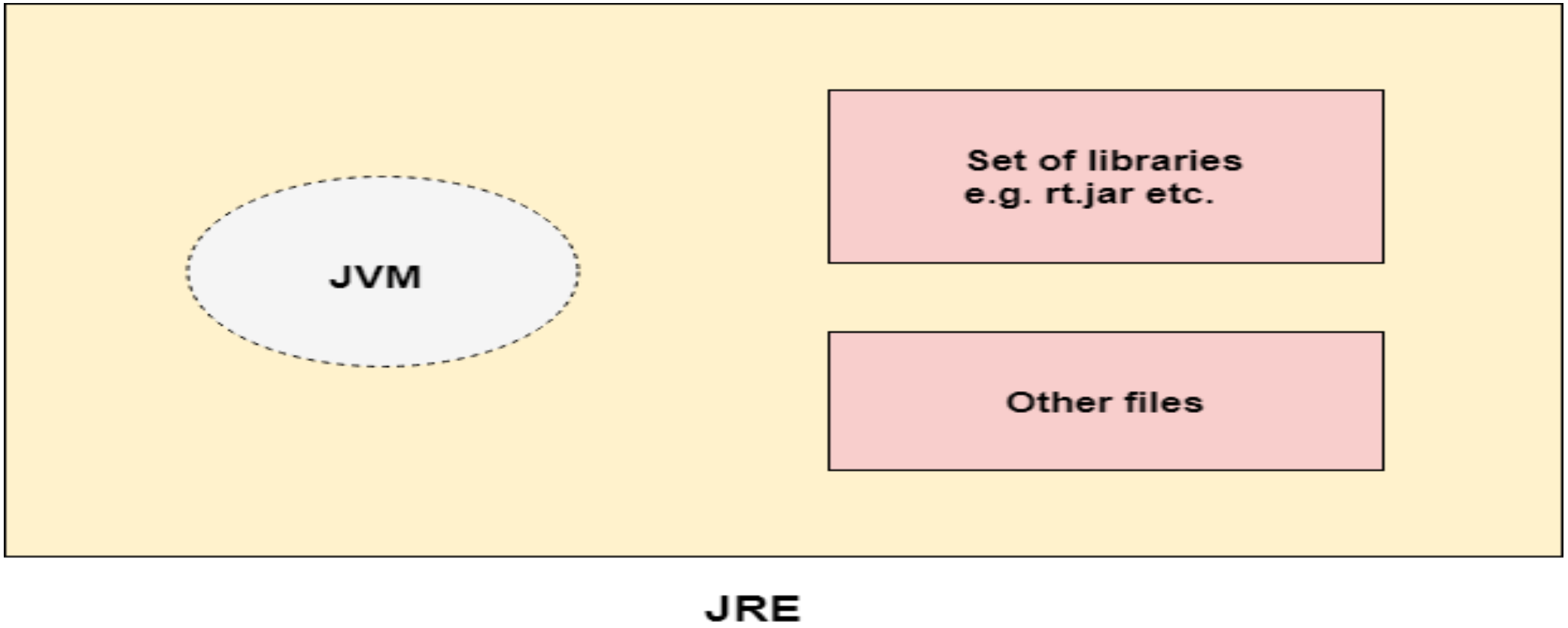
- Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc.

JRE

- JRE Stands for Java Runtime Environment.
- It is used to provide the runtime environment.
- It physically exists.

Cont...

- It contains a set of libraries + other files that JVM uses at runtime.



JDK

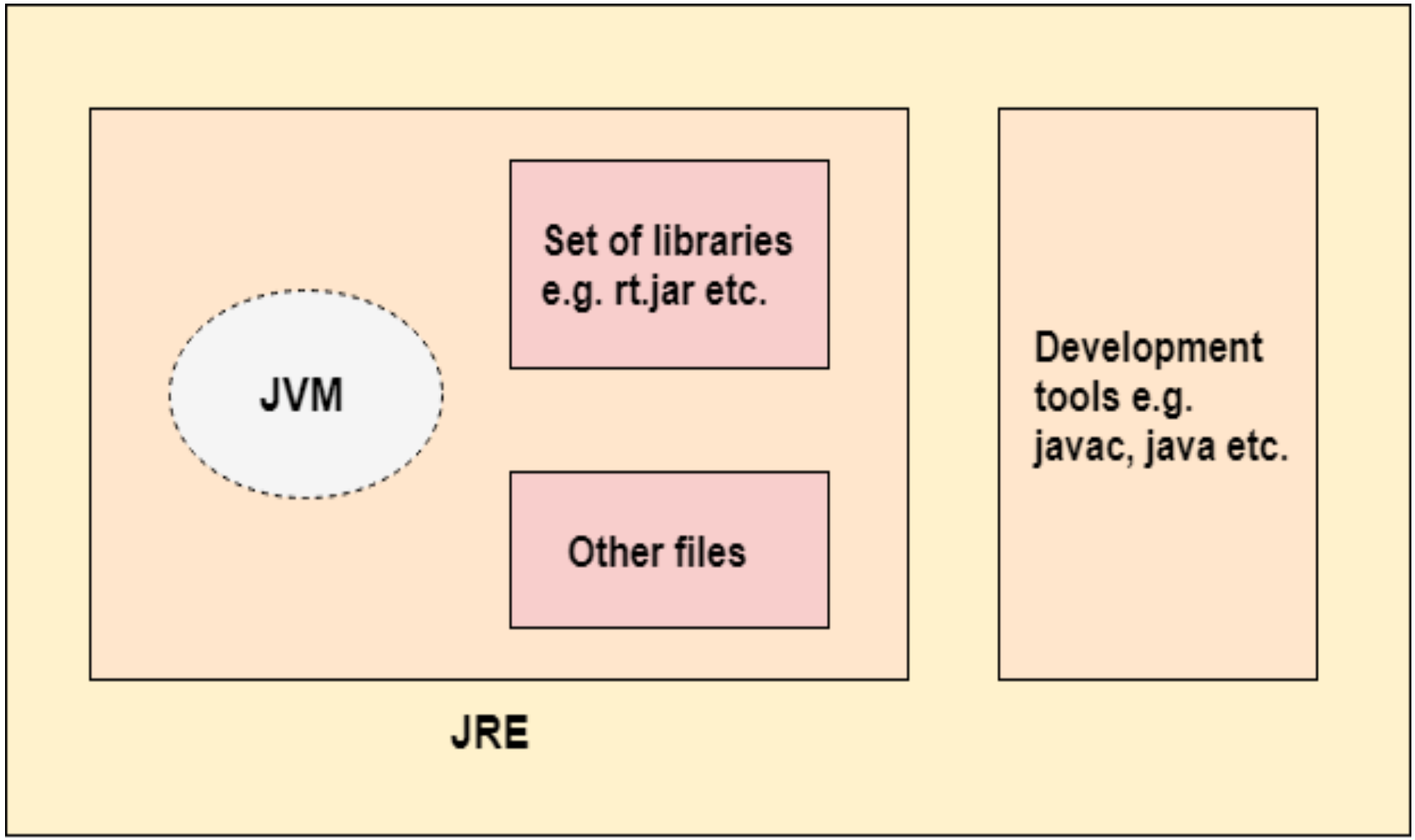
- JDK stands for Java Development Kit.

- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications

Cont...

- It physically exists.

- It contains JRE + development tools.


Set of libraries
e.g. rt.jar etc.
JVM
Other files
Development
tools e.g.
javac, java etc.
JRE
JDK

Just In Time (JIT) Compiler

- The Just-In-Time (JIT) compiler is a an essential part of the JRE i.e. Java Runtime Environment.

- It is responsible for performance optimization of java based applications at run time.

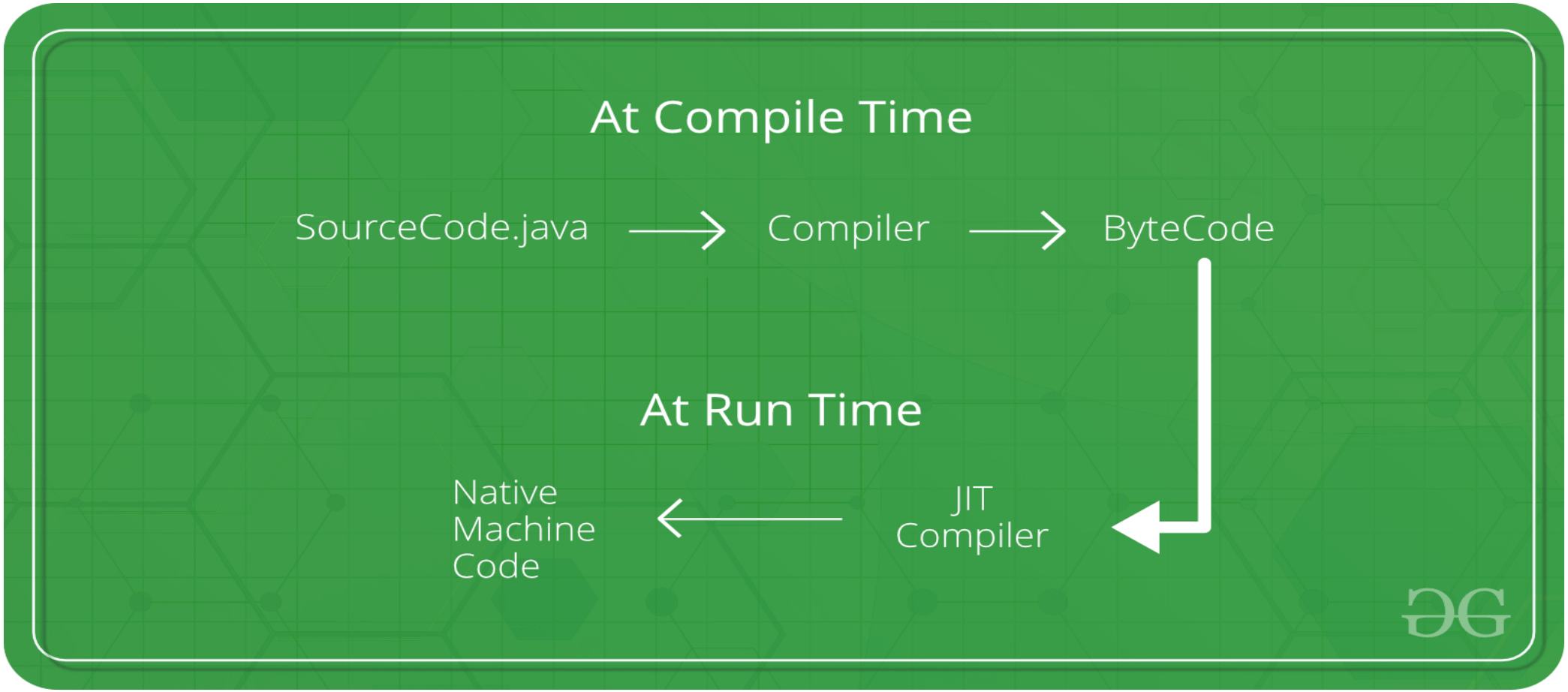
Cont...

- In order to speed up the performance, JIT compiler communicates with JVM at the execution time in order to compile byte code sequences into native machine code.

- Basically, when using JIT Compiler, the native code is easily executed by the hardware.

Cont...

- By doing so, there will be a huge gain in execution speed.



Java Tokens

- A **token** is the smallest element of a program that is meaningful to the compiler.

Types of Tokens

- Keywords
- Identifiers
- Literals
- Operators
- Separators
- Comments

Keywords

- These are the **pre-defined** reserved words of any programming language.
- Each keyword has a special meaning.
- It is always written in lower case. Java provides the following keywords:

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. throw	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

Cont…

- The keywords **const** and **goto** are reserved but not used.

Identifiers

- Identifiers are used to name a variable, constant, function, class, and array.

- It usually defined by the user.

Cont...

- It uses letters, underscores, or a dollar sign as the first character.

- Remember that the identifier name must be different from the reserved keywords.

Cont...

- There are some rules to declare identifiers are:
  - The first letter of an identifier must be a letter, underscore or a dollar sign. It cannot start with digits but may contain digits.
  - The whitespace cannot be included in the identifier.
  - Identifiers are case sensitive.

Cont...

- Some valid identifiers are:
  - PhoneNumber
  - PRICE
  - radius
  - a
  - a1
  - _phonenumber
  - $circumference
  - jagged_array

Cont...

- Invalid identifier names include these:
  - 2count
  - high-temp
  - not/ok

Literals

- In programming literal is a notation that represents a fixed value (constant) in the source code.

- It can be categorized as an integer literal, string literal, Boolean literal, etc.

Cont…

- It is defined by the programmer.

- Once it has been defined cannot be changed.

Cont...

- Java provides five types of literals are as follows:
  - Integer (Ex: 2, 4)
  - Floating Point (Ex: 12.3, 16.8)
  - Character (Ex: ‘A’, ‘a’)
  - String (Ex: “Ram”, “Hello”)
  - Boolean (Ex: true, false)

Separators

- The separators in Java is also known as **punctuators**.

- **Square Brackets []:** It is used to define array elements. A pair of square brackets represents the single-dimensional array, two pairs of square brackets represent the two-dimensional array.

- **Parentheses ():** It is used to call the functions.

Cont...

- **Curly Braces {}:** The curly braces denote the starting and ending of a code block.
- **Comma (,):** It is used to separate two values and parameters.
- **Assignment Operator (=):** It is used to assign a variable and constant.

Cont…

- **Semicolon (;):** It is the symbol that can be found at end of the statements. It separates the two statements.

- **Period (.):** It separates the package name form the sub-packages and class. It also separates a variable or method from a reference variable.

Java Comments

- The Java comments are the statements that are not executed by the compiler and interpreter.

- The comments can be used to provide information or explanation about the variable, method, class or any statement.

Cont…

- It can also be used to hide program code.

Types of Java Comments

- There are three types of comments in Java.
  - Single Line Comment
  - Multi Line Comment
  - Documentation Comment

1) Java Single Line Comment

- The single line comment is used to comment only one line.

- **Syntax:**

  //This is single line comment

Example:

```
class CommentExample1
{
        public static void main(String a[])
        {
                int i=10; //Here, i is a variable
                System.out.println(i);
        }
}
```

Output: 10

2) Java Multi Line Comment

- The multi line comment is used to comment multiple lines of code.

- **Syntax:**

```
/*
      This
      is
      multi line
      comment
*/
```

Example:

```
class CommentExample2
{
        public static void main(String a[])
        {
                /* Let's declare and
                    print variable in java. */

                int i=10;
                System.out.println(i);
        }
}
```

Output: 10

3) Java Documentation Comment

- The documentation comment is used to create documentation API.

- To create documentation API, you need to use **javadoc tool**.

- **Syntax:**

```
/**
        This
        is
        documentation
        comment
*/
```

Example:

```
/** The Calculator class provides methods to get addition and subtraction of given 2 numbers.*/
public class Calculator
{
        /** The add() method returns addition of given numbers.*/
        public int add(int a, int b)
        {
                return a+b;
        }
/** The sub() method returns subtraction of given numbers.*/
        public int sub(int a, int b)
        {
                return a-b;
        }
}
```

Cont...

- Compile it by javac tool:
  - javac Calculator.java

- Create Documentation API by javadoc tool:
  - javadoc Calculator.java

- Now, there will be HTML files created for your Calculator class in the current directory.

Whitespace

- Java is a free-form language.

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

Cont…

- **For example**, Program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator.

- In java, whitespace is a space, tab, or new line.

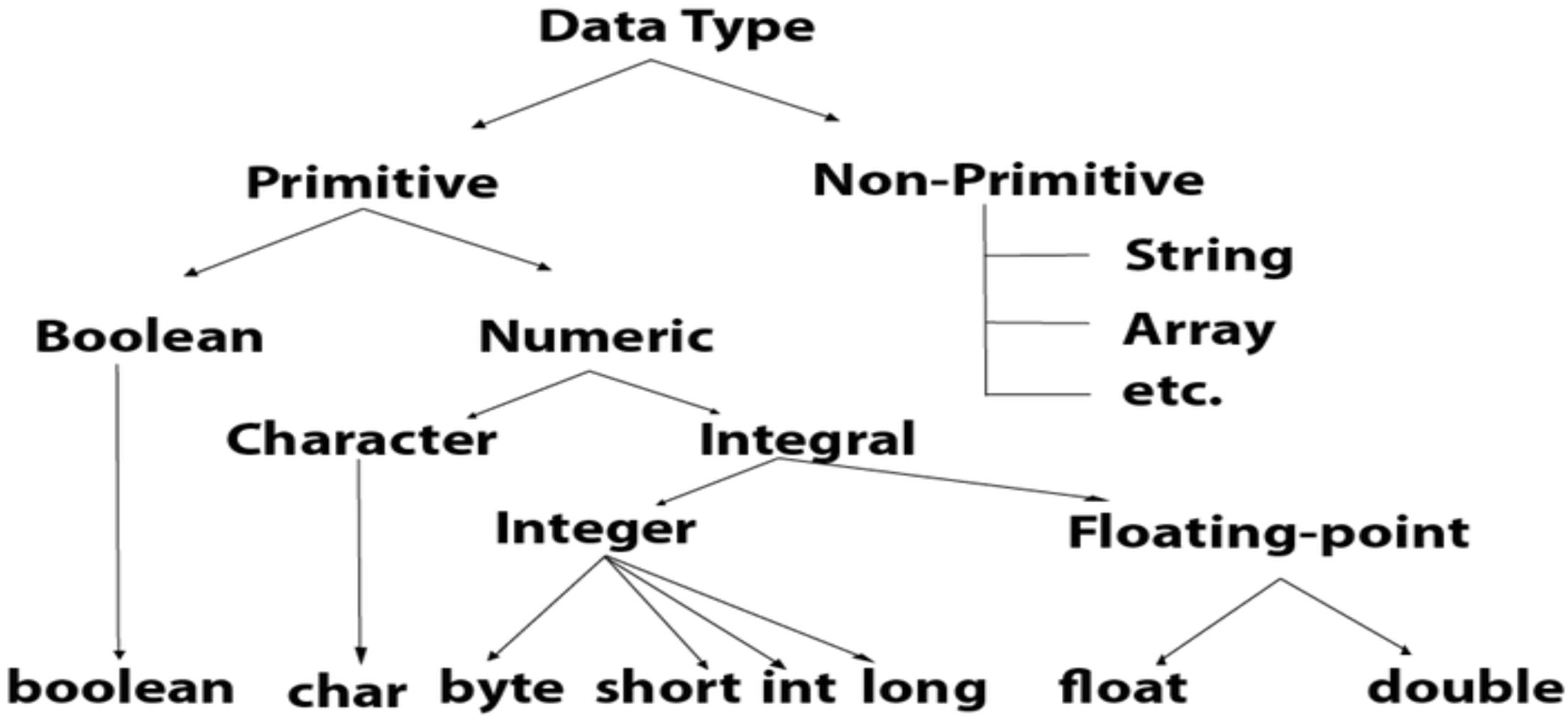
Data Types in Java

- Type of the data is called data type.

- Data types specify the different sizes and values that can be stored in the variable.

Cont...

- There are two types of data types in Java:
  - **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
  - **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Boolean Data Type

- The Boolean data type is used to store only two possible values: true and false.

- **Example:** boolean one = false;

Why char uses 2 byte in java and what is \u0000 ?

- It is because java uses Unicode system not ASCII code system.

- The \u0000 is the lowest range of Unicode system

Unicode System

- Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

- In unicode, character holds 2 byte, so java also uses 2 byte for characters.
  - lowest value:\u0000
  - highest value:\uFFFF

Java Variables

- **Variable** is name of reserved area allocated in memory.

- In other words, it is a name of memory location.

- A variable is a container which holds the value.

Cont...

- A variable is assigned with a data type.

- It is a combination of "vary + able" that means its value can be changed.

Declaration of a Variable

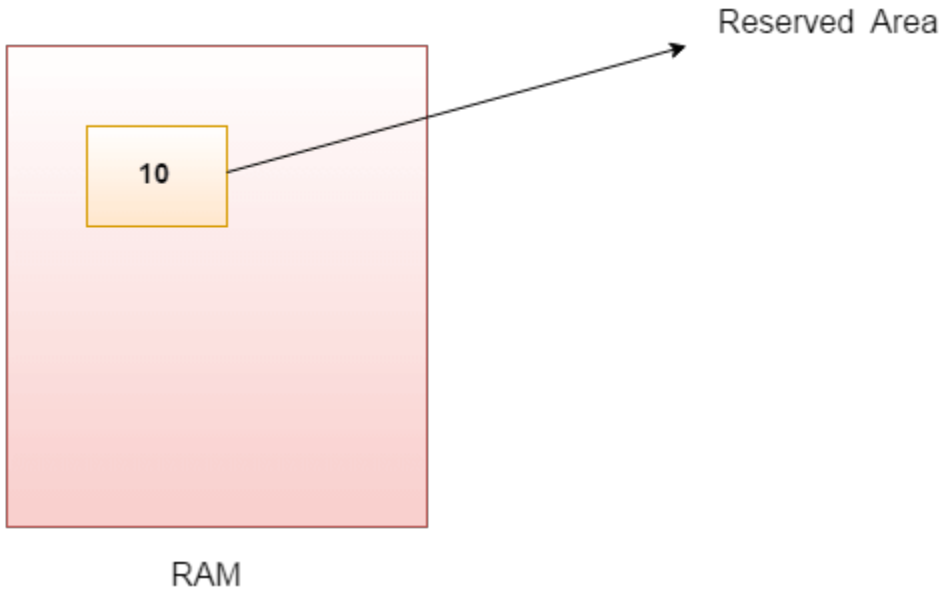
- **Syntax:** datatype variablename ;

- **Example:** int data;

Initialization of a Variable

- **Syntax:** datatype variablename = value ;

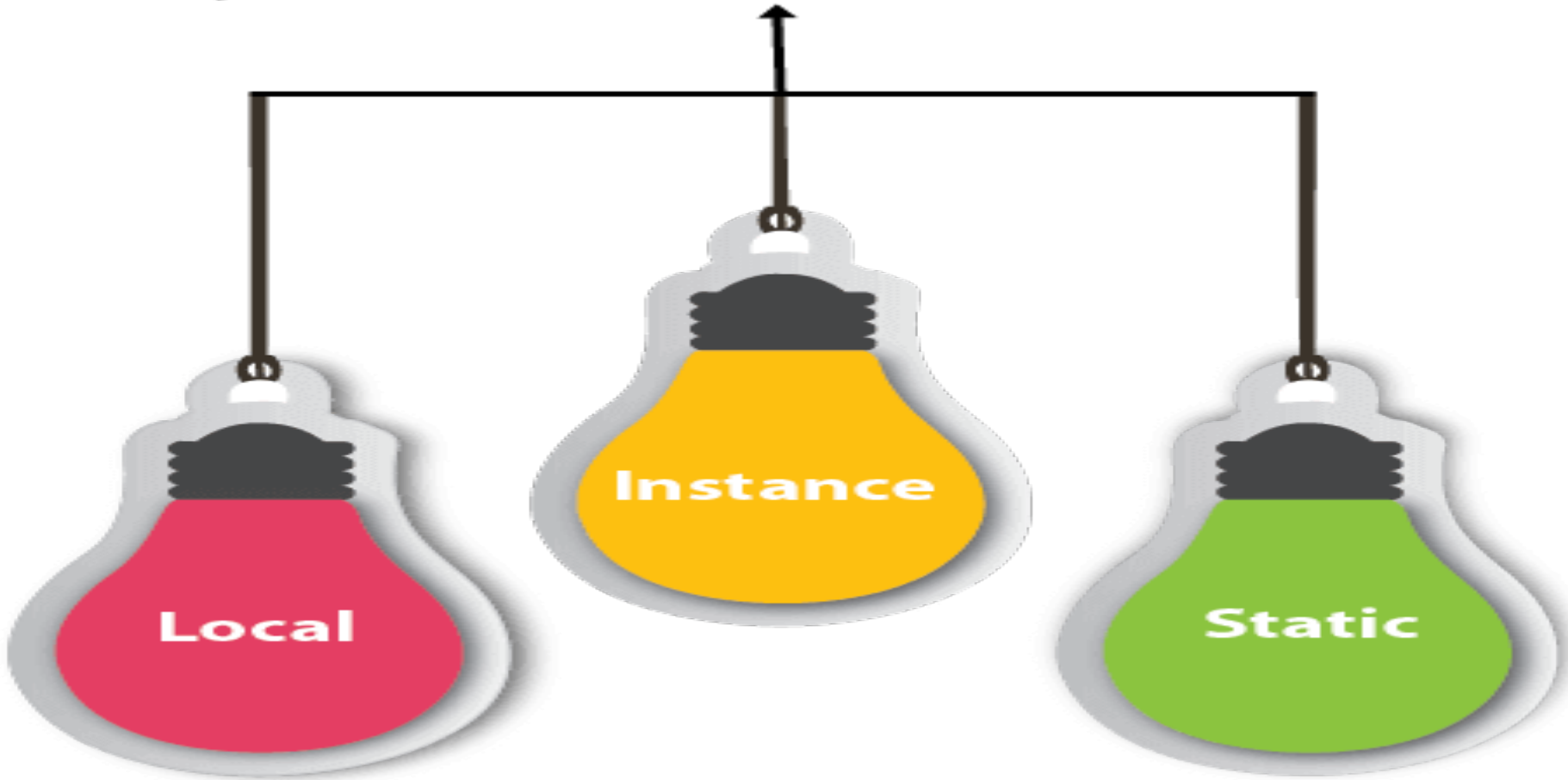
- **Example:** int data = 10;



Types of Variables

- There are three types of variables in Java:
  - local variable
  - instance variable
  - static variable

Types of Variables



1) Local Variable

- A variable declared inside the body of the method, block and constructor is called local variable.

- You can use local variable only within that method, block and constructor in which it is declared.

Cont…

- A local variable cannot be defined with "static" keyword.

2) Instance Variable

- A variable declared inside the class but outside the body of the method, block and constructor is called instance variable.

- It is not declared as static.

Cont…

- It is called instance variable because its value is instance (object) specific and is not shared among instances (objects).

3) Static variable

- A variable which is declared as static is called static variable.

- It can not be local.

Cont…

- You can create a single copy of static variable and share among all the instances (object) of the class.

- Memory allocation for static variable happens only once when the class is loaded in the memory.

Example

```
class A
{
        int data=50;//instance variable
        static int m=100;//static variable
        void method()
        {
                int n=90;//local variable
        }
}//end of class
```

Java Constant

- **Constant** is a value that can not be changed after assigning it.

- In Java, to declare any variable as constant, we use final modifiers (keyword).

Syntax to declare a constant :

- **final** datatype variablename = value;

- **Example: final double** PRICE=432.78;

Example

```
class HelloWorld
{
        public static void main(String  a[])
        {
                final int b =60;
                System.out.println(b);
        }
}
```

Type Casting in Java

- Convert a value from one data type to another data type is known as **type casting**.

Types of Type Casting

- There are two types of type casting:
  - Widening Type Casting
  - Narrowing Type Casting

Widening Type Casting

- Converting a lower data type into a higher one is called **widening** type casting.

- It is also known as **implicit conversion** or **upcasting**.

- It is done automatically.

Cont...

- It takes place when:
  - Both data types must be compatible with each other.
  - The target type must be larger than the source type.

- **byte -> short -> char -> int -> long -> float -> double**

Example

```
class A
{
        public static void main(String a[])
        {
                int x = 7;
                //automatically converts the integer type into long type
                long y = x;
                //automatically converts the long type into float type
                float z = y;
                System.out.println(x);
                System.out.println(y);
                System.out.println(z);
        }
}
```

Output

7

7

7.0

Narrowing Type Casting

- Converting a higher data type into a lower one is called **narrowing** type casting.

- It is also known as **explicit conversion** or down **casting**.

- It is done manually by the programmer.

Cont…

- If we do not perform casting then the compiler reports a compile-time error.

- **double -> float -> long -> int -> char -> short -> byte**

Example

```
class A
{
        public static void main(String  a[])
        {
                double d = 166.66;
                //converting double data type into long data type
                long l = (long)d;
                //converting long data type into int data type
                int i = (int)l;
                System.out.println(d);
                //fractional part lost
                System.out.println(l);
                //fractional part lost
                System.out.println(i);
        }
}
```

Output

166.66

166

166

Java Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.

- The arguments passed from the console can be received in the java program and it can be used as an input.

Simple example of command-line argument in java

```
class A
{
        public static void main(String a[])
        {
                System.out.println(a[0]);
        }
}
```

Compile

javac A.java

run

java A 10

Output

10

Give Output-

1.
```
class Test {
    public static void main(String[] args) {
        int a = 5, b = 2;
        System.out.println(a % b);
    }
}
```

2.
```
class Test {
    public static void main(String[] args) {
        int a = 7;
        System.out.println(++a);
    }
}
```

3.
```
class Test {
    public static void main(String[] args) {
        int x = 10;
        System.out.println(x++ + ++x);
    }
}
```

4.
```
class Test {
    public static void main(String[] args) {
        boolean a = true, b = false;
        System.out.println(a && b);
    }
}
```

Give Output-

5.
```
class Test {
    public static void main(String[] args) {
        int a = 4;
        System.out.println(a << 1);
    }
}
```

6.
```
class Test {
    public static void main(String[] args) {
        int a = 10, b = 20;
        System.out.println(a < b && a++ < 15);
        System.out.println(a);
    }
}
```

7.
```
class Test {
    public static void main(String[] args) {
        int a = 10, b = 5, c = 20;
        System.out.println(a > b ? (a > c ? a : c) : (b > c ? b : c));
    }
}
```

8.
```
class Test {
    public static void main(String[] args) {
        int a = 5, b = 10;
        System.out.println(a == b);
    }
}
```

Give Output-

9.class Test {
public static void main(String[] args) {
int x = 10;
x += 5 * 2;
System.out.println(x);
}
}

10. class Test {
public static void main(String[] args) {
int a = 7, b = 3;
System.out.println(a & b);
}
}

11. class Test {
public static void main(String[] args) {
int a = 12, b = 25;
System.out.println(a | b);
}
}

12. class Test {
public static void main(String[] args) {
int x = 5;
System.out.println(x++ + x++ + ++x);
}
}

Give Output-

13.class Test {
public static void main(String[] args) {
int a = 5;
System.out.println(a++ * --a + a++);
}}

14.class CastDemo {
public static void main(String[] args) {
char c = 'A';
int i = c;
System.out.println(i);
}}

15. class CastDemo {
public static void main(String[] args) {
int a = 257;
byte b = (byte) a;
System.out.println(b);
}}

16.class Test {
public static void main(String[] args) {
int a = 5, b = 10;
System.out.println((a > b) ? (a - b) : (b - a));
}
}

17.class Test {
int a = 5; // instance variable
public static void main(String[] args) {
Test t1 = new Test();
Test t2 = new Test();
t1.a = 10;
System.out.println(t1.a);
System.out.println(t2.a);
}
}

1. 1
2. 8
3. 22
4. false
5. 8
6. true 11
7. 20
8. false
9. 20
10. 3
11. 29
12. 19
13. 30
14. 65
15. 1
16. 5
17. 10
5

result = a % 256
(if result > 127 → subtract 256 to bring into range -128 to 127)

Java Arrays

- Normally, an array is a collection of similar type of elements which has contiguous memory location.

- **Java array** is an object which contains elements of a similar data type.

Cont...

- Additionally, The elements of an array are stored in a contiguous memory location.

- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Cont…

- Unlike C/C++, we can get the length of the array using the **length member.**

Advantages

- **Code Optimization:** It makes the code optimized.

- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

- To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

- There are two types of array.
  - Single Dimensional Array
  - Multidimensional Array

Single Dimensional (1-D) Array in Java

- An Array that will contain only one subscript is called Single Dimensional Array.

Syntax to Declare 1-D Array in Java

- dataType arrayname[]; (or)
- dataType[] arrayname; (or)
- dataType []arrayname;

Example

- int a[];

Instantiation of 1-D Array in Java

- **Syntax:**
  - arrayname =**new** datatype[size];

- **Example:**
  - a=new int[3];

Declaration and instantiation in one line

- **Example:**
  - int a[] = new int[3];

Example of Java 1-D Array

```
class Testarray
{
        public static void main(String a1[])
        {
                int a[]=new int[5]; //declaration and instantiation
                a[0]=10; //initialization
                a[1]=20;
                a[2]=70;
                a[3]=40;
                a[4]=50;
                //traversing array
                for(int i=0; i<a.length; i++) //length is the property of array
                        System.out.println(a[i]);
        }
}
```

Declaration, Instantiation and Initialization of Java 1-D Array

- We can declare, instantiate and initialize the java array together.

- **Example:**
  - **int** a[]={33,3,4,5};//declaration, instantiation and initialization

Example

```
class Testarray1
{
        public static void main(String a[])
        {
                int a[]={33,3,4,5};//declaration, instantiation and initialization

                //printing array
                for(int i=0;i<a.length;i++)  //length is the property of array
                        System.out.println(a[i]);
        }
}
```

For-each Loop for Java Array

- The Java for-each loop prints the array elements one by one.

The syntax of the for-each loop is given below:

```
for(datatype variablename:arrayname)
{
        //body of the loop
}
```

Example

```
//Java Program to print the array elements using for-each loop
class Testarray1
{
        public static void main(String a[])
        {
                int arr[]={33,3,4,5};
                //printing array using for-each loop
                for(int i : arr)
                        System.out.println(i);
        }
}
```

Multidimensional Array in Java

- An Array that will contain more than one subscript is called Multi Dimensional Array.

Types of Multidimensional Array

- 2-D Array
- 3-D Array
- n-D Array

2-D Array

- An Array that will contain two subscript is called Two Dimensional Array.

Syntax to Declare 2-D Array in Java

- dataType[][] arrayname; (or)
- dataType [][]arrayname; (or)
- dataType arrayname[][];
- **Example**
  - int a[][] ;

Instantiation of 2-D Array in Java

- **Syntax:**
  - arrayname =**new** datatype[rowsize][columnsize];

- **Example:**
  - a=new int[3][3];
-

Declaration and instantiation in one line

- **Example:**
  - int a[][] = new int[3][3];

Example to initialize 2-D Array in Java

- arr[0][0]=1;
- arr[0][1]=2;
- arr[0][2]=3;
- arr[1][0]=4;
- arr[1][1]=5;
- arr[1][2]=6;
- arr[2][0]=7;
- arr[2][1]=8;
- arr[2][2]=9;

Declaration, Instantiation and Initialization of Java 2-D Array

- We can declare, instantiate and initialize the java array together.

- **Example:**
  - **int** a[][]={{1,2,3},{4,5,6}};//declaration, instantiation and initialization

Example of 2-D Array

```
class A
{
        public static void main(String a[])
        {
                int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
                for(int i=0;i<3;i++)
                {
                        for(int j=0;j<3;j++)
                        {
                                System.out.print(arr[i][j]);
                        }
                        System.out.println();
                }
        }
}
```

Java String

- Generally, String is a sequence of characters.
- But in Java, string is an object that represents a sequence of characters.
- The java.lang.String class is used to create a string object.

How to create a string object?

- There are two ways to create String object:
  - By string literal
  - By new keyword

1) String Literal

- Java String literal is created by using double quotes.

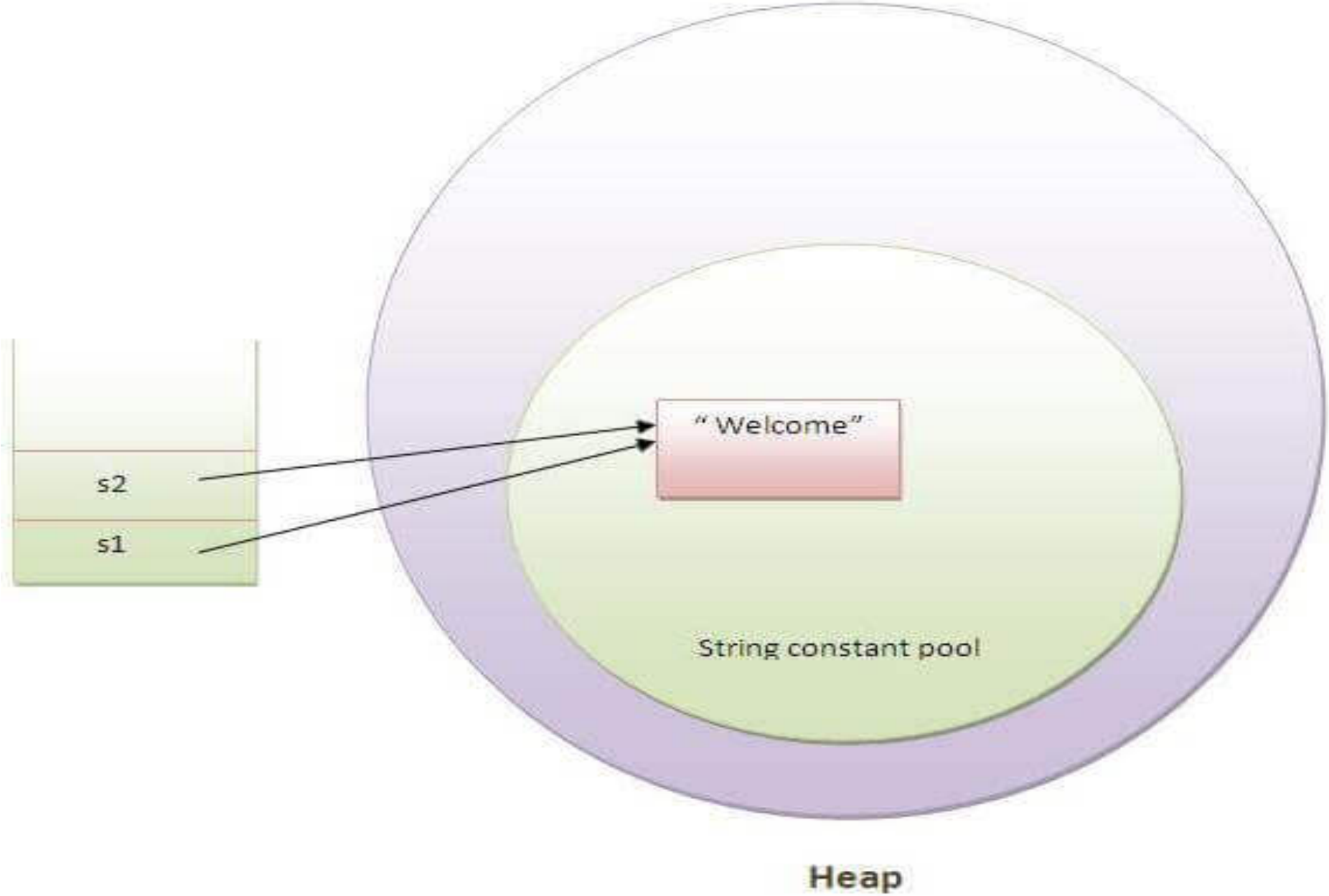
- Example:
    - String s="welcome";

Cont…

- Each time you create a string literal, the JVM checks the "string constant pool" first.
- If the string already exists in the pool, a reference to the pooled instance is returned.
- If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

Cont...

- Example:
  - String s1="Welcome";
  - String s2="Welcome";//It doesn't create a new instance


s2
s1
" Welcome"
String constant pool
Heap

Why Java uses the concept of String literal?

- To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

2) By new keyword

- Example:

  String s=**new** String("Welcome");//creates two objects and one reference variable

- In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Immutable String in Java

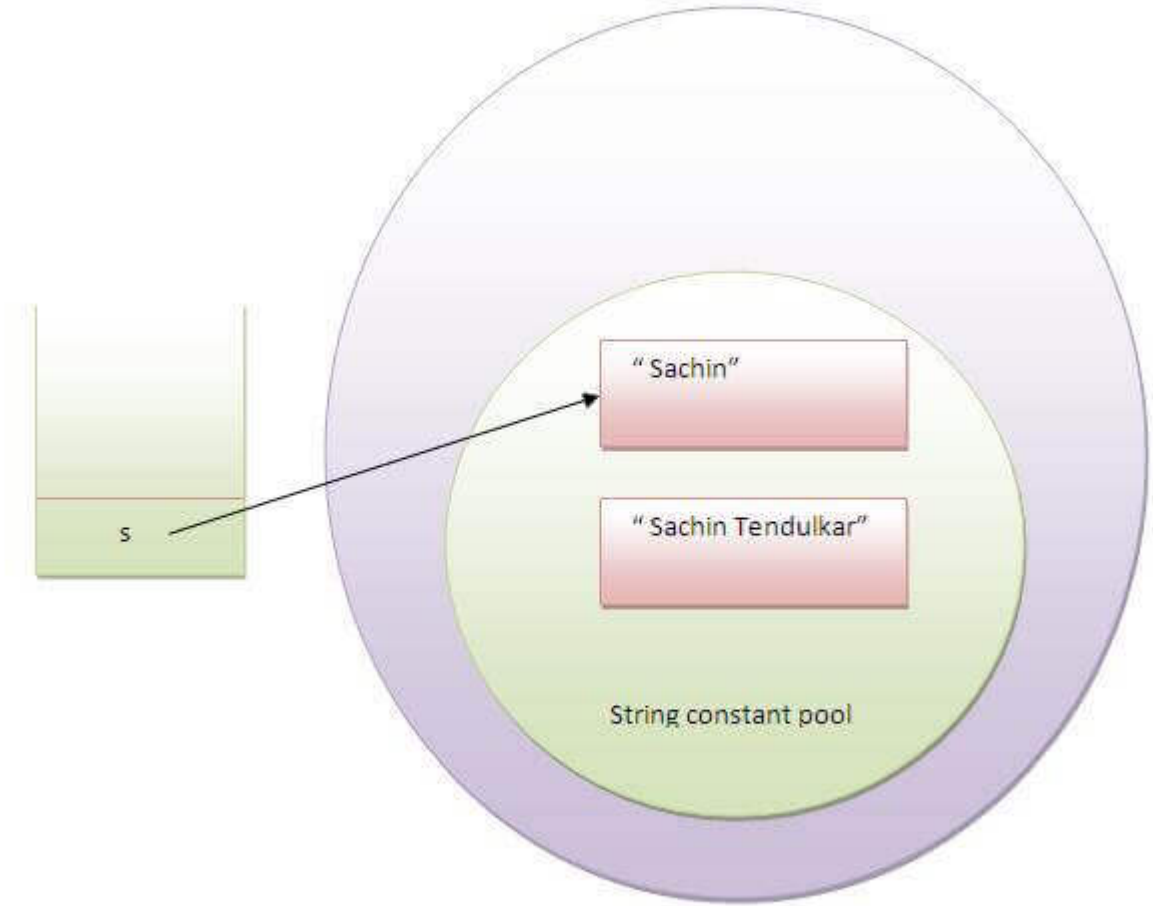
- In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

- Once string object is created its data can't be changed but a new string object is created.

Example

```
class A
{
        public static void main(String args[])
        {
                String s="Sachin";
                s.concat(" Tendulkar");//concat() method appends the string at the end
              System.out.println(s);//will print Sachin because strings are immutable objects
        }
}
```

Output:Sachin


" Sachin"
" Sachin Tendulkar"
s
String constant pool
Heap

Why string objects are immutable in java?

- Because java uses the concept of string literal.

- Suppose there are 3 reference variables,all referes to one object "sachin".

- If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

Java String class methods

- The java.lang.String class provides many useful methods to perform operations on sequence of char values (String).

Cont...

1. char charAt(int index): returns char value for the particular index.

Example

```
public class CharAtExample
{
        public static void main(String args[])
        {
                String name="javatpoint";
              char ch=name.charAt(4);//returns the char value at the 4th index
                System.out.println(ch);
        }
}
```

Output: t

Cont...

2. int length(): returns string length.

Example:

```
public class LengthExample
{
        public static void main(String a[])
        {
                String s1="javatpoint";
                System.out.println(s1.length());//10 is the length of javatpoint string
        }
}
```

Output:

10

Cont…

3. String substring(int beginIndex, int endIndex): returns substring for given begin index and end index.

Example

```
public class SubstringExample
{
        public static void main(String a[])
        {
                String s1="javatpoint";
                System.out.println(s1.substring(2,4));//returns va
        }
}
```

Output:

va

Cont…

- String toLowerCase(): returns a string in lowercase.

Example

```
public class StringLowerExample
{
        public static void main(String a[])
        {
                String s1="JAVATPOINT HELLO stRIng";
                System.out.println(s1.toLowerCase());
        }
}
```

Output:

javatpoint hello string

Cont…

- String toUpperCase(): returns a string in uppercase.

Example

```
public class StringUpperExample
{
        public static void main(String a[])
        {
                String s1="hello string";
                System.out.println(s1.toUpperCase());
        }
}
```

Output:
HELLO STRING

String Concatenation in Java

- There are two ways to concat string in java:
  - By + (string concatenation) operator
  - By concat() method

1) String Concatenation by + (string concatenation) operator

- Java string concatenation operator (+) is used to add strings.

Example 1

```
class TestStringConcatenation1
{
        public static void main(String a[])
        {
                String s="Sachin"+"Tendulkar";
                System.out.println(s);
        }
}
```

Output: SachinTendulkar

Example 2

```
class TestStringConcatenation2
{
        public static void main(String a[])
        {
                String s=50+30+"Sachin"+40+40;
                System.out.println(s);
        }
}
```

Output: 80Sachin4040

Cont…

- **Note:** After a string literal, all the + will be treated as string concatenation operator.

2) String Concatenation by concat() method

- The String concat() method concatenates the specified string to the end of current string.

- **Syntax:**
  - **public** String concat(String another)

Example

```
class TestStringConcatenation3
{
        public static void main(String a[])
        {
                String s1="Sachin";
                String s2="Tendulkar";
                String s3=s1.concat(s2);
                System.out.println(s3);
        }
}
```

Output: SachinTendulkar

Some More String Functions

- toString()
- getChars()
- equalsIgnoreCase()
- compareTo()
- lastIndexOf()
- valueOf()…..

- And many more…….

Give Output

1.
```
public class Test {
    public static void main(String[] args) {
        String s = “Java programming”;
        System.out.println(s.length());
    }
}
```

2.
```
public class Test {
    public static void main(String[] args) {
        String s = "Programming";
        System.out.println(s.indexOf('g'));
    }
}
```

3.
```
public class Test {
    public static void main(String[] args) {
        String s1 = new String("Java");
        String s2 = new String("Java");
        System.out.println(s1.equals(s2));
    }
}
```

4.
```
public class Test {
    public static void main(String[] args) {
        String str = "Hello";
        System.out.println(str.charAt(str.length() - 1));
    }
}
```

Give Output

5.
```
public class Test {
    public static void main(String[] args) {
        int[] arr = new int[3];
        arr[1] = 10;
        System.out.println(arr[0] + arr[1]);
    }
}
```

7.
```
public class Test {
    public static void main(String[] args) {
        String s = "Hello";
        System.out.println(s.isEmpty());
    }
}
```

6.
```
public class Test {
    public static void main(String[] args) {
        String s = "abcd";
        System.out.println(s.contains(“cd"));
    }
}
```

8.
```
import java.util.Arrays;
public class Test {
    public static void main(String[] args) {
        int arr[] = {2, 4, 6, 8};
        System.out.println(Arrays.toString(arr));
    }
}
```

Give Output

9.
```
public class Test {
    public static void main(String[] args) {
        String s = "Hello World";
        System.out.println(s.startsWith("Hell"));
    }
}
```

10.
```
public class Test {
    public static void main(String[] args) {
        String s = "banana";
        System.out.println(s.indexOf("na"));
    }
}
```

11.
```
public class Test {
    public static void main(String[] args) {
        String s1 = "Test";
        String s2 = s1;
        s1 = s1 + "ing";
        System.out.println(s1 + " " + s2);
    }
}
```

12.
```
public class Test {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.println(arr[0] + arr[arr.length - 1]);
    }
}
```

Give Output

13.
```
import java.util.Arrays;
public class Test {
    public static void main(String[] args) {
        int[][] matrix = {{1, 2}, {3, 4}, {5, 6}};
        System.out.println(matrix[2][0] * matrix[0][1]);
    }
}
```

14.
```
public class Test {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4};
        String result = "";
        for (int i : arr) {
            result += i;
        }
        System.out.println(result);
    }
}
```

Answers

- 1. 16
- 2. 3
- 3. true
- 4. o
- 5. 10
- 6. true
- 7. false
- 8. [2,4,6,8]
- 9. true
- 10. 2
- 11. Testing Test
- 12. 6
- 13. 10
- 14.1234