JAVA ENTERPRISE APPLICATION DEVELOPMENT

Ms. Archana A

Department of Computer Applications

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Java Fundamentals

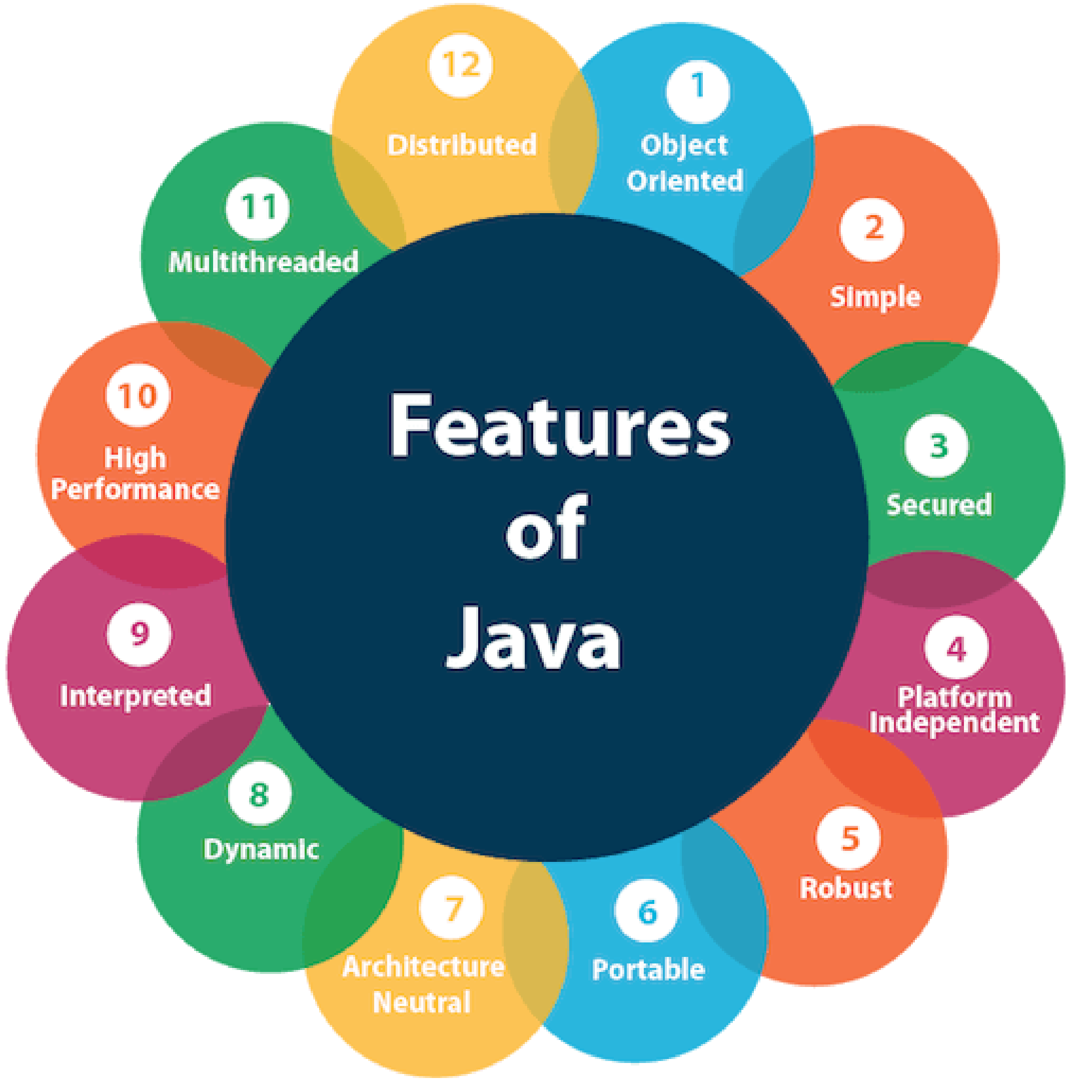
Ms. Archana A
Department of Computer Applications

Java Evolution

Java History:

- Java is a general-purpose, object-oriented programming language developed by **Sun Microsystems of USA in 1995**.
- Originally called **Oak** by **James Gosling**, one of the inventors of the language.
- Java was designed for the development of software for **consumer electronic devices** like **TVs, VCRs, toasters** and such other electronic machines.
- Java is **simple**, **portable** and **highly** reliable language.

Features of Java



Java Features

- Inventors of Java wanted to design a language that is not **only reliable, portable, and distributed** but also **simple, compact and interactive**.

- Sun Microsystems officially describes Java with the following features:
  1. **Compiled and Interpreted**
  2. **Platform-independent and Portable**
  3. **Object-oriented**
  4. **Robust and Secure**
  5. **Distributed**
  6. **Familiar, simple and small**
  7. **Multithreaded and interactive**
  8. **High Performance**
  9. **Dynamic and Extensible**

Java Features

1. Compiled and Interpreted

- Usually a computer language is either compiled or interpreted. Java combines both these approaches, Thus Java is a **two-stage system**.

- First java compiler translates source code into **bytecode** instructions. Bytecodes are not machine instructions and therefore, in second stage **Java interpreter** generates machine code that can be directly executed by the machine that is running the java program.

Java Features

2. Platform-independent and Portable

- The most significant contribution of java over other languages is its **portability**.

- Java programs can be easily moved from one computer to another, anywhere and anytime.

- Java ensures portability in two ways, first java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine-independent.

Java Features

3. Object-oriented

- Java is a true object oriented language. Almost everything in java is an **object.**
- All program code and data reside within objects and classes.

- Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by **inheritance.**

- The object model in java is simple and easy to extend.

4. Robust and Secure

- Java is a robust language. It provides many safeguards to ensure reliable code. It has **strict compile time and run time checking for data types**.
- It is designed as garbage-collected language which helps programmer to **handle all memory management problem virtually**.

- **Security** becomes an important issue for a language that is used for **programming on internet**.
- Java systems not only verify all memory access but also ensure that no viruses are communicated. The **absence of pointers** in java ensures that programmers **cannot gain access to memory locations without proper authorization**.

Java Features

5. Distributed

- Java is designed as a distributed language for creating applications on networks.

- It has ability to share both data and programs.

- Java applications can **open and access remote objects** on internet easily. This enables **multiple programmers at multiple remote locations to collaborate and work together on a single project.**

Java Features

6. Familiar, simple and small

- Redundant or unreliable code of c and c++ are not part of java.
- For eg: java does not use **pointers, preprocessor header files, operator overloading, multiple inheritance** and many others.

- To make the language look familiar to the existing programmers **java was modelled on C and C++ language**. Java uses many constructs of C and C++ therefore java code "look like C++" code, infact java is simplified version of C++.

Java Features

7. Multithreaded and interactive

- Multithreaded means **handling multiple tasks simultaneously**.
- Java supports multithreaded, which means we need not wait for one application to finish one task before beginning another.
- For eg: we can listen to an audio clip while scrolling a page and at same time download an app.
- Java runtime comes with tools that **support multiprocess synchronization** and construct smoothly running interactive system.

8. High Performance

- Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode.
- According to Sun, Java architecture is also designed to reduce overheads during runtime.
- Multithreading feature enhances the overall execution speed of java program.

9. Dynamic and Extensible

- Java is capable of dynamically linking in **new class libraries, methods and objects.**
- Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.
- Java programs support functions written in other languages such as C/C++. These functions are known as ***native* methods**. Native methods are linked dynamically at runtime.

Java Applications

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

1. Desktop Applications such as **acrobat reader, media player, antivirus**, etc.
2. Web Applications such as **irctc.co.in**,, etc.
3. Enterprise Applications such as **banking applications**.
4. **Mobile**
5. **Embedded System**
6. **Smart Card**
7. **Robotics**
8. **Games,** etc.

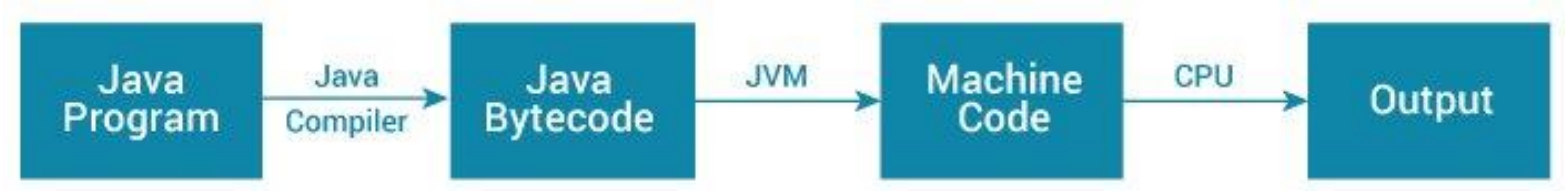
JAVA ENTERPRISE APPLICATION DEVELOPMENT

How Java works ?

Ms. Archana A

Department of Computer Applications

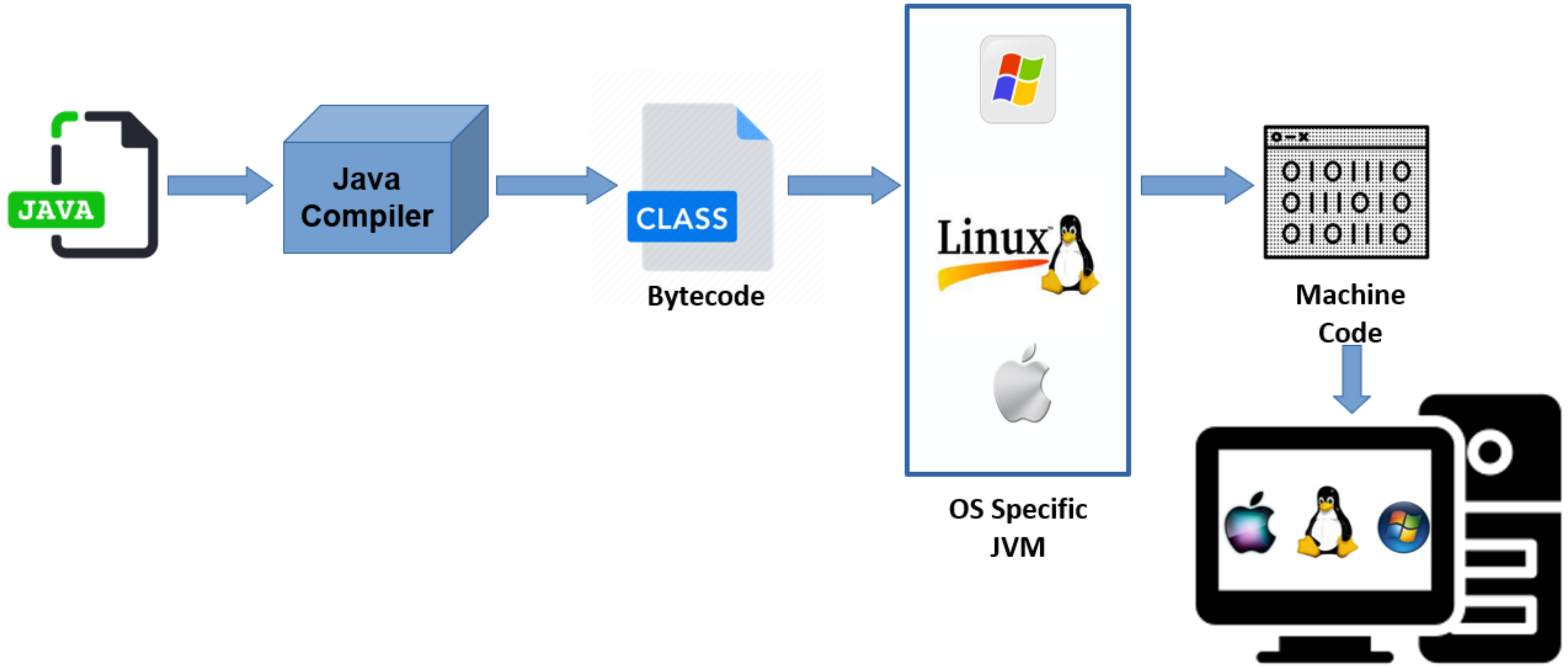
Working of Java program



- Java is platform independent but JVM is dependent
- Every system have JVM installed in it in-spite of any operating system

Working of Java

How does bytecode works ?



Java fundamentals…

– **Key Points**

- A bytecode in Java is a set of byte-long instructions that Java compiler produces and Java interpreter (JVM) executes.
- When Java compiler compiles **.java file**, it generates a series of bytecode (machine independent code)and stores them in a .class file.
- JVM then interprets and executes the byte code stored in the **.class** file and converts them into machine code.
- The byte code remains the same on different platforms such as Windows, Linux, and Mac OS.

Java fundamentals…

- **Java Interpreter**
  - It is a program that is implemented in C and C++ with the name java.exe.
  - It is **platform-dependent**.
    - Java interpreter converts the bytecode into the native code line by line.
    - It executes the program on your system.

JVM…

What is JVM?

- JVM (**Java Virtual Machine**) is an **abstract machine that enables the computer to run a Java program.**

- Java is a platform-independent language. It is because when we write Java code, it is ultimately written for JVM but not for physical machine (computer). Since JVM executes the Java bytecode which is platform-independent, hence Java is platform-independent.

JVM…

- **Java Virtual Machine…**
  - JVM is the core of the Java ecosystem, and makes it possible for Java-based software programs to follow the "**write once, run anywhere**" approach.
  - We can write Java code on one machine, and run it on any other machine using the JVM.
  - JVM was initially designed **to support only Java**.
  - However, over the time, many other languages such as **Scala, Kotlin and Groovy** were adopted on the Java platform.
    - All of these languages are collectively known as **JVM languages**

JVM…

- In programming languages like C and C++, the code is first compiled into platform-specific machine code.
  - **These languages are called compiled languages.**
- Languages like JavaScript and Python, the computer executes the instructions directly without having to compile them.
  - **These languages are called interpreted languages**

JVM…

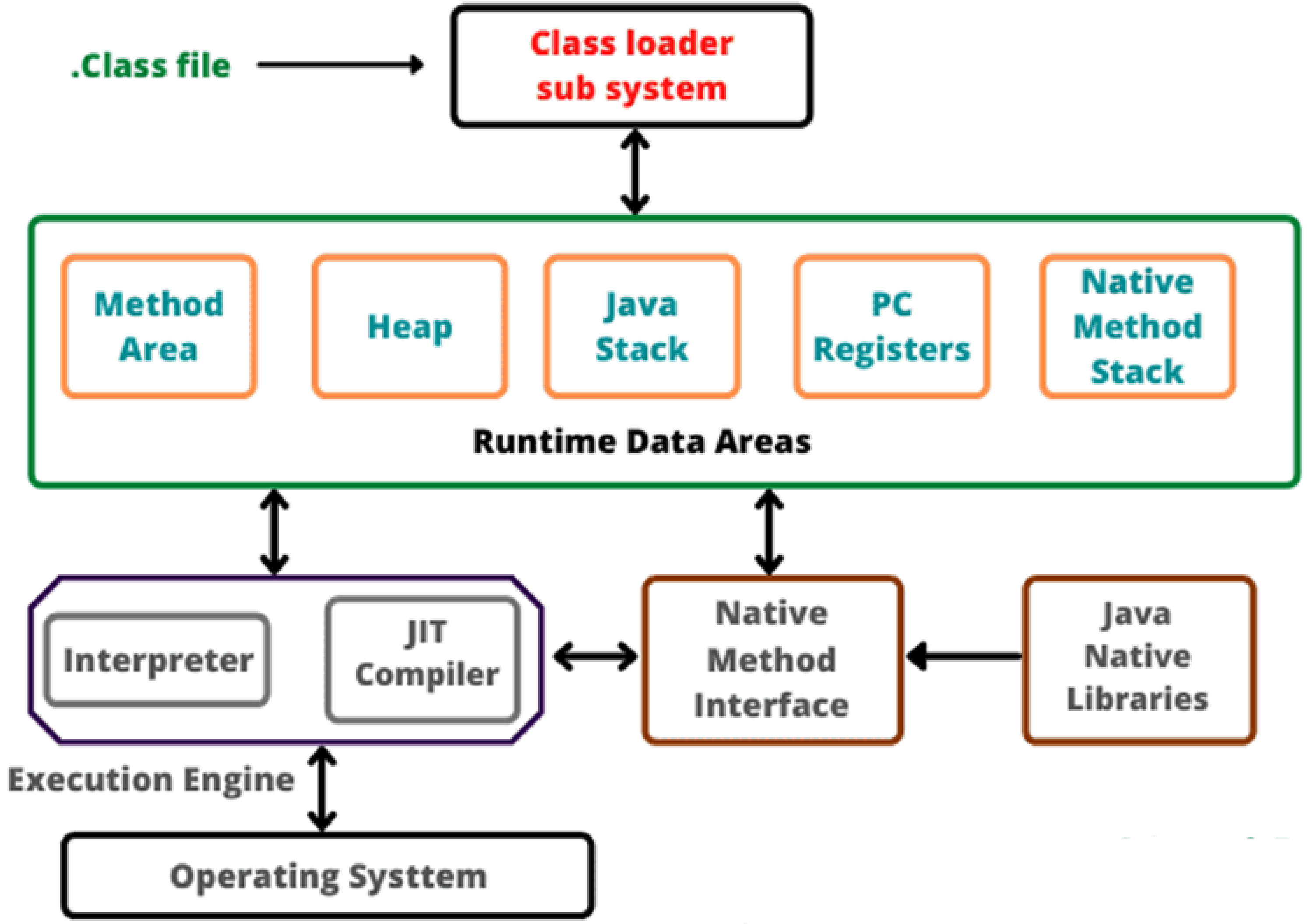
- The JVM creates an isolated space on a host machine.

- This space can be used to execute Java programs irrespective of the platform or operating system of the machine.

- The JVM consists of **three distinct components**:
  - Class Loader
  - Runtime Memory/Data Area
  - Execution Engine

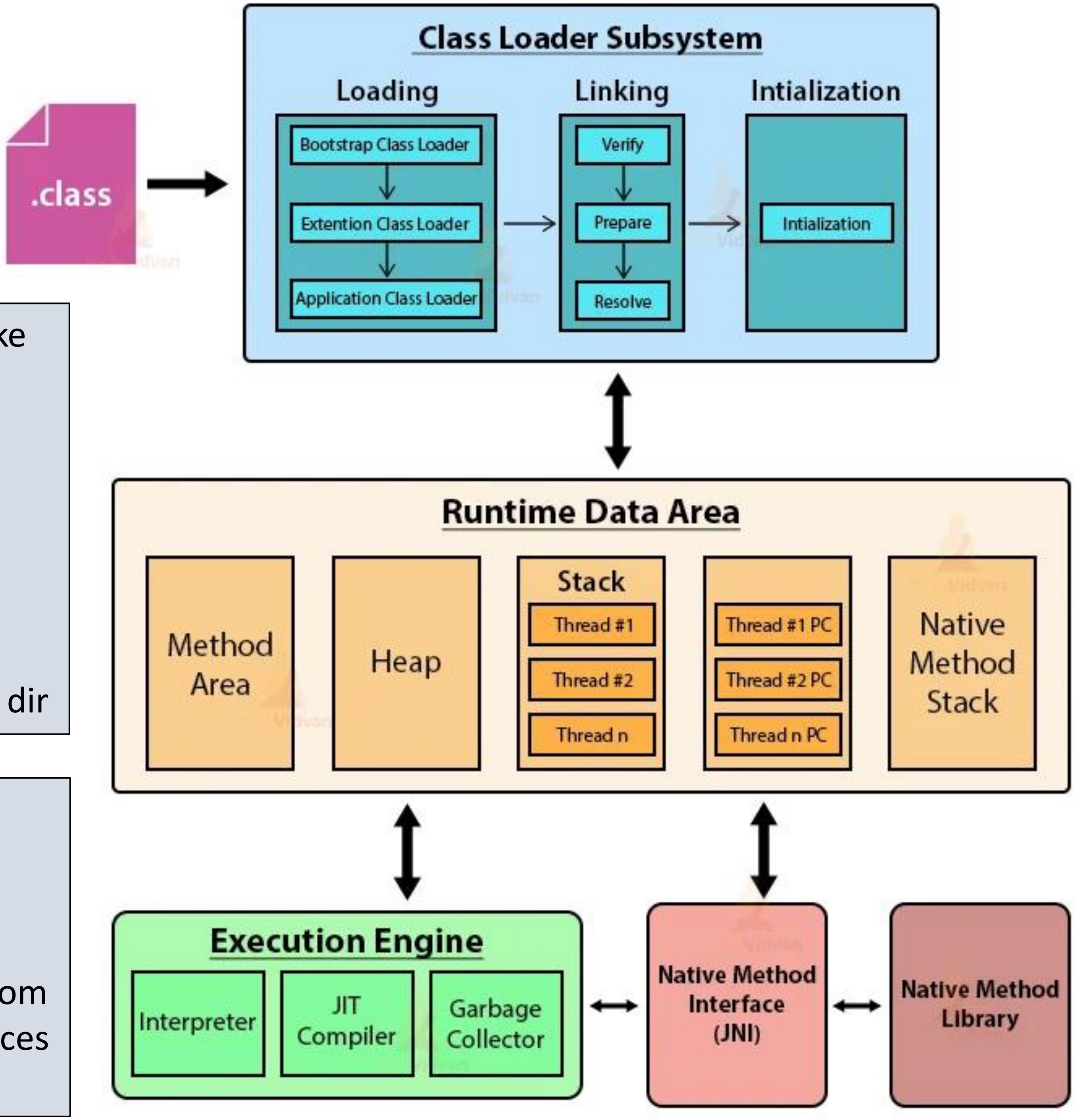
JRE…

JVM →


.Class file
Class loader sub system
Method Area
Heap
Java Stack
PC Registers
Native Method Stack
Runtime Data Areas
Interpreter
JIT Compiler
Execution Engine
Native Method Interface
Java Native Libraries
Operating Systtem

JEAD

JVM - Architecture



Bootstrap: Loads std. classes like
lang, util, io etc.
(lib folder)

Extention: std. libraries
(lib/ext folder)

Application: libraries from
ClassPath-current dir

Verify: Java 11 on Java 8?

Prepare: init. Static variables

Resolve: Symbolic constants from
other classes/references

JRE…

What is JRE?

- JRE (**Java Runtime Environment**) is a software package that provides **Java class libraries**, **Java Virtual Machine** (JVM), and other components that are required to run Java applications.

- **JRE** is the **superset of JVM**.

- If you need to run Java programs, but not develop them, JRE is what you need. You can download JRE from **Java SE Runtime Environment 8 Downloads** page

JDK…

What is JDK?

- JDK (**Java Development Kit**) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.

- In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).

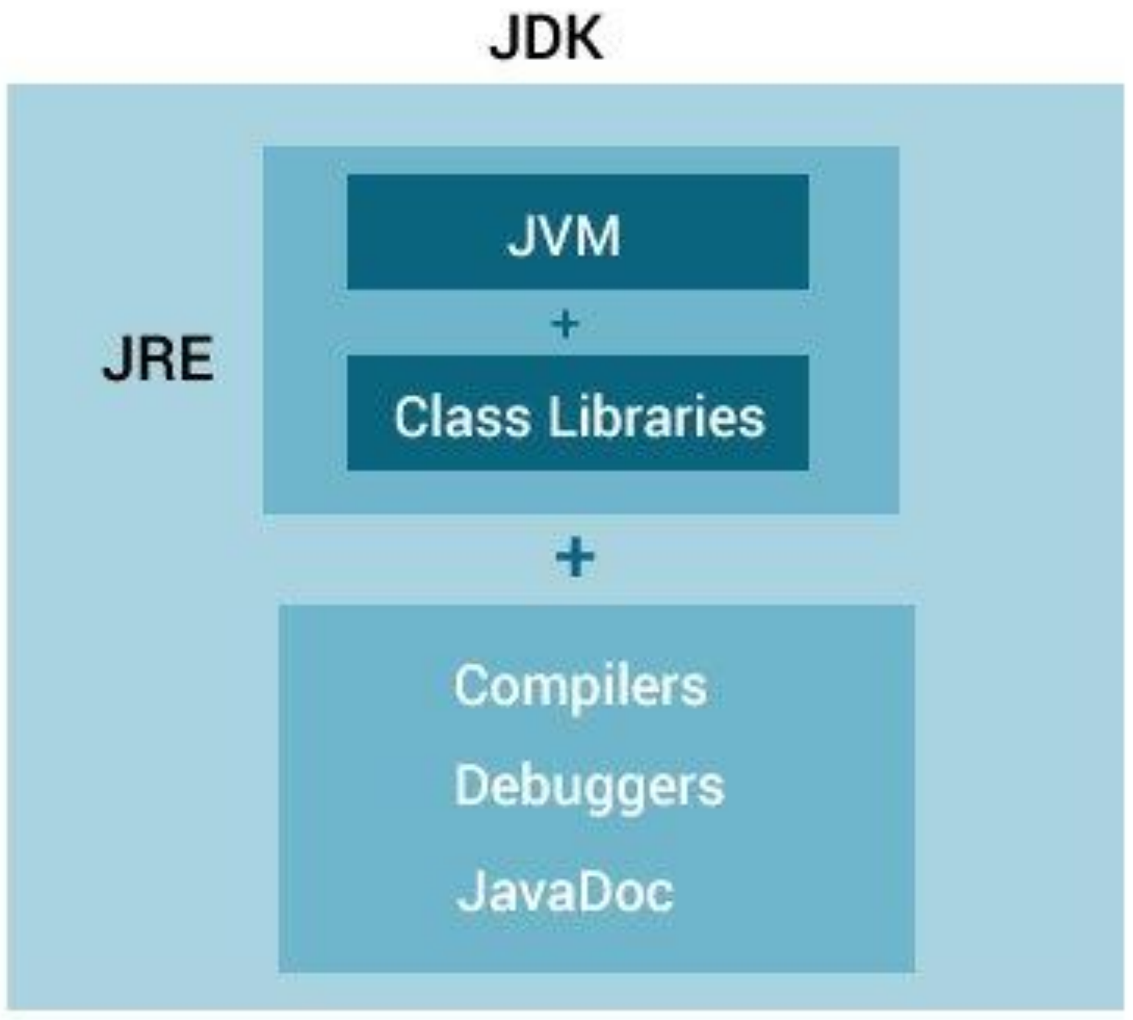
JDK…

Java Development Kit:
The Java development kit comes with a **collection of tools** that are used for developing and running Java programs. They include:

- **javac (Java compiler)**
- **java (Java interpreter)**
- **javap (Java disassembler) –** which enables to convert bytecode files into program description
- **javah (for C header files)**
- **javadoc (for creating HTML documents)**
- **jdb (Java debugger)**

JEAD

JDK, JVM, JRE



Assignment

- Java versions
- Differentiate between C, C++ and Java
- Compare – JDK vs JRE vs JVM

Java Installation

- Download Java (oracle.com)
  https://www.oracle.com/java/technologies/downloads/

- Install Java (open JDK) Guide
  System Requirements:
  https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html#A1097784

  Installation Guide:
  https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_jdk_install.html#CHDEBCCJ

Java Compiler

- ‘**javac**’:
  - Checks for Syntax errors
  - Converts **.java** file to **.class** intermediate file (byte code)

Execution of Java Program

- Execute .class files (without extension)- java
- '**java**' - JVM
  - ✓ loads the class file specified
  - ✓ Calls main() method for execution

JEAD
Java Coding conventions

- Class / Interface names
- Method names / variable names
- Constants

```
class Test {
            int num = 10;
            int Num = 20;
            int NUM = 30;
            int hhhhhhhhhhhhhhhhhhhhhhhhHhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh=20;
}
```

Java coding conventions

Naming Conventions:

- **Class Names:** Should start with an uppercase letter and use CamelCase (e.g., MyClass).
- **Interface Names:** Should start with an uppercase I and use CamelCase (e.g., MyInterface).
- **Method Names:** Should start with a lowercase letter and use CamelCase (e.g., myMethod).
- **Variable Names:** Should start with a lowercase letter and use CamelCase (e.g., myVariable).
- **Constant Names:** Should be in uppercase with words separated by underscores (e.g., MAX_VALUE).

Java coding conventions…

Indentation and Formatting:

- Use 4 spaces for each level of indentation.
- Use curly braces even for one-line blocks.
- Place opening curly braces at the end of the line, not on a new line.

Java Tokens

Tokens are the various elements in the java program that are identified by Java compiler. A token is the smallest individual element (unit) in a program that is meaningful to the compiler.

Java language contains different types of tokens that are as follows:

- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators
- Termination

Java Tokens

```
class Test {
    public static void main(String[] args) {
            int x = 10;
    }
}
```

How many - ?

Reserved Keywords – class, public, static, void, String, int

Identifiers – Test, main, String, args, x

Literals - 10

Operators - =

Separators – { }, (), ;

Reserved Keywords

Data Type: 8
byte
short
int
long
float
double
char
boolean

Object and Classes: 7
class
interface
extends
implements
this
super
new

Access Modifiers: 3+1
private
protected
Public
default

Control Statement: 11
else
switch
case
default
for
while
do
continue
break
goto [not used]
if

Reserved Keywords…

Modifiers: 8
static
abstract
synchronized
volatile
native
transient
strictfp
final

Exceptions: 5
try
catch
finally
throw
throws

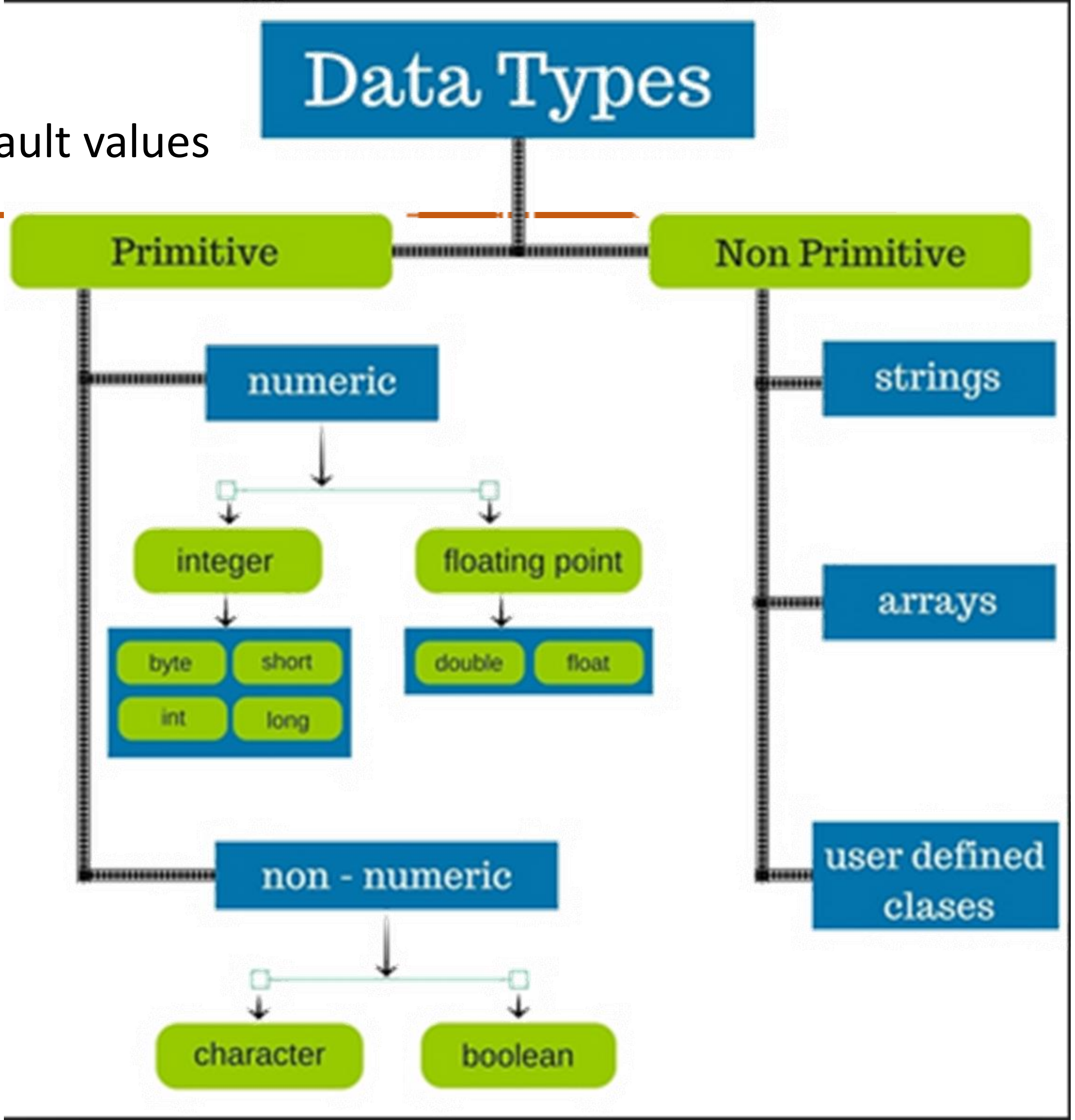
Package: 2
package
import

Miscellaneous : 7
return
const (*)
instanceof
void
assert
default
enum

Data types and default values

- Numbers : **byte, short, int, long (default: 0)**
- Decimal: **float and double** (default: 0.0) – difference??
- Character: **char** (default: ‘ ‘)
- Boolean : **False**
- Non-primitive – **builtin class - String**

JEAD
Data types and default values



Data types and default values

Comments in Java are statements that describe the features of a program. It allows the programmers to compose and express their thoughts related to the code independently.

- Single line comment - //
- Multi line comment - /*   …   */
- Java documentation comment - /**   */
    - https://www.oracle.com/in/technical-resources/articles/java/javadoc-tool.html

JAVA ENTERPRISE APPLICATION DEVELOPMENT

Introduction…

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

JAVA ENTERPRISE APPLICATION DEVELOPMENT

Introduction…

Try These:

1. WAP to add two integers
2. WAP to multiply two floating point numbers
3. Find ASCII Value of a character
4. Swap two numbers

JAVA ENTERPRISE APPLICATION DEVELOPMENT

Course Description

How Java "Hello, World!" Program Works?

1. // Your First Program

In Java, any line starting with // is a comment. Comments are intended for users reading the code to understand the intent and functionality of the program. It is completely ignored by the Java compiler (an application that translates Java program to Java bytecode that computer can execute). To learn more, visit Java comments.

Java program

2. class HelloWorld { ... }

In Java, every application begins with a class definition. In the program, HelloWorld is the name of the class, and the class definition is:

Java User input

- The **Scanner class** is used to get user input, and it is found in the **java.util package.**

- To use the Scanner class, **create an object of the class** and use any of the **available methods** found in the **Scanner class documentation**.

- In our example, we will use the **nextLine()** method, which is used to read Strings:

JAVA ENTERPRISE APPLICATION DEVELOPMENT

Scanner class example-1

```
import java.util.Scanner;  // Import the Scanner class

class Simpleio1 {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

JAVA ENTERPRISE APPLICATION DEVELOPMENT

User input

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Scanner class example2

```
import java.util.Scanner;
public class SumOfTwoNumbers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the first number: ");
        int num1 = scanner.nextInt();
        System.out.print("Enter the second number: ");
        int num2 = scanner.nextInt();

        int sum = num1 + num2;
        System.out.println("Sum: " + sum);
        scanner.close();
    }
}
```

JAVA ENTERPRISE APPLICATION DEVELOPMENT

Scanner class example1

```
import java.util.Scanner;  // Import the Scanner class
public class Simpleio2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();
        System.out.println("Hello, " + name + "! You are " + age + " years old.");
        scanner.close(); // Don't forget to close the scanner when you're done!
    }
}
```

User input

Try these:

1. Find the average of the numbers
2. Check whether entered number is prime or not
3. Check whether palindrome or not
4. Find odd or even number

Java Variables

- A variable is a **location in memory** (storage area) **to hold data**.

- To indicate the storage area, each variable should be given a unique name (identifier).

- int a, b, c; // Declares three ints, a, b, and c.
- int a = 10, b = 10; // Example of initialization
- byte B = 22; // initializes a byte type variable B.
- double pi = 3.14159; // declares and assigns a value of PI.
- char a = 'a'; // the char variable a is initialized with value 'a'

Java Variables

There are 3 types of variables in java:

1. **Local variable**
2. **Instance variable**
3. **Class / static variable**

- Local variables are declared in methods, constructors, or blocks.
- Local variables are visible only within the declared method, constructor, or block.

- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Java Variables

Example: Local variable

```
public class variableType {
    public void localVariable() {
        String name = “Ajay";
        int marks = 95;
        System.out.println(name + " Scored " + marks + "%.");
    }

    public static void main(String[] args) {
        variableType vt = new variableType();
        vt.localVariable();
    }
}
```

Java Variables

- **Instance variables** are declared in a class, but outside a method, constructor or any block.

- When a space is allocated for an object in the heap, a slot for each instance variable value is created.

- Instance variables are created when an object is created with the use of the **keyword 'new'** and destroyed when the object is destroyed.

Example: Instance variables

```
public class variableType {

    public String name = "Ben";
    public int marks = 95;

    public void instanceVariable() {
        System.out.println(name + " Scored " + marks + "%.");
    }

    public static void main(String[] args) {
        variableType vt = new variableType();
        vt.instanceVariable();
    }
}
```

Java Variables

- **Class variables** also known as **static variables** are declared with the static keyword in a class, but outside a method, constructor or a block.

- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. **Constant variables never change from their initial value.**

- Static variables are stored in the static memory. It is rare to use static variables other than declared **final** and used as either public or private constants.

- Static variables can be accessed by calling with the class name **ClassName.VariableName**.

Java Variables

Example: Class Variable

```
public class variableType {

    public static String name;
    public static int marks;

    public static void main(String[] args) {
        name = “Ajay";
        marks = 95;
        System.out.println(name + " Scored " + marks + "%.");
    }
}
```

Java Variables

```
import java.io.*;
public class Employee {
   // salary  variable is a private static variable
   private static double salary;
   // DEPARTMENT is a constant
   public static final String DEPARTMENT = "Development ";
   public static void main(String args[]) {
      salary = 1000;
      System.out.println(DEPARTMENT + "average salary:" + salary);
   }
}
```

Java - Operators

1.Arithmetic Operators	:	+, -, *, /, %
2.Assignment Operators	:	=, +=, -=, *=, /=, %=
3.Relational Operators	:	==, !=, <, <=, >, >=
4.Logical Operators	:	&&, ||, !
5.Unary Operators	:	+, -,++, --, !
6.Bitwise Operators	:	~, <<, >>, >>>, &, ^

Java supports Operator overloading?

Demonstration of ‘+’ operator implicitly overloaded

Sop(“Sum”+ “ is :”);

Sop(5 + 10);

Sop(5 + 10 + “is the sum”);

Sop(“Sum is: “ + 5 + 10); ??

Java Operators

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

    // declare variables
    int a = 12, b = 5;

    // addition operator
    System.out.println("a + b = " + (a + b));

    // subtraction operator
    System.out.println("a - b = " + (a - b));

    // multiplication operator
    System.out.println("a * b = " + (a * b));

    // division operator
    System.out.println("a / b = " + (a / b));

    // modulo operator
    System.out.println("a % b = " + (a % b));
  }
}
```

Java Assignment Operator

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Java Variables

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

    // create variables
    int a = 4;
    int var;

    // assign value using =
    var = a;
    System.out.println("var using =: " + var);

    // assign value using =+
    var += a;
    System.out.println("var using +=: " + var);

    // assign value using =*
    var *= a;
    System.out.println("var using *=: " + var);
  }
}
```

Java Operators…

Java instanceof Operator

- The instanceof operator checks whether an object is an instanceof a particular class. For example,

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

    String str = “Java Programming";
    boolean result;

    // checks if str is an instance of
    // the String class
    result = str instanceof String;
    System.out.println("Is str an object of String? " + result);
  }
}
```

Java Operators…

Java Ternary Operator

- The ternary operator (conditional operator) is shorthand for the if-then-else statement. For example,

```
variable = Expression ? expression1 : expression2
```

Working of above statement is:

If the Expression is true, expression1 is assigned to the variable.
If the Expression is false, expression2 is assigned to the variable.

Java Operators…

Ternary operator Ex:

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

    int februaryDays = 29;
    String result;

    // ternary operator
    result = (februaryDays == 28) ? "Not a leap year" : "Leap
year";
    System.out.println(result);
  }
```

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Java Input/output

Ms. Archana A

Department of Computer Applications

Taking Input

- **Java provides different ways to get input from the user. However, get input from user using the object of Scanner class.**
- In order to use the object of Scanner, we need to import **java.util.Scanner** package.

- Then, we need to create an object of the **Scanner class**. We can use the object to take input from the user.

```
Scanner sc = new Scanner(System.in);

// take input from the user
int number = sc.nextInt();
//ScannerExample.java
```

Java input statement…

- There are four ways to create an object of the scanner class. Let us see how we can create Scanner objects:

1. Reading keyboard input:

```
Scanner sc = new Scanner(System.in)
```

2. Reading String input:

```
Scanner sc = new Scanner(String str)
```

3. Reading input stream:

```
Scanner sc = new Scanner(InputStream input)
```

4. Reading File input:

```
Scanner sc = new Scanner(File file)
```

Java Input statements

- Scanner class
- BufferedReader
- BufferedReader + InputStreamReader

Assignment:

Find the difference between the above classes.

Java input statement…

```
import java.util.Scanner;

class Input {
  public static void main(String[] args) {
      Scanner input = new Scanner(System.in);

    // Getting float input
    System.out.print("Enter float: ");
    float myFloat = input.nextFloat();
    System.out.println("Float entered = " + myFloat);

    // Getting double input
    System.out.print("Enter double: ");
    double myDouble = input.nextDouble();
    System.out.println("Double entered = " + myDouble);

    // Getting String input
    System.out.print("Enter text: ");
    String myString = input.next();
    System.out.println("Text entered = " + myString);
  }
}
```

JEAD

Java output statements

Java Output

In Java, you can simply use

- **System.out.println(); or**
- **System.out.print(); or**
- **System.out.printf();**

to send output to standard output (screen).

Java output statements

Difference between **println(), print()** and **printf()**

- **print()** - It prints string inside the quotes.
- **println()** - It prints string inside the quotes similar like print() method. Then the cursor moves to the beginning of the **next line.**
- **printf()** - It provides string formatting (similar to printf in C/C++ programming).

Java output statements

Example”

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

        System.out.println("1. println ");
        System.out.println("2. println ");

        System.out.print("1. print ");
        System.out.print("2. print");
    }
}
```

JEAD
Java output statements

Example”

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

        System.out.println("1. println ");
        System.out.println("2. println ");

        System.out.print("1. print ");
        System.out.print("2. print");
    }
}
```

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Java Constructs

Ms. Archana A
Department of Computer Applications

JEAD

Basic constructs

- **Conditional statements**: if, if-else, switch
- **Iterative statements** : for, while, do-while
- **Transfer statements** : break, continue, go to, return, try

Basic constructs

```
if ( condition ) {

    statements…

}
```

Conditional statement…

- Example for **If condition**

```
public class JavaIf {
    public static void main(String[] args) {
        String word = "Mobile";
        int wordsize = 6;
        if (word == "Mobile" && wordsize == 6) {
            System.out.println("Word size matches to word");
        }
    }
}
```

Basic constructs

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

Conditional statement…

- Example for **If…else** condition

```
public class JavaIf {
    public static void main(String[] args) {
 String word = "Mobile";
        int wordsize = 6;
        if (word == "Mobile" && wordsize == 6)
        } else {
            System.out.println("Invalid size");
        }
    }
}
```

Conditional statement…

Nested if Statements

if statements inside other if statements are called nested if statements.

Example:

```
public class Nested {
    public static void main(String[] args) {
        String name = "Mohan";
        int Roll = 25;
        int marks = 85;
        if (name == "Mohan" && Roll == 25) {
            if (marks > 35) {
                System.out.println("Mohan has been promoted to next class.");
            } else {
                System.out.println("Mohan needs to give re-exam.");
            }
        }
    }
```

Basic constructs

```
switch (constant) {

    case const_1 : {   statements; break; }

     case const_2 : {   statements; break; }

     case const_n : {   statements; break; }

     default:   {   statements; break; }

}
```

Conditional statement…

Example for Switch **case statements**

- In a switch statement, a block of code is executed from many based on switch case.

```
public class JavaSwitch {
   public static void main(String[] args) {
      String day = “Thursday";
      switch (day) {
        case "Sunday":
          System.out.println("Today is Sunday");
          break;
        case "Monday":
          System.out.println("Today is Monday");
          break;
        case "Tuesday":
          System.out.println("Today is Tuesday");
          break;
        case "Wednesday":
          System.out.println("Today is Wednesday");
          break;
        case "Thursday":
          System.out.println("Today is Thursday");
          break;
        case "Friday":
          System.out.println("Today is Friday");
          break;
        case "Saturday":
          System.out.println("Today is Saturday");
          break;
      }
   }
}
```

Loop statement…

There are three types of loops in java:

- for loop
- while loop
- do……while loop

1. For loop:

Whenever a loop needs to be run a specific number of times, we use a for loop.

Syntax:

```
for (initializeVariable, testCondition, increment/decrement){
//block of code
}
```

JEAD
Basic constructs

```
for ( initialisation; condition; incr / decr) {

    statements;

}
```

```
for ( i=1; i<=10; i++) {

    System.out.println(“Welcome to Java World”);

}
```

Loop statement…

For-each loop:

We also have a special syntax for **iterating through arrays** and other collection objects, also called as a **for-each loop**.

Syntax:

```
for (type variable : collectionObject){
   //block of code
}
```

Loop statement…

Example : for-each loop:

```
public class ForEach1 {
    public static void main(String[] args) {
        int[] prime = {1,3,5,7,11,13,17,19};
        System.out.println("Prime numbers are:");
        for(int i : prime) {
            System.out.println(i);
        }
    }
}
```

Loop statement…

While loop:

Whenever **we are not sure** about the **number of times the loop needs to be run**, we use a while loop. A while loop keeps on running till the condition is true, as soon as the condition is false, control is returned to the main body of the program.

Syntax:

```
while (baseBooleanCondition) {
   //block of code
}
```

baseBooleanCondition: it checks if the condition is true or false after each iteration. If true, run block of code. If false, terminate the loop.

Basic constructs

```
while (condition) {

	statements;

}
```

```
i=1

while (i<=10) {

	Sop(“Welcome to Java World”);

	i++;

}
```

Loop statement…

do-while Loop

A do…..while loop is a special kind of loop that **runs the loop at least once** even if the **base condition is false**. This is because the base condition in this loop is checked after executing the block of code. As such, even if the condition is false, the loop is bound to run atleast once. Hence, do…..while loop is also called as an **Exit Control Loop**.

Syntax:

```
do {
   //block of code
} while (baseBooleanCondition);
```

.

Loop construct…

Nested Loops

Loops inside other loops are called nested loops.

Example:

```
public class Nested {
    public static void main(String[] args) {
        for (int i=1; i<5; i++) {
            for (int j=1; j<5; j++) {
                System.out.println(i+" * "+j+" = "+i*j);
            }
            System.out.println();
        }
    }
}
```

Break/continue statement…

break/continue Statement

break statement

- In java, break statement is used when working with any kind of a loop or a switch statement.
- It breaks out of the loop or a switch statement and returns the control to the main body of the program.
- In the case of nested loops, it breaks the inner loop and control is returned to the outer loop.

Break/continue statement…

break/continue Statement

continue statement

- The continue statement breaks the **current iteration in the loop**, if a **specified condition occurs**, moves to the end of the loop, and continues with the next iteration in the loop..

Try These:

1. WAP to display largest among three integers and also display second largest.

2. WAP to accept a student marks in five subjects and display the grade he/she achieved

Exercise 2: Ticket Pricing

Scenario1:
You are developing a ticket booking system for a cinema. The pricing depends on the age of the customer and the type of movie:

Regular movies:
Adults (age >= 18): Rs.10
Children (age < 18): Rs.7
3D movies:
Adults: Rs. 15
Children: Rs.10
Write a Java program that takes the age and movie type as input and calculates the ticket price.

Exercise 3: Password Strength Checker

Scenario2:

You are building a user registration system. You need to ensure that the passwords meet certain criteria for security. The criteria are:

At least 8 characters long
Contains at least one uppercase letter, one lowercase letter, and one digit
Write a Java program that checks if a given password meets these criteria.

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Java Strings

Ms. Archana A
Department of Computer Applications

String Basics

Strings in java is a sequence of characters that Is **enclosed in double quotes.**
Whenever java comes across a String literal in the code, it creates a string literal with the value of string.

```
public class string {
    public static void main(String[] args) {
        String name;
        name = "Deepa";
        System.out.println("My name is " + name);
    }
}
```

Java String Basics…

Concatenate Strings:

Concatenation between two strings in java is done using the + operator.

Example:

```
public class string {
    public static void main(String[] args) {
        String fname, lname;
        fname = “Hard";
        lname = “Ware";
        System.out.println(fname + " " + lname);
    }
}
```

Java String Basics…

Concatenate Strings…

Alternatively, we can use the concat() method to concatenate two strings.

Example:

```
public class string {
   public static void main(String[] args) {
      String fname, lname;
      fname = “Hard";
      lname = " Ware";
      System.out.println(fname.concat(lname));
   }
}
```

Java String Basics…

What if we concatenate string with an integer? Try this

- **Well concatenating a string and an integer will give us a string.**

```
public class string {
    public static void main(String[] args) {
        String name;
        int quantity;
        quantity = 12;
        name = " Apples";
        System.out.println(quantity + name);
    }
}
```

Java Escape character

Escape Characters

- Try running the code given below in your java compiler.

```
public class string {
    public static void main(String[] args) {
        System.out.println("He said, "I believe that the Earth is Flat".");
    }
}
```

- As we can see that the **code gives an error**. This is because the compiler assumes that the string ends after the 2nd quotation mark.
- This can be solved by using **'\' (backslash)**.
- Backslash acts as an escape character allowing us to use quotation marks in strings.

Java Escape character

Escape Characters example:

```
public class string {
    public static void main(String[] args) {
        System.out.println("He said, \"I believe that the Earth is Flat\".");
        System.out.println("she said, \'But the Earth is spherical\'.");
    }
}
```

Java Escape character

Similarly to use a backslash in the string we must escape it with another backslash.

Example:

```
public class string {
    public static void main(String[] args) {
        System.out.println("The path is D:\\Docs\\Java\\Strings");
    }
}
```

Java Escape character

We also have an escape character for printing on a new line(\n), inserting a tab(\t), backspacing(\b), etc.

Example:

```
public class string {
    public static void main(String[] args) {
        System.out.println("My name is Arjun. \nI'm an Artist.");
        System.out.println();
        System.out.println("Age:\t38");
        System.out.println("Addresss\b: Washington DC");
    }
}
```

String Methods

- **Here we will see some of the popular methods we can use with strings.**

length(): This method is used to find the length of a string.

Example:

```
public class string {
    public static void main(String[] args) {
        String quote = "To be or not to be";
        System.out.println(quote.length());
    }
}
```

String Methods

indexOf(): This method returns the first occurrence of a specified character or text in a string.

Example:

```
public class string {
    public static void main(String[] args) {
        String quote = "To be or not to be";
        System.out.println(quote.indexOf("be"));    //index of text
        System.out.println(quote.indexOf("r"));     //index of character
    }
}
```

String Methods

toLowerCase(): Converts string to lower case characters.
toUpperCase(): **Converts string to upper case characters.**

Example:

```
public class string {
   public static void main(String[] args) {
      String quote = “PES UNIVERSITY";
      System.out.println(quote.toLowerCase());
          System.out.println(quote.toUpperCase());
   }
}
```

String Methods

toLowerCase(): Converts string to lower case characters.

Example:

```
public class string {
    public static void main(String[] args) {
        String quote = "THOR: Love and Thunder";
        System.out.println(quote.toLowerCase());
    }
}
```

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Java Arrays

Ms. Archana A
Department of Computer Applications

Arrays

Array Basics

- Indexed collection of fixed number of **homogeneous elements**.
- An array is a **container object that holds a fixed number of values of a single type**. We do not need to create different variables to store many values, instead we store them in different indices of the same objects and refer them by these indices whenever we need to call them.

Syntax:

Datatype[] arrayName = {val1, val2, val3,……. valN}

Note: array indexing in java starts from [0].

Array Basics…

There are two types of array in java:

- **Single Dimensional Array**
- **Multi-Dimensionalal Array**

We will see how to perform different operations on both the type of arrays,

1) Length of an array:

- Finding out the length of an array is very crucial when performing major operations. This is done using the **.length** property:

Arrays…

2. Accessing array elements:

Array elements can be accessed using indexing. In java, **indexing starts from 0 rather than 1.**

Example:

```
public class ArrayExample {
    public static void main(String[] args) {
        //creating array objects
        String[] cities = {"Delhi", "Mumbai", "Lucknow", "Pune", "Chennai"};
        int[] numbers = {25,93,48,95,74,63,87,11,36};

        //accessing array elements using indexing
        System.out.println(cities[3]);
        System.out.println(numbers[2]);
    }
}
```

JAVA ENTERPRISE APPLICATION DEVELOPMENT

Arrays…

3. Change array elements:

The value of any element within the array can be changed by referring to its index number.

Example:

```
public class ArrayExample {
    public static void main(String[] args) {
        //creating array objects
        String[] cities = {"Delhi", "Mumbai", "Lucknow", "Pune", "Chennai"};

        cities[2] = "Bangalore";
        System.out.println(cities[2]);
    }
}
```

Multidimensional Arrays

- We can even create multidimensional arrays i.e. arrays within arrays. We access values by providing an index for the array and another one for the value inside it.

```
public class ArrayExample {
   public static void main(String[] args) {
      //creating array objects
      String[][] objects = {{"Spoon", "Fork", "Bowl"}, {"Salt", "Pepper"}};

      //accessing array elements using indexing
      System.out.println(objects[0][2]);
      System.out.println(objects[1][1]);
   }
}
```

Arrays…

We can also print the multi-dimensional array.

Example:

```
public class ArrayExample {
    public static void main(String[] args) {
        //creating array objects
        int[][] objects = {{1,2,3}, {4,5,6}};

        for (int i = 0; i < objects.length; i++) {
            for (int j = 0; j < objects[i].length; j++) {
                System.out.print(objects[i][j] + "\t");
            }
            System.out.println();
        }
    }
```

Arrays…

Use loops with arrays:

- The simplest way of looping through an array is to use a simple **for-each** loop.

Example:

```
public class ArrayExample {
    public static void main(String[] args) {
        //creating array objects
        String[] objects = {"Spoon", "Fork", "Bowl", "Salt", "Pepper"};

        for (String i : objects) {
            System.out.println(i);
        }
    }
}
```

Arrays…

Alternatively, you can also use a simple for loop to do the same.

Example:

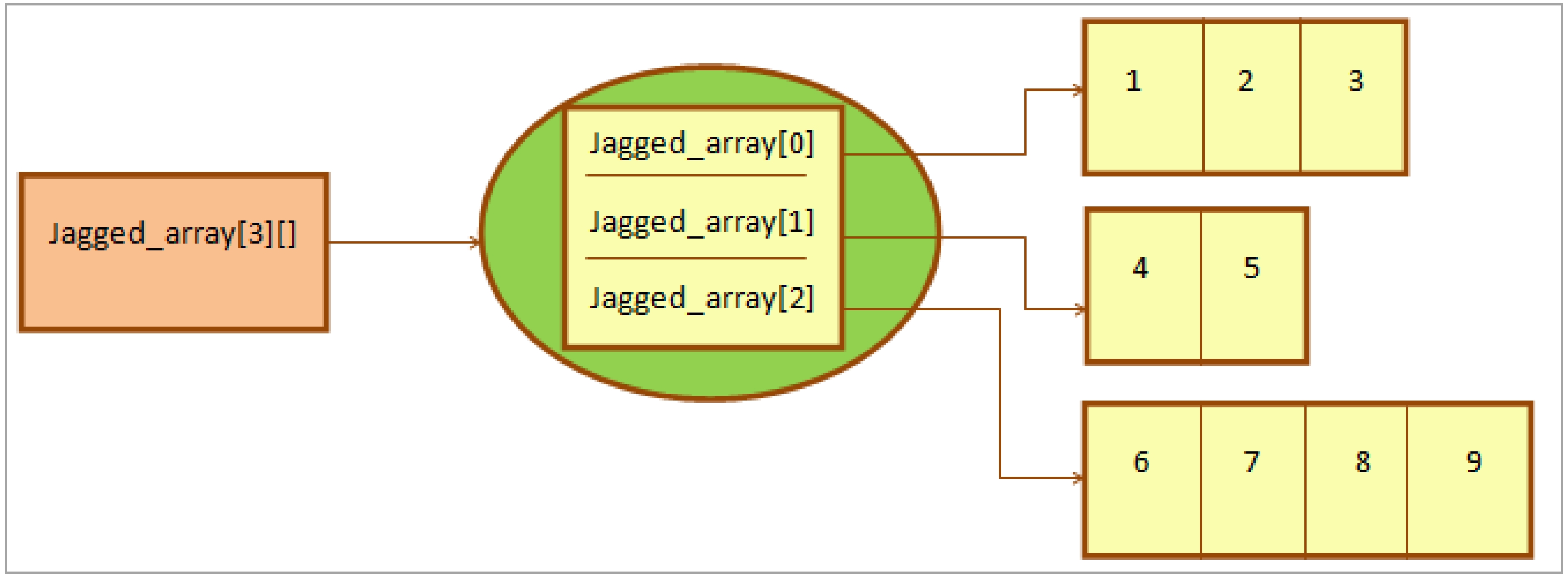
```
public class ArrayExample {
    public static void main(String[] args) {
        //creating array objects
        String[] objects = {"Spoon", "Fork", "Bowl", "Salt", "Pepper"};

        for (int i = 0; i < objects.length; i++) {
            System.out.println(objects[i]);
        }
    }
}
```

Arrays…

Java Jagged Arrays

- **It is an array of arrays where each element is, in turn, an array.**
- A special feature of this type of array is that it is a **Multidimensional array whose each element can have different sizes**.
- **For Example, a two-dimensional array in Java is an array of single dimension array. In the case of a two-dimensional array, each one-dimensional array will have different columns.**

Arrays...



The above pictorial representation, gives an idea of how does it look. **Above shown is a two-dimensional Jagged array.** Each individual element of this array is a one-dimensional array that has **varied sizes** as shown above.

The first 1D array has 3 columns; the second row has 2 columns while the third has 4 columns.

Jagged Arrays…

Create & Initialize Jagged Array

- While creating an array of arrays **specify only the first dimension that represents a number of rows in the array.**

- To create a two-dimensional jagged array as shown:

```
int myarray[][] = new int[3][];
```

- In the above declaration, a two-dimensional array is declared with three rows.

Jagged Arrays…

Once the array is declared, you can define it as a Jagged array as shown below:

```
myarray[1] = new int[2];
myarray[2] = new int[3];
myarray[3] = new int[4];
```

- The first statement above indicates that the first row in the 2D array will have 2 columns. The second row will have 3 columns while the third row will have 4 columns thereby making it a Jagged array.

- Once the array is created, you can initialize it with values. Note that if you don’t explicitly initialize this array (as in the above case), then it will take the default values as initial values depending on the data type of the array.

Jagged Arrays…

Alternatively, you can also initialize an array as follows:

```
int myarray[][] = new int[][]{
    new int[] { 1, 2, 3 };
    new int[] { 4, 5, 6, 7 };
    new int[] { 8, 9 };
};
```

Jagged Arrays…

Yet another way of initializing a Jagged array is by **omitting the first new operator** as shown below:

```
int[][]myarray ={
   new int[] { 1, 2, 3 };
   new int[] { 4, 5, 6, 7 };
   new int[] { 8, 9 };
};
```

As you can see above, the new operator is omitted and the array is initialized as well as declared in the same statement.

Jagged Arrays…

```
public class JagArr{
    public static void main(String[] args) {
        // Create a jagged array
        int[][] jaggedArray = {
            {1, 2, 3},
            {4, 5},
            {6, 7, 8, 9}
        };

        // Print the jagged array
        for (int i = 0; i < jaggedArray.length; i++) {
            for (int j = 0; j < jaggedArray[i].length; j++) {
                System.out.print(jaggedArray[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Jagged Arrays – practice exercise

1. Write a program to find the sum of elements in a jagged array
2. Write a Java program that finds and prints the maximum element in a jagged array.
3. Write a Java program that calculates and prints the sum of elements in each row of a jagged array.

2D Array Exercise:

1. Write a Java program that finds and prints the diagonal elements of a square matrix.
2. Write a Java program that transposes a square matrix (i.e., swaps rows with columns).
3. Write a Java program to find and print the frequency of a given element in a 2D array.
4. Write a Java program to find and print the sum of elements in each row of a 2D array.
5. Write a Java program to find and print the sum of elements in each column of a 2D array.

1D Array Exercise:

WAP

1. Finding the Sum of Array Elements
2. Finding the Largest Element in an Array
3. Finding the Smallest Element in an Array
4. Reversing an Array
5. Finding the Average of Array Elements

Built-in methods in Arrays class :

- binarySearch
- compare
- copyOf
- copyOfRange
- deepEquals
- equals
- fill
- sort
- toString
- parallelSort
- mismatch
- hashCode
- arraycopy
- clone

Arrays is part of Collections framework with static methods

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Classes, Objects and Methods

Ms. Archana A
Department of Computer Applications

Classes, Objects and Methods

What is a Class?

- A class is a model to create objects. It means that we write properties and actions of objects in the class.

- **Properties** are represented by **variables**, and **actions** are represented by **methods**. So, a class consists of variables and methods.

- A class is basically user-defined data types that act as a **template for creating objects** of the identical type. Every Java class contains attributes and methods

Classes, Objects and Methods

- A class in Java is a fundamental building block of object-oriented programming (OOP) language. In other words, a class is the basic unit of OOP.

- According to OOPs concept in Java, a class is the **blueprint/template** of an object. It contains the **similar types of objects** having the same states (properties) and behaviour

Classes, Objects and Methods

What is an object?

- **An object is an entity that has state and behavior. Here, the state represents properties and behavior represents actions or functionality.**

Classes, Objects and Methods

An object in Java has three characteristics:

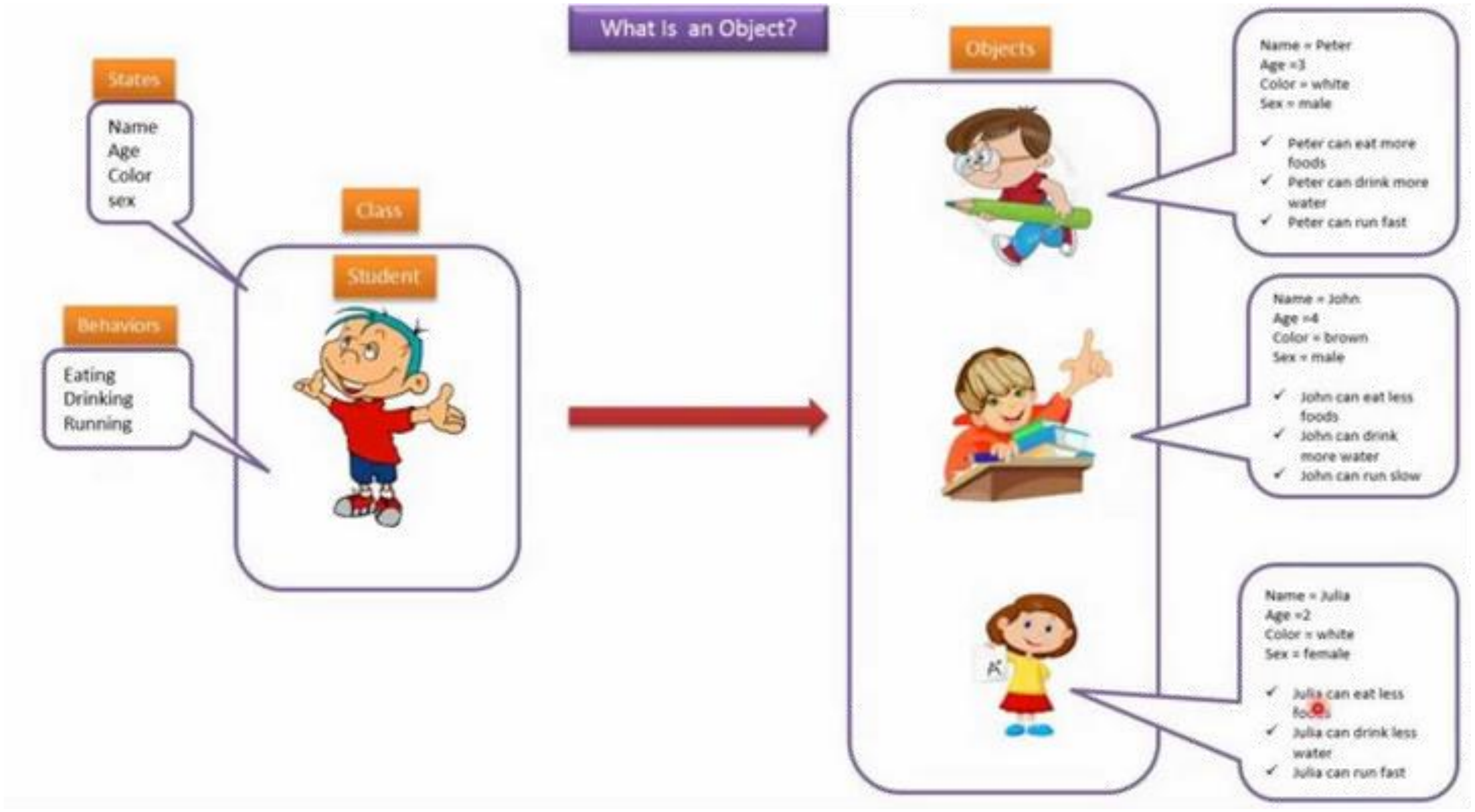
1. State: State **represents** **properties or attributes** of an object. It is represented by instance variable. The properties of an object are important because the outcome of functions depends on the properties.

2. Behavior: Behavior represents **functionality or actions**. It is represented by methods in Java.

3. Identity: Identity represents the **unique name** of an object. It differentiates one object from the other. The unique name of an object is used to identify the object.
Eg: Apple

JAVA ENTERPRISE APPLICATION DEVELOPMENT

Classes, Objects and Methods



Object & Class

- An **object** can be any real-world entity such as a book, cupboard, pen, paper, fan, etc. While a **class** is a group of objects with similar properties.

- An object is an instance of a class while a class is a blueprint from which we create objects.

i.

create objects:

we can create a class and from it, create an object.

Syntax to create object from class:

className objectName = new className();

Classes, Objects and Methods

To create an object for Example1 class we will use **the new keyword**. new keyword **allocates heap memory** to object at run time.

```
public class Example1 {

    String name = "Ramesh";
    int age = 20;

    public static void main(String[] args) {
        Example1 ex = new Example1();
        System.out.println("Name: " + ex.name);
        System.out.println("Age: " + ex.age);
    }
}
```

Classes, Objects and Methods

We can even create **multiple instances of the same class**.

Example:

```
public class Example2 {

    String name = "Raksha";
    int age = 25;

    public static void main(String[] args) {
        Example2 ex1 = new Example2();
        Example2 ex2 = new Example2();
        System.out.println("Name: " + ex1.name);
        System.out.println("Age: " + ex2.age);
    }
}
```

ii. Class attributes/fields:

Class attributes are the variables created inside a class.

Example:

```
public class Example3 {
   String name = "Ramesh";
   int age = 20;
}
```

Here **name and age** are attributes or fields of class Example3**.**

But the attributes declared in the class can be overridden in the main method().

Example:

```
public class Example1 {

    String name = "Ramesh";
    int age = 20;

    public static void main(String[] args) {
        Example1 ex = new Example1();
        ex.age = 25;
        System.out.println("Name: " + ex.name);
        System.out.println("Age: " + ex.age);
    }
}
```

As we can see, we changed the value of **age attribute in the main method and** the output was affected by it**. But what if we don't want the main method to override any declared value?**

This can be done using the final keyword. Essentially what final keyword does is that, once the attribute holds certain value, then it cannot be overridden.

Example:

Classes, Objects and Methods

```java
public class Example1 {

    final String name = "Ramesh";
    final int age = 20;

    public static void main(String[] args) {
        Example1 ex = new Example1();
        ex.name = "Sanjay";
        ex.age = 25;
        System.out.println("Name: " + ex.name);
        System.out.println("Age: " + ex.age);
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved
compilation problems:
   The final field Example1.name cannot be assigned
   The final field Example1.age cannot be assigned
   at javaFiles/ObjectNClass.Example1.main(Example1.java:10)
```

iii. Class Methods:

- Methods are a block of code that accept certain parameters and perform actions whenever they are called. Methods are always written inside a java class and can be called by simply referring to the method name.

In java we have public and static methods. So what is the difference between?

public methods are accessed by making objects of the class whereas static methods do not need objects to be accessed, we can directly access static methods.

Example:

Classes, Objects and Methods

```
public class Example5 {
    //public method
    public void fruits() {
        String fruits[] = {"Apple", "Banana", "Mango", "Strawberry"};
        System.out.println("Fruits:");
        for (int i=0; i<fruits.length; i++) {
            System.out.println(fruits[i]);
        }
    }

    //static method
    static void vegetables() {
        String vegies[] = {"Onion", "Potato", "Carrot", "Raddish"};
        System.out.println("Vegetables:");
        for (int i=0; i<vegies.length; i++) {
            System.out.println(vegies[i]);
        }
    }
```

Classes, Objects and Methods

```
  public static void main(String[] args) {
      Example5 ex5 = new Example5();        //need to create object
      ex5.fruits();
      System.out.println();

      vegetables();                        // no need to create object
   }
}
```

As we can see, we have created two methods, inside which each of the method has array and for loop to print them. But public method needs an object to be declared in the main() method while we can directly use the static method inside the main() method.

Classes, Objects and Methods

In Java, there are **two types of methods**:

- **User-defined Methods**: We can create our own method based on our requirements.
- **Standard Library Methods**: These are built-in methods in Java that are available to use.

The **syntax** to declare a method is:

```
returnType methodName() {
  // method body
}
```

Classes, Objects and Methods

This is the simple syntax of declaring a method. However, the complete syntax of declaring a method is

```
modifier static returnType nameOfMethod (parameter1, parameter2, ...) {
  // method body
}
```

Let's first learn about user-defined methods.

Java Methods

- Methods or Functions are a block of code that accept certain parameters and perform actions whenever they are called. Methods are always written inside a java class and can be called by simply referring to the method name.

Passing Parameters:

- Parameters are nothing but the variables that are passed inside the parenthesis of a method. We can pass a single or more than one parameter of different datatypes inside our method.

Example 1: Passing single parameter

Java Methods…

```
public class MethodExample {

    static void Details(String name) {
        System.out.println("Welcome " + name);
    }

    public static void main(String[] args) {
        Details("Meena");
    }
}
```

Java Methods...

Example 2: Passing multiple parameters

```
public class MethodExample {

    static void Details(String name, int roll) {
        System.out.println("Welcome " + name);
        System.out.println("Your roll number is "+ roll);
    }

    public static void main(String[] args) {
        Details("Mitali", 37);
    }
}
```

Java Methods…

Example 3: Method with loop

```
public class Methodloop {
    static int Details(int num) {
        int fact = 1;
        for (int i=2; i<=num; i++) {
            fact = fact * i;
        }
        return fact;
    }

    public static void main(String[] args) {
        System.out.println(Details(3));
        System.out.println(Details(4));
        System.out.println(Details(5));
    }
}
```

Java Methods…

Example 4: Method with control statements

```
public class Methodif {
    static void Details(int marks) {
        if (marks < 35) {
            System.out.println("Fail");
        } else if (marks >= 35 && marks < 65) {
            System.out.println("B grade");
        } else if (marks >= 65 && marks < 75) {
            System.out.println("A grade");
        } else if (marks >= 75 && marks < 100) {
            System.out.println("O grade");
        } else {
            System.out.println("Invalid marks entered");
        }
    }
```

continued…

Java Methods…

```
    public static void main(String[] args) {
        Details(25);
        Details(90);
        Details(60);
    }
}
```

Exercise-1:

- Create a class called Person with the following attributes: name (String), age (int), and email (String).
- Create an instance of the Person class and initialize its attributes.
- Print out the details of the person.
- Add a method to the Person class called introduce() that prints a message like: "Hi, I'm [name] and I am [age] years old."
- Call the introduce() method on the person object you created before/

Exercise 2: Bank Account Class

- Create a class called BankAccount with attributes: accountNumber (String), balance (double), and owner (String).
- Add methods deposit(double amount) and withdraw(double amount) to deposit and withdraw money from the account.
- Add a method getBalance() that returns the current balance.
- Create an instance of the BankAccount class, perform some deposits and withdrawals, and print out the final balance.

Exercise 3: Book Class

- **Create a class called Book with attributes: title (String), author (String), and numPages (int).**
- **Add a method displayInfo() that prints out the book's information.**
- **Create multiple instances of the Book class and call the displayInfo() method for each..**

Exercise 4: Calculator Class

- Create a class called Calculator with methods for basic arithmetic operations (addition, subtraction, multiplication, division).
- Each method should take two parameters and return the result.
- Create an instance of the Calculator class and perform some calculations.

Standard Library Methods

Standard Library Methods

- The standard library methods are built-in methods in Java that are readily available for use. These standard libraries come along with the Java Class Library (JCL) in a Java archive (*.jar) file with JVM and JRE.

For example,

print() is a method of java.io.PrintStream. The print("...") method prints the string inside quotation marks.
sqrt() is a method of Math class. It returns the square root of a number.

Example: Standard Library Methods

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

    // using the sqrt() method
    System.out.print("Square root of 4 is: " + Math.sqrt(4));
  }
}
```

Method Overloading

- In Java, **two or more methods may have the same name** if they **differ in parameters** (different number of parameters, different types of parameters, or both). These methods are called **overloaded methods** and this feature is called **method overloading.** For example:

```
void func() { ... }
void func(int a) { ... }
float func(double a) { ... }
float func(int a, float b) { ... }
```

Here, the **func() method is overloaded**. These methods have the same name but accept different arguments.

Method Overloading

Note:

- The return types of the these methods are not the same. It is because **method overloading is not associated with return types**.
- Overloaded methods may have the same or different return types, but they must differ in parameters.

Method Overloading

How to perform method overloading in Java?

- Here are different ways to perform method overloading:

1. Overloading by changing the number of parameters

```
class MethodOverloading {
    private static void display(int a){
        System.out.println("Arguments: " + a);
    }
    private static void display(int a, int b){
        System.out.println("Arguments: " + a + " and " + b);
    }
    public static void main(String[] args) {
        display(1);
        display(1, 4);
    }
}
```

Method Overloading

2. Method Overloading by changing the data type of parameters

```
class MethodOverloading {
    // this method accepts int
    private static void display(int a){
        System.out.println("Got Integer data.");
    }
    // this method  accepts String object
    private static void display(String a){
        System.out.println("Got String object.");
    }
    public static void main(String[] args) {
        display(1);
        display("Hello");
    }
}
```

Method Overloading

Important Points

- Two or more methods can have the same name inside the same class if they accept different arguments. This feature is known as method overloading.
- Method overloading is achieved by either:
  - changing the number of arguments.
  - or changing the data type of arguments.
- It is not method overloading if we only change the return type of methods. There must be differences in the number of parameters.

Recursive Function

Java allows us to **recursively call a method** many times, which helps us solve different type of problems. For ex:

```
public class RecursiveMethod {
      static int fact(int num) {
       if (num != 0)
          return num * fact(num-1);
       else
          return 1;
    }
    public static void main(String[] args) {
       int num1 = 6, res;
       res = fact(num1);
       System.out.println("Factorial of " + num1 + " is " + res);
    }
}
```

Practice Exercise on Method Overloading

Try These:

1. Write a Java class with a method called add that performs addition for different data types (int, double). The method should return the result.

2. Create a class with a method named findMax that finds the maximum of two integers, and another method findMax that finds the maximum of three integers.

3. Write a class with a method named calculateArea that calculates the area of a rectangle, a circle, and a triangle. Overload the method to accept different parameters for each shape.

THANK YOU

Ms. Archana A
Department of Computer Applications

archana@pes.edu

+91 80 6666 3333 Extn 392

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Java Constructors

Ms. Archana A
Department of Computer Applications

Java Constructor

What is a Constructor?

- A constructor in Java is similar to a method that **is invoked when an object of the class is created.**

- Unlike Java methods, a **constructor has the same name as that of the class** and does not have any return type. For example,

```
class Test {
  Test() {
    // constructor body
  }
}
```

Here, Test() is a constructor. It has the same name as that of the class and doesn't have a return type.

Constructors

Points to remember:

- Constructors are similar to methods, but they are **used to initialize an object**.
- Constructors **do not** have any **return type**(**not even void**).
- Every time we create an object by using the new() keyword, a constructor is called.
- If we do not create a constructor by ourself, then the **default constructor** (created by Java compiler) is called**.**

Constructors…

Rules for creating a Constructor :

- **The class name and constructor name should be the same.**
- **It must have no explicit return type.**
- **It can not be abstract, static, final, and synchronized.**

Types of Constructors in Java

- In Java, constructors can be divided into 3 types:

1. **Default Constructor**
2. **No-Arg Constructor**
3. **Parameterized Constructor**

Constructors

1. **Defaut constructor :** A constructor with 0 parameters is known as default constructor.

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

Syntax :

```
<class_name>(){
//code to be executed on the execution of the constructor
}
```

Default Constructors Example

```
class DefaultConst {
  int a;
  boolean b;

  public static void main(String[] args) {

    // A default constructor is called
 DefaultConst obj = new DefaultConst();

    System.out.println("Default Value:");
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
  }
}
```

Constructors

2. No-Arg Constructors

- Similar to methods, a Java constructor may or may not have any parameters (arguments).

- If a constructor does not accept any parameters, it is known as a no-argument constructor.

Example : No-Arg Constructors

```
class Num1 {
  int i;
  // constructor with no parameter
  private Num1() {
    i = 5;
    System.out.println("Constructor is called");
  }
  public static void main(String[] args) {
    // calling the constructor without any parameter
    Num1 obj = new Num1();
    System.out.println("Value of i: " + obj.i);
  }
}
```

Constructors

Another Example : No-Arg Constructors

```
class Company {
  String name;

  // public constructor
  public Company() {
    name = “IBM";
  }
}

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

    // object is created in another class
    Company obj = new Company();
    System.out.println("Company name = " + obj.name);
  }
}
```

Constructors

3. Paramerterized constructor :

- A constructor with some specified number of parameters is known as a parameterized constructor.

Syntax :

```
<class-name>(<data-type> param1, <data-type> param2,......){
//code to be executed on the invocation of the constructor
}
```

Constructors

```
class ParaCon {

  String languages;

  // constructor accepting single value
  ParaCon(String lang) {
    languages = lang;
    System.out.println(languages + " Programming Language");
  }

  public static void main(String[] args) {

    // call constructor by passing a single value
 ParaCon obj1 = new ParaCon("Java");
 ParaCon obj2 = new ParaCon("Python");
 ParaCon obj3 = new ParaCon("C");
  }
}
```

Constructors

Constructor Overloading in Java

- Just like methods, constructors can also be overloaded in Java.
- For example: We can overload the Employe constructor like below:

```
public class Employee{
    public Employee (String n){
        name = n;
      }
```

Constructors

Note:

- Constructors can take parameters without being overloaded
- There can be more than two overloaded constructors
- Let's take an example to understand the concept of constructor overloading.

Example :

In the below example, the class Employee has a constructor named Employee(). It takes two arguments, i.e., string s & int i. The same constructor is overloaded and then it accepts three arguments i.e., string s, int i & int salary.

Constructors

```
class Employee {
// First constructor
    Employee(String s, int i){
        System.out.println("The name of the first employee is : " + s);
        System.out.println("The id of the first employee is : " + i);
    }
//    Constructor overloaded
    Employee(String s, int i, int salary){
        System.out.println("The name of the second employee is : " + s);
        System.out.println("The id of the second employee is : " + i);
        System.out.println("The salary of second employee is : " + salary);
    }
}
public class ConsOverload {
    public static void main(String[] args) {
        Employee girish = new Employee(“Girish",20);
        Employee harish = new Employee("Harish",24,70000);
    }
}
```

Exercise on Constructors

Create a class Game, which allows a user to play "Guess the Number" game once.

Game should have the following methods:

- Constructor to generate the random number
- takeUserInput() to take a user input of number
- isCorrectNumber() to detect whether the number entered by the user is true
- getter and setter for noOfGuesses
- Use properties such as noOfGuesses(int), etc to get this task done!

Java this Keyword

this Keyword

- In Java, this keyword is used to refer to the current object inside a method or a constructor. For example,

```
class Main {
    int instVar;

    Main(int instVar){
        this.instVar = instVar;
        System.out.println("this reference = " + this);
    }

    public static void main(String[] args) {
        Main obj = new Main(8);
        System.out.println("object reference = " + obj);
    }
}
```

Java this Keyword

Using this for Ambiguity Variable Names

- In Java, it is **not allowed** to declare **two or more variables having the same name** inside a scope (class scope or method scope). However, instance variables and parameters may have the same name. For example,

```
class MyClass {
    // instance variable
    int age;

    // parameter
    MyClass(int age){
        age = age;
    }
}
```

Here, the instance variable and the parameter have the same name: age. Here, the **Java compiler is confused due to name ambiguity.**

In such a situation, we **use this keyword**. For example

Java this Keyword

First, let's see an example without using this keyword:

```
class Main {

    int age;
    Main(int age){
        age = age;
    }

    public static void main(String[] args) {
        Main obj = new Main(8);
        System.out.println("obj.age = " + obj.age);
    }
}
```

Output: 0

we have passed 8 as a value to the constructor. However, we are getting 0 as an output. This is because the Java compiler gets confused because of the ambiguity in names between instance the variable and the parameter.

Java this Keyword

Now, let's rewrite the above code using this keyword.

```
class Main {

    int age;
    Main(int age){
        this.age = age;
    }

    public static void main(String[] args) {
        Main obj = new Main(8);
        System.out.println("obj.age = " + obj.age);
    }
}
```

Output:

```
obj.age = 8
```

THANK YOU

Ms. Archana A
Department of Computer Applications

archana@pes.edu

+91 80 6666 3333 Extn 392

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Garbage Collection and finalizer

Ms. Archana A
Department of Computer Applications

Garbage collection

What is Garbage?

What is Garbage collection?

When do objects become Garbage?

What is Garbage?

In Java, garbage means unreferenced objects.

What is Garbage Collection?

Java garbage collection (GC) is the process by which the Java Virtual Machine (JVM) **automatically reclaims memory that is no longer being used by objects**.

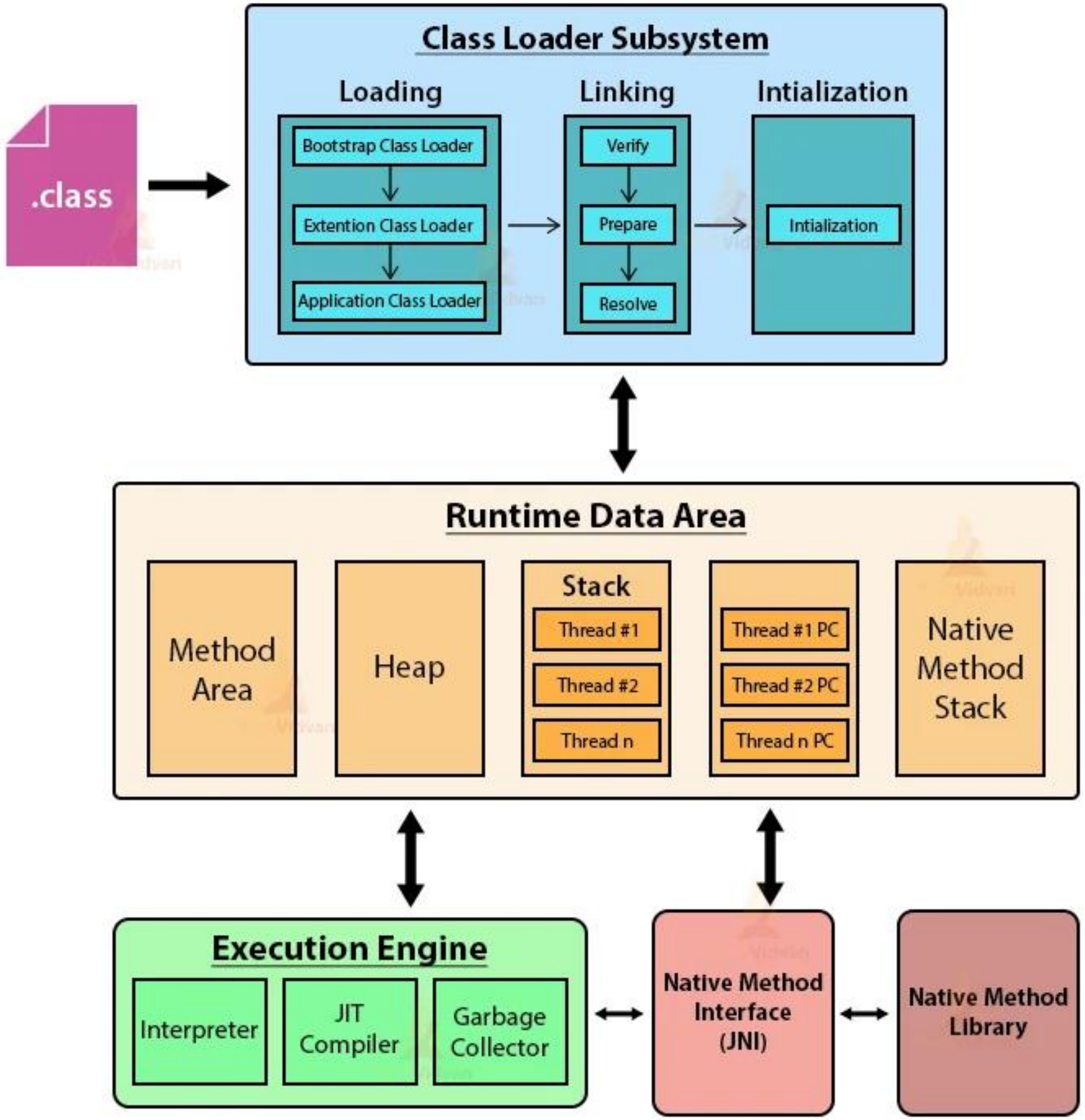
- This is done by identifying and deleting objects that are no longer referenced by any other objects in the program

Garbage collection

- Garbage collection in Java is the process by which Java programs perform **automatic memory management**..
- When Java programs run on the JVM, **objects are created** on the heap, which is a portion of memory dedicated to the program.
- Eventually, some **objects will no longer be needed**. The garbage collector finds these unused objects and deletes them to free up memory.

JEAD

JVM - Architecture



Garbage collection…

How to unreferenced an object?

- There are **three ways** to unreferenced an object. They are

1. By nulling the reference

```
for eg:  Fruits f = new Fruits();
         f=null;
```

2.By assigning a reference to another

```
for eg: Fruits f1 = new Fruits();
        Fruits f2 = new Fruits();
        f1=f2;  //now f1 can be destroyed
```

3. By anonymous object

```
for eg:  new Fruits();
```

Methods used to request JVM to run GC:

- **System.gc() – static method**
- **Runtime class gc() - instance method**
- **finalize() - Finalization**

Garbage collection…

The **System.gc()** method in Java is used to request that the Java Virtual Machine (JVM) run the garbage collector. The garbage collector is a process that automatically reclaims memory that is no longer being used by objects.

The **Runtime.gc()** method in Java is the same as the System.gc() method. It sends a hint to the Java Virtual Machine (JVM) that it should run the garbage collector. The garbage collector is a process that automatically reclaims memory that is no longer being used by objects.

The **finalize()** method is a special method that can be used to clean up resources before an object is garbage collected.

Garbage collection…

```
public class GarbageCollectionExample {
    public static void main(String[] args) {
        // Create a new object.
        Object obj = new Object();

        // Set the reference to the object to null.
        obj = null;

        // Request garbage collection.
        System.gc();
    }
}
```

JAVA ENTERPRISE APPLICATION DEVELOPMENT (JEAD)

Nested Classes

Ms. Archana A

Department of Computer Applications

Java Nested and Inner class

A class within another class is known as nested class.

For example,

```
class OuterClass {
   // ...
   class NestedClass {
      // ...
   }
}
```

There are two types of nested classes you can create in Java.

- **Non-static nested class (inner class)**
- **Static nested class**

Non-Static Nested Class (Inner Class)

- A non-static nested class is a class within another class. It has access to members of the enclosing class (outer class). It is commonly known as **inner class.**

- Since the inner class exists within the outer class, you must **instantiate the outer class first, in order to instantiate the inner class.**

Java Nested and Inner class

```
class OuterClass {
    private int data;

    class InnerClass {
        public void display() {
            System.out.println("Inner class method invoked");
        }
    }

    public void setData(int value) {
        this.data = value;
    }

    public void invokeInnerClass() {
        InnerClass inner = new InnerClass();
        inner.display();
    }
}
```

Java Nested and Inner class

```
public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.setData(10);
        outer.invokeInnerClass();
    }
}
```

Java Nested and Inner class

Practice Exercises:

1. Create an outer class called "Bank" with a variable "balance".
Create an inner class called "Account" with a method "getBalance" that accesses the "balance" variable of the outer class.
Create an instance of the outer class and an instance of the inner class. Use the inner class to retrieve the balance.

Java Nested and Inner class

Practice Exercises:

2. Create an outer class called "University" with a variable "name".
Create an inner class called "Department" with a method "getUniversityName" that accesses the "name" variable of the outer class.
Create an instance of the outer class and an instance of the inner class. Use the inner class to retrieve the university name.

Java Nested and Inner class

Practice Exercises:

3. Create an outer class called "Library" with a variable "bookCount".
Create an inner class called "Book" with a method "displayBookCount" that displays the "bookCount" variable of the outer class.
Create an instance of the outer class and an instance of the inner class. Use the inner class to display the book count.

Java Nested and Inner class

Practice Exercises:

4. Create an outer class called "Car" with a variable "model".
Create an inner class called "Engine" with a method "startEngine" that prints a message saying the engine is starting along with the car model.
Create an instance of the outer class and an instance of the inner class. Use the inner class to start the engine.

Accessing Members of Outer Class within Inner Class

We can access the members of the outer class by using **this** keyword

```
class OuterClass {
    private int outerData = 10;

    class InnerClass {
        private int innerData = 20;

        public void displayData() {
            System.out.println("Outer data: " + OuterClass.this.outerData);
            System.out.println("Inner data: " + this.innerData);
        }
    }
}
```

Java Nested and Inner class

```
public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner = outer.new InnerClass();

        inner.displayData();
    }
}
```

Java Nested and Inner class

Static Nested Class

- In Java, we can also define a static class inside another class. Such class is known as **static nested class**. Static nested classes are **not** called **static inner classes**.

- Unlike inner class, a static nested class cannot access the member variables of the outer class. It is because the static nested class doesn't require you to create an instance of the outer class.

```
OuterClass.NestedClass obj = new OuterClass.NestedClass();
```

- Here, we are creating an object of the static nested class by simply using the class name of the outer class. Hence, the outer class cannot be referenced using OuterClass.this.

Java Nested and Inner class

```
class MotherBoard {

   // static nested class
   static class USB{
      int usb2 = 2;
      int usb3 = 1;
      int getTotalPorts(){
         return usb2 + usb3;
      }
   }

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

      // create an object of the static nested class
      // using the name of the outer class
      MotherBoard.USB usb = new MotherBoard.USB();
      System.out.println("Total Ports = " + usb.getTotalPorts());
   }
}
```

Key Points to Remember

- **Java treats the inner class as a regular member of a class. They are just like methods and variables declared inside a class.**
- **Since inner classes are members of the outer class, you can apply any access modifiers like private, protected to your inner class which is not possible in normal classes.**
- **Since the nested class is a member of its enclosing outer class, you can use the dot (.) notation to access the nested class and its members.**
- **Using the nested class will make your code more readable and provide better encapsulation.**
- **Non-static nested classes (inner classes) have access to other members of the outer/enclosing class, even if they are declared private.**

THANK YOU

Ms. Archana A
Department of Computer Applications

archana@pes.edu

+91 80 6666 3333 Extn 392