CS6308 Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

Day 1

V P Jayachitra

Syllabus

MODULE I FUNDAMENTALS OF JAVA LANGUAGE	L	T	P	EL
	3	0	4	3

Introduction to Java, Java basics – Variables, Operators, Expressions, Control flow Statements, Methods, Arrays

SUGGESTED ACTIVITIES :

- Practical-Implementation of simple Java programs Using Java Basic Constructs and Arrays using any standard IDE like NETBEANS / ECLIPSE
- EL – Understanding JVM

SUGGESTED EVALUATION METHODS:

- Assignment problems
- Quizzes

TEXT BOOKS:

1. Y. Daniel Liang, "Introduction to Java Programming and Data Structures, Comprehensive Version", 11th Edition, Pearson Education, 2018.
2. Herbert Schildt, "Java: The Complete Reference", 11th Edition, McGraw-Hill Education, 2018.

REFERENCES:

1. Paul Dietel and Harvey Deitel, "Java - How to Program Early Objects", 11th Edition, Pearson Education, 2017.
2. Sachin Malhotra, Sourabh Choudhary, "Programming in Java", Revised 2nd Edition, Oxford University Press, 2018.
3. Cay S. Horstmann, "Core Java - Vol. 1, Fundamentals", 11th Edition, Pearson Education, 2018.

Web references:

1. NPTEL
2. MIT OCW

EVALUATION PATTERN:

Category of Course	Continuous Assessment	Mid –Semester Assessment	End Semester
Theory Integrated with Practical	15(T) + 25 (P)	20	40

V P Jayachitra

The Origins of Java

- Java is conceived by
  - James Gosling
  - Patrick Naughton
  - Chris Warth
  - Ed Frank, and
  - Mike Sheridan

  at Sun Microsystems in 1991.
- Known as Oak initially and renamed as Java in 1995.

Motivation

- First, to create a platform-independent language
  - To create software for various consumer electronic devices, such as toasters, microwave ovens, and remote controls.
  - Why?
    - computer languages were designed to be compiled into machine code that was targeted for a specific type of CPU.
    - compilers are expensive and time consuming to create for all kind of CPU.
  - What?
    - portable, cross-platform language that could produce code that would run on a variety of CPUs under differing environments.

V P Jayachitra

Motivation contd…

- Second focus is the World Wide Web
  - Why?
    - Web, demanded portable programs
    - Java was an obscure language for consumer electronics.
  - What?
    - Java was propelled to the forefront of computer language design due to WWW.
    - architecture-neutral programming language
    - focus switch from consumer electronics to Internet programming.

V P Jayachitra

Motivation contd…

- Third, Security
  - Java's support for networking.
- Why?
  - library of ready-to-use functionality enabled programmers to easily write programs that accessed or made use of the Internet.
  - Downloading and executing program at multiplatformed environment may harm the system (virus, Trojan etc.,)
- What?
  - Confine an application to the Java execution environment and prevent it from accessing other parts of the computer.

Java's Lineage: C and C++

- From C, Java inherits its syntax.
- Java's object model is adapted from C++.
  - Java is not an enhanced version of C++
  - Neither upwardly nor downwardly compatible with C++
  - C++ was designed to solve a different set of problems.
  - Java was designed to solve a certain set of problems.

V P Jayachitra

Key features of Java

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

V P Jayachitra

Key features of Java contd…

- **Simple**
  - Easy to learn and use effectively.
    - no pointers/stack concerns
  - Because Java inherits the C/C++ syntax and many of the object-oriented features of C++
- **Object-Oriented**
  - “everything is an object” paradigm
  - Only primitive types, such as integers, are kept as high-performance nonobjects.

V P Jayachitra

Key features of Java contd…

- **Robust**
  - Java is a strongly typed language
    - checks the code at compile time, also checks the code at run time
  - Garbage collector
    - Automatic memory management-
  - Exception handling
- **Architecture-Neutral**
  - JVM
    - code longevity and portability.
    - “write once; run anywhere, any time, forever.”

V P Jayachitra

Key features of Java contd…

- **Multithreaded**
  - write programs that do many things simultaneously.
  - multiprocess synchronization -enables to construct smoothly running interactive systems
- **Interpreted and High Performance**
  - Java bytecode -carefully designed to translate directly into native machine code
  - For very high performance -just-in-time compiler

Key features of Java contd…

- **Distributed**
  - Java is designed for the distributed environment of the Internet as it handles TCP/IP protocols.
  - Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.

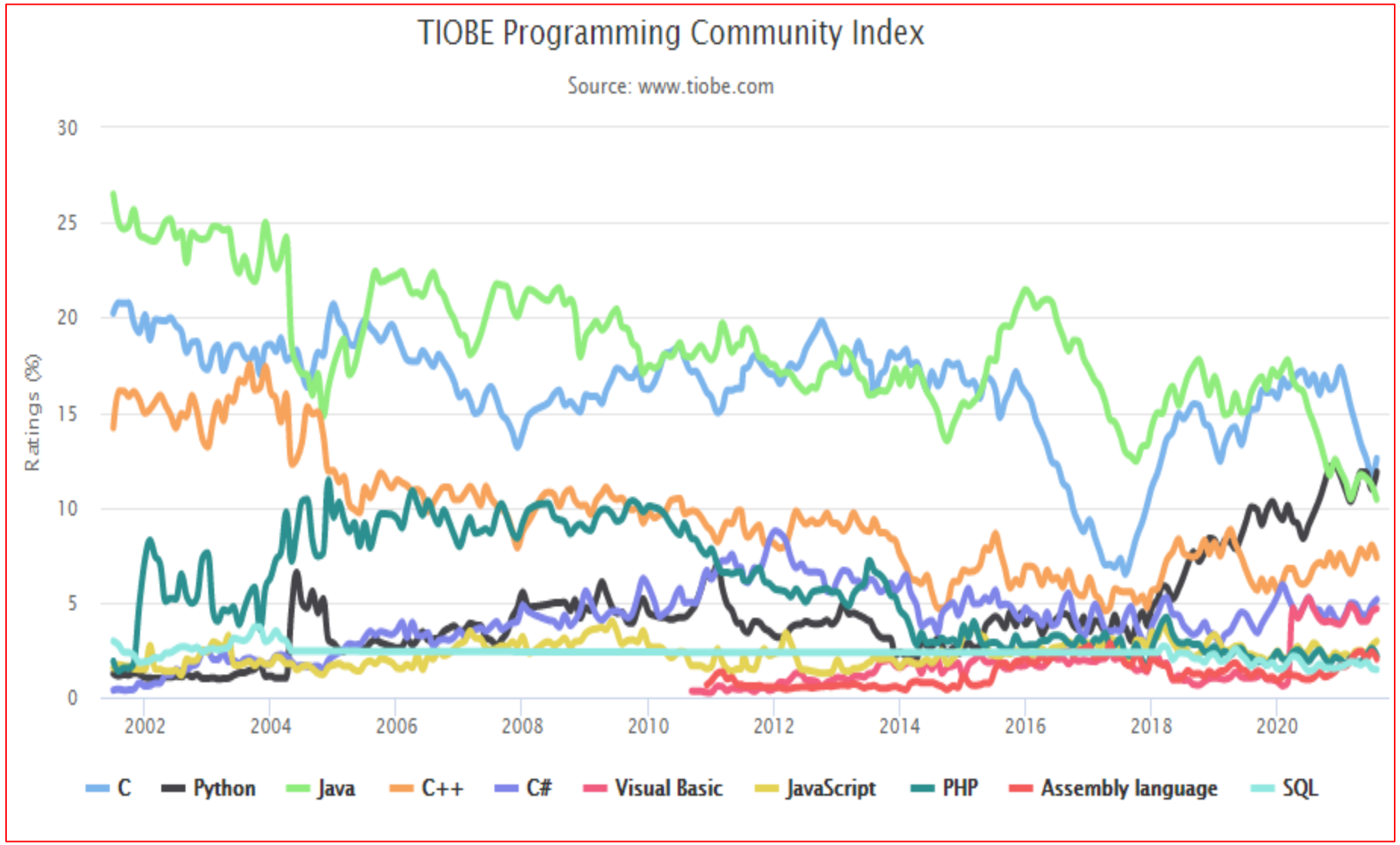
- **Interpreted and High Performance**
  - Java bytecode -carefully designed to translate directly into native machine code
  - For very high performance -just-in-time compiler

V P Jayachitra

Key features of Java contd…

- Dynamic
  - run-time type information -used to verify and resolve accesses to objects at run time.

Java – 2020 popular language



V P Jayachitra

Java Editions

- Java 2 Platform, Standard Edition (J2SE)
  - Desktop based application and networking applications
- Java 2 Platform, Enterprise Edition (J2EE)
  - Large-scale, distributed networking applications and Web-based applications
- Java 2 Platform, Micro Edition (J2ME)
  - Small memory-constrained devices, such as cell phones, pagers and PDAs

V P Jayachitra

Java development environment

- Text editor
- IDE (Integrated development environment)

Java IDE

- Integrated Development Environment(IDE)
  - a GUI based application that facilitates application development such as coding, compilation or interpretation and debug programs more easily.
  - An IDE contains
    - Code editor-syntax highlights
    - Build tools -Compiler /Debugger/Interpreter
    - Auto documentation-comments
    - Libraries
  - Top 3 lightweight IDE -2021
    - IntelliJ IDEA
    - Eclipse
    - NetBeans

V P Jayachitra

Overview of Java

- Two Paradigms
  - Procedural oriented paradigm
    - process-oriented model
    - code acting on data
    - problems with this approach appear as programs grow larger and more complex.
  - object-oriented programming paradigm
    - data controlling access to code.
      - organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data.
    - problems with this approach appear as programs grow larger and more complex.

Overview of Java

- Abstraction
  - A powerful way to manage abstraction is through the use of hierarchical classifications.
  - To manage complexity

Overview of Java

- The Three OOP Principle
  - Encapsulation
  - Inheritance, and
  - Polymorphism

Overview of Java

- Encapsulation:
  - mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
  - protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper.
  - knows how to access it and thus can use it regardless of the implementation details—and without fear of unexpected side effects.
  - Class
    - To encapsulate complexity
    - A class defines the structure and behavior (data and code) that will be shared by a set of objects.
    - Members-code and data
      - Data-member variables or instance variables.
      - Code-member method or method
        - behavior and interface of a class are defined by the methods that operate on its instance data.
  - objects are referred to as instances of a class.
  - Thus, a class is a logical construct; an object has physical reality.
  - Each method or variable in a class may be marked private or public.
  - he *public* interface of a class represents everything that external users of the class need to know, or may know.
  - The *private* methods and data can only be accessed by code that is a member of the class

V P Jayachitra

Overview of Java

- **Inheritance**
  - *Inheritance* is the process by which one object acquires the properties of another object.
  - Without the use of hierarchies, each object would need to define all of its characteristics explicitly.
  - an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its paren

V P Jayachitra

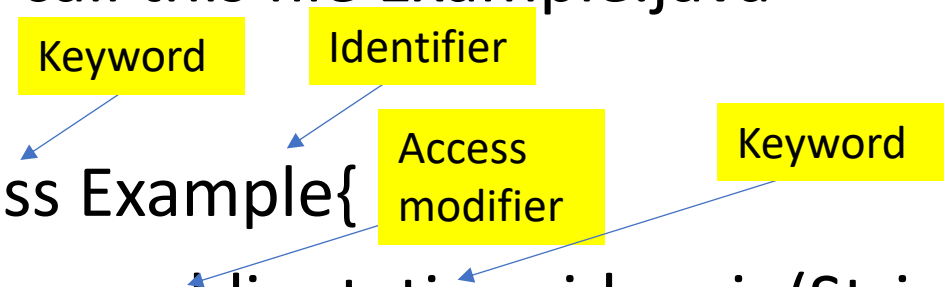
Overview of Java

- **Polymorphism**
  - is a feature that allows one interface to be used for a general class of actions.
  - The specific action is determined by the exact nature of the situation

V P Jayachitra

A First Simple Program

```
/* First Java program
     call this file Example.java
*/
class Example{
        public static void main(String args[]){
                System.out.println("This is java template");
}
```



A First Simple Program

- The keyword **class** to declare that a new class is being defined.
- **Example** is an *identifier* that is the name of the class.
- The entire class definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}).
- The public keyword is an access modifier, which allows the programmer to control the visibility of class members.
- The keyword static allows main( ) to be called without having to instantiate a particular instance of the class.
- This is necessary since main( ) is called by the Java Virtual Machine before any objects are made.
- The keyword void simply tells the compiler that main( ) does not return a value.
- Java compiler will compile classes that do not contain a main( ) method. But java has no way to run these classes
- String args[ ] declares a parameter named args, which is an array of instances of the class String.
- **args** receives any command-line arguments present when the program is executed.
- A complex program will have dozens of classes, only one of which will need to have a **main( )** method to get things started.
- **System** is a predefined class that provides access to the system, and **out** is the output stream that is connected to the console.
- println() is the method

V P Jayachitra

Java program

- In Java, a source file is officially called a compilation unit.
- The Java compiler requires that a source file use the .java filename extension.
- In Java, all code must reside inside a class.
- Java is case-sensitive
- Filename should match the class name

V P Jayachitra

Compiling the Program

- Execute the compiler, **javac**, specifying the name of the source file on the command line
- C:\>javac Example.java
- The **javac** compiler creates a file called **Example.class** that contains the bytecode version of the program.
- Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute.
- The output of **javac** is not code that can be directly executed.

V P Jayachitra

Run the program

- Use Java application launcher called **java**.
- Pass the class name **Example** as a command-line argument
- C:\>java Example
- Output:
  - This is java template

Comment

- The contents of a comment are ignored by the compiler.
- Java supports three styles of comments.
  - multiline comment.
    - begin with /* and end with */
  - Single line comment
    - //
  - Documentation comment
    - begin with /** and end with */.

```
/**
* <h1> sum of two numbers !</h1>
* program finds the sum
*and gives the output on
*the screen.
*
*/
```

V P Jayachitra

JAVA‘S COMPONENT

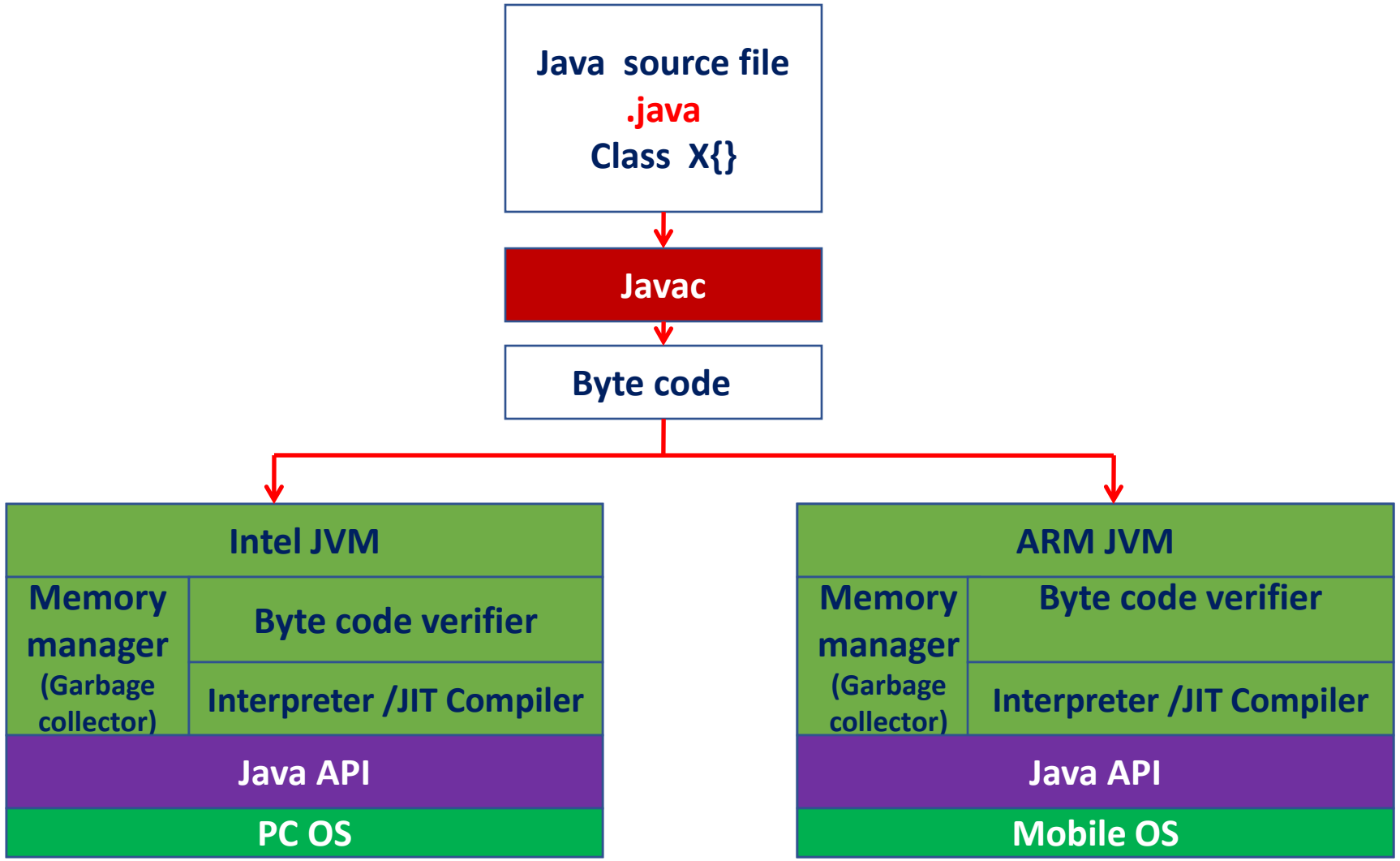
- JVM: Platform-independent environment for running Java programs by loading, validating, and executing code.

- JRE: Software bundle that includes the JVM, Java class libraries, and class loader to create an environment for running Java files.

- JDK: Software development environment including a private JVM, tools, and libraries for creating Java programs

V P Jayachitra

JVM Architecture



JVM architecture

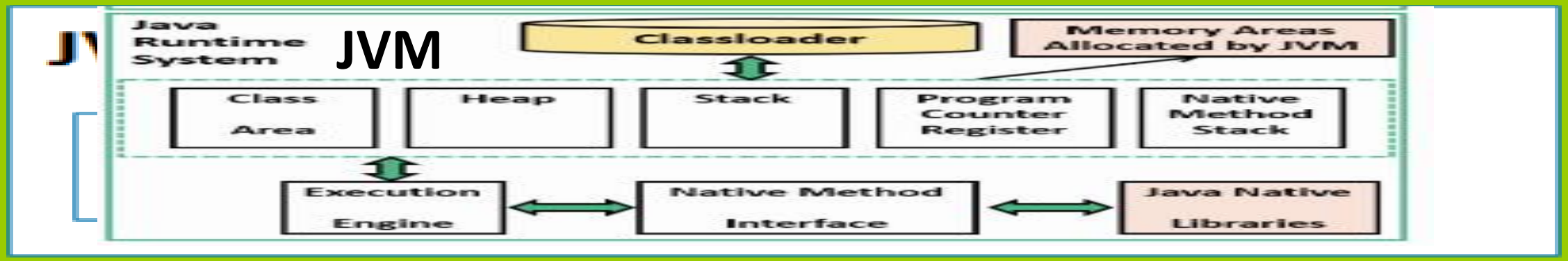
JVM

JDK

javac, jar, debugging tools, javap

JRE

java, javaw, libraries, rt.jar



V P Jayachitra

JVM

- Bootstrap ClassLoader: java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes
- Extension ClassLoader: loades the jar files located inside $JAVA_HOME/jre/lib/ext directory.
- System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath.
- Class(Method) Area Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.
- Heap It is the runtime data area in which objects are allocated.
- Stack Java Stackstores frames. It holds local variables and partial results, and plays a part in method invocation and

V P Jayachitra

JVM

- Program Counter Register PC (program counter) register: contains the address of the Java virtual machine instruction currently being executed.
- Native Method Stack It contains all the native methods used in the application.
- Execution Engine It contains:
  - 1. A virtual processor
  - 2. Interpreter: Read bytecode stream then execute the instructions.
  - 3. Just-In-Time(JIT) compiler:
- 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.
  - Java uses JNI framework to send output to the Console or interact with OS libraries.

V P Jayachitra

JDK VS JRE VS JVM

JDK	**JRE**	**JVM**
Java Development Kit (JDK) a software development kit used to develop Java applications.	JavaRuntime Environment (JRE) a software package that provides Java Virtual Machine (JVM), class libraries, and other components to run applications in Java.	Java Virtual Machine (JVM) **an abstract machine that provides an environment for the execution of Java ByteCodes.**
JDK contains tools for developing, monitoring, and debugging Java codes.	JRE contains class libraries and other supporting files required by JVM for executing Java programs.	JVM does not include any software development tools.
platform-dependent i.e. different platforms require different JDK.	platform-dependent.	platform-dependent.
Used to create an environment to write Java programs that JRE and JVM can execute.	Used to create an environment for the execution of Java programs.	JVM specifies all the implementations.
JDK = JRE + Development tools	JRE = JVM + Class libraries	JVM = provides a runtime environment.

V P Jayachitra

write once and run anywhere

- It is the bytecode.

- Java compiler converts the Java programs into the .class file known as the Byte Code,
- Byte code is the intermediate language between source code and machine code.
- This bytecode is not platform-specific and can be executed on any machine.

JVM?

- JVM (Java Virtual Machine): A virtual machine that interprets Java bytecode.
- Responsibilities:
  - Loading Bytecode: Loads .class files containing Java bytecode.
  - Verifying Bytecode: Ensures that the bytecode adheres to Java's security and integrity rules.
  - Executing Bytecode: Executes the bytecode by converting it into machine code.
- Implementation: Known as the Java Runtime Environment (JRE).
- Platform Dependency: The JVM software is tailored for different operating systems, making it platform-dependent.
- Platform Independence: Despite its platform dependency, the JVM enables Java to be platform-independent by allowing the same bytecode to run on any system with a compatible JVM.

V P Jayachitra

JIT Compiler

- Write Java Code: Develop your program using Java.
- Compile with javac: The Java compiler (javac) converts the source code into bytecode, which is saved in .class files.
- Load Bytecode: The Java Virtual Machine (JVM) loads the .class files at runtime.
- Interpret Bytecode: The JVM's interpreter converts the bytecode into machine-understandable instructions.
- JIT Compilation: The Just-In-Time (JIT) compiler within the JVM analyzes frequently executed methods and compiles them into optimized native machine code.
- Execute Optimized Code: The JVM executes the optimized native code directly, bypassing the need for interpretation.
- Improved Performance: This process results in faster execution and better performance of your Java program.

V P Jayachitra

public static void main(String args[]) i

- Public: the public is the access modifier responsible for making the main function globally available. It is made public so that JVM can invoke it from outside the class, as it is not present in the current class.
- Static: It is a keyword to use the element without initiating the class to avoid the unnecessary allocation of the memory.
- Void: void is a keyword used to specify that a method doesn’t return anything. The main method doesn’t return anything.
- Main: main helps JVM to identify that the declared function is the main function to begin the execution
- String args[]: It stores Java command-line arguments and is an array of type java. lang.String class.

Java command line argument

```
class CommandLineArgs {
 public static void main(String[] args) {
if (args.length > 0) {
 System.out.println("The command line arguments are:");
for (String value : args)
 System.out.println(value);
 } else
 System.out.println("No command line arguments found.");
 }
}
```

```
javac CommandLineArgs.java
java CommandLineArgs Welcome to Java
The command line arguments are:
Welcome
 to
Java
```

Can main method in java be overloaded?

Yes, the main method can be overloaded.

We can create as many overloaded main methods as we want.

However, JVM has a predefined calling method that JVM will only call the main method with the definition of

public static void main(string[] args)

V P Jayachitra

Can main method in java be overloaded?

```
class Main {
 public static void main(String[] args) {
 System.out.println("This is the main method");
 }

 public static void main(int[] args) {
 System.out.println("Overloaded integer array main method");
 }
```

```
This is the main method
```

What is System.out.println?

```
public final class System {
    // Static field representing the standard output stream
    public static final PrintStream out = new PrintStream(new
FileOutputStream(FileDescriptor.out));

    // Other fields and methods of System class
}
```

```
package java.io;
 public class PrintStream extends OutputStream{
  public void println() { /* implementation */ } {
public void println() { /* implementation */ }
}
```

- **System** - It is a class present in java.lang package.
- **out** is the **static variable** of type PrintStream class present in the System class
- Key Members of System:
  - **out**: A static field of type PrintStream which is used for outputting text to the console.
  - **in**: A static field of type InputStream used for reading input from the console.
- **.println()** is the method present in the PrintStream class.

V P Jayachitra

Elements of a Java Application

Elements	
Java Classes Basic building blocks Define structure and behavior	**Constructors** Special methods for initialization Same name as the class, no return type
Objects Instances of classes Hold state and perform actions	**Main Method** Entry point of a Java application public static void main(String[] args)
Methods Functions within a class Define actions and behaviors	**Comments** Document code Help with code readability
Fields (Attributes) Variables in a class Store data related to objects	

V P Jayachitra

```
public class Car {
 // Class variables
 String color;
 int modelYear;
 // Class constructor
 public Car(String color, int modelYear) {
   this.color = color;
   this.modelYear = modelYear;
 }
 // Class methods
 public void start() {
   System.out.println("The car is starting.");
 }
 public void stop() {
   System.out.println("The car is stopping.");
 }
}
```

```
public class Main {
 public static void main(String[] args) {
  // Create instances of Car
    Car myCar = new Car("Red", 2024);
    Car yourCar = new Car("Blue", 2022);
     // Display details and use methods of Car
  System.out.println("My car color: " + myCar.color + ", Model year: " +
myCar.modelYear);
   myCar.start();
   myCar.stop();
   System.out.println("Your car color: " + yourCar.color + ", Model year: "
yourCar.modelYear);
    yourCar.start();
    yourCar.stop();    }}
```

```
The car is starting.
The car is stopping.
```

V P Jayachitra

```
//creating instance of a class using new keywor
ClassName objectName = new ClassName();
```

Class vs Object

A class in Java is a blueprint or template that determines the structure and behavior of objects

An object is an instance of a class that represents a real-world entity, whereas

Aspect	Class	Object
Definition	Blueprint or template defining properties and behavior	Instance of a class with actual values
Usage	Used to create objects	Created using a class constructor
Properties	Defines properties (variables)	Holds values for those properties
Methods	Defines behavior (methods)	Can call methods to perform tasks
Memory Allocation	Not allocated memory in JVM	Allocated memory when created with new keyword

V P Jayachitra

```
public class Phone {
   // Class variables (attributes)
   private String brand;
   private String model;
   private int storageCapacity; // in GB
   private double screenSize;   // in inches
   private boolean isSmartphone;
   // Constructor
   public Phone(String brand, String model, int storageCapacity,
    double screenSize, boolean isSmartphone) {
      this.brand = brand;
      this.model = model;
      this.storageCapacity = storageCapacity;
      this.screenSize = screenSize;
      this.isSmartphone = isSmartphone;
   }
   // Method to display phone details
   public void displayDetails() {
      System.out.println("Phone Brand: " + brand);
      System.out.println("Phone Model: " + model);
      System.out.println("Storage Capacity: " + storageCapacity + " GB");
      System.out.println("Screen Size: " + screenSize + " inches");
      System.out.println("Smartphone: " + (isSmartphone ? "Yes" : "No"));
   }
      public void setSmartphone(boolean isSmartphone) {
      this.isSmartphone = isSmartphone;
   } }
```

```
// Method to make a call (example of functionality)
   public void makeCall(String phoneNumber) {
      System.out.println("Making a call to: " + phoneNumber);
   }
   // Method to send a message (example of functionality)
   public void sendMessage(String phoneNumber, String message) {
      System.out.println("Sending message to: " + phoneNumber);
      System.out.println("Message: " + message);
   }
public class Amazon {
 public static void main(String[] args) {
  // Create an instance of Phone
  Phone myPhone = new Phone("Samsung", "Galaxy S21", 128, 6.8, true);
   // Display phone details
   myPhone.displayDetails();
  // Make a call
   myPhone.makeCall("+123456789");
  // Send a message
   myPhone.sendMessage("+123456789", "Hello, this is a test message.");
}}
```

V P Jayachitra

```
// Define the class
public class ClassName {
    // Fields (Instance variables)
    private DataType variableName;
    // Constructor
    public ClassName() {
        // Initialize fields
        variableName = initialValue; // Example: message = "";
    }
    // Method to perform some action
    public void methodName(ParameterType parameter) {
        // Method body
        variableName = parameter; // Example: message = newMessage;
        System.out.println("Message posted: " + variableName);
    }
    // Method to perform another action
    public ReturnType anotherMethod() {
        // Method body
        return value; // Example: return message;
    }
```

```
// Main method to test the class
    public static void main(String[] args) {
        // Create an instance of the class
        ClassName instanceName = new ClassName();
        // Call methods on the instance
        instanceName.methodName(parameterValue);
        // Example: app.postMessage("Hello, World!");
        instanceName.anotherMethod();
        // Example: app.readMessage();
    }
}
```

V P Jayachitra

```
public class WatsApp {

   private String message;
private int messageCount;
   // Constructor
   public SimpleMessagingApp() {
      message = "";
      messageCount = 0;
   }
   // Method to post a message
   public void postMessage(String newMessage) {
      message = newMessage;
      messageCount++;
      System.out.println("Message posted: " + newMessage);
   }
   // Method to read the current message
   public void readMessage() {
      if (messageCount > 0) {
         System.out.println("Reading message: " + message);
      } else {
         System.out.println("No messages to read.");
      }
   }
```

```
public static void main(String[] args) {
      // Create an instance of SimpleMessagingApp
      SimpleMessagingApp app = new SimpleMessagingApp();
      app.postMessage("Hello, World!");
      app.readMessage();
      app.postMessage("How are you?");
      app.readMessage();
   }
}
```

V P Jayachitra

Default values

Primitive Data Types:
byte: 0
short: 0
int: 0
long: 0L
float: 0.0f
double: 0.0d
char: '\u0000' (null character)
boolean: false

Reference Data Types:
Object: null
String: null
Arrays: null (since arrays are objects in Java)

V P Jayachitra

Elements of a Java Application

Packages Organize related classes and interfaces Avoid name conflicts	**Polymorphism** Treat objects as instances of a common class Allows generic handlin
Interfaces Define a set of methods without implementations Used for abstraction	**Encapsulation** Hide internal state Access via methods (getters and setters)
Inheritance Mechanism to reuse code Subclass inherits fields and methods from superclass	

V P Jayachitra

Encapsulation

- Encapsulation groups data and methods into a class while hiding implementation-specifics and exposing a public interface.
- Encapsulation in Java includes limiting direct access by defining instance variables as private.
- There are defined public getters and setters for these variables.
- Advantages of Encapsulation in Java
  - Make a class read-only or write-only using only getter or setter methods, respectively.
  - Control data by providing logic in setter methods.
  - Hide data so other classes can't access private members directly.

```
public class User {
    // Public username for direct access
    public String username;

    // Private password with encapsulation
    private String password;

    // Constructor
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    // Setter for password with validation
    public void setPassword(String password) {
        if (password != null && password.length() >= 6) {
            this.password = password;
        } else {
            System.out.println("Password must be at least 6
characters long.");
        }
    }
```

```
    // Main method to demonstrate usage
    public static void main(String[] args) {
        // Create a User object
        User user = new User("Alice", "securePass123");

        // Access and modify public field directly
        user.username = "Bob";

        // Access and modify private field using getter and setter
        user.setPassword("newPass456");

        // Display user information using object fields and
methods
        System.out.println("Username: " + user.username);
        System.out.println("Password: " + user.getPassword());
    }
}
```

ENCAPSULATION

V P Jayachitra

Abstraction in Java

- Abstraction in Java is achieved through interfaces and abstract classes.
- Data abstraction involves identifying only the essential characteristics of an object while ignoring irrelevant details.

```
abstract class Shape {
 // Abstract method
 public abstract double calculateArea();
}
```

```
// Concrete class representing a Circle
class Circle extends Shape {
 private double radius; public Circle(double radius) {
 this.radius = radius;
 } @Override
 public double calculateArea() {
 return Math.PI * radius * radius; }
```

```
// Concrete class representing a Rectangle
class Rectangle extends Shape {
 private double length;
 private double width;
public Rectangle(double length, double width) {
 this.length = length;
 this.width = width;
 } @Override
 public double calculateArea() {
 return length * width;
 }
```

```
Circle - Area: 78.53981633974483
Rectangle - Area: 24.0
```

```
public class AbstractionExample {
 public static void main(String[] args) {
 // Creating objects of Circle and Rectangle
 Circle circle = new Circle(5);
 Rectangle rectangle = new Rectangle(4, 6); // Using the abstracted methods to calculate area
and perimeter
 System.out.println("Circle - Area: " + circle.calculateArea() + ", Perimeter: " +
circle.calculatePerimeter());
 System.out.println("Rectangle - Area: " + rectangle.calculateArea() + ", Perimeter: " +
rectangle.calculatePerimeter());
 }
}
```

V P Jayachitra

Inheritance

- In Java, inheritance encourages the reuse of existing code and the development of hierarchies of related classes.
- The superclass or base class is the one from which the subclass inherits.
- The properties and methods of a superclass can be used by a subclass.
- To increase the functionality of the superclass, subclasses can also add additional fields and methods.

V P Jayachitra

```
// Base class (superclass)
class Animal {
 String name;
  public Animal(String name) {
 this.name = name;
 }
 public void eat() {
 System.out.println(name + " is eating.");
 }
public void sleep() {
 System.out.println(name + " is sleeping.");
 }
}
```

```
// Derived class (subclass)
class Dog extends Animal {
 public Dog(String name) {
 super(name);
 } public void bark() {
 System.out.println(name + " is barking.");
 }
}
```

```
Buddy is eating.
Buddy is sleeping.
Buddy is barking
```

```
public class InheritanceExample {
 public static void main(String[] args) {
 // Create a Dog object
 Dog myDog = new Dog("Buddy"); // Call methods from the base class (Animal)
 myDog.eat();
 myDog.sleep(); // Call a method from the derived class (Dog)
 myDog.bark();
 }
}
```

V P Jayachitra

Polymorphism

- One interface or function signature may have many implementations
- It supports OOP's flexibility and code reuse.
- Method overloading and method overriding are frequently used to achieve polymorphism.
- Removing the underlying implementation details facilitates code maintenance and improves code readability.
- To achieve dynamic binding, also known as late binding, where the exact method to be executed is determined at runtime, polymorphism is an essential concept.

```
class Shape {
 public void draw() {
 System.out.println("Drawing a shape");
 }

}
```

```
class Circle extends Shape {
 @Override
 public void draw() {
 System.out.println("Drawing a circle");
 }
}
```

```
class Rectangle extends Shape {
 @Override
 public void draw() {
 System.out.println("Drawing a
rectangle");
 }
```

```
public class PolymorphismExample {
 public static void main(String[] args) {
 Shape shape1 = new Circle();
 Shape shape2 = new Rectangle();
 shape1.draw(); // Calls the draw() method of Circle
 shape2.draw(); // Calls the draw() method of Rectangle
 }
}
```

V P Jayachitra

String literals vs String Object

```
public class StringExample {
    public static void main(String[] args) {
        String s = "sasf";       // String literal
        String s1 = new String("sasf"); // String object

        // Compare references
        System.out.println(s == s1);   // false, different references
        System.out.println(s.equals(s1)); // true, same value
    }
}
```

V P Jayachitra

String literals vs String Object

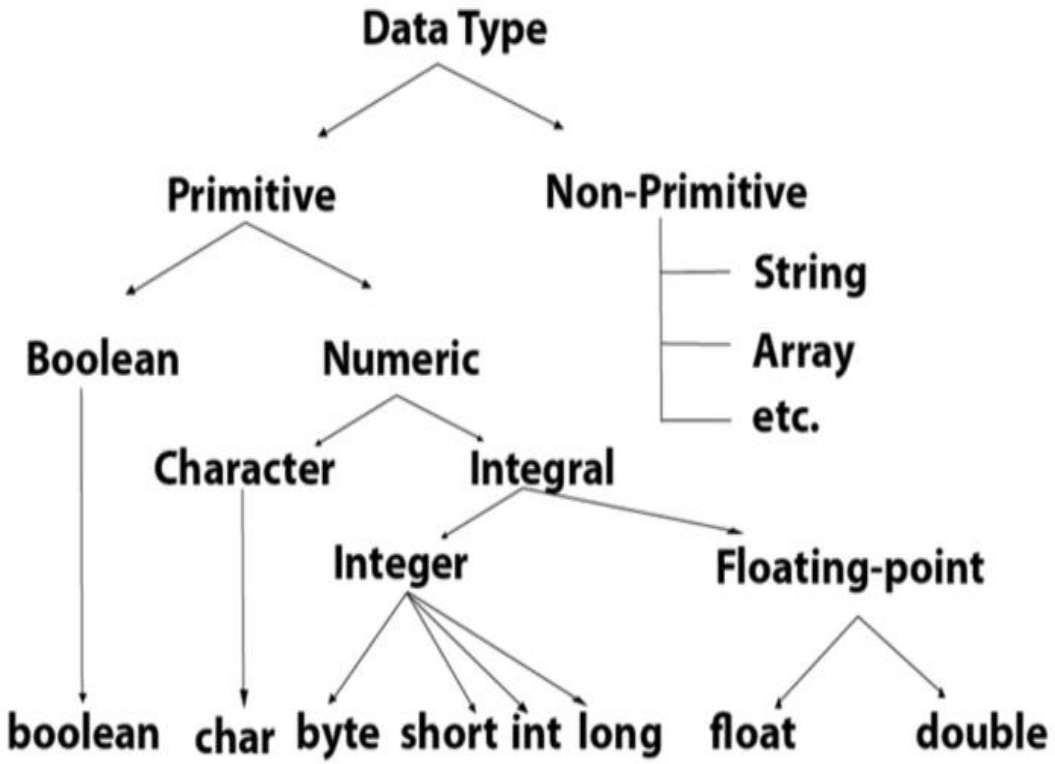
```
public class StringAddressExample {
    public static void main(String[] args) {
        String s = "sasf";       // Step 1: String literal
        String s1 = new String("sasf"); // Step 2: String object

        // Print references
        System.out.println("Address of s: " + System.identityHashCode(s));
        System.out.println("Address of s1: " + System.identityHashCode(s1));

        // Compare references and values
        System.out.println("s == s1: " + (s == s1));   // false, different references
        System.out.println("s.equals(s1): " + s.equals(s1)); // true, same value
    }
}
```

V P Jayachitra

Data Type



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

Java vs python

- Java: Suitable for web development, Big Data, and IoT applications in addition to mobile apps.

- Python:used in machine learning, image processing, multimedia applications, and other fields.