Object Oriented Programming

Asif Iqbal Middya
Assistant Professor
Dept of CSE, FIEM

C language (Platform Dependent)

C language (Platform Dependent)
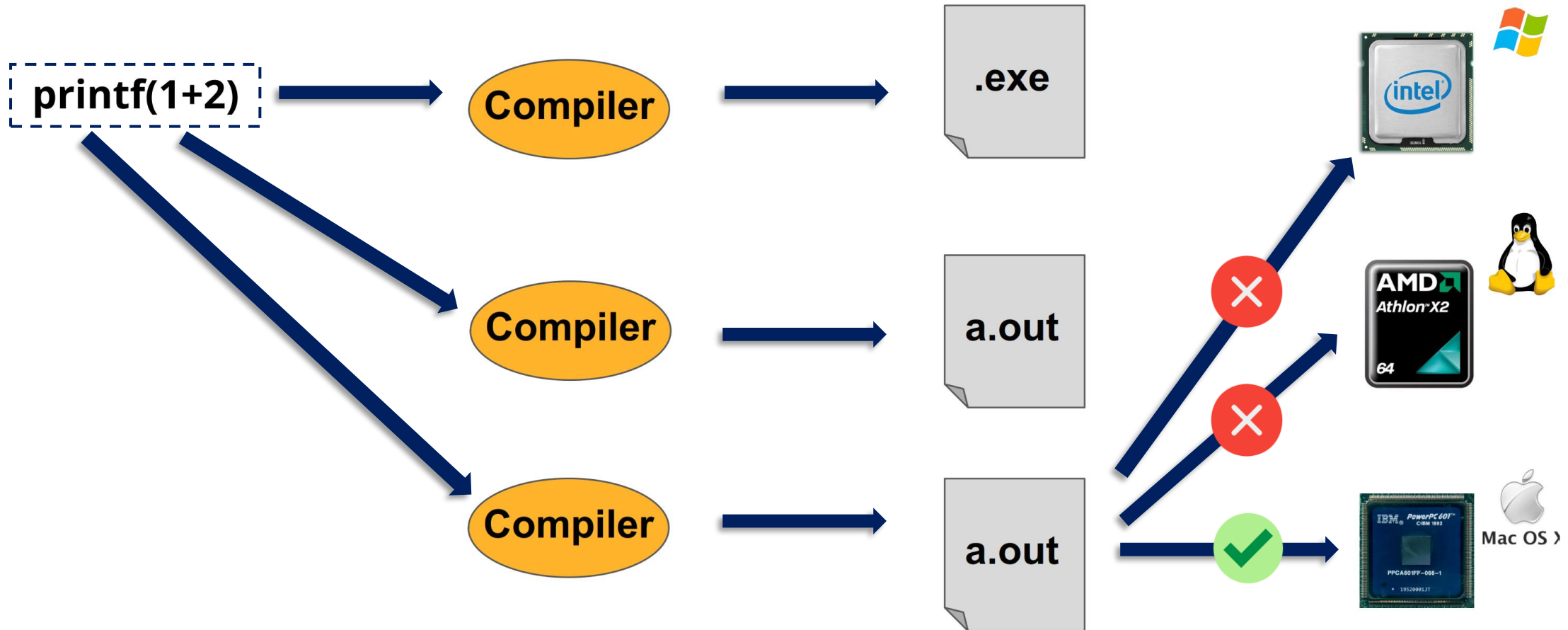
printf(1+2)
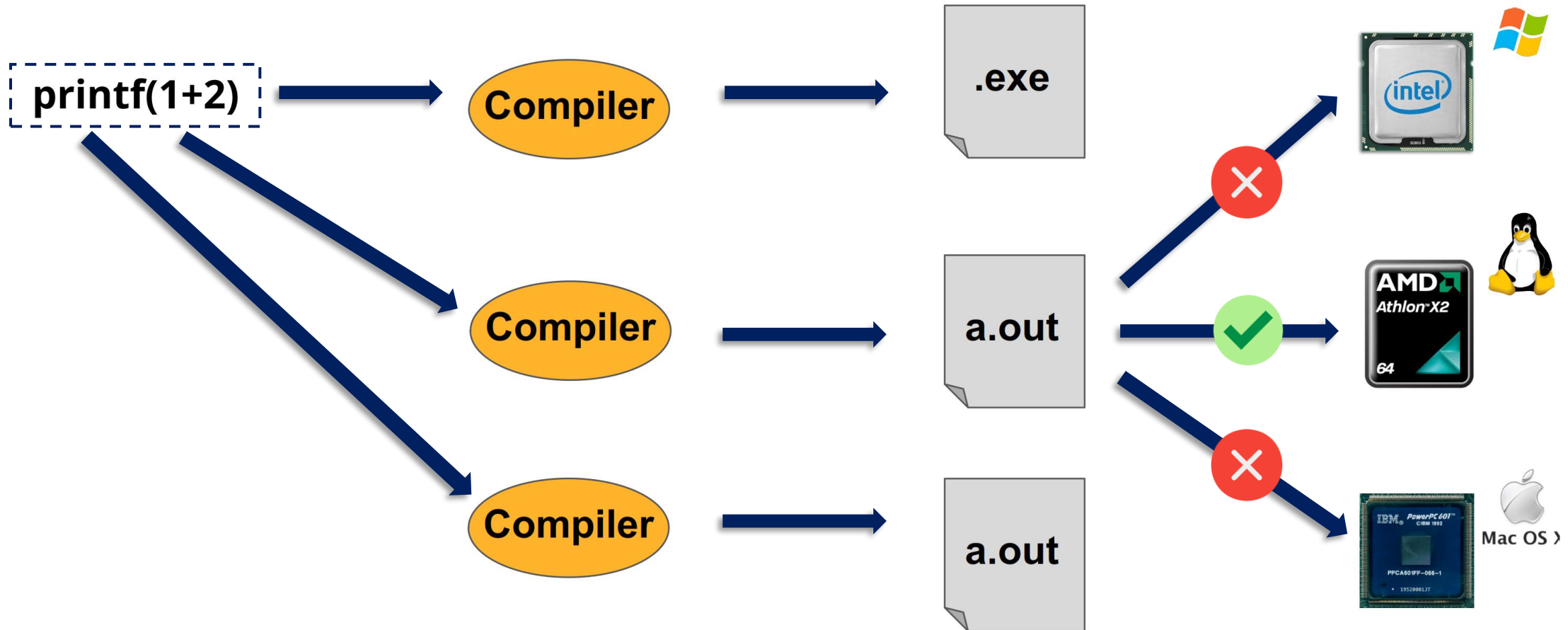
C language (Platform Dependent)

Java (Platform Independent)

system.out.println(1+2); **.java file**

Java Compiler → 00011010110 BYTECODE **.classfile**

Windows JVM → 101110101010

Linux JVM → 111110101111

Mac O.S JVM → 000110101100
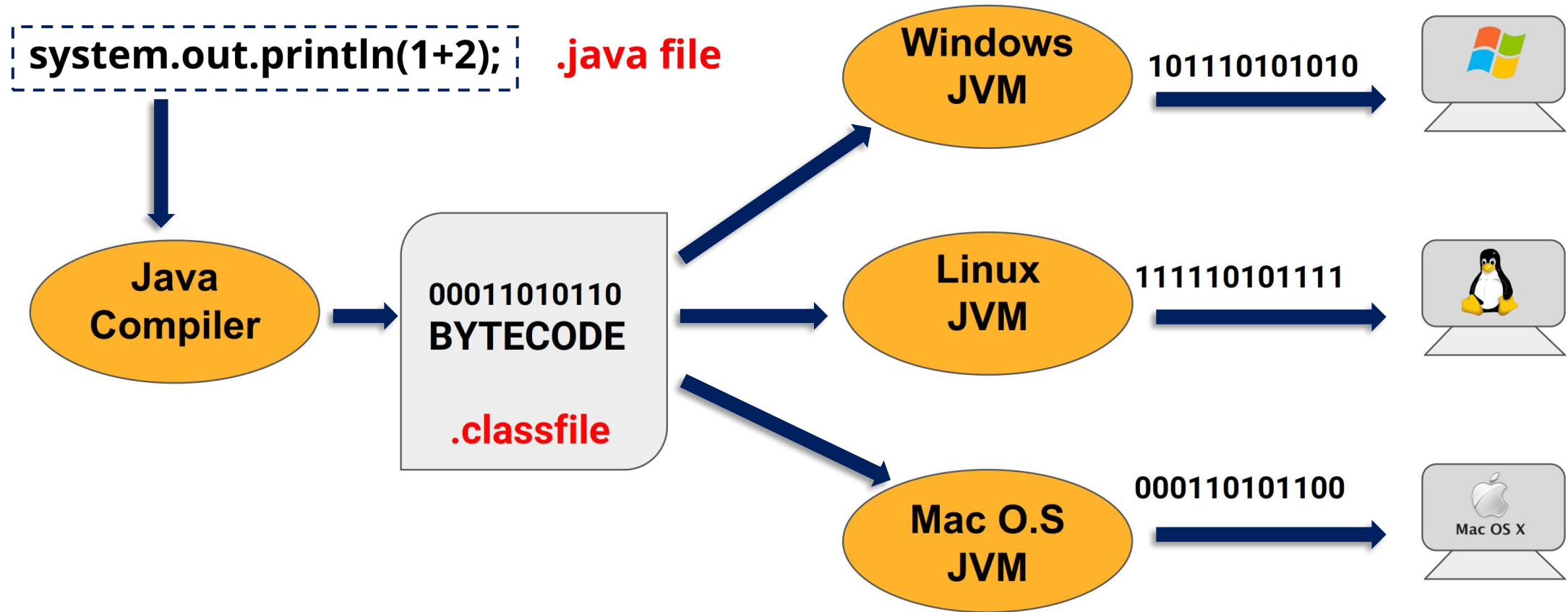
Programming Paradigms

A programming paradigm is a style or way of programming.

Procedural Programming Vs Object Oriented Programming

Procedural Programming

Object Oriented Programming

Procedural Programming Vs Object Oriented Programming

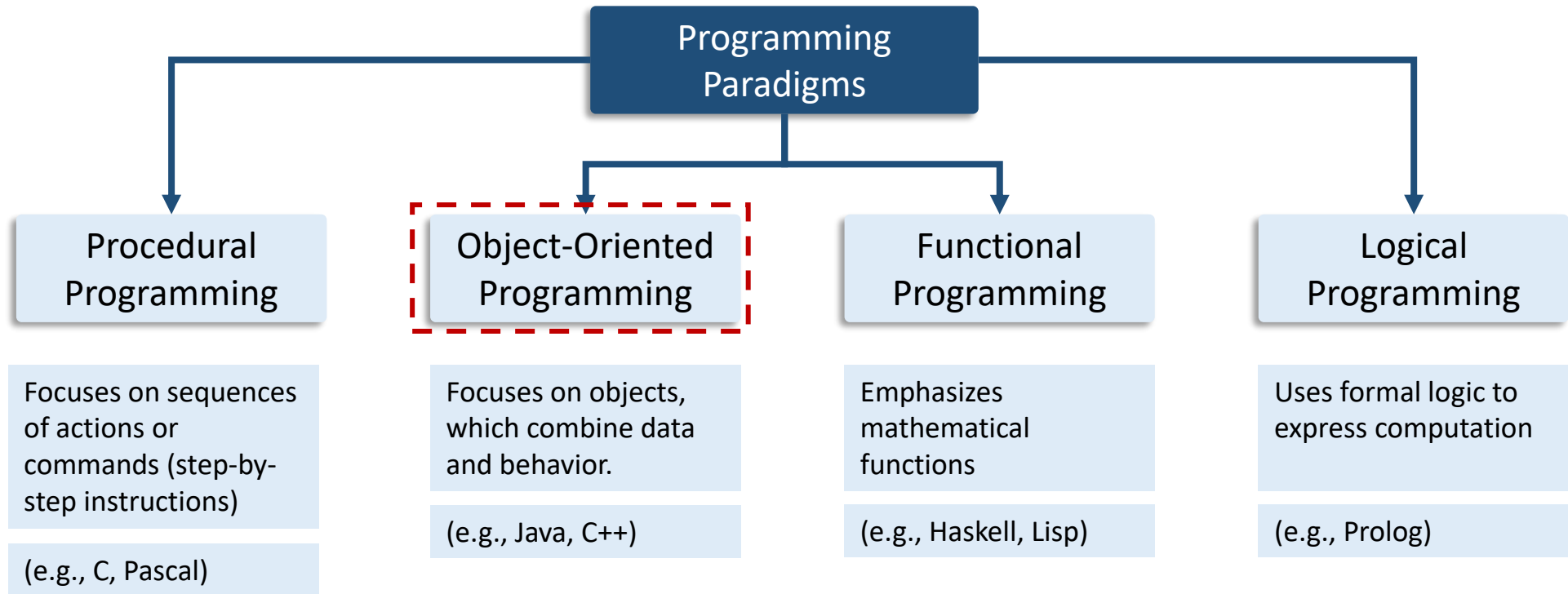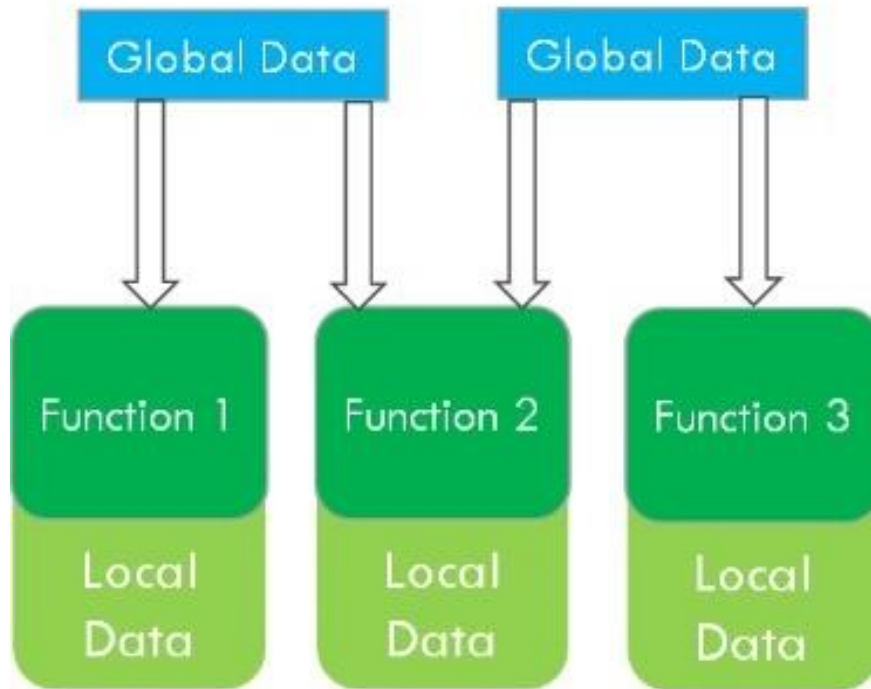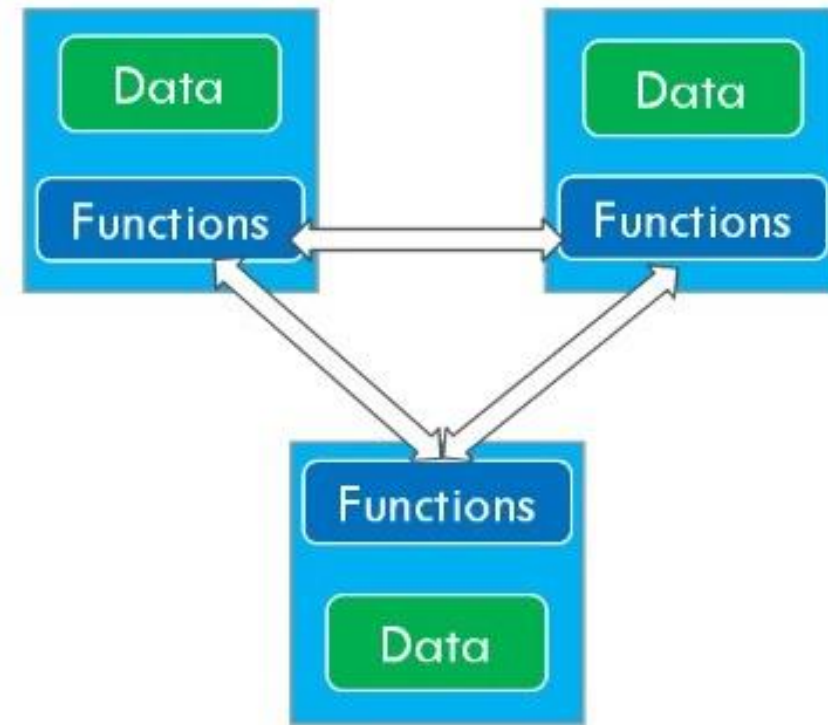| Feature	Procedural Programming (PP)	Object-Oriented Programming (OOP)
Organization	Divides a program into functions (procedures) that operate on data.	Structures a program around objects, which are instances of classes that encapsulate data and behavior (methods).
Focus	Focuses on the sequence of steps (procedures) to be executed to achieve a desired outcome.	Focuses on modeling real-world entities as objects, with attributes (data) and methods (actions) that define their behavior.
Approach	Top-down approach: Starts with the overall problem and breaks it down into smaller subproblems.	Bottom-up approach: Starts by modeling the entities (objects) and their interactions.
Code Reusability	Limited code reusability.	High code reusability through inheritance and polymorphism.
Languages	C, FORTRAN, Pascal	C++, Java, Python, C#
Use Cases	Scripting, tasks that involve a clear sequence of steps (e.g., data processing, file manipulation), smaller projects.	Large-scale applications, complex systems, projects where modeling real-world entities is beneficial (e.g., simulations, games, business applications), projects where code reusability and maintainability are crucial.

Key Concepts of OOP

Class

A class is a blueprint / design / template that describes something

Example: Design of a Car

Properties	Methods - behaviors
color	start()
price	backward()
km	forward()
model	stop()

Object

An object is an instance of a class. (Create based on the blueprint / design / template)

Property values	Methods
color: red	start()
price: 23,000	backward()
km: 1,200	forward()
model: Audi	stop()

Simple Java Program Example: Printing "Hello, World!"

The class is accessible from other classes

Keyword used to declare a class in Java

Name of the class (should start with a capital letter)

Class Declaration ➡

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

Simple Java Program Example: Printing "Hello, World!"

The method is accessible from other classes.

The method can be called without creating an instance of the class

The method does not return a value

Main method - the entry point for any Java application.

An array of command-line arguments, not used in this example

Main Method

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

Simple Java Program Example: Printing "Hello, World!"

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

Print statement →

A class that provides system-related utilities.

A static field in the System class

A method used to print messages to the console.

The the string to be printed.

Compiling and Running a Java Program

File name:
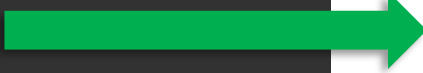HelloWorld.java

To compile:
javac HelloWorld.java

To run:
java HelloWorld

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

Java Program : A popular Template

Scanner Class: To Read Different Types of Data from the User

Example: Read **String** Input

```java
import java.util.Scanner;

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

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        System.out.println("Name: " + name);

        sc.close();
    }
}
```

Import the Scanner class

Create a Scanner object

Print a message to the user

Reads a line of text from the user

Display the collected information

Close the Scanner (good practice)

Scanner Class: To Read Different Types of Data from the User

Example: Read **Integer** Input

```java
1  import java.util.Scanner;
2
3  public class UserInputExample {
4      public static void main(String args[]) {
5
6          Scanner sc = new Scanner(System.in);
7
8          System.out.print("Enter your age: ");
9          int age = sc.nextInt();
10
11         System.out.println("Age: " + age);
12
13         sc.close();
14     }
15 }
16
```

Import the Scanner class

Create a Scanner object

Print a message to the user

Reads an integer input from the user

Display the collected information

Close the Scanner (good practice)

Scanner Class: To Read Different Types of Data from the User

Example: Read **Float** Input

```java
1  import java.util.Scanner;
2
3  public class UserInputExample {
4      public static void main(String args[]) {
5
6          Scanner sc = new Scanner(System.in);
7
8          System.out.print("Enter your height in meters: ");
9          float height = sc.nextFloat();
10
11         System.out.println("Height: " + height + " meters");
12
13         sc.close();
14     }
15 }
16
```

Import the Scanner class

Create a Scanner object

Print a message to the user

Reads a float input from the user

Display the collected information

Close the Scanner (good practice)

Q1

Write a Java program that takes the following inputs from the user and then displays the information:

- Student's full name (String)
- Roll number (Integer)
- Marks in three subjects (Float for each subject)

Calculate and display the total and average marks.

```java
import java.util.Scanner;

public class StudentInfo {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner scanner = new Scanner(System.in);

        // Taking String input for full name
        System.out.print("Enter the student's full name: ");
        String fullName = scanner.nextLine();

        // Taking Integer input for roll number
        System.out.print("Enter the student's roll number: ");
        int rollNumber = scanner.nextInt();

        // Taking Float input for marks in three subjects
        System.out.print("Enter the marks for subject 1: ");
        float marks1 = scanner.nextFloat();

        System.out.print("Enter the marks for subject 2: ");
        float marks2 = scanner.nextFloat();

        System.out.print("Enter the marks for subject 3: ");
        float marks3 = scanner.nextFloat();

        // Calculating total and average marks
        float totalMarks = marks1 + marks2 + marks3;
        float averageMarks = totalMarks / 3;

        // Displaying the student's information
        System.out.println("\nStudent Information:");
        System.out.println("Full Name: " + fullName);
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Marks in Subject 1: " + marks1);
        System.out.println("Marks in Subject 2: " + marks2);
        System.out.println("Marks in Subject 3: " + marks3);
        System.out.println("Total Marks: " + totalMarks);
        System.out.println("Average Marks: " + averageMarks);

        // Close the scanner
        scanner.close();
    }
}
```

Features of Object-Oriented Programming (OOP)

- **Classes and Objects**

- **Encapsulation**

- **Inheritance**

- **Polymorphism**

- **Abstraction**

Classes and Objects in Java

Class

A class is a blueprint / design / template that describes something

Example: Design of a Car

Properties	Methods - behaviors
color	start()
price	backward()
km	forward()
model	stop()

Object

An object is an instance of a class. (Create based on the blueprint / design / template)

Property values	Methods
color: red	start()
price: 23,000	backward()
km: 1,200	forward()
model: Audi	stop()

Creating / Defining a Class

Syntax →

```
class ClassName {
    // properties
    // methods
}
```

Example of Class Definition

```
1  class Car {
2      // Properties (Attributes)
3      String color;
4      String model;
5      int year;
6
7      // Methods (Behaviors)
8      void displayDetails() {
9          System.out.println("Model: " + model + ", Color: " + color + ", Year: " + year);
10     }
11 }
12
```

Class Definition

```java
class Car {
    // Properties (Attributes)
    String color;
    String model;
    int year;

    // Methods (Behaviors)
    void displayDetails() {
        System.out.println("Model: " + model + ", Color: " + color + ", Year: " + year);
    }
}

public class Main {
    public static void main(String args[]) {
        Car myCar = new Car();
        myCar.color = "Red";
        myCar.model = "Toyota";
        myCar.year = 2024;
        myCar.displayDetails();
    }
}
```

Create the car object

Setting Properties

Calling Methods

Constructors

- Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.

- A constructor in Java is a special type of method that is called when an object is instantiated.

Key Points about Constructors:

1. *Name Same as Class*: The name of the constructor must be the same as the name of the class.
2. *No Return Type*: Constructors do not have a return type.
3. *Called Automatically*: When an object is created, the constructor is called automatically.

Constructor : Example

```
1  class Box {
2      double width;
3      double height;
4      double depth;
5
6      // Constructor used when all dimensions are specified
7      Box(double w, double h, double d) {
8          width = w;
9          height = h;
10         depth = d;
11     }
12
13     // Constructor used when no dimensions are specified
14     Box() {
15         width = -1;   // Use -1 to indicate an uninitialized box
16         height = -1;
17         depth = -1;
18     }
19
20     // Constructor used when cube is created
21     Box(double len) {
22         width = height = depth = len;
23     }
24
25     // Method to compute and return the volume
26     double volume() {
27         return width * height * depth;
28     }
29 }
```

Constructor : Example

```
1   class Box {
2       double width;
3       double height;
4       double depth;
5
6       // Constructor used when all dimensions are specified
7       Box(double w, double h, double d) {
8           width = w;
9           height = h;
10          depth = d;
11      }
12
13      // Constructor used when no dimensions are specified
14      Box() {
15          width = -1;   // Use -1 to indicate an uninitialized box
16          height = -1;
17          depth = -1;
18      }
19
20      // Constructor used when cube is created
21      Box(double len) {
22          width = height = depth = len;
23      }
24
25      // Method to compute and return the volume
26      double volume() {
27          return width * height * depth;
28      }
29  }
```

Class name

Parameterized Constructor

Default Constructor

Parameterized Constructor

Constructor : Example

Contd.

```java
public class Main {
    public static void main(String args[]) {
        // Create boxes using the various constructors
        Box myBox1 = new Box(10, 20, 15); // Box with specified dimensions
        Box myBox2 = new Box();            // Box with default dimensions
        Box myCube = new Box(7);           // Cube with side length 7

        double vol;

        // Get volume of first box
        vol = myBox1.volume();
        System.out.println("Volume of myBox1 is " + vol);

        // Get volume of second box
        vol = myBox2.volume();
        System.out.println("Volume of myBox2 is " + vol);

        // Get volume of cube
        vol = myCube.volume();
        System.out.println("Volume of myCube is " + vol);
    }
}
```

- When you do not explicitly define a constructor for a class, then java creates a default constructor for the class
- The default constructor automatically initializes all the variables to zero.

The "`this`" Keyword in Java

- The "`this`" keyword in Java is a reference variable that refers to the current object.
- It is primarily used to differentiate between instance variables and parameters with the same name, but it has other uses as well.
    - *Referencing Instance Variables*: To differentiate between instance variables and local variables/parameters.
    - *Calling Another Constructor*: To implement constructor chaining within the same class.
    - *Returning the Current Class Instance*: To facilitate method chaining.
    - *Passing the Current Class Instance as a Parameter*: To pass the current object to methods or constructors.

The "`this`" Keyword: Referencing Instance Variables

When local variables (parameters) and instance variables have the same name, "`this`" is used to refer to the instance variables.

```java
1  class Box {
2      double width;
3      double height;
4      double depth;
5
6      // Constructor with parameters having the same names as instance variables
7      Box(double width, double height, double depth) {
8          this.width = width;   // `this.width` refers to the instance variable
9          this.height = height; // `this.height` refers to the instance variable
10         this.depth = depth;   // `this.depth` refers to the instance variable
11     }
12
13     void displayDimensions() {
14         System.out.println("Width: " + this.width + ", Height: " + this.height + ", Depth: " + this.depth);
15     }
16 }
17
18 public class Main {
19     public static void main(String args[]) {
20         Box myBox = new Box(10, 20, 15);
21         myBox.displayDimensions();
22     }
23 }
```

Garbage Collection in Java

- Garbage collection in Java is the process by which the Java Virtual Machine (JVM) automatically identifies and discards objects that are no longer needed in order to reclaim and reuse their memory.

- Key Concepts
    - *Automatic Memory Management*: Java handles memory allocation and deallocation automatically. Programmers do not need to explicitly free memory as in languages like C or C++.
    - *Garbage Collector*: The garbage collector is a part of the JVM that performs garbage collection. It runs in the background, identifying objects that are no longer reachable and freeing their

"`finalize()`" Method

Provides a mechanism for cleanup before an object is garbage collected, but it is not reliable and has been deprecated in Java 9.

Method Overloading

- Method overloading is a feature in Java that allows a class to have **more than one method with the same name**, provided their **parameter lists are different**.

- This allows methods to perform similar tasks with different inputs, enhancing code readability and reusability.

- Key Points
  - *Same Name*: All overloaded methods must have the same name.
  - *Different Parameter Lists*: Overloaded methods must have different parameter lists (different number of parameters, different types of parameters, or both).
  - *Return Type*: Overloading is not determined by the return type of the method. Methods can have the same return type or different return types.

Method Overloading : Example

```java
class MathOperations {

    // Method to add two integers
    int add(int a, int b) {
        return a + b;
    }

    // Overloaded method to add three integers
    int add(int a, int b, int c) {
        return a + b + c;
    }

    // Overloaded method to add two double values
    double add(double a, double b) {
        return a + b;
    }

    // Overloaded method to add three double values
    double add(double a, double b, double c) {
        return a + b + c;
    }
}

public class Main {
    public static void main(String[] args) {
        MathOperations math = new MathOperations();

        // Calling overloaded methods
        System.out.println("Sum of 2 integers: " + math.add(10, 20));
        System.out.println("Sum of 3 integers: " + math.add(10, 20, 30));
        System.out.println("Sum of 2 doubles: " + math.add(10.5, 20.5));
        System.out.println("Sum of 3 doubles: " + math.add(10.5, 20.5, 30.5));
    }
}
```