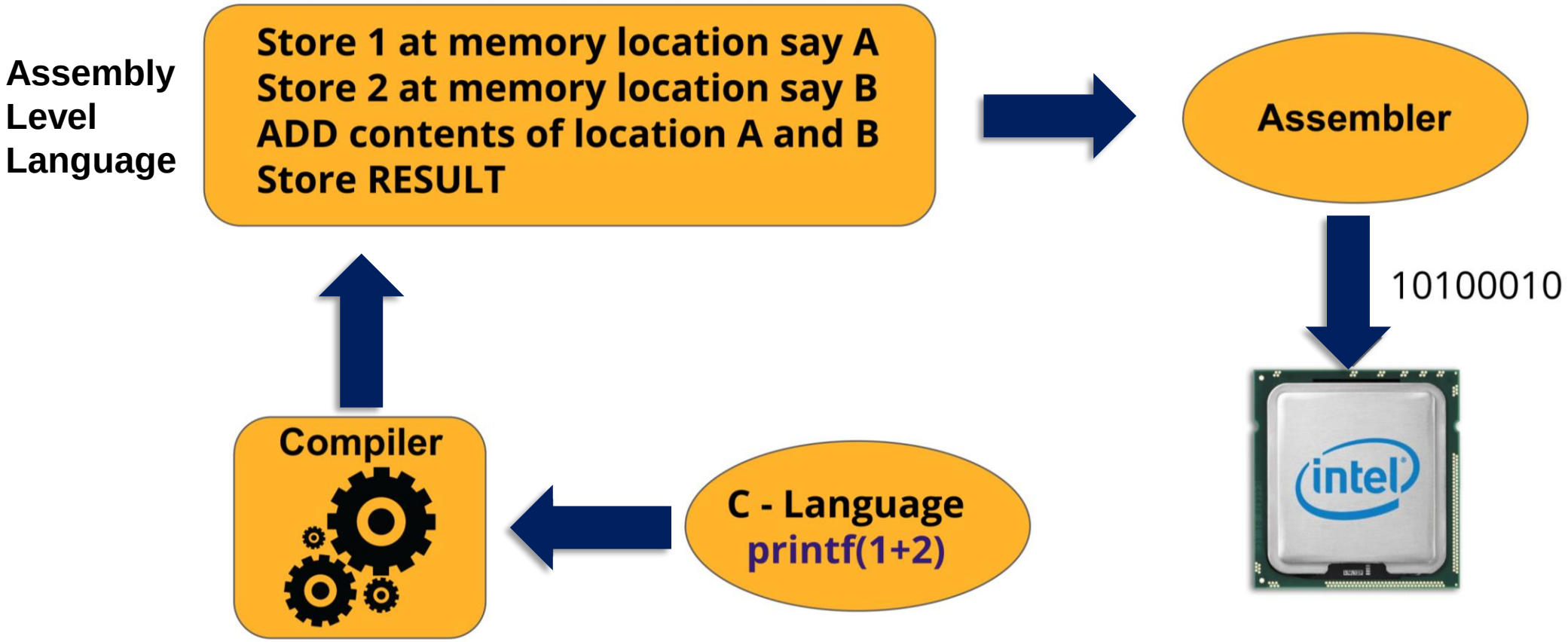
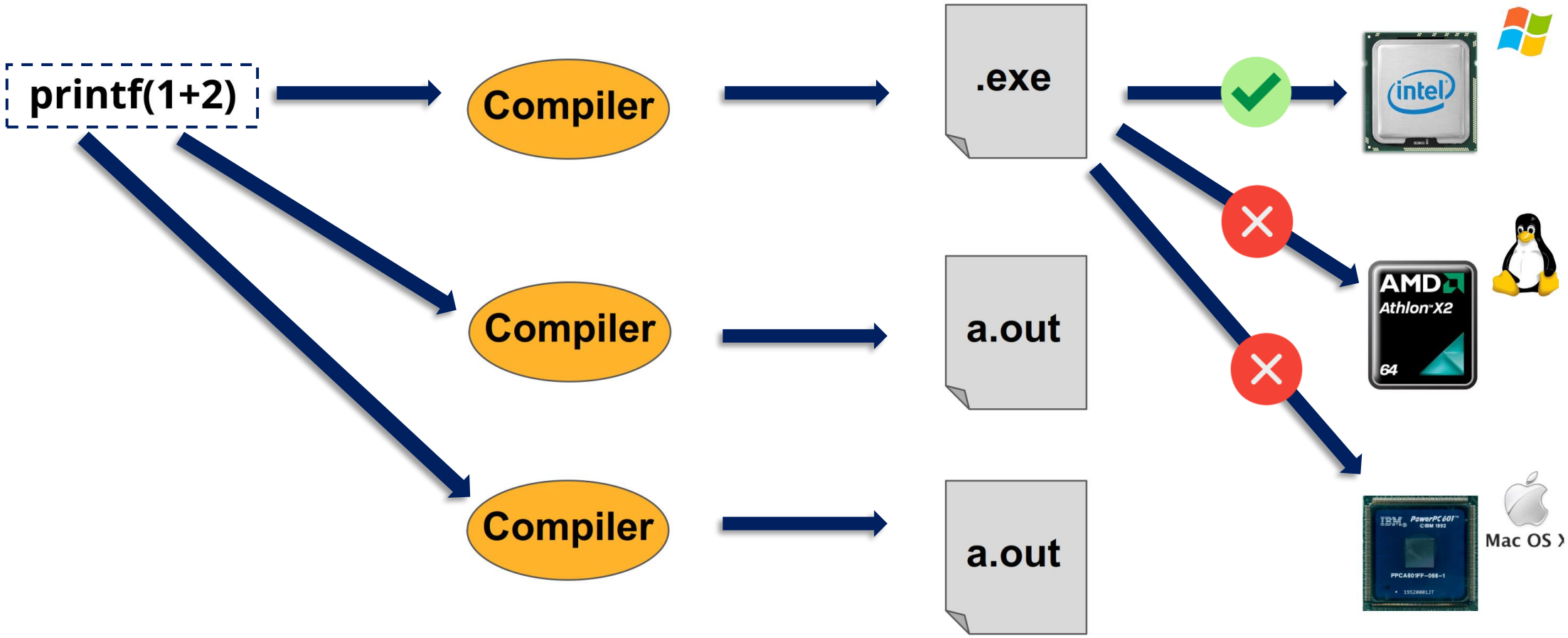
Object Oriented Programming

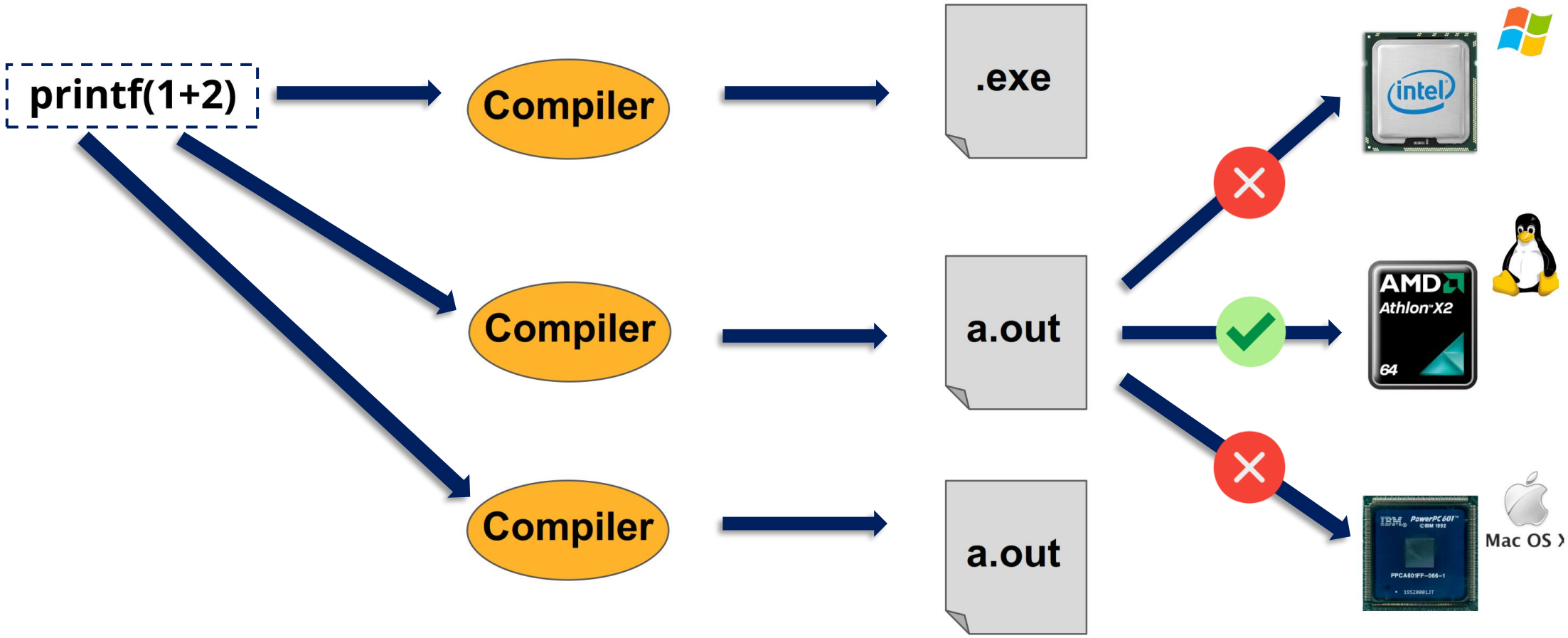
Asif Iqbal Middya
Assistant Professor
Dept of CSE, FIEM


Assembly Level Language
Store 1 at memory location say A
Store 2 at memory location say B
ADD contents of location A and B
Store RESULT
Assembler
10100010
intel
Compiler
C - Language
printf(1+2)

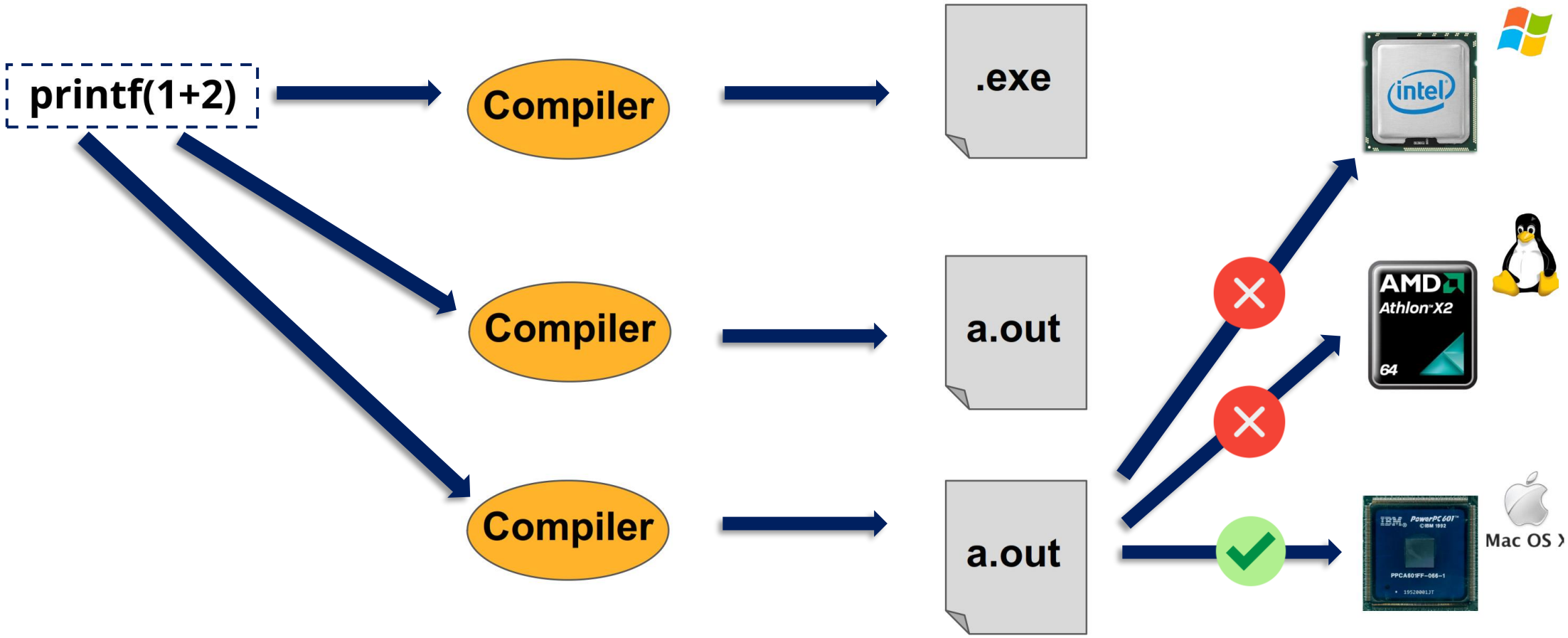
C language (Platform Dependent)



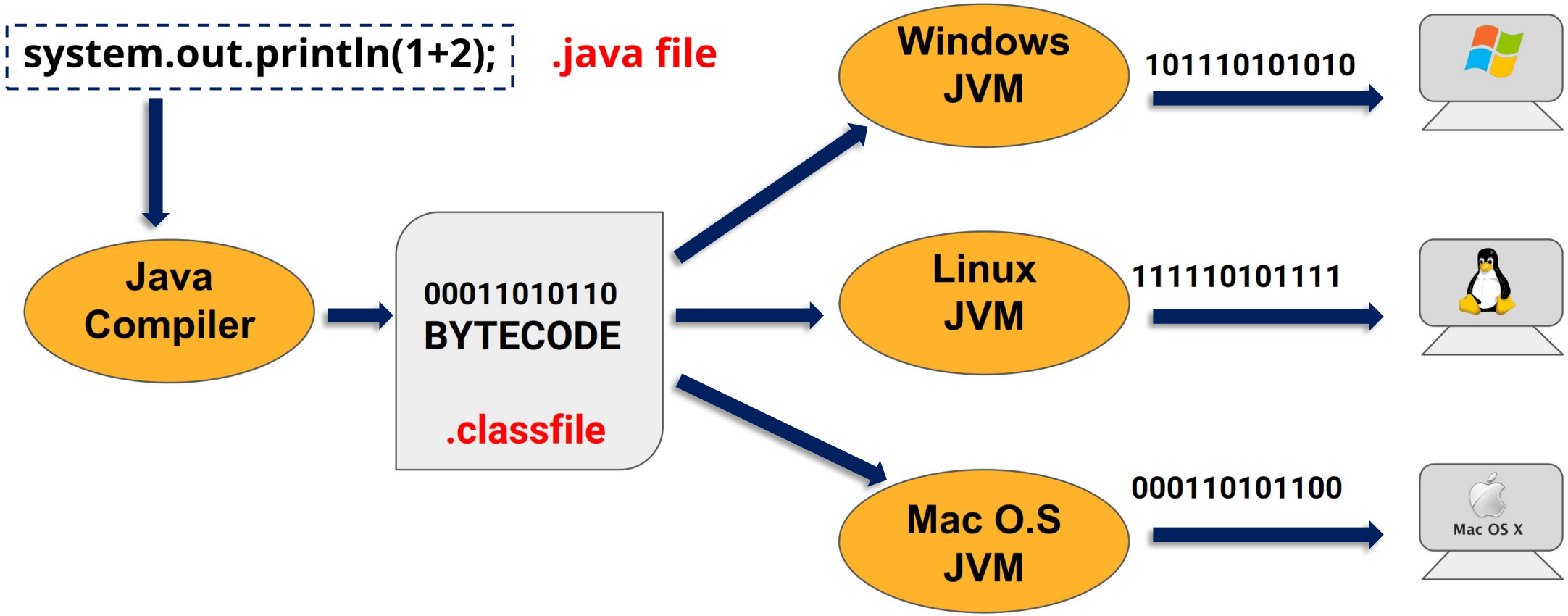
C language (Platform Dependent)



C language (Platform Dependent)

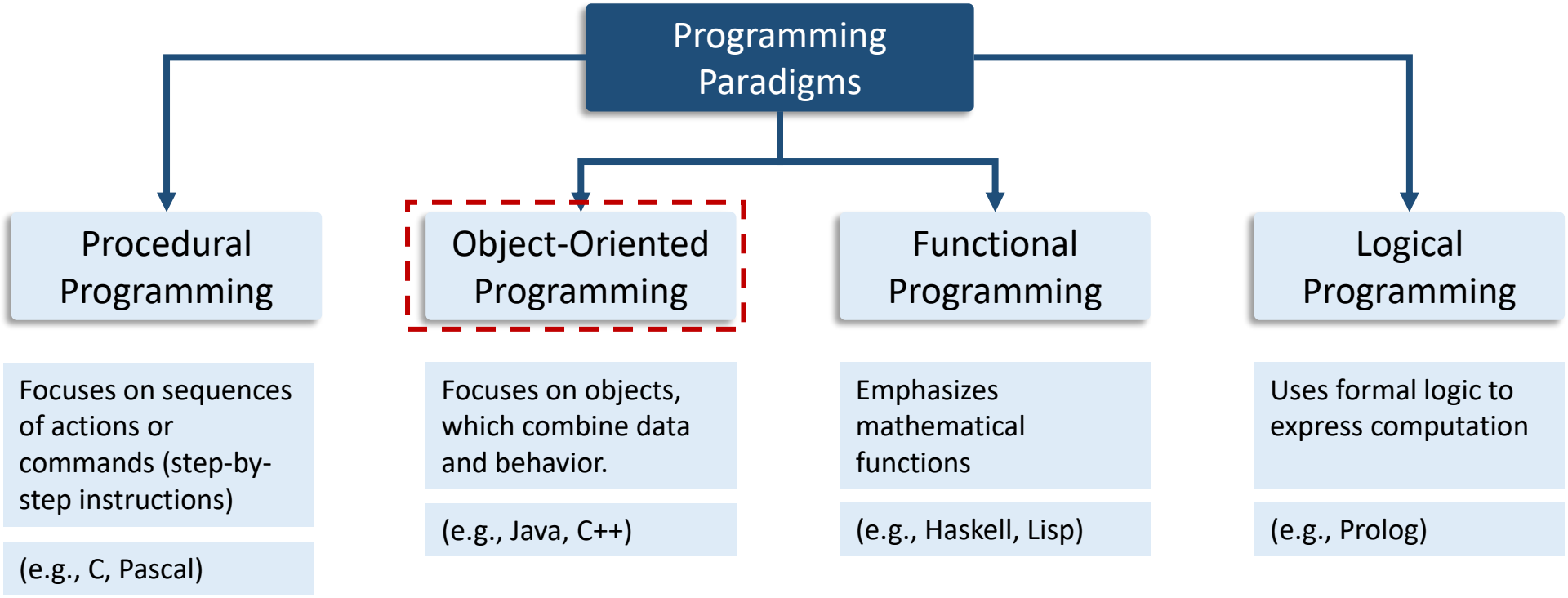


Java (Platform Independent)



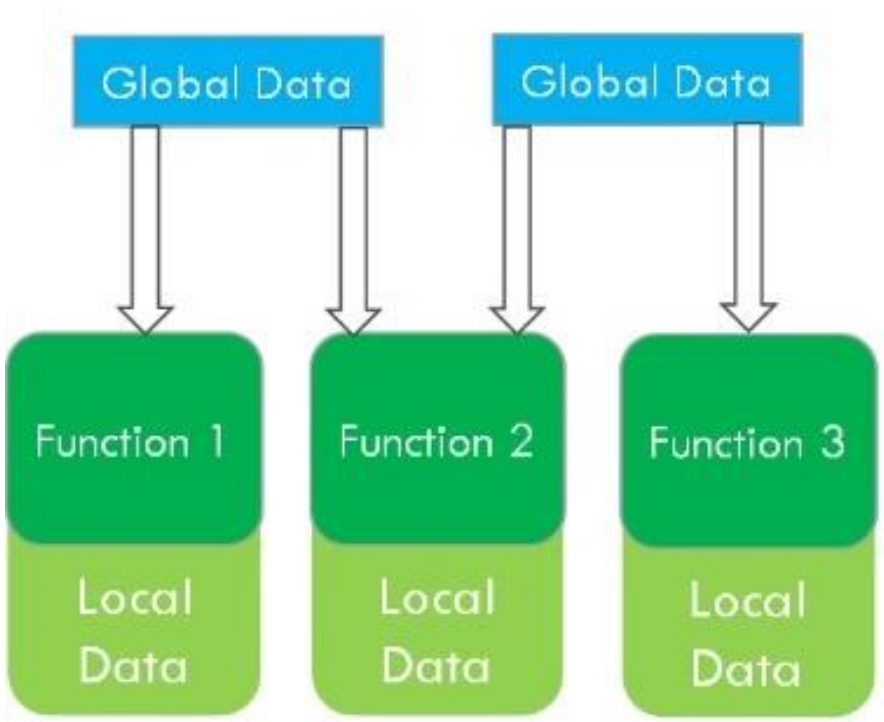
Programming Paradigms

A programming paradigm is a style or way of programming.

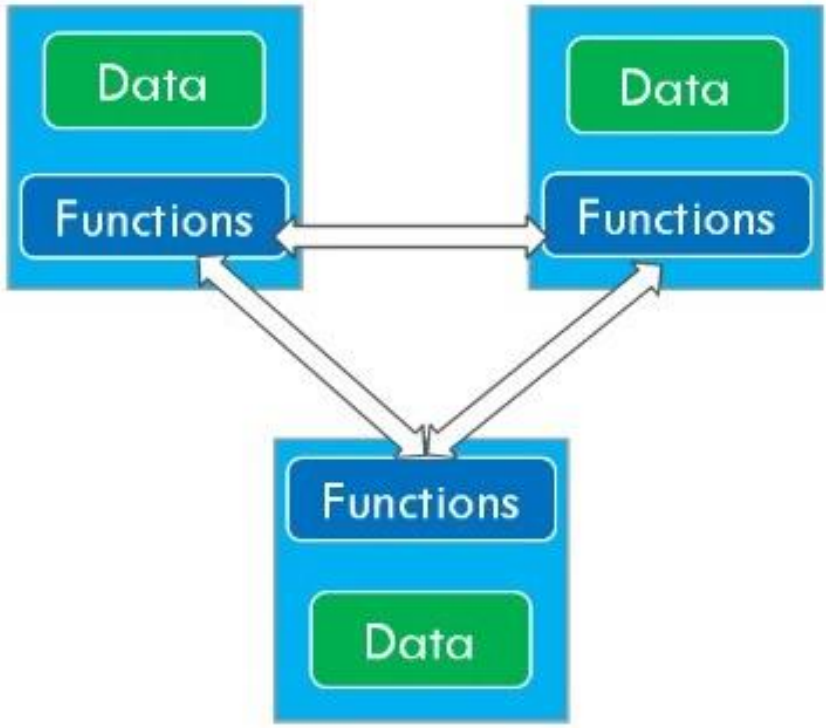


Procedural Programming Vs Object Oriented Programming

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

Object

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

Example: Design of a Car

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

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

```
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

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

Simple Java Program Example: Printing “Hello, World!”

Print statement

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

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

```
1
2
3
4  public class ClassName {
5      public static void main(String args[]) {
6
7
8
9
10
11     }
12 }
13
```

You may have to write some import statement(s) here

Write the code here

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
import java.util.Scanner;

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

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your age: ");
        int age = sc.nextInt();

        System.out.println("Age: " + age);

        sc.close();
    }
}
```

- 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
import java.util.Scanner;

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

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your height in meters: ");
        float height = sc.nextFloat();

        System.out.println("Height: " + height + " meters");

        sc.close();
    }
}
```

- 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

Object

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

Example: Design of a Car

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

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

```java
class Box {
    double width;
    double height;
    double depth;

    // Constructor used when all dimensions are specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    // Constructor used when no dimensions are specified
    Box() {
        width = -1;  // Use -1 to indicate an uninitialized box
        height = -1;
        depth = -1;
    }

    // Constructor used when cube is created
    Box(double len) {
        width = height = depth = len;
    }

    // Method to compute and return the volume
    double volume() {
        return width * height * depth;
    }
}
```

Constructor : Example

```java
class Box {
    double width;
    double height;
    double depth;

    // Constructor used when all dimensions are specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    // Constructor used when no dimensions are specified
    Box() {
        width = -1;  // Use -1 to indicate an uninitialized box
        height = -1;
        depth = -1;
    }

    // Constructor used when cube is created
    Box(double len) {
        width = height = depth = len;
    }

    // Method to compute and return the volume
    double volume() {
        return width * height * depth;
    }
}
```

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
class Box {
    double width;
    double height;
    double depth;

    // Constructor with parameters having the same names as instance variables
    Box(double width, double height, double depth) {
        this.width = width;   // `this.width` refers to the instance variable
        this.height = height; // `this.height` refers to the instance variable
        this.depth = depth;   // `this.depth` refers to the instance variable
    }

    void displayDimensions() {
        System.out.println("Width: " + this.width + ", Height: " + this.height + ", Depth: " + this.depth);
    }
}

public class Main {
    public static void main(String args[]) {
        Box myBox = new Box(10, 20, 15);
        myBox.displayDimensions();
    }
}
```

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));
    }
}
```

Continuous Assessment 1

1. Encapsulation in JAVA (Roll 104 - 109)
2. Abstraction in JAVA (Roll 110 - 115)
3. Polymorphism in JAVA (Roll 116 - 121)
4. Object Oriented Paradigm (Roll 122 - 150)
5. Inheritance in JAVA (Roll 151 - 156)
6. Wrapper Class in JAVA (Roll 157 – 160, 124YL, 162C)
7. ADT (Roll 161C, L1 – L5)
8. Method Overloading vs Method Overriding (Roll L6 – L12)

4 < = No of Slides < = 8

The `static` Keyword

- When a member is declared static, it belongs to the class rather than instances of the class.
- It can be applied to variables, methods, blocks, and nested classes.

Static Variables

A static variable is *shared among all instances / objects* of a class. It is a *global variable*.

```java
class Counter {
    static int count = 0; // static variable

    Counter() {
        count++; // incrementing the value of static variable
        System.out.println(count);
    }
}

public class TestStaticVariable {
    public static void main(String args[]) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();
        Counter c3 = new Counter();
    }
}
```

Output

```
1
2
3
```

Explanation:

- The `count` variable is static, so it is shared among all instances of the Counter class.
- When a new Counter object is created, the static variable `count` is incremented.
- Each time a new Counter instance is created, it reflects the incremented value.

Static Methods

Static methods can be called without creating an instance of the class. They can access static variables and other static methods directly.

```
1  class MathOperations {
2      // static method
3      static int add(int a, int b) {
4          return a + b;
5      }
6
7      // non-static method
8      int multiply(int a, int b) {
9          return a * b;
10     }
11 }
12
13 public class TestStaticMethod {
14     public static void main(String args[]) {
15         // calling static method
16         int result = MathOperations.add(10, 20);
17         System.out.println("Sum: " + result);
18
19         // calling non-static method
20         MathOperations obj = new MathOperations();
21         int mulResult = obj.multiply(10, 20);
22         System.out.println("Product: " + mulResult);
23     }
24 }
```

Output

```
Sum: 30
Product: 200
```

Explanation:

- The static method `add` can be called directly using the class name without creating an instance of the `MathOperations` class (see line 16).
- The non-static method `multiply` requires an instance/object of the class to be called.

What will be the output?

```java
class A {
    static int x = 10;

    static void display() {
        System.out.println("Class A: " + x);
    }
}

class B {
    public static void main(String args[]) {
        A.display();
        A.x = 20;
        A.display();
    }
}
```

Is there an error in the code?

```java
class A {
    static int x = 10;
    int y = -10;

    static void display() {
        System.out.println("Class A: " + x);
    }
}

class B {
    public static void main(String args[]) {
        A.display();
        A.x = 20;
        A.display();
        A.y = -20;
    }
}
```

Inheritance in Java

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that *allows one class to inherit properties and behaviors (fields and methods) from another class*.

Inheritance provides a way to *create a new class* (called the subclass or derived class) *based on an existing class* (called the superclass or base class).

The subclass inherits the fields and methods of the superclass, allowing code reuse and the creation of a hierarchical relationship between classes.

In Java, inheritance is implemented using the extends keyword

Superclass (Base Class / parent): The class whose properties are inherited.
Subclass (Derived Class / child): The class that inherits the properties of another class.

Syntax of Inheritance

Super Class

```
class Superclass {
    // Properies (Attributes / fields)
    // Methods (Behaviour)
}
```

```
class Subclass extends Superclass {
    // Additional Properies
    // Additional Methods
}
```

Sub Class

Super Class

```java
// Superclass
class Vehicle {
    String brand;
    int year;

    // Constructor
    Vehicle(String brand, int year) {
        this.brand = brand;
        this.year = year;
    }

    // Method
    void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Year: " + year);
    }
}
```

Sub Class

```java
// Subclass
class Car extends Vehicle {
    String model;

    // Constructor
    Car(String brand, int year, String model) {
        super(brand, year); // Call the constructor of the superclass
        this.model = model;
    }

    // Method
    void displayCarInfo() {
        displayInfo(); // Call the superclass method
        System.out.println("Model: " + model);
    }
}
```

```java
// Main class
public class Main {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", 2022, "Corolla");
        myCar.displayCarInfo();
    }
}
```

Main Class

Super Class

```java
// Superclass
class Vehicle {
    String brand;
    int year;

    // Constructor
    Vehicle(String brand, int year) {
        this.brand = brand;
        this.year = year;
    }

    // Method
    void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Year: " + year);
    }
}
```

Sub Class

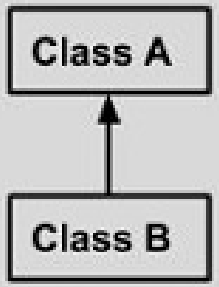
```java
// Subclass
class Car extends Vehicle {
    String model;

    // Constructor
    Car(String brand, int year, String model) {
        super(brand, year); // Call the constructor of the superclass
        this.model = model;
    }

    // Method
    void displayCarInfo() {
        displayInfo(); // Call the superclass method
        System.out.println("Model: " + model);
    }
}
```

```java
// Main class
public class Main {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", 2022, "Corolla");
        myCar.displayCarInfo();
    }
}
```

Main Class

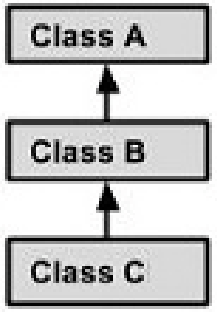
Types of Inheritance

Single Inheritance



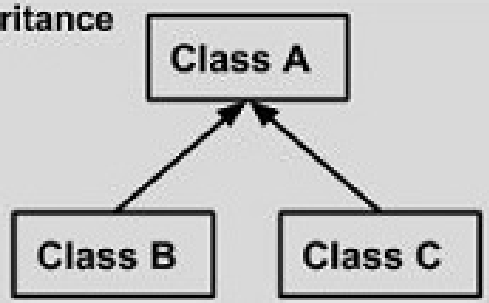

```
public class A {
   .......
}
public class B extends A {
   .........
}
```

Multi Level Inheritance



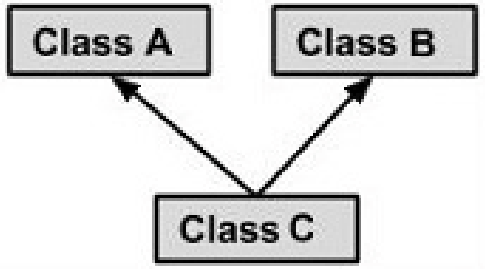

```
public class A { ...................}

public class B extends A {.....................}

public class C extends  B {...................... }
```

Hierarchical Inheritance




```
public class A { ...................}

public class B extends A {.....................}

public class C extends A {...................... }
```

Multiple Inheritance




```
public class A { ...................}

public class B {.....................}

public class C extends  A,B {
   ......................
} // Java does not support mutiple Inheritance
```

Method Overriding

Method Overriding occurs when a subclass (child class) provides a specific implementation for a method that is already defined in its superclass (parent class).

Key points

- The method in the subclass must have the *same name, return type, and parameters* as the method in the superclass.
- Method overriding *allows a subclass to provide a specific implementation* that is different from the one provided by its superclass.
- The overridden method in the subclass will be *called at runtime* based on the object being referenced.

Method Overriding

Super Class

```java
class Animal {
    // Method in the superclass
    void sound() {
        System.out.println("The animal makes a sound");
    }
}
```

Sub Class

```java
class Dog extends Animal {
    // Overriding the sound method in the subclass
    @Override
    void sound() {
        System.out.println("The dog barks");
    }
}
```

Main Class

```java
public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();  // Create an Animal object
        Animal myDog = new Dog();        // Create a Dog object

        myAnimal.sound();  // Outputs: The animal makes a sound
        myDog.sound();     // Outputs: The dog barks
    }
}
```

Super Class

```java
class Animal {
    // Method in the superclass
    void sound() {
        System.out.println("The animal makes a sound");
    }
}
```

Sub Class

```java
class Dog extends Animal {
    // Overriding the sound method in the subclass
    @Override
    void sound() {
        System.out.println("The dog barks");
    }
}
```

Main Class

```java
public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();  // Create an Animal object
        Animal myDog = new Dog();        // Create a Dog object

        myAnimal.sound();  // Outputs: The animal makes a sound
        myDog.sound();     // Outputs: The dog barks
    }
}
```

Both methods having same name, return type, and parameters

Difference Between Method Overloading and Method Overriding

Feature	Method Overloading	Method Overriding
Definition	Multiple methods in the same class with the same name but different parameters.	A method in a subclass has the same name, return type, and parameters as a method in its superclass.
Parameters	Must be different (either in number, type, or both).	Must be the same as the method in the superclass.
Return Type	Can be different.	Must be the same with the return type of the superclass method.
Polymorphism	Compile-time polymorphism (static binding).	Runtime polymorphism (dynamic binding).
Purpose	To increase the readability of the program by defining multiple behaviors for a method.	To provide a specific implementation of a method already defined in a superclass.

Problem Statement

Create a class `Employee` with attributes `name` and `salary`. Then create a subclass `Manager` that adds an additional attribute `bonus`. Write a method `displayDetails()` in both classes to display the details of the employee and manager. In the main class, create an object of `Manager`, set its attributes, and display its details.

Solution

```java
// Superclass
class Employee {
    String name;
    double salary;

    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
    }
}

// Subclass Manager
class Manager extends Employee {
    double bonus;

    Manager(String name, double salary, double bonus) {
        super(name, salary);  // Calling the superclass constructor
        this.bonus = bonus;
    }

    @Override
    void displayDetails() {
        super.displayDetails();  // Calling the superclass method
        System.out.println("Bonus: " + bonus);
        System.out.println("Total Compensation: " + (salary + bonus));
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        Manager mgr = new Manager("Alice", 75000, 15000);
        mgr.displayDetails();  // Displaying Manager's details
    }
}
```