# TheKiranAcademy

## Solve : Any 3
## Time : 1 hour

## **Problem Statement: Implementing Polymorphism in Java**

****Background:****

You have been given the task of designing and implementing a Java program that showcases the concept of polymorphism. The program should emphasize the ability of different objects to respond to the same method calls in distinct ways based on their individual implementations.

****Problem Description:****

Develop a Java program that demonstrates polymorphism by creating a set of shapes and performing various calculations and operations on them using a common interface.

****Requirements:****

1. ****Interface and Classes:****

Define an interface called `Shape` with methods like `calculateArea()` and `calculatePerimeter()`. Implement this interface in the following classes:

- `Circle`: A class representing a circle with attributes like radius.

- `Rectangle`: A class representing a rectangle with attributes like width and height.

- `Triangle`: A class representing a triangle with attributes like base and height.

2. ****Polymorphism:****

Create an array or collection of `Shape` objects that includes instances of `Circle`, `Rectangle`, and `Triangle`. Demonstrate polymorphism by iterating through this collection and calling the `calculateArea()` and `calculatePerimeter()` methods on each object, even though they are of different types.

# **Problem Statement: Implementing Interfaces in Java for Vehicles**

**Background:**

You have been given a task to create a Java program that helps understand interfaces. Think of interfaces as a set of promises that classes make to have certain behaviors. The program should show how different vehicles can share common actions using interfaces.

**Problem Description:**

Design a Java program that demonstrates how interfaces work by creating different types of vehicles and making them start, accelerate, and brake.

**Requirements:**

1. **Interface:**

   Create a rulebook named `Vehicle` that lists what actions a vehicle must promise to perform:

   - `start()`: Vehicle must know how to start.

   - `accelerate()`: Vehicle must know how to speed up.

   - `brake()`: Vehicle must know how to stop.

2. **Vehicle Classes:**

   Make different vehicles (like cars and motorcycles) follow the `Vehicle` rulebook:

   - `Car`: A car should know how to start its engine, accelerate, and apply brakes.

   - `Motorcycle`: A motorcycle should know how to start its engine, accelerate, and apply brakes.

3. **User Interaction:**

   Create a simple way for users to "play" with these vehicles. They should be able to start, speed up, and stop each vehicle.

4. **Polymorphism:**

   Show how vehicles can be treated in a similar way, regardless of whether they are cars or motorcycles. Use the rulebook to ensure that all vehicles can perform the same actions.

# **Problem Statement: Exploring Constructors in Java**

**Background:**

You have been assigned a task to create a Java program that delves into the concept of constructors. Constructors are special methods used to initialize objects when they are created. The program should demonstrate how constructors work and how they can be used to set up object states.

**Problem Description:**

Develop a Java program that illustrates the use of constructors by modeling a simple student information system.

**Requirements:**

1. **Class Definition:**

   Define a class named `Student` with the following attributes:

   - `name`: A string representing the student's name.

   - `age`: An integer representing the student's age.

   - `studentId`: A unique identifier for each student.

2. **Constructors:**

   Implement the following constructors for the `Student` class:

   - A default constructor that initializes the student's name as "Unknown", age as 0, and assigns a unique student ID.

   - A parameterized constructor that accepts the student's name and age and assigns a unique student ID.

   - Overload the parameterized constructor to also accept the student's age.

3. **User Interaction:**

   Create a simple user interface where users can create instances of the `Student` class using different constructors. Allow users to input the student's name and age, and display the student's details.

# **Problem Statement: Building an Inheritance-based Employee System in Java**

**Background:**

You have been assigned the task of designing a Java program that explores the concept of inheritance in the context of an employee management system. Inheritance allows you to create specialized classes that inherit attributes and methods from a base class. The program should showcase how inheritance can be used to model different types of employees with shared and distinct characteristics.

**Problem Description:**

Develop a Java program that demonstrates inheritance by creating a hierarchy of employees in an organization.

**Requirements:**

1. **Employee Classes:**

   Create a base class named `Employee` with the following attributes:

   - `id`: An integer representing the employee's unique identifier.

   - `name`: A string representing the employee's name.

   Implement methods for getting and setting these attributes.

2. **Derived Classes:**

   Implement the following derived classes that inherit from the `Employee` class:

   - `Manager`: A class representing managers. Add an attribute `department` to store the department they manage.

   - `Developer`: A class representing developers. Add an attribute `programmingLanguage` to store the primary programming language they use.

3. **User Interaction:**

   Create a simple user interface where users can create instances of the `Manager` and `Developer` classes. Allow users to input attributes like name, id, department (for managers), and programming language (for developers). Display the details of each employee and

4. **target**

   Illustrate the hierarchy of the employee classes, emphasizing the parent-child relationship between `Employee`, `Manager`, and `Developer`.