FirstBitSolutions.com
...Learn IT Bit by Bit...

Assignment: Implementing Inheritance

Objective

To understand and implement inheritance in Java by modelling real-world relationships using IS-A hierarchies and class reuse.

Problem Statement

You are given multiple class hierarchies representing real-world scenarios.
Each hierarchy consists of one superclass and one or more subclasses.
Your task is to:

1. Implement the given hierarchies using inheritance in Java.
2. Create additional hierarchies following the same inheritance principles.

Given Hierarchies

1. Employee Hierarchy

Employee

- id
- name
- salary

Admin is-a Employee

- allowance

SalesManager is-a Employee

- incentive
- target

HR is-a Employee

- commission

2. Vehicle Hierarchy

Vehicle

- vehicleNumber
- model
- companyName
- noOfWheels
- price

Bike is-a Vehicle

- noOfStands
- noOfHelmets
- bikeCategory

Car is-a Vehicle

- hasPowerSteering
- driveMode
- parkingAssistSensors

Bus is-a Vehicle

- passengerCapacity
- standingCapacity

3. Shape Hierarchy

Shape

- area

Circle is-a Shape

- radius

Triangle is-a Shape

- base
- height

Rectangle is-a Shape

- length
- breadth

4. Player Hierarchy

Player

- name
- age
- country
- matchesPlayed
- jerseyNumber

CricketPlayer is-a Player

- totalRuns
- totalWickets
- battingStyle
- bowlingStyle

FootballPlayer is-a Player

- totalGoals
- playingPosition

5. Artist Hierarchy

Artist

- name
- age

Painter is-a Artist

- paintingStyle
- mediumUsed
- numberOfPaintings

Musician is-a Artist

- instrument
- musicGenre
- numberOfAlbums

Actor is-a Artist

- filmIndustry
- numberOfMovies

Student Tasks

1. Implement **all the given hierarchies** using Java inheritance.
2. Create **any five (5) additional hierarchies** of your own.
3. Each new hierarchy must include:
    - One superclass
    - At least two subclass
4. Ensure all relationships follow proper **IS-A logic**.

Important Rules

• Do **not** modify the given hierarchies
• Do **not** duplicate attributes between superclass and subclass
• Use appropriate access modifiers
• Follow proper **Java naming conventions**
• Write clean, readable, and well-structured code