POLYMORPHISM PRACTICE PROBLEMS(Any Six)(1;2;3;5;6;8 done)

# PROBLEM 1: Food Delivery App

**Concept:** Method Overloading

You're creating a food ordering system. Design a class that can calculate delivery charges in different ways:

* Basic delivery (just distance)
* Premium delivery (distance + priority fee)
* Group delivery (distance + number of orders discount)
* Festival special (distance + discount percentage + free delivery over certain amount)

Each calculation should show a different message about the delivery cost breakdown.

**Hint:** Same method name, different parameters - let Java pick the right one!

Sol:

class DeliveryCharges {

void calculate(int distance) {

System.out.println("Basic Delivery: Distance = " + distance + "km, Cost = " + (distance \* 10));

}

void calculate(int distance, int priorityFee) {

System.out.println("Premium Delivery: Distance = " + distance + "km, Priority Fee = " + priorityFee +

", Cost = " + (distance \* 10 + priorityFee));

}

void calculate(int distance, int orders, double discountPerOrder) {

double discount = orders \* discountPerOrder;

double cost = distance \* 10 - discount;

System.out.println("Group Delivery: Distance = " + distance + "km, Orders = " + orders +

", Cost = " + cost);

}

void calculate(int distance, double discountPercent, double freeLimit) {

double cost = distance \* 10;

if (cost > freeLimit) {

System.out.println("Festival Special: Free Delivery!");

} else {

cost = cost - (cost \* discountPercent / 100);

System.out.println("Festival Special: Distance = " + distance + "km, Cost = " + cost);

}

}

}

public class FoodDeliveryApp {

public static void main(String[] args) {

DeliveryCharges d = new DeliveryCharges();

d.calculate(5);

d.calculate(5, 20);

d.calculate(5, 3, 2.5);

d.calculate(5, 10, 100);

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

# PROBLEM 2: Social Media Feed

**Concept:** Method Overriding

Build a social media post system where different platforms display posts differently:

* Instagram posts show with hashtags and likes
* Twitter posts show with character count and retweets
* LinkedIn posts show with professional formatting and connections

All posts share common info (author, content, time) but display uniquely for each platform.

**Hint:** Parent class defines the structure, child classes customize the display!

Sol:

class Post {

String author, content, time;

Post(String a, String c, String t) {

author = a; content = c; time = t;

}

void display() {

System.out.println(author + ": " + content + " (" + time + ")");

}

}

class InstagramPost extends Post {

int likes;

InstagramPost(String a, String c, String t, int l) { super(a,c,t); likes = l; }

void display() {

System.out.println("[Instagram] " + author + ": " + content + " ❤️ Likes: " + likes);

}

}

class TwitterPost extends Post {

int retweets;

TwitterPost(String a, String c, String t, int r) { super(a,c,t); retweets = r; }

void display() {

System.out.println("[Twitter] " + author + ": " + content + " (" + content.length() + " chars) 🔁 Retweets: " + retweets);

}

}

class LinkedInPost extends Post {

int connections;

LinkedInPost(String a, String c, String t, int con) { super(a,c,t); connections = con; }

void display() {

System.out.println("[LinkedIn] " + author + " shared: " + content + " 👥 Connections: " + connections);

}

}

public class SocialMediaFeed {

public static void main(String[] args) {

Post p1 = new InstagramPost("Alice", "Enjoying vacation!", "10AM", 120);

Post p2 = new TwitterPost("Bob", "Java is awesome!", "11AM", 50);

Post p3 = new LinkedInPost("Carol", "Looking for opportunities!", "12PM", 300);

p1.display();

p2.display();

p3.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

# PROBLEM 3: Gaming Character System

**Concept:** Dynamic Method Dispatch

Create a battle system with different character types:

* Warriors attack with weapons and have high defense
* Mages cast spells and use mana
* Archers shoot arrows with long-range damage

Design it so the same "attack" command produces different results based on the character type, even when stored in a mixed army array.

**Hint:** Same reference, different objects - let runtime decide the behavior!

Sol:

abstract class Character {

abstract void attack();

}

class Warrior extends Character {

void attack() { System.out.println("Warrior attacks with sword and shields!"); }

}

class Mage extends Character {

void attack() { System.out.println("Mage casts a powerful fireball!"); }

}

class Archer extends Character {

void attack() { System.out.println("Archer shoots an arrow from long range!"); }

}

public class GameeBattle {

public static void main(String[] args) {

Character[] army = { new Warrior(), new Mage(), new Archer() };

for (Character c : army) {

c.attack(); // runtime decision

}

}

}

A screenshot of a computer

AI-generated content may be incorrect.

# PROBLEM 4: University Library System

**Concept:** Upcasting

Design a library system with different types of users:

* Students can borrow books and access computers
* Faculty can reserve books and access research databases
* Guests can only browse books

Create a general "LibraryUser" system that can handle any user type for common operations like entry logging and basic info display.

**Hint:** Think bigger picture - store specialists as generalists safely!

# PROBLEM 5: Movie Streaming Platform

**Concept:** Downcasting

Build a streaming service that handles different content types:

* Movies have ratings, duration, and subtitle options
* TV Series have seasons, episodes, and next episode suggestions
* Documentaries have educational tags and related content

Sometimes you need to access specific features based on what the user is actually watching.

**Hint:** Go from general to specific - but be careful, not everything is what it seems!

Sol:

class Content {

void show() { System.out.println("General Content"); }

}

class Movie extends Content {

void movieInfo() { System.out.println("Movie: Rating PG, Duration 2hrs, Subtitles available"); }

}

class TVSeries extends Content {

void seriesInfo() { System.out.println("TV Series: Season 2, Episode 5, Next episode suggested"); }

}

class Documentary extends Content {

void docInfo() { System.out.println("Documentary: Educational tags, Related content"); }

}

public class StreamingPlatform {

public static void main(String[] args) {

Content c = new TVSeries(); // general type

c.show();

if (c instanceof TVSeries) {

TVSeries t = (TVSeries) c; // downcasting

t.seriesInfo();

}

}

}

A screen shot of a computer

AI-generated content may be incorrect.

# PROBLEM 6: Smart Campus IoT System

**Concept:** Safe Downcasting with instanceof

Create a campus management system with different smart devices:

* Smart classrooms control lighting, AC, and projectors
* Smart labs manage equipment and safety systems
* Smart libraries track occupancy and book availability

Process mixed device collections safely, applying the right controls to each device type without crashing.

**Hint:** Check first, cast second - safety matters in the real world!

Sol:

class Device { void control() { System.out.println("Generic Device Control"); } }

class SmartClassroom extends Device {

void control() { System.out.println("Classroom: Lights, AC, Projector managed"); }

}

class SmartLab extends Device {

void control() { System.out.println("Lab: Equipment and safety systems managed"); }

}

class SmartLibrary extends Device {

void control() { System.out.println("Library: Occupancy and book availability tracked"); }

}

public class CampusIoT {

public static void main(String[] args) {

Device[] devices = { new SmartClassroom(), new SmartLab(), new SmartLibrary() };

for (Device d : devices) {

if (d instanceof SmartClassroom) ((SmartClassroom) d).control();

else if (d instanceof SmartLab) ((SmartLab) d).control();

else if (d instanceof SmartLibrary) ((SmartLibrary) d).control();

else d.control();

}

}

}

A screen shot of a computer screen

AI-generated content may be incorrect.

# PROBLEM 7: E-Commerce Recommendation Engine

**Concept:** Multiple Polymorphism Integration

Design a product recommendation system handling:

* Electronics with warranty info and tech specs
* Clothing with size charts and style matching
* Books with author details and genre recommendations

The system should recommend products differently, update product info in multiple ways, and safely process mixed product catalogs.

**Hint:** Combine overloading for different updates, overriding for custom recommendations, and safe casting for mixed processing!

# PROBLEM 8: Virtual Pet Simulator

**Concept:** Complete Polymorphism Mastery Create a virtual pet game with different pet types:

* Dogs need walks, play fetch, and bark at strangers
* Cats need grooming, hunt mice, and sleep a lot
* Birds need flying time, sing songs, and learn words Design interaction systems that:
* Handle feeding all pets the same way (inheritance)
* Play with pets differently based on type (overriding)
* Care for pets with multiple care options (overloading)
* Process mixed pet collections safely (casting)

**Hint:** Everything together - inheritance hierarchy, method variations, runtime decisions, and safe type handling!

Sol:

class Pet {

void feed() {

System.out.println("Feeding pet...");

}

void play() {

System.out.println("Generic play with pet...");

}

}

class Dog extends Pet {

@Override

void play() {

System.out.println("Dog plays fetch and barks!");

}

void care(String activity) {

System.out.println("Dog care: " + activity);

}

}

class Cat extends Pet {

@Override

void play() {

System.out.println("Cat hunts mice and sleeps a lot!");

}

void care(int minutes) {

System.out.println("Cat grooming for " + minutes + " minutes");

}

}

class Bird extends Pet {

@Override

void play() {

System.out.println("Bird flies around and sings songs!");

}

void care(boolean teachWords) {

if (teachWords) {

System.out.println("Bird care: Teaching new words");

} else {

System.out.println("Bird care: Normal care");

}

}

}

public class VirtualPetGame {

public static void main(String[] args) {

Pet[] pets = new Pet[3]; // ✅ Correct: Pet[], not Dog[]

pets[0] = new Dog();

pets[1] = new Cat();

pets[2] = new Bird();

for (Pet p : pets) {

p.feed();

p.play();

if (p instanceof Dog) {

((Dog) p).care("Morning walk");

} else if (p instanceof Cat) {

((Cat) p).care(15);

} else if (p instanceof Bird) {

((Bird) p).care(true);

}

System.out.println("------------------------");

}

}

}

