

**POLYMORPHISM PRACTICE PROBLEMS(Any Six)**

**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 FoodDelivery {

public void calculateDelivery(double distance) {

double cost = distance \* 10;

System.out.println("Basic Delivery: Distance = " + distance + " km");

System.out.println("Delivery Charge = ₹" + cost + "\n");

}

public void calculateDelivery(double distance, double priorityFee) {

double cost = distance \* 10 + priorityFee;

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

System.out.println("Delivery Charge = ₹" + cost + "\n");

}

public void calculateDelivery(double distance, int numberOfOrders) {

double cost = distance \* 10;

double discount = numberOfOrders \* 5;

double finalCost = cost - discount;

if (finalCost < 0) finalCost = 0;

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

System.out.println("Base Charge = ₹" + cost + ", Discount = ₹" + discount);

System.out.println("Final Delivery Charge = ₹" + finalCost + "\n");

}

public void calculateDelivery(double distance, double discountPercent, double freeAbove) {

double cost = distance \* 10;

if (cost > freeAbove) {

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

System.out.println("Order above ₹" + freeAbove + " → Free Delivery! 🎉\n");

} else {

double discount = cost \* (discountPercent / 100);

double finalCost = cost - discount;

System.out.println("Festival Special: Distance = " + distance + " km, Discount = " + discountPercent + "%");

System.out.println("Base Charge = ₹" + cost + ", Discount = ₹" + discount);

System.out.println("Final Delivery Charge = ₹" + finalCost + "\n");

}

}

}

public class FoodDeliveryApp {

public static void main(String[] args) {

FoodDelivery fd = new FoodDelivery();

fd.calculateDelivery(5);

fd.calculateDelivery(5, 50.0);

fd.calculateDelivery(10, 3);

fd.calculateDelivery(20, 20.0, 150.0);

}

}

**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! 1

SOL.

class Post {

String author;

String content;

String time;

Post(String author, String content, String time) {

this.author = author;

this.content = content;

this.time = time;

}

public void display() {

System.out.println(author + " posted at " + time);

System.out.println(content);

}

}

class InstagramPost extends Post {

int likes;

String hashtags;

InstagramPost(String author, String content, String time, int likes, String hashtags) {

super(author, content, time);

this.likes = likes;

this.hashtags = hashtags;

}

public void display() {

System.out.println("Instagram Post");

System.out.println(author + " | " + time);

System.out.println(content);

System.out.println("Hashtags: " + hashtags);

System.out.println("Likes: " + likes + "\n");

}

}

class TwitterPost extends Post {

int retweets;

TwitterPost(String author, String content, String time, int retweets) {

super(author, content, time);

this.retweets = retweets;

}

public void display() {

System.out.println("Twitter Post");

System.out.println(author + " | " + time);

System.out.println(content);

System.out.println("Characters: " + content.length());

System.out.println("Retweets: " + retweets + "\n");

}

}

class LinkedInPost extends Post {

int connections;

LinkedInPost(String author, String content, String time, int connections) {

super(author, content, time);

this.connections = connections;

}

public void display() {

System.out.println("LinkedIn Post");

System.out.println("By " + author + " | " + time);

System.out.println("Content: " + content);

System.out.println("Seen by " + connections + " connections\n");

}

}

public class SocialMediaFeed {

public static void main(String[] args) {

Post p1 = new InstagramPost("Alice", "Enjoying vacation!", "10:30 AM", 120, "#travel #fun");

Post p2 = new TwitterPost("Bob", "Learning Java Method Overriding today!", "11:00 AM", 45);

Post p3 = new LinkedInPost("Charlie", "Excited to start a new job role!", "2:00 PM", 300);

p1.display();

p2.display();

p3.display();

}

}



**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.

class Character {

String name;

Character(String name) {

this.name = name;

}

public void attack() {

System.out.println(name + " attacks!");

}

}

class Warrior extends Character {

Warrior(String name) {

super(name);

}

public void attack() {

System.out.println(name + " swings a mighty sword with high defense!");

}

}

class Mage extends Character {

Mage(String name) {

super(name);

}

public void attack() {

System.out.println(name + " casts a powerful spell using mana!");

}

}

class Archer extends Character {

Archer(String name) {

super(name);

}

public void attack() {

System.out.println(name + " shoots a precise arrow from long range!");

}

}

public class BattleSystem {

public static void main(String[] args) {

Character[] army = new Character[3];

army[0] = new Warrior("Thor");

army[1] = new Mage("Merlin");

army[2] = new Archer("Legolas");

for (Character c : army) {

c.attack();

}

}

}

**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!

SOL.

class LibraryUser {

String name;

int id;

LibraryUser(String name, int id) {

this.name = name;

this.id = id;

}

public void displayInfo() {

System.out.println("User: " + name + " | ID: " + id);

}

public void accessLibrary() {

System.out.println(name + " is accessing the library.");

}

}

class Student extends LibraryUser {

Student(String name, int id) {

super(name, id);

}

public void borrowBooks() {

System.out.println(name + " borrowed a book.");

}

public void accessComputers() {

System.out.println(name + " is using a computer.");

}

}

class Faculty extends LibraryUser {

Faculty(String name, int id) {

super(name, id);

}

public void reserveBooks() {

System.out.println(name + " reserved a book.");

}

public void accessResearchDatabase() {

System.out.println(name + " is accessing research databases.");

}

}

class Guest extends LibraryUser {

Guest(String name, int id) {

super(name, id);

}

public void browseBooks() {

System.out.println(name + " is browsing books.");

}

}

public class UniversityLibrarySystem {

public static void main(String[] args) {

LibraryUser u1 = new Student("Alice", 101);

LibraryUser u2 = new Faculty("Bob", 201);

LibraryUser u3 = new Guest("Charlie", 301);

LibraryUser[] users = {u1, u2, u3};

for (LibraryUser user : users) {

user.displayInfo();

user.accessLibrary();

System.out.println();

}

((Student) u1).borrowBooks();

((Student) u1).accessComputers();

((Faculty) u2).reserveBooks();

((Faculty) u2).accessResearchDatabase();

((Guest) u3).browseBooks();

}

}

**PROBLEM 5: Movie Streaming Platform**

**Concept:** Downcasting

Build a streaming service that handles different content types: 2



● 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 {

String title;

String genre;

Content(String title, String genre) {

this.title = title;

this.genre = genre;

}

public void play() {

System.out.println("Playing: " + title + " (" + genre + ")");

}

}

class Movie extends Content {

double rating;

int duration;

boolean subtitles;

Movie(String title, String genre, double rating, int duration, boolean subtitles) {

super(title, genre);

this.rating = rating;

this.duration = duration;

this.subtitles = subtitles;

}

public void showMovieDetails() {

System.out.println("Movie: " + title);

System.out.println("Rating: " + rating + " | Duration: " + duration + " mins");

System.out.println("Subtitles Available: " + (subtitles ? "Yes" : "No"));

}

}

class TVSeries extends Content {

int seasons;

int episodes;

TVSeries(String title, String genre, int seasons, int episodes) {

super(title, genre);

this.seasons = seasons;

this.episodes = episodes;

}

public void suggestNextEpisode() {

System.out.println("TV Series: " + title);

System.out.println("Seasons: " + seasons + " | Episodes: " + episodes);

System.out.println("Suggested: Watch next episode!");

}

}

class Documentary extends Content {

String tags;

String relatedContent;

Documentary(String title, String genre, String tags, String relatedContent) {

super(title, genre);

this.tags = tags;

this.relatedContent = relatedContent;

}

public void showEducationalContent() {

System.out.println("Documentary: " + title);

System.out.println("Tags: " + tags);

System.out.println("You may also like: " + relatedContent);

}

}

public class StreamingPlatform {

public static void main(String[] args) {

Content c1 = new Movie("Inception", "Sci-Fi", 8.8, 148, true);

Content c2 = new TVSeries("Breaking Bad", "Crime Drama", 5, 62);

Content c3 = new Documentary("Planet Earth", "Nature", "Wildlife, Ecology", "Blue Planet");

Content[] watchlist = {c1, c2, c3};

for (Content c : watchlist) {

c.play();

if (c instanceof Movie)

((Movie) c).showMovieDetails();

else if (c instanceof TVSeries)

((TVSeries) c).suggestNextEpisode();

else if (c instanceof Documentary)

((Documentary) c).showEducationalContent();

System.out.println();

}

}

}

**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 SmartDevice {

String deviceId;

String location;

SmartDevice(String deviceId, String location) {

this.deviceId = deviceId;

this.location = location;

}

public void showStatus() {

System.out.println("Device ID: " + deviceId + " | Location: " + location);

}

}

class SmartClassroom extends SmartDevice {

SmartClassroom(String deviceId, String location) {

super(deviceId, location);

}

public void controlLighting() {

System.out.println("Lighting adjusted in classroom at " + location);

}

public void controlAC() {

System.out.println("AC temperature optimized in classroom at " + location);

}

public void controlProjector() {

System.out.println("Projector turned on in classroom at " + location);

}

}

class SmartLab extends SmartDevice {

SmartLab(String deviceId, String location) {

super(deviceId, location);

}

public void manageEquipment() {

System.out.println("Lab equipment calibrated in lab at " + location);

}

public void checkSafetySystems() {

System.out.println("Safety systems checked in lab at " + location);

}

}

class SmartLibrary extends SmartDevice {

SmartLibrary(String deviceId, String location) {

super(deviceId, location);

}

public void trackOccupancy() {

System.out.println("Library occupancy tracked at " + location);

}

public void checkBookAvailability() {

System.out.println("Book availability updated in library at " + location);

}

}

public class SmartCampusIoT {

public static void main(String[] args) {

SmartDevice d1 = new SmartClassroom("C101", "Block A");

SmartDevice d2 = new SmartLab("L202", "Block B");

SmartDevice d3 = new SmartLibrary("Lib1", "Central Building");

SmartDevice[] devices = {d1, d2, d3};

for (SmartDevice d : devices) {

d.showStatus();

if (d instanceof SmartClassroom) {

SmartClassroom sc = (SmartClassroom) d;

sc.controlLighting();

sc.controlAC();

sc.controlProjector();

} else if (d instanceof SmartLab) {

SmartLab sl = (SmartLab) d;

sl.manageEquipment();

sl.checkSafetySystems();

} else if (d instanceof SmartLibrary) {

SmartLibrary lib = (SmartLibrary) d;

lib.trackOccupancy();

lib.checkBookAvailability();

}

System.out.println();

}

}

}

**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

3



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!

4