

**POLYMORPHISM ASSIGNMENT (HW) PRACTICE PROBLEMS (Any 6)**

**PROBLEM 1: Hotel Booking System**

**Concept: Method Overloading**

You're building a hotel reservation system that calculates room prices in various ways:

● Standard booking (just room type and nights)

● Seasonal booking (room type, nights + seasonal multiplier)

● Corporate booking (room type, nights + corporate discount + meal package) ● Wedding package (room type, nights + guest count + decoration fee + catering options)   
  
Each calculation should display a detailed breakdown of costs and savings applied.

**Hint: Multiple ways to book the same room - let method signatures handle the complexity!**

PROGRAM:

class HotelBooking {

void bookRoom(String roomType, int nights) {

double price = nights \* 1000; // base price

System.out.println("Standard Booking: Room=" + roomType + ", Nights=" + nights + ", Price=" + price);

}

void bookRoom(String roomType, int nights, double seasonalMultiplier) {

double price = nights \* 1000 \* seasonalMultiplier;

System.out.println("Seasonal Booking: Room=" + roomType + ", Nights=" + nights + ", Price=" + price);

}

void bookRoom(String roomType, int nights, double corporateDiscount, boolean meal) {

double base = nights \* 1000;

double discount = base \* corporateDiscount;

double price = base - discount + (meal ? 500 : 0);

System.out.println("Corporate Booking: Room=" + roomType + ", Nights=" + nights + ", Price=" + price);

}

void bookRoom(String roomType, int nights, int guestCount, double decorationFee, double cateringFee) {

double price = nights \* 2000 + decorationFee + cateringFee \* guestCount;

System.out.println("Wedding Package: Room=" + roomType + ", Guests=" + guestCount + ", Price=" + price);

}

}

public class Problem1\_HotelBooking {

public static void main(String[] args) {

HotelBooking hb = new HotelBooking();

hb.bookRoom("Deluxe", 3);

hb.bookRoom("Deluxe", 3, 1.2);

hb.bookRoom("Suite", 5, 0.1, true);

hb.bookRoom("Hall", 1, 100, 5000, 200);

}

}

OUTPUT:

Standard Booking: Room=Deluxe, Nights=3, Price=3000.0

Seasonal Booking: Room=Deluxe, Nights=3, Price=3600.0

Corporate Booking: Room=Suite, Nights=5, Price=5000.0

Wedding Package: Room=Hall, Guests=100, Price=27000.0

**PROBLEM 2: Online Learning Platform**

**Concept: Method Overriding**

Create an educational content system where different course types display progress differently:

● Video courses show completion percentage and watch time

● Interactive courses show quiz scores and hands-on projects completed ● Reading courses show pages read and note-taking progress

● Certification courses show exam attempts and certification status

All courses share basic info (title, instructor, enrollment date) but track and display progress uniquely.

1



**Hint: Common learning foundation, specialized progress tracking per course type!**

PROGRAM:

class Course {

String title, instructor;

Course(String title, String instructor) {

this.title = title; this.instructor = instructor;

}

void showProgress() {

System.out.println("Generic course progress...");

}

}

class VideoCourse extends Course {

int percentage, watchTime;

VideoCourse(String t, String i, int p, int w) { super(t,i); percentage=p; watchTime=w; }

@Override void showProgress() {

System.out.println(title + ": " + percentage + "% completed, WatchTime=" + watchTime+" hrs");

}

}

class InteractiveCourse extends Course {

int quizzes, projects;

InteractiveCourse(String t, String i, int q, int p) { super(t,i); quizzes=q; projects=p; }

@Override void showProgress() {

System.out.println(title + ": Quizzes=" + quizzes + ", Projects=" + projects);

}

}

class ReadingCourse extends Course {

int pages, notes;

ReadingCourse(String t, String i, int p, int n) { super(t,i); pages=p; notes=n; }

@Override void showProgress() {

System.out.println(title + ": Pages Read=" + pages + ", Notes=" + notes);

}

}

class CertificationCourse extends Course {

int attempts; boolean certified;

CertificationCourse(String t, String i, int a, boolean c) { super(t,i); attempts=a; certified=c; }

@Override void showProgress() {

System.out.println(title + ": Exam Attempts=" + attempts + ", Certified=" + certified);

}

}

public class Problem2\_OnlineLearning {

public static void main(String[] args) {

Course[] courses = {

new VideoCourse("Java", "Mr. A", 80, 15),

new InteractiveCourse("Python", "Ms. B", 5, 2),

new ReadingCourse("History", "Mr. C", 120, 20),

new CertificationCourse("AWS", "Ms. D", 2, true)

};

for (Course c : courses) c.showProgress();

}

}

OUTPUT:

Java: 80% completed, WatchTime=15 hrs

Python: Quizzes=5, Projects=2

History: Pages Read=120, Notes=20

AWS: Exam Attempts=2, Certified=true

**PROBLEM 3: Transportation Fleet Management Concept: Dynamic Method Dispatch**

Design a city transport system with different vehicle types:

● Buses follow fixed routes and track passenger capacity

● Taxis provide door-to-door service and calculate fare by distance ● Trains operate on schedules and manage multiple car capacity ● Bikes are available for short-distance eco-friendly trips

Create a unified "dispatch" system where the same command produces appropriate transportation behavior based on vehicle type.

**Hint: One dispatch call, many transport solutions - runtime polymorphism in action!**

PROGRAM:

abstract class Vehicle {

abstract void operate();

}

class Bus extends Vehicle {

@Override void operate() { System.out.println("Bus follows fixed routes and manages passengers."); }

}

class Taxi extends Vehicle {

@Override void operate() { System.out.println("Taxi calculates fare by distance."); }

}

class Train extends Vehicle {

@Override void operate() { System.out.println("Train runs on schedule and manages cars."); }

}

class Bike extends Vehicle {

@Override void operate() { System.out.println("Bike provides eco-friendly short trips."); }

}

public class Problem3\_TransportFleet {

public static void main(String[] args) {

Vehicle[] fleet = { new Bus(), new Taxi(), new Train(), new Bike() };

for (Vehicle v : fleet) v.operate();

}

}

OUTPUT:

Bus follows fixed routes and manages passengers.

Taxi calculates fare by distance.

Train runs on schedule and manages cars.

Bike provides eco-friendly short trips.

**PROBLEM 4: Hospital Management System Concept: Upcasting**

Build a hospital system managing different types of medical staff:

● Doctors can diagnose patients, prescribe medicine, and perform surgeries ● Nurses can administer medicine, monitor patients, and assist procedures ● Technicians can operate equipment, run tests, and maintain instruments ● Administrators can schedule appointments and manage records

Design a general "MedicalStaff" system for common operations like shift scheduling, ID card access, and payroll processing.

**Hint: Different specialties, common professional needs - think institutional level!** 2

PROGRAM:

class MedicalStaff {

void shiftSchedule() { System.out.println("Shift scheduled."); }

}

class Doctor extends MedicalStaff {

void diagnose() { System.out.println("Doctor diagnoses patients."); }

}

class Nurse extends MedicalStaff {

void assist() { System.out.println("Nurse assists in procedures."); }

}

class Technician extends MedicalStaff {

void operateEquip() { System.out.println("Technician runs equipment."); }

}

class Administrator extends MedicalStaff {

void manageRecords() { System.out.println("Administrator manages hospital records."); }

}

public class Problem4\_HospitalManagement {

public static void main(String[] args) {

MedicalStaff staff = new Doctor(); // upcasting

staff.shiftSchedule(); // only common method accessible

}

}

OUTPUT:

Shift scheduled.



**PROBLEM 5: Digital Art Gallery**

**Concept: Downcasting**

Create an art gallery system handling different artwork types:

● Paintings have brush techniques, color palettes, and frame specifications ● Sculptures have material composition, dimensions, and lighting requirements ● Digital art has resolution, file formats, and interactive elements

● Photography has camera settings, editing details, and print specifications Sometimes curators need access to specific artwork features for exhibition planning. **Hint: From general art piece to specific medium - unlock the details when needed!**

PROGRAM:

class Artwork {

String title;

Artwork(String t){ title = t; }

}

class Painting extends Artwork {

String technique;

Painting(String t,String tech){ super(t); technique=tech; }

void showPainting(){ System.out.println(title + " uses technique: " + technique); }

}

class Sculpture extends Artwork {

String material;

Sculpture(String t,String m){ super(t); material=m; }

void showSculpture(){ System.out.println(title + " made of: " + material); }

}

public class Problem5\_DigitalArtGallery {

public static void main(String[] args) {

Artwork art = new Painting("Mona Lisa", "Oil on Canvas");

if (art instanceof Painting) {

Painting p = (Painting) art; // downcasting

p.showPainting();

}

}

}

OUTPUT:

Mona Lisa uses technique: Oil on Canvas

**PROBLEM 6: Smart Home Automation**

**Concept: Safe Downcasting with instanceof**

Design a home automation system controlling various smart devices:

● Smart TVs manage channels, volume, and streaming apps

● Smart thermostats control temperature, humidity, and energy saving modes ● Smart security systems handle cameras, alarms, and access controls ● Smart kitchen appliances manage cooking times, temperatures, and recipes

Process mixed device collections safely, applying appropriate controls without system   
crashes.

**Hint: Identify before you control - each device has its own smart features!**

PROGRAM:

abstract class SmartDevice {}

class SmartTV extends SmartDevice {

void stream(){ System.out.println("Streaming on Smart TV"); }

}

class SmartThermostat extends SmartDevice {

void setTemp(){ System.out.println("Temperature set by Thermostat"); }

}

class SmartSecurity extends SmartDevice {

void arm(){ System.out.println("Security System Armed"); }

}

public class Problem6\_SmartHomeAutomation {

public static void main(String[] args) {

SmartDevice[] devices = { new SmartTV(), new SmartThermostat(), new SmartSecurity() };

for (SmartDevice d : devices) {

if (d instanceof SmartTV) ((SmartTV)d).stream();

else if (d instanceof SmartThermostat) ((SmartThermostat)d).setTemp();

else if (d instanceof SmartSecurity) ((SmartSecurity)d).arm();

}

}

}

OUTPUT:

Streaming on Smart TV

Temperature set by Thermostat

Security System Armed

**PROBLEM 7: Banking Transaction System**

3



**Concept: Multiple Polymorphism Integration**

Create a comprehensive banking system handling:

● Savings accounts with interest calculation and withdrawal limits ● Checking accounts with overdraft protection and fee structures ● Investment accounts with portfolio management and risk assessment ● Business accounts with bulk transactions and merchant services

The system should process transactions differently, calculate fees in multiple ways, and safely handle mixed account portfolios.

**Hint: Combine method overloading for different fee structures, overriding for account-specific processing, and safe casting for mixed operations!**

**PROBLEM 8: Theme Park Management System Concept: Complete Polymorphism Mastery**

Build a comprehensive theme park system with different attraction types:

● Roller coasters have height requirements, thrill levels, and safety checks ● Water rides need swim ability checks, weather dependency, and equipment rental

● Shows have seating capacity, showtimes, and age-appropriate content ratings ● Games have skill levels, prize tiers, and group participation options

Design management systems that:

● Handle visitor entry the same way for all attractions (inheritance) ● Operate attractions differently based on type (overriding)

● Process tickets with multiple pricing options (overloading)

● Manage mixed attraction collections safely during maintenance (casting)

**Hint: The ultimate polymorphism playground - inheritance hierarchies, method variations, runtime decisions, and type-safe operations all working together!**

4



**Additional Challenge Problems**

**PROBLEM 9: Cryptocurrency Exchange Platform**

**Concept: Advanced Polymorphism with Interfaces**

Design a trading platform supporting multiple cryptocurrencies:

● Bitcoin with blockchain verification and mining difficulty

● Ethereum with smart contract capabilities and gas fees

● Stablecoins with peg mechanisms and reserve backing

● NFTs with metadata, ownership history, and marketplace features

Implement trading algorithms that work uniformly across all crypto types while accessing specific features when needed.

**PROBLEM 10: Weather Monitoring Network**

**Concept: Factory Pattern with Polymorphism**

Create a weather station network with different sensor types:

● Temperature sensors with calibration data and range specifications ● Humidity sensors with moisture detection and dew point calculation ● Wind sensors with direction tracking and gust measurement

● Rain sensors with precipitation accumulation and intensity levels

Build a monitoring system that creates appropriate sensors dynamically and processes their data polymorphically while maintaining type safety.

5