**POLYMORPHISM ASSIGNMENT (HW) PRACTICE PROBLEMS**

**(Any 6)(1;2;5;6;7;8;9 done)**

**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!**

**Sol: class HotelBooking {**

**void bookRoom(String roomType, int nights) {**

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

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

**}**

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

**double price = nights \* 1000 \* seasonalMultiplier;**

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

**}**

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

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

**double discount = base \* corporateDiscount / 100;**

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

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

**}**

**void bookRoom(String roomType, int nights, int guests, double decorationFee, double catering) {**

**double price = nights \* 1000 + decorationFee + catering \* guests;**

**System.out.println("Wedding Booking: " + roomType + ", Price = " + price);**

**}**

**public static void main(String[] args) {**

**HotelBooking hb = new HotelBooking();**

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

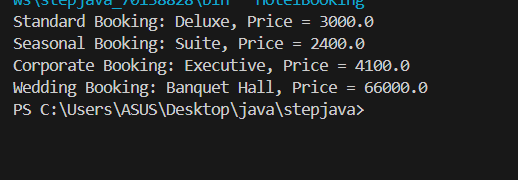
**hb.bookRoom("Suite", 2, 1.2);**

**hb.bookRoom("Executive", 4, 10, true);**

**hb.bookRoom("Banquet Hall", 1, 200, 5000, 300);**

**}**

**}**

****

# 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.

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

**Sol:**

**class Course {**

**String title, instructor;**

**Course(String title, String instructor) {**

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

**}**

**void showProgress() {**

**System.out.println("General progress...");**

**}**

**}**

**class VideoCourse extends Course {**

**int percent, watchTime;**

**VideoCourse(String t, String i, int p, int wt) { super(t,i); percent=p; watchTime=wt; }**

**void showProgress() {**

**System.out.println(title + ": " + percent + "% completed, Watch time: " + watchTime + " hrs");**

**}**

**}**

**class InteractiveCourse extends Course {**

**int quizScore, projects;**

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

**void showProgress() {**

**System.out.println(title + ": Quiz " + quizScore + "%, Projects done: " + projects);**

**}**

**}**

**public class Maain {**

**public static void main(String[] args) {**

**Course c1 = new VideoCourse("Java Basics","Alice",80,10);**

**Course c2 = new InteractiveCourse("Data Structures","Bob",90,3);**

**c1.showProgress();**

**c2.showProgress();**

**}**

**}**

**A screen shot of a computer

AI-generated content may be incorrect.**

# 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!**

**Sol:**

# 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!**

# 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!**

**Sol:**

**class Artwork {**

**String title;**

**Artwork(String title){ this.title=title; }**

**}**

**class Painting extends Artwork {**

**String style;**

**Painting(String t,String s){ super(t); style=s; }**

**void showDetails(){ System.out.println("Painting: " + title + ", Style: " + style); }**

**}**

**public class Gallery {**

**public static void main(String[] args) {**

**Artwork art = new Painting("Sunset","Impressionism");**

**if(art instanceof Painting){**

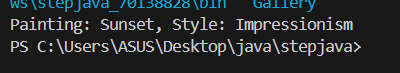
**Painting p = (Painting) art; // Downcasting**

**p.showDetails();**

**}**

**}**

**}**

****

# 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**

**Sol:**

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

**class SmartTV extends Device {**

**void controlTV(){ System.out.println("SmartTV: Switching channel."); }**

**}**

**class Thermostat extends Device {**

**void controlThermo(){ System.out.println("Thermostat: Adjusting temperature."); }**

**}**

**public class HomeAutomation {**

**public static void main(String[] args) {**

**Device[] devices = { new SmartTV(), new Thermostat() };**

**for(Device d : devices){**

**if(d instanceof SmartTV) ((SmartTV)d).controlTV();**

**else if(d instanceof Thermostat) ((Thermostat)d).controlThermo();**

**}**

**}**

**}**

**A screen shot of a computer

AI-generated content may be incorrect.**

# PROBLEM 7: Banking Transaction System

**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!**

**Sol:**

**class Account {**

**String accType;**

**double balance;**

**Account(String accType, double balance){**

**this.accType = accType; this.balance = balance;**

**}**

**void processTransaction(double amount){**

**System.out.println("Processing generic transaction...");**

**}**

**void calcFee(double amount){**

**System.out.println("Generic fee calculation");**

**}**

**}**

**class SavingsAccount extends Account {**

**SavingsAccount(double bal){ super("Savings", bal); }**

**void processTransaction(double amount){**

**if(balance - amount >= 500) {**

**balance -= amount;**

**System.out.println("Savings: Withdrawal done. Balance: " + balance);**

**} else System.out.println("Savings: Minimum balance required!");**

**}**

**void calcFee(double amount){**

**System.out.println("Savings: No extra fee. Interest will apply monthly.");**

**}**

**}**

**class CheckingAccount extends Account {**

**CheckingAccount(double bal){ super("Checking", bal); }**

**void processTransaction(double amount){**

**balance -= amount;**

**System.out.println("Checking: Transaction done. Balance: " + balance);**

**}**

**void calcFee(double amount){**

**double fee = amount \* 0.02;**

**balance -= fee;**

**System.out.println("Checking: Fee charged = " + fee);**

**}**

**}**

**public class BankSystem {**

**public static void main(String[] args) {**

**Account a1 = new SavingsAccount(2000);**

**Account a2 = new CheckingAccount(3000);**

**a1.processTransaction(1000); a1.calcFee(1000);**

**a2.processTransaction(500); a2.calcFee(500);**

**}**

**}**

**A screen shot of a computer code

AI-generated content may be incorrect.**

# 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!**

**Sol:**

**class Attraction {**

**String name;**

**Attraction(String name){ this.name = name; }**

**void enter(){ System.out.println("Visitor entered attraction: " + name); }**

**void operate(){ System.out.println("Generic attraction running..."); }**

**void ticketPrice(int people){ System.out.println("Ticket price: " + (people\*100)); }**

**}**

**class RollerCoaster extends Attraction {**

**RollerCoaster(String n){ super(n); }**

**void operate(){ System.out.println("RollerCoaster: Thrill ride with safety checks!"); }**

**void ticketPrice(int people){ System.out.println("Coaster price: " + (people\*200)); }**

**}**

**class WaterRide extends Attraction {**

**WaterRide(String n){ super(n); }**

**void operate(){ System.out.println("WaterRide: Check weather & swim ability."); }**

**void ticketPrice(int people, boolean rent){**

**int cost = people\*150 + (rent? 50:0);**

**System.out.println("WaterRide price: " + cost);**

**}**

**}**

**public class ThemePark {**

**public static void main(String[] args) {**

**Attraction a1 = new RollerCoaster("Dragon Coaster");**

**Attraction a2 = new WaterRide("Splash River");**

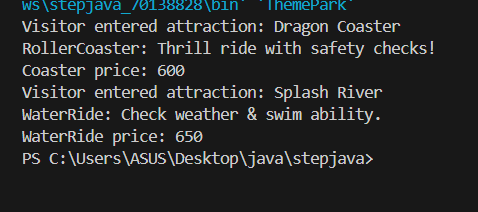
**a1.enter(); a1.operate(); a1.ticketPrice(3);**

**a2.enter(); a2.operate();**

**((WaterRide)a2).ticketPrice(4,true); // casting for overloaded method**

**}**

**}**

****

# 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.

Sol:

interface Crypto {

void trade(double amount);

void specialFeature();

}

class Bitcoin implements Crypto {

public void trade(double amount){ System.out.println("Bitcoin trade: " + amount + " BTC"); }

public void specialFeature(){ System.out.println("Bitcoin: Blockchain verification"); }

}

class Ethereum implements Crypto {

public void trade(double amount){ System.out.println("Ethereum trade: " + amount + " ETH"); }

public void specialFeature(){ System.out.println("Ethereum: Smart contract executed"); }

}

class StableCoin implements Crypto {

public void trade(double amount){ System.out.println("StableCoin trade: $" + amount); }

public void specialFeature(){ System.out.println("StableCoin: Peg mechanism working"); }

}

public class Exchange {

public static void main(String[] args) {

Crypto[] coins = { new Bitcoin(), new Ethereum(), new StableCoin() };

for(Crypto c : coins){

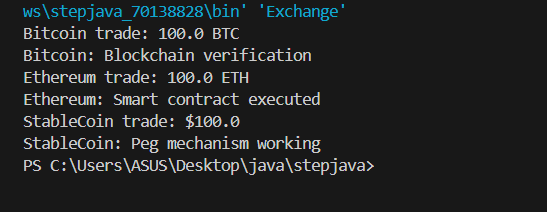
c.trade(100);

c.specialFeature();

}

}

}



## 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.