```java
// 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!



public class HotelBooking {
    public static void main(String[] args) {
        HotelBooking hb = new HotelBooking();

        System.out.println("Standard Booking:");
        hb.calculatePrice("Deluxe", 3);
        System.out.println();

        System.out.println("Seasonal Booking:");
        hb.calculatePrice("Deluxe", 3, 1.25); // seasonal multiplier
        System.out.println();

        System.out.println("Corporate Booking:");
        hb.calculatePrice("Deluxe", 5, 0.10, true); // discount 10%, meal
package true
        System.out.println();

        System.out.println("Wedding Package:");
        hb.calculatePrice("Suite", 2, 50, 500.0, true); // guests,
decoration, catering
```

```
        System.out.println();
    }

    // Standard booking: room type and nights
    public void calculatePrice(String roomType, int nights) {
        double rate = baseRate(roomType);
        double total = rate * nights;
        System.out.println(String.format("Room: %s | Nights: %d | Rate:
%.2f | Total: %.2f", roomType, nights, rate, total));
    }

    // Seasonal booking: multiplier applies
    public void calculatePrice(String roomType, int nights, double
seasonalMultiplier) {
        double rate = baseRate(roomType);
        double subtotal = rate * nights;
        double total = subtotal * seasonalMultiplier;
        System.out.println(String.format("Room: %s | Nights: %d | Base:
%.2f | Multiplier: %.2f | Total: %.2f", roomType, nights, subtotal,
seasonalMultiplier, total));
    }

    // Corporate booking: discount and optional meal package
    public void calculatePrice(String roomType, int nights, double
corporateDiscount, boolean mealPackage) {
        double rate = baseRate(roomType);
        double base = rate * nights;
        double discount = base * corporateDiscount;
        double meal = mealPackage ? 20.0 * nights : 0.0; // per-night meal
cost
        double total = base - discount + meal;
        System.out.println(String.format("Room: %s | Nights: %d | Base:
%.2f | Discount: %.2f | Meal: %.2f | Total: %.2f", roomType, nights, base,
discount, meal, total));
    }

    // Wedding package: guests, decoration fee, catering per guest
    public void calculatePrice(String roomType, int nights, int
guestCount, double decorationFee, boolean catering) {
        double rate = baseRate(roomType);
```

```
        double base = rate * nights;
        double cateringCost = catering ? guestCount * 15.0 : 0.0;
        double total = base + decorationFee + cateringCost;
        System.out.println(String.format("Room: %s | Nights: %d | Base:
%.2f | Guests: %d | Decoration: %.2f | Catering: %.2f | Total: %.2f",
roomType, nights, base, guestCount, decorationFee, cateringCost, total));
    }

    private double baseRate(String roomType) {
        switch (roomType.toLowerCase()) {
            case "suite": return 500.0;
            case "deluxe": return 200.0;
            default: return 100.0;
        }
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\assignment> cd "e:\JAVA PROGRAMS\steparyansingh\year2
\oops\week7\assignment"; java HotelBooking
Room: Deluxe | Nights: 5 | Base: 1000.00 | Discount: 100.00 | Meal: 100.00 | Total: 1000.00

Wedding Package:
Room: Suite | Nights: 2 | Base: 1000.00 | Guests: 50 | Decoration: 500.00 | Catering: 750.00 | Total: 225
0.00
```

```
// 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!




import java.time.LocalDate;

public class OnlineLearningDemo {
    public static void main(String[] args) {
        Course[] courses = new Course[4];
        courses[0] = new VideoCourse("Java 101", "Dr. A",
LocalDate.of(2025,1,15), 75, 120);
        courses[1] = new InteractiveCourse("Hands-on Web", "Prof. B",
LocalDate.of(2025,3,1), 85, 3);
        courses[2] = new ReadingCourse("Algorithms Book", "Dr. C",
LocalDate.of(2025,2,10), 120, 30);
        courses[3] = new CertificationCourse("Cloud Cert", "Ms. D",
LocalDate.of(2024,11,5), 2, true);

        for (Course c : courses) {
            c.displayProgress();
            System.out.println();
        }
    }
}

class Course {
    protected String title;
    protected String instructor;
    protected LocalDate enrolled;

    public Course(String title, String instructor, LocalDate enrolled) {
        this.title = title;
        this.instructor = instructor;
        this.enrolled = enrolled;
    }
```

```
    public void displayProgress() {
        System.out.println("Course: " + title + " | Instructor: " +
instructor + " | Enrolled: " + enrolled);
    }
}

class VideoCourse extends Course {
    private int completionPercent;
    private int watchTimeMins;

    public VideoCourse(String title, String instructor, LocalDate
enrolled, int completionPercent, int watchTimeMins) {
        super(title, instructor, enrolled);
        this.completionPercent = completionPercent;
        this.watchTimeMins = watchTimeMins;
    }

    @Override
    public void displayProgress() {
        super.displayProgress();
        System.out.println("Video Progress: " + completionPercent + "% |
Watch time: " + watchTimeMins + " mins");
    }
}

class InteractiveCourse extends Course {
    private int quizScore;
    private int projectsCompleted;

    public InteractiveCourse(String title, String instructor, LocalDate
enrolled, int quizScore, int projectsCompleted) {
        super(title, instructor, enrolled);
        this.quizScore = quizScore;
        this.projectsCompleted = projectsCompleted;
    }

    @Override
    public void displayProgress() {
        super.displayProgress();
```

```
        System.out.println("Interactive Progress: Quiz " + quizScore + "%
| Projects: " + projectsCompleted);
    }
}

class ReadingCourse extends Course {
    private int pagesRead;
    private int notesTaken;

    public ReadingCourse(String title, String instructor, LocalDate
enrolled, int pagesRead, int notesTaken) {
        super(title, instructor, enrolled);
        this.pagesRead = pagesRead;
        this.notesTaken = notesTaken;
    }

    @Override
    public void displayProgress() {
        super.displayProgress();
        System.out.println("Reading Progress: Pages read " + pagesRead + "
| Notes: " + notesTaken);
    }
}

class CertificationCourse extends Course {
    private int examAttempts;
    private boolean certified;

    public CertificationCourse(String title, String instructor, LocalDate
enrolled, int examAttempts, boolean certified) {
        super(title, instructor, enrolled);
        this.examAttempts = examAttempts;
        this.certified = certified;
    }

    @Override
    public void displayProgress() {
        super.displayProgress();
        System.out.println("Certification Progress: Attempts " +
examAttempts + " | Certified: " + (certified ? "Yes" : "No"));
```

```
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\assignment> cd "e:\JAVA PROGRAMS\steparyansingh\year2
\oops\week7\assignment"; java OnlineLearningDemo
Course: Algorithms Book | Instructor: Dr. C | Enrolled: 2025-02-10
Reading Progress: Pages read 120 | Notes: 30

Course: Cloud Cert | Instructor: Ms. D | Enrolled: 2024-11-05
Certification Progress: Attempts 2 | Certified: Yes
```

```java
// PROBLEM 4: Hospital Management System
// Concept: Upcasting
// Demonstrates a general MedicalStaff system with specialized staff types
// and common institutional operations.

public class HospitalManagementDemo {
    public static void main(String[] args) {
        MedicalStaff[] staffList = new MedicalStaff[] {
            new Doctor("Dr. Smith", 101),
            new Nurse("Nurse Amy", 202),
            new Technician("Tech Bob", 303),
            new Administrator("Ms. Clark", 404)
        };

        // Common institutional operations via upcasting
        for (MedicalStaff staff : staffList) {
            staff.shiftSchedule();
            staff.accessIDCard();
            staff.processPayroll();
            System.out.println();
        }

        // Specialized operations via downcasting (example)
```

```
        for (MedicalStaff staff : staffList) {
            if (staff instanceof Doctor) {
                Doctor d = (Doctor) staff;
                d.diagnosePatient();
                d.prescribeMedicine();
                d.performSurgery();
            } else if (staff instanceof Nurse) {
                Nurse n = (Nurse) staff;
                n.administerMedicine();
                n.monitorPatient();
                n.assistProcedure();
            } else if (staff instanceof Technician) {
                Technician t = (Technician) staff;
                t.operateEquipment();
                t.runTest();
                t.maintainInstrument();
            } else if (staff instanceof Administrator) {
                Administrator a = (Administrator) staff;
                a.scheduleAppointment();
                a.manageRecords();
            }
            System.out.println();
        }
    }
}

class MedicalStaff {
    protected String name;
    protected int staffId;

    public MedicalStaff(String name, int staffId) {
        this.name = name;
        this.staffId = staffId;
    }

    public void shiftSchedule() {
        System.out.println(name + " (ID: " + staffId + ") scheduled for
shift.");
    }
```

```
    public void accessIDCard() {
        System.out.println(name + " (ID: " + staffId + ") accessed
hospital ID card.");
    }

    public void processPayroll() {
        System.out.println(name + " (ID: " + staffId + ") payroll
processed.");
    }
}

class Doctor extends MedicalStaff {
    public Doctor(String name, int staffId) {
        super(name, staffId);
    }
    public void diagnosePatient() {
        System.out.println(name + " is diagnosing a patient.");
    }
    public void prescribeMedicine() {
        System.out.println(name + " is prescribing medicine.");
    }
    public void performSurgery() {
        System.out.println(name + " is performing surgery.");
    }
}

class Nurse extends MedicalStaff {
    public Nurse(String name, int staffId) {
        super(name, staffId);
    }
    public void administerMedicine() {
        System.out.println(name + " is administering medicine.");
    }
    public void monitorPatient() {
        System.out.println(name + " is monitoring a patient.");
    }
    public void assistProcedure() {
        System.out.println(name + " is assisting in a procedure.");
    }
}
```

```
class Technician extends MedicalStaff {
    public Technician(String name, int staffId) {
        super(name, staffId);
    }
    public void operateEquipment() {
        System.out.println(name + " is operating medical equipment.");
    }
    public void runTest() {
        System.out.println(name + " is running a diagnostic test.");
    }
    public void maintainInstrument() {
        System.out.println(name + " is maintaining instruments.");
    }
}

class Administrator extends MedicalStaff {
    public Administrator(String name, int staffId) {
        super(name, staffId);
    }
    public void scheduleAppointment() {
        System.out.println(name + " is scheduling an appointment.");
    }
    public void manageRecords() {
        System.out.println(name + " is managing hospital records.");
    }
}
```

```
Caused by: java.lang.ClassNotFoundException: HospitalManagementDemo
PS E:\JAVA PROGRAMS\steparyansingh\year2> cd oops\week7\assignment; javac HospitalManagementDemo.java; ja
va HospitalManagementDemo
Dr. Smith (ID: 101) scheduled for shift.
Dr. Smith (ID: 101) accessed hospital ID card.
Dr. Smith (ID: 101) payroll processed.

Nurse Amy (ID: 202) scheduled for shift.
Nurse Amy (ID: 202) accessed hospital ID card.
Nurse Amy (ID: 202) payroll processed.

Tech Bob (ID: 303) scheduled for shift.
Tech Bob (ID: 303) accessed hospital ID card.
Tech Bob (ID: 303) payroll processed.

Ms. Clark (ID: 404) scheduled for shift.
Ms. Clark (ID: 404) accessed hospital ID card.
Ms. Clark (ID: 404) payroll processed.

Dr. Smith is diagnosing a patient.
Dr. Smith is prescribing medicine.
Dr. Smith is performing surgery.

Nurse Amy is administering medicine.
Nurse Amy is monitoring a patient.
Nurse Amy is assisting in a procedure.

Tech Bob is operating medical equipment.
Tech Bob is running a diagnostic test.
Tech Bob is maintaining instruments.

Ms. Clark is scheduling an appointment.
Ms. Clark is managing hospital records.
```

```java
// PROBLEM 5: Digital Art Gallery
// Concept: Downcasting
// Demonstrates a general ArtPiece system with specialized artwork types
// and curator access to specific features via downcasting.

public class DigitalArtGalleryDemo {
    public static void main(String[] args) {
        ArtPiece[] gallery = new ArtPiece[] {
```

```
            new Painting("Starry Night", "Van Gogh", "Impasto", "Vivid
Blue-Yellow", "Ornate Gold"),
            new Sculpture("The Thinker", "Rodin", "Bronze", "2.5m x 1.2m x
1.5m", "Spotlight"),
            new DigitalArt("Neon Dreams", "A. Lee", "4K", "PNG, GIF",
true),
            new Photography("Moonrise", "A. Adams", "f/8 ISO100", "HDR,
Crop", "Matte 20x30cm")
        };

        // General exhibition info
        for (ArtPiece art : gallery) {
            art.displayInfo();
            System.out.println();
        }

        // Curator needs specific details for planning
        for (ArtPiece art : gallery) {
            if (art instanceof Painting) {
                Painting p = (Painting) art;
                System.out.println("Painting details:");
                p.showBrushTechnique();
                p.showColorPalette();
                p.showFrameSpec();
            } else if (art instanceof Sculpture) {
                Sculpture s = (Sculpture) art;
                System.out.println("Sculpture details:");
                s.showMaterial();
                s.showDimensions();
                s.showLighting();
            } else if (art instanceof DigitalArt) {
                DigitalArt d = (DigitalArt) art;
                System.out.println("Digital Art details:");
                d.showResolution();
                d.showFileFormats();
                d.showInteractive();
            } else if (art instanceof Photography) {
                Photography ph = (Photography) art;
                System.out.println("Photography details:");
                ph.showCameraSettings();
```

```
                ph.showEditingDetails();
                ph.showPrintSpec();
            }
            System.out.println();
        }
    }
}

class ArtPiece {
    protected String title;
    protected String artist;

    public ArtPiece(String title, String artist) {
        this.title = title;
        this.artist = artist;
    }

    public void displayInfo() {
        System.out.println("Title: " + title + " | Artist: " + artist);
    }
}

class Painting extends ArtPiece {
    private String brushTechnique;
    private String colorPalette;
    private String frameSpec;

    public Painting(String title, String artist, String brushTechnique,
String colorPalette, String frameSpec) {
        super(title, artist);
        this.brushTechnique = brushTechnique;
        this.colorPalette = colorPalette;
        this.frameSpec = frameSpec;
    }
    public void showBrushTechnique() {
        System.out.println("Brush technique: " + brushTechnique);
    }
    public void showColorPalette() {
        System.out.println("Color palette: " + colorPalette);
    }
```

```
    public void showFrameSpec() {
        System.out.println("Frame: " + frameSpec);
    }
}

class Sculpture extends ArtPiece {
    private String material;
    private String dimensions;
    private String lighting;

    public Sculpture(String title, String artist, String material, String
dimensions, String lighting) {
        super(title, artist);
        this.material = material;
        this.dimensions = dimensions;
        this.lighting = lighting;
    }
    public void showMaterial() {
        System.out.println("Material: " + material);
    }
    public void showDimensions() {
        System.out.println("Dimensions: " + dimensions);
    }
    public void showLighting() {
        System.out.println("Lighting: " + lighting);
    }
}

class DigitalArt extends ArtPiece {
    private String resolution;
    private String fileFormats;
    private boolean interactive;

    public DigitalArt(String title, String artist, String resolution,
String fileFormats, boolean interactive) {
        super(title, artist);
        this.resolution = resolution;
        this.fileFormats = fileFormats;
        this.interactive = interactive;
    }
```

```
    public void showResolution() {
        System.out.println("Resolution: " + resolution);
    }
    public void showFileFormats() {
        System.out.println("File formats: " + fileFormats);
    }
    public void showInteractive() {
        System.out.println("Interactive elements: " + (interactive ? "Yes"
: "No"));
    }
}

class Photography extends ArtPiece {
    private String cameraSettings;
    private String editingDetails;
    private String printSpec;

    public Photography(String title, String artist, String cameraSettings,
String editingDetails, String printSpec) {
        super(title, artist);
        this.cameraSettings = cameraSettings;
        this.editingDetails = editingDetails;
        this.printSpec = printSpec;
    }
    public void showCameraSettings() {
        System.out.println("Camera settings: " + cameraSettings);
    }
    public void showEditingDetails() {
        System.out.println("Editing: " + editingDetails);
    }
    public void showPrintSpec() {
        System.out.println("Print: " + printSpec);
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\assignment> java DigitalArtGalleryDemo
Title: Starry Night | Artist: Van Gogh

Title: The Thinker | Artist: Rodin

Title: Neon Dreams | Artist: A. Lee

Title: Moonrise | Artist: A. Adams

Painting details:
Brush technique: Impasto
Color palette: Vivid Blue-Yellow
Frame: Ornate Gold

Sculpture details:
Material: Bronze
Dimensions: 2.5m x 1.2m x 1.5m
Lighting: Spotlight

Digital Art details:
Resolution: 4K
File formats: PNG, GIF
Interactive elements: Yes

Photography details:
Camera settings: f/8 ISO100
Editing: HDR, Crop
Print: Matte 20x30cm
```

```java
// PROBLEM 6: Smart Home Automation
// Concept: Safe Downcasting with instanceof
// Demonstrates a home automation system with mixed smart devices
// and safe control using instanceof before downcasting.

public class SmartHomeAutomationDemo {
    public static void main(String[] args) {
        SmartDevice[] devices = new SmartDevice[] {
            new SmartTV("Living Room TV", 45, 12, "Netflix"),
            new SmartThermostat("Nest Thermostat", 22, 45, true),
```

```
            new SmartSecuritySystem("Front Door Security", 4, true,
"1234"),
            new SmartKitchenAppliance("Oven", 180, 45, "Pizza")
        };

        for (SmartDevice device : devices) {
            System.out.println("Device: " + device.getName());
            device.basicStatus();

            // Safe downcasting with instanceof
            if (device instanceof SmartTV) {
                SmartTV tv = (SmartTV) device;
                tv.changeChannel(7);
                tv.adjustVolume(20);
                tv.launchStreamingApp("YouTube");
            } else if (device instanceof SmartThermostat) {
                SmartThermostat thermo = (SmartThermostat) device;
                thermo.setTemperature(24);
                thermo.setHumidity(50);
                thermo.toggleEnergySaving();
            } else if (device instanceof SmartSecuritySystem) {
                SmartSecuritySystem sec = (SmartSecuritySystem) device;
                sec.activateCamera();
                sec.triggerAlarm();
                sec.setAccessCode("5678");
            } else if (device instanceof SmartKitchenAppliance) {
                SmartKitchenAppliance kitchen = (SmartKitchenAppliance)
device;
                kitchen.setCookingTime(30);
                kitchen.setCookingTemp(200);
                kitchen.selectRecipe("Cake");
            }
            System.out.println();
        }
    }
}

abstract class SmartDevice {
    protected String name;
    public SmartDevice(String name) {
```

```
        this.name = name;
    }
    public String getName() { return name; }
    public void basicStatus() {
        System.out.println("Status: " + name + " is online.");
    }
}

class SmartTV extends SmartDevice {
    private int channel;
    private int volume;
    private String streamingApp;
    public SmartTV(String name, int channel, int volume, String
streamingApp) {
        super(name);
        this.channel = channel;
        this.volume = volume;
        this.streamingApp = streamingApp;
    }
    public void changeChannel(int ch) {
        channel = ch;
        System.out.println(name + " channel set to " + channel);
    }
    public void adjustVolume(int vol) {
        volume = vol;
        System.out.println(name + " volume set to " + volume);
    }
    public void launchStreamingApp(String app) {
        streamingApp = app;
        System.out.println(name + " streaming app launched: " +
streamingApp);
    }
}

class SmartThermostat extends SmartDevice {
    private int temperature;
    private int humidity;
    private boolean energySaving;
    public SmartThermostat(String name, int temperature, int humidity,
boolean energySaving) {
```

```
        super(name);
        this.temperature = temperature;
        this.humidity = humidity;
        this.energySaving = energySaving;
    }
    public void setTemperature(int temp) {
        temperature = temp;
        System.out.println(name + " temperature set to " + temperature +
"°C");
    }
    public void setHumidity(int hum) {
        humidity = hum;
        System.out.println(name + " humidity set to " + humidity + "%");
    }
    public void toggleEnergySaving() {
        energySaving = !energySaving;
        System.out.println(name + " energy saving mode: " + (energySaving
? "ON" : "OFF"));
    }
}

class SmartSecuritySystem extends SmartDevice {
    private int cameras;
    private boolean alarmActive;
    private String accessCode;
    public SmartSecuritySystem(String name, int cameras, boolean
alarmActive, String accessCode) {
        super(name);
        this.cameras = cameras;
        this.alarmActive = alarmActive;
        this.accessCode = accessCode;
    }
    public void activateCamera() {
        System.out.println(name + " cameras activated: " + cameras);
    }
    public void triggerAlarm() {
        alarmActive = true;
        System.out.println(name + " alarm triggered!");
    }
    public void setAccessCode(String code) {
```

```
        accessCode = code;
        System.out.println(name + " access code set to " + accessCode);
    }
}

class SmartKitchenAppliance extends SmartDevice {
    private int cookingTemp;
    private int cookingTime;
    private String recipe;
    public SmartKitchenAppliance(String name, int cookingTemp, int
cookingTime, String recipe) {
        super(name);
        this.cookingTemp = cookingTemp;
        this.cookingTime = cookingTime;
        this.recipe = recipe;
    }
    public void setCookingTemp(int temp) {
        cookingTemp = temp;
        System.out.println(name + " cooking temperature set to " +
cookingTemp + "°C");
    }
    public void setCookingTime(int time) {
        cookingTime = time;
        System.out.println(name + " cooking time set to " + cookingTime +
" min");
    }
    public void selectRecipe(String r) {
        recipe = r;
        System.out.println(name + " recipe selected: " + recipe);
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\assignment> java SmartHomeAutomationDemo
Device: Living Room TV
Status: Living Room TV is online.
Living Room TV channel set to 7
Living Room TV volume set to 20
Living Room TV streaming app launched: YouTube

Device: Nest Thermostat
Status: Nest Thermostat is online.
Nest Thermostat temperature set to 24°C
Nest Thermostat humidity set to 50%
Nest Thermostat energy saving mode: OFF

Device: Front Door Security
Status: Front Door Security is online.
Front Door Security cameras activated: 4
Front Door Security alarm triggered!
Front Door Security access code set to 5678

Device: Oven
Status: Oven is online.
Oven cooking time set to 30 min
Oven cooking temperature set to 200°C
Oven recipe selected: Cake
```