```java
// 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!



public class FoodDeliveryCharges {
    public static void main(String[] args) {
        FoodDeliveryCharges calc = new FoodDeliveryCharges();

        System.out.println("Basic Delivery:");
        calc.calculateCharge(10.0); // distance in km
        System.out.println();

        System.out.println("Premium Delivery:");
        calc.calculateCharge(10.0, 5.0); // distance + priority fee
        System.out.println();

        System.out.println("Group Delivery:");
        calc.calculateCharge(10.0, 4); // distance + number of orders
discount
        System.out.println();

        System.out.println("Festival Special:");
        calc.calculateCharge(150.0, 10.0, 1000.0); // distance,
discountPercent, freeOverAmount
        System.out.println();
    }
```

```
    // Basic delivery: base rate per km
    public void calculateCharge(double distanceKm) {
        double ratePerKm = 5.0; // currency units per km
        double charge = distanceKm * ratePerKm;
        System.out.println(String.format("Distance: %.1f km | Rate: %.2f
per km | Total: %.2f", distanceKm, ratePerKm, charge));
    }

    // Premium delivery: distance + priority fee
    public void calculateCharge(double distanceKm, double priorityFee) {
        double ratePerKm = 5.0;
        double base = distanceKm * ratePerKm;
        double total = base + priorityFee;
        System.out.println(String.format("Distance: %.1f km | Base: %.2f |
Priority fee: %.2f | Total: %.2f", distanceKm, base, priorityFee, total));
    }

    // Group delivery: distance + number of orders (discount per extra
order)
    public void calculateCharge(double distanceKm, int numberOfOrders) {
        double ratePerKm = 5.0;
        double base = distanceKm * ratePerKm;
        double discountPerOrder = 1.0; // discount per extra order
        double discount = Math.max(0, (numberOfOrders - 1) *
discountPerOrder);
        double total = Math.max(0, base - discount);
        System.out.println(String.format("Distance: %.1f km | Base: %.2f |
Orders: %d | Discount: %.2f | Total: %.2f", distanceKm, base,
numberOfOrders, discount, total));
    }

    // Festival special: distance + percentage discount, free over certain
amount
    public void calculateCharge(double distanceKm, double discountPercent,
double freeOverAmount) {
        double ratePerKm = 5.0;
        double base = distanceKm * ratePerKm;
        double discounted = base * (1 - discountPercent / 100.0);
        double total = discounted;
```

```
        boolean free = discounted >= freeOverAmount;
        if (free) {
            total = 0.0;
        }
        System.out.println(String.format("Distance: %.1f km | Base: %.2f |
Discount: %.1f%% | After discount: %.2f | Free over: %.2f | Final: %.2f",
distanceKm, base, discountPercent, discounted, freeOverAmount, total));
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\lab-work> cd "e:\JAVA PROGRAMS\steparyansingh\year2\o
ops\week7\lab-work"; java FoodDeliveryCharges

Premium Delivery:
Distance: 10.0 km | Base: 50.00 | Priority fee: 5.00 | Total: 55.00

Group Delivery:
Distance: 10.0 km | Base: 50.00 | Orders: 4 | Discount: 3.00 | Total: 47.00

Festival Special:
Distance: 150.0 km | Base: 750.00 | Discount: 10.0% | After discount: 675.00 | Free over: 1000.00 | Final
: 675.00
```

```
// 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!
```

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class SocialMediaFeed {
    public static void main(String[] args) {
        SocialMediaPost[] feed = new SocialMediaPost[4];
        feed[0] = new InstagramPost("Loving the sunset #sunset #vibes",
"john_doe", 245);
        feed[1] = new TwitterPost("Java 17 features are great!",
"code_ninja", 89);
        feed[2] = new LinkedInPost("Excited to announce my promotion to
Senior Engineer.", "pro_user", 350);
        feed[3] = new SocialMediaPost("Hello from a generic post",
"anon_user");

        for (SocialMediaPost post : feed) {
            post.display();
            System.out.println();
        }
    }
}

class SocialMediaPost {
    protected String content;
    protected String author;
    protected String time;
    private static final DateTimeFormatter FMT =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    public SocialMediaPost(String content, String author) {
        this.content = content;
        this.author = author;
        this.time = LocalDateTime.now().format(FMT);
    }

    public void display() {
        System.out.println("[Generic Post] " + time);
        System.out.println("Author: " + author);
        System.out.println("Content: " + content);
```

```
    }
}

class InstagramPost extends SocialMediaPost {
    private int likes;

    public InstagramPost(String content, String author, int likes) {
        super(content, author);
        this.likes = likes;
    }

    @Override
    public void display() {
        System.out.println("[Instagram] " + time);
        System.out.println("@" + author + " • " + likes + " likes");
        System.out.println(content);
        System.out.println("Hashtags: " + extractHashtags(content));
    }

    private String extractHashtags(String text) {
        StringBuilder sb = new StringBuilder();
        for (String token : text.split("\\s+")) {
            if (token.startsWith("#")) {
                if (sb.length() > 0) sb.append(", ");
                sb.append(token);
            }
        }
        return sb.length() > 0 ? sb.toString() : "(none)";
    }
}

class TwitterPost extends SocialMediaPost {
    private int retweets;

    public TwitterPost(String content, String author, int retweets) {
        super(content, author);
        this.retweets = retweets;
    }

    @Override
```

```
    public void display() {
        System.out.println("[Twitter] " + time);
        System.out.println("@" + author + " • " + content.length() + "
chars" + " • " + retweets + " retweets");
        System.out.println(truncate(content));
    }

    private String truncate(String text) {
        int limit = 280;
        return text.length() <= limit ? text : text.substring(0, limit -
3) + "...";
    }
}

class LinkedInPost extends SocialMediaPost {
    private int connections;

    public LinkedInPost(String content, String author, int connections) {
        super(content, author);
        this.connections = connections;
    }

    @Override
    public void display() {
        System.out.println("[LinkedIn] " + time);
        System.out.println(author + " — " + connections + " connections");
        System.out.println(formatProfessional(content));
    }

    private String formatProfessional(String text) {
        return "***\n" + text + "\n***";
    }
}
```

```
[LinkedIn] 2025-09-24 08:22:30
pro_user ? 350 connections
***
Excited to announce my promotion to Senior Engineer.
***

[Generic Post] 2025-09-24 08:22:30
Author: anon_user
Content: Hello from a generic post
```

```java
public class GameCharactersDemo {
    public static void main(String[] args) {
        Character[] army = new Character[3];
        army[0] = new Warrior("Thorin", "Axe", 80);
        army[1] = new Mage("Gandalf", 120);
        army[2] = new Archer("Legolas", 60);

        System.out.println("Battle Start:\n");
        for (Character c : army) {
            c.attack();
            System.out.println();
        }

        System.out.println("Special actions:");
        ((Warrior) army[0]).defend();
        ((Mage) army[1]).castSpell("Fireball");
        ((Archer) army[2]).longRangeShot();
    }
```

```
}

abstract class Character {
    protected String name;

    public Character(String name) {
        this.name = name;
    }

    public abstract void attack();
}

class Warrior extends Character {
    private String weapon;
    private int defense;

    public Warrior(String name, String weapon, int defense) {
        super(name);
        this.weapon = weapon;
        this.defense = defense;
    }

    @Override
    public void attack() {
        System.out.println(name + " swings " + weapon + " dealing heavy
melee damage!");
    }

    public void defend() {
        System.out.println(name + " raises shield, defense increased by "
+ defense);
    }
}

class Mage extends Character {
    private int mana;

    public Mage(String name, int mana) {
        super(name);
        this.mana = mana;
```

```
    }

    @Override
    public void attack() {
        System.out.println(name + " casts a magic bolt, consuming 10
mana.");
        mana -= 10;
        System.out.println(name + " remaining mana: " + mana);
    }

    public void castSpell(String spell) {
        System.out.println(name + " casts " + spell + " with spectacular
effects!");
    }
}

class Archer extends Character {
    private int range;

    public Archer(String name, int range) {
        super(name);
        this.range = range;
    }

    @Override
    public void attack() {
        System.out.println(name + " fires a precise arrow dealing
long-range damage (range " + range + ")");
    }

    public void longRangeShot() {
        System.out.println(name + " performs a long-range shot at distance
" + range + " meters!");
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\lab-work> cd "e:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\lab-work"; java GameCharactersDemo
Legolas fires a precise arrow dealing long-range damage (range 60)

Special actions:
Thorin raises shield, defense increased by 80
Gandalf casts Fireball with spectacular effects!
Legolas performs a long-range shot at distance 60 meters!
```

```java
// 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!



public class LibrarySystemDemo {
    public static void main(String[] args) {
        LibraryUser[] users = new LibraryUser[3];
        users[0] = new Student("Alice", "CS2021");
        users[1] = new Faculty("Dr. Brown", "Physics");
        users[2] = new Guest("Visitor");
```

```
        for (LibraryUser u : users) {
            u.logEntry();
            u.displayInfo();
            System.out.println();
        }

        // Demonstrate upcasting limitations and downcasting
        LibraryUser lu = users[0]; // upcast Student -> LibraryUser
        // lu.borrowBook(); // compile error if uncommented
        if (lu instanceof Student) {
            ((Student) lu).borrowBook("Introduction to Algorithms");
            ((Student) lu).accessComputer();
        }

        LibraryUser fac = users[1];
        if (fac instanceof Faculty) {
            ((Faculty) fac).reserveBook("Advanced Quantum Mechanics");
            ((Faculty) fac).accessResearchDB();
        }
    }
}

class LibraryUser {
    protected String name;

    public LibraryUser(String name) {
        this.name = name;
    }

    public void logEntry() {
        System.out.println(name + " entered the library.");
    }

    public void displayInfo() {
        System.out.println("User: " + name + " (General library user)");
    }
}

class Student extends LibraryUser {
    private String studentId;
```

```
    public Student(String name, String studentId) {
        super(name);
        this.studentId = studentId;
    }

    public void borrowBook(String title) {
        System.out.println(name + " borrowed: " + title + " (ID: " +
studentId + ")");
    }

    public void accessComputer() {
        System.out.println(name + " is accessing a public computer.");
    }

    @Override
    public void displayInfo() {
        System.out.println("Student: " + name + " (" + studentId + ")");
    }
}

class Faculty extends LibraryUser {
    private String department;

    public Faculty(String name, String department) {
        super(name);
        this.department = department;
    }

    public void reserveBook(String title) {
        System.out.println(name + " reserved: " + title);
    }

    public void accessResearchDB() {
        System.out.println(name + " is accessing a research database.");
    }

    @Override
    public void displayInfo() {
        System.out.println("Faculty: " + name + " (" + department + ")");
```

```
    }
}

class Guest extends LibraryUser {
    public Guest(String name) {
        super(name);
    }

    public void browseBooks() {
        System.out.println(name + " is browsing books.");
    }

    @Override
    public void displayInfo() {
        System.out.println("Guest: " + name + " (limited access)");
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\lab-work> cd "e:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\lab-work"; java LibrarySystemDemo
Guest: Visitor (limited access)

Alice borrowed: Introduction to Algorithms (ID: CS2021)
Alice is accessing a public computer.
Dr. Brown reserved: Advanced Quantum Mechanics
Dr. Brown is accessing a research database.
```

```
// 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!




public class StreamingPlatformDemo {
    public static void main(String[] args) {
        Content[] library = new Content[3];
        library[0] = new Movie("Inception", 148, "PG-13", true);
        library[1] = new TVSeries("Strange Shows", 3, 10);
        library[2] = new Documentary("Planet Earth", new
String[]{"Nature", "Wildlife"}, "Related: Planet Earth II");

        for (Content c : library) {
            c.play();
            if (c instanceof Movie) {
                Movie m = (Movie) c;
                System.out.println("Subtitle available: " +
m.hasSubtitles());
            } else if (c instanceof TVSeries) {
                TVSeries t = (TVSeries) c;
                t.suggestNextEpisode();
            } else if (c instanceof Documentary) {
                Documentary d = (Documentary) c;
                d.showEducationalTags();
            }
            System.out.println();
        }

        // Danger check: downcasting wrong type
        Content maybeMovie = library[1];
        if (maybeMovie instanceof Movie) {
            Movie m = (Movie) maybeMovie; // safe only if instanceof true
        } else {
```

```
            System.out.println(maybeMovie.getTitle() + " is not a Movie -
cannot access movie-specific features.");
        }
    }
}

class Content {
    protected String title;

    public Content(String title) {
        this.title = title;
    }

    public void play() {
        System.out.println("Now playing: " + title);
    }

    public String getTitle() { return title; }
}

class Movie extends Content {
    private int durationMinutes;
    private String rating;
    private boolean subtitles;

    public Movie(String title, int durationMinutes, String rating, boolean
subtitles) {
        super(title);
        this.durationMinutes = durationMinutes;
        this.rating = rating;
        this.subtitles = subtitles;
    }

    @Override
    public void play() {
        super.play();
        System.out.println("Movie | Duration: " + durationMinutes + " mins
| Rating: " + rating);
    }
```

```
    public boolean hasSubtitles() { return subtitles; }
}

class TVSeries extends Content {
    private int seasons;
    private int episodesPerSeason;

    public TVSeries(String title, int seasons, int episodesPerSeason) {
        super(title);
        this.seasons = seasons;
        this.episodesPerSeason = episodesPerSeason;
    }

    @Override
    public void play() {
        super.play();
        System.out.println("TV Series | Seasons: " + seasons + " |
Episodes/season: " + episodesPerSeason);
    }

    public void suggestNextEpisode() {
        System.out.println("Suggested next episode: S1:E1 (placeholder)");
    }
}

class Documentary extends Content {
    private String[] tags;
    private String related;

    public Documentary(String title, String[] tags, String related) {
        super(title);
        this.tags = tags;
        this.related = related;
    }

    @Override
    public void play() {
        super.play();
        System.out.println("Documentary | Tags: " + String.join(",",
tags));
```

```
    }

    public void showEducationalTags() {
        System.out.println("Educational tags: " + String.join(", ", tags)
+ "; " + related);
    }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\lab-work> cd "e:\JAVA PROGRAMS\steparyansingh\year2\o
ops\week7\lab-work"; java StreamingPlatformDemo

Now playing: Planet Earth
Documentary | Tags: Nature,Wildlife
Educational tags: Nature, Wildlife; Related: Planet Earth II

Strange Shows is not a Movie - cannot access movie-specific features.
```

```
// 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!
```

```
public class SmartCampusDemo {
    public static void main(String[] args) {
        Device[] devices = new Device[3];
        devices[0] = new SmartClassroom("Room 101");
        devices[1] = new SmartLab("Lab A");
        devices[2] = new SmartLibrary("Central Library");

        for (Device d : devices) {
            d.status();
            if (d instanceof SmartClassroom) {
                SmartClassroom c = (SmartClassroom) d;
                c.setLights(true);
                c.setAC(22);
                c.lowerProjector();
            } else if (d instanceof SmartLab) {
                SmartLab l = (SmartLab) d;
                l.checkSafetySystems();
                l.activateEquipment("3D Printer");
            } else if (d instanceof SmartLibrary) {
                SmartLibrary lib = (SmartLibrary) d;
                lib.updateOccupancy(120);
                lib.checkBookAvailability("Data Structures");
            }
            System.out.println();
        }
    }
}

abstract class Device {
    protected String location;

    public Device(String location) {
        this.location = location;
    }

    public void status() {
        System.out.println("Device at " + location + " reporting
status.");
    }
```

```
}

class SmartClassroom extends Device {
    public SmartClassroom(String location) { super(location); }

    public void setLights(boolean on) { System.out.println(location + "
lights set to " + (on ? "ON" : "OFF")); }
    public void setAC(int temp) { System.out.println(location + " AC set
to " + temp + "C"); }
    public void lowerProjector() { System.out.println(location + "
projector lowered."); }
}

class SmartLab extends Device {
    public SmartLab(String location) { super(location); }

    public void checkSafetySystems() { System.out.println(location + "
safety systems OK."); }
    public void activateEquipment(String eq) { System.out.println(location
+ " activated equipment: " + eq); }
}

class SmartLibrary extends Device {
    public SmartLibrary(String location) { super(location); }

    public void updateOccupancy(int count) { System.out.println(location +
" occupancy updated: " + count); }
    public void checkBookAvailability(String title) {
System.out.println(location + " checked availability for: " + title); }
}
```

```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\lab-work> cd "e:\JAVA PROGRAMS\steparyansingh\year2\o
ops\week7\lab-work"; java SmartCampusDemo
Lab A activated equipment: 3D Printer

Device at Central Library reporting status.
Central Library occupancy updated: 120
Central Library checked availability for: Data Structures
```