

**POLYMORPHISM PRACTICE PROBLEMS**

**PRACTICE PROBLEM 1: Gaming Arena - Method Overloading**

**Understanding compile-time polymorphism through method overloading in a gaming context.**

// File: GameBattle.java

import java.util.Arrays;

public class GameBattle {

public void attack(int damage) {

System.out.println("Basic attack for " + damage + " points!");

}

public void attack(int damage, String weapon) {

System.out.println("Attacking with " + weapon + " for " + damage + " points!");

}

public void attack(int damage, String weapon, boolean isCritical) {

if (isCritical) {

System.out.println("CRITICAL HIT! " + weapon + " deals " + (damage \* 2) + " points!");

} else {

attack(damage, weapon);

}

}

public void attack(int damage, String[] teammates) {

String team = Arrays.toString(teammates);

int totalDamage = damage \* teammates.length;

System.out.println("Team attack with " + team + " for " + totalDamage + " total damage!");

}

public static void main(String[] args) {

GameBattle battle = new GameBattle();

System.out.println("--- Gaming Battle Simulation ---");

battle.attack(50);

battle.attack(75, "Sword");

battle.attack(60, "Bow", true);

battle.attack(60, "Bow", false);

String[] team = {"Alice", "Bob"};

battle.attack(40, team);

}

}

OUTPUT::

--- Gaming Battle Simulation ---

Basic attack for 50 points!

Attacking with Sword for 75 points!

CRITICAL HIT! Bow deals 120 points!

Attacking with Bow for 60 points!

Team attack with [Alice, Bob] for 80 total damage!

**PRACTICE PROBLEM 2: Social Media Platform - Method Overriding**

**Demonstrating runtime polymorphism through method overriding in social media context.**

// File: SocialMediaDemo.java

class SocialMediaPost {

protected String content;

protected String author;

public SocialMediaPost(String content, String author) {

this.content = content;

this.author = author;

}

public void share() {

System.out.println("Sharing: \"" + content + "\" by " + author);

}

}

class InstagramPost extends SocialMediaPost {

private int likes;

public InstagramPost(String content, String author, int likes) {

super(content, author);

this.likes = likes;

}

@Override

public void share() {

System.out.println("Instagram: \"" + content + "\" by @" + author + " - " + likes + " likes");

}

}

class TwitterPost extends SocialMediaPost {

private int retweets;

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

super(content, author);

this.retweets = retweets;

}

@Override

public void share() {

System.out.println("Tweet: \"" + content + "\" by @" + author + " - " + retweets + " retweets");

}

}

public class SocialMediaDemo {

public static void main(String[] args) {

SocialMediaPost[] feed = new SocialMediaPost[3];

feed[0] = new InstagramPost("Sunset vibes!", "john\_doe", 245);

feed[1] = new TwitterPost("Java is awesome!", "code\_ninja", 89);

feed[2] = new SocialMediaPost("Hello world!", "beginner");

System.out.println("--- Social Media Feed ---");

for (SocialMediaPost post : feed) {

post.share();

}

}

}

OUTPUT::

--- Social Media Feed ---

Instagram: "Sunset vibes!" by @john\_doe - 245 likes

Tweet: "Java is awesome!" by @code\_ninja - 89 retweets

Sharing: "Hello world!" by beginner

**PRACTICE PROBLEM 3: Food Delivery App - Dynamic Method Dispatch**

**Exploring how JVM resolves method calls at runtime based on actual object type.**

// File: FoodDelivery.java

class Restaurant {

protected String name;

public Restaurant(String name) {

this.name = name;

}

public void prepareFood() {

System.out.println(name + " is preparing generic food.");

}

public void estimateTime() {

System.out.println("Estimated time: 30 minutes");

}

}

class PizzaPlace extends Restaurant {

public PizzaPlace(String name) {

super(name);

}

@Override

public void prepareFood() {

System.out.println(name + " is making delicious pizza with fresh toppings!");

}

@Override

public void estimateTime() {

System.out.println("Pizza ready in 20 minutes!");

}

}

class SushiBar extends Restaurant {

public SushiBar(String name) {

super(name);

}

@Override

public void prepareFood() {

System.out.println(name + " is crafting fresh sushi with precision!");

}

@Override

public void estimateTime() {

System.out.println("Sushi will be ready in 25 minutes!");

}

}

public class FoodDelivery {

public static void main(String[] args) {

Restaurant currentOrder;

System.out.println("--- Ordering Pizza ---");

currentOrder = new PizzaPlace("Mario's Pizza");

currentOrder.prepareFood();

currentOrder.estimateTime();

System.out.println("\n--- Ordering Sushi ---");

currentOrder = new SushiBar("Tokyo Sushi");

currentOrder.prepareFood();

currentOrder.estimateTime();

}

}

OUTPUT::

--- Ordering Pizza ---

Mario's Pizza is making delicious pizza with fresh toppings!

Pizza ready in 20 minutes!

--- Ordering Sushi ---

Tokyo Sushi is crafting fresh sushi with precision!

Sushi will be ready in 25 minutes!

**PRACTICE PROBLEM 4: University System - Upcasting Adventures**

**Learning safe upcasting and accessing inherited members in university context.**

// File: UniversitySystem.java

class Person {

protected String name;

protected int age;

protected String email;

public Person(String name, int age, String email) {

this.name = name;

this.age = age;

this.email = email;

}

public void introduce() {

System.out.println("Hi! I'm " + name + ", " + age + " years old.");

}

public void getContactInfo() {

System.out.println("Email: " + email);

}

}

class Student extends Person {

private String studentId;

private String major;

public Student(String name, int age, String email, String studentId, String major) {

super(name, age, email);

this.studentId = studentId;

this.major = major;

}

public void attendLecture() {

System.out.println(name + " is attending " + major + " lecture.");

}

public void submitAssignment() {

System.out.println("Assignment submitted by " + studentId);

}

}

class Professor extends Person {

private String department;

public Professor(String name, int age, String email, String department) {

super(name, age, email);

this.department = department;

}

public void conductClass() {

System.out.println("Prof. " + name + " is teaching " + department);

}

}

public class UniversitySystem {

public static void main(String[] args) {

Student studentAlice = new Student("Alice", 20, "alice@uni.edu", "CS2021", "Computer Science");

Person person = studentAlice;

System.out.println("--- Accessing through Person reference ---");

person.introduce();

person.getContactInfo();

System.out.println("Accessed name directly: " + person.name);

// person.attendLecture();

}

}

OUTPUT::  
--- Accessing through Person reference ---

Hi! I'm Modi, 60 years old.

Email: modi@gov.in

Accessed name directly: Modi

**PRACTICE PROBLEM 5: Entertainment System - Mastering Downcasting**

**Learning explicit downcasting to access subclass-specific functionality.**

// File: EntertainmentHub.java

class Entertainment {

protected String title;

public Entertainment(String title) {

this.title = title;

}

public void start() {

System.out.println("Starting " + title);

}

public void stop() {

System.out.println("Stopping " + title);

}

}

class Movie extends Entertainment {

private String genre;

public Movie(String title, String genre) {

super(title);

this.genre = genre;

}

public void showSubtitles() {

System.out.println("Showing subtitles for " + title + " (" + genre + ")");

}

public void adjustQuality() {

System.out.println("Adjusting video quality for " + title);

}

}

class Game extends Entertainment {

private String platform;

public Game(String title, String platform) {

super(title);

this.platform = platform;

}

public void saveProgress() {

System.out.println("Saving " + title + " progress on " + platform);

}

public void showLeaderboard() {

System.out.println(title + " leaderboard on " + platform);

}

}

public class EntertainmentHub {

public static void main(String[] args) {

System.out.println("--- Playing a Movie ---");

Entertainment myMedia = new Movie("Avengers", "Action");

myMedia.start();

Movie myMovie = (Movie) myMedia;

myMovie.showSubtitles();

myMovie.adjustQuality();

System.out.println("\n--- Playing a Game ---");

myMedia = new Game("FIFA 24", "PlayStation");

myMedia.start();

Game myGame = (Game) myMedia;

myGame.saveProgress();

myGame.showLeaderboard();

System.out.println("\n--- Demonstrating Wrong Downcast ---");

Entertainment anotherMedia = new Movie("Inception", "Sci-Fi");

try {

Game wrongGame = (Game) anotherMedia;

wrongGame.saveProgress();

} catch (ClassCastException e) {

System.out.println("ERROR: " + e.getMessage());

System.out.println("Cannot cast a Movie object to a Game reference!");

}

}

}

OUTPUT::

--- Playing a Movie ---

Starting Avengers

Showing subtitles for Avengers (Action)

Adjusting video quality for Avengers

--- Playing a Game ---

Starting FIFA 24

Saving FIFA 24 progress on PlayStation

FIFA 24 leaderboard on PlayStation

--- Demonstrating Wrong Downcast ---

ERROR: class Movie cannot be cast to class Game (Movie and Game are in unnamed module of loader 'app')

Cannot cast a Movie object to a Game reference!

**PRACTICE PROBLEM 6: Smart Home - Safe Downcasting with instanceof**

**Mastering the instanceof operator for safe type checking and downcasting.**

**// File: SmartHome.java**

**class SmartDevice {**

**protected String name;**

**public SmartDevice(String name) {**

**this.name = name;**

**}**

**public void turnOn() {**

**System.out.println(name + " is turning on.");**

**}**

**public void turnOff() {**

**System.out.println(name + " is turning off.");**

**}**

**}**

**class SmartLight extends SmartDevice {**

**public SmartLight(String name) {**

**super(name);**

**}**

**public void changeColor(String color) {**

**System.out.println(name + " is changing color to " + color + ".");**

**}**

**}**

**class SmartThermostat extends SmartDevice {**

**public SmartThermostat(String name) {**

**super(name);**

**}**

**public void setTemperature(int degrees) {**

**System.out.println(name + " is setting temperature to " + degrees + " degrees.");**

**}**

**}**

**public class SmartHome {**

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

**SmartDevice[] devices = {**

**new SmartLight("Living Room Light"),**

**new SmartThermostat("Main Thermostat"),**

**new SmartLight("Bedroom Lamp")**

**};**

**System.out.println("--- Controlling Smart Home Devices ---");**

**for (SmartDevice device : devices) {**

**device.turnOn();**

**if (device instanceof SmartLight) {**

**SmartLight light = (SmartLight) device;**

**light.changeColor("Blue");**

**} else if (device instanceof SmartThermostat) {**

**SmartThermostat thermostat = (SmartThermostat) device;**

**thermostat.setTemperature(22);**

**}**

**device.turnOff();**

**System.out.println("----------");**

**}**

**}**

**}**

OUTPUT::

--- Controlling Smart Home Devices ---

Living Room Light is turning on.

Living Room Light is changing color to Blue.

Living Room Light is turning off.

----------

Main Thermostat is turning on.

Main Thermostat is setting temperature to 22 degrees.

Main Thermostat is turning off.

----------

Bedroom Lamp is turning on.

Bedroom Lamp is changing color to Blue.

Bedroom Lamp is turning off.

----------



**Learning Objectives Summary**

**Problem 1 - Method Overloading (Compile-time Polymorphism)**

● Understanding method signatures and parameter differences ● How compiler resolves overloaded methods at compile time ● Practical application in gaming scenarios

**Problem 2 - Method Overriding (Runtime Polymorphism)**

● Overriding inherited methods in subclasses

● Using @Override annotation for clarity

● Social media context makes concepts relatable

**Problem 3 - Dynamic Method Dispatch**

● How JVM resolves method calls at runtime

● Understanding the difference between reference type and object type ● Real-world food delivery scenario

**Problem 4 - Upcasting**

● Safe conversion from subclass to superclass reference ● Accessing inherited members through superclass reference ● Understanding compile-time method binding limitations

**Problem 5 - Downcasting**

● Explicit casting from superclass to subclass reference ● Accessing subclass-specific methods and fields

● Understanding potential ClassCastException risks

**Problem 6 - Safe Downcasting with instanceof**

● Using instanceof operator for type checking

● Preventing runtime exceptions through safe downcasting ● Writing robust and flexible polymorphic code

12

**Challenge Tips**

****

● **Start Simple**: Begin with basic implementations, then add creative touches ● **Test Everything**: Try different scenarios to see polymorphism in action ● **Think Real-World**: These problems simulate actual software scenarios ● **Debug Creatively**: If something doesn't work, experiment with different approaches

● **Time Goal**: Aim to complete all 6 problems within 60 minutes! **Happy Coding!**

13