Polymorphism Assignment - Answers (First 6 Questions)

# Problem 1: Hotel Booking System (Method Overloading)

In this problem, method overloading is used to calculate hotel booking costs in multiple ways. We create overloaded methods with different parameters for standard booking, seasonal booking, corporate booking, and wedding packages. Each method calculates cost accordingly.

Example (Java):

class HotelBooking {  
 void calculateCost(String roomType, int nights) {  
 System.out.println("Standard Booking: " + (nights \* 2000));  
 }  
  
 void calculateCost(String roomType, int nights, double seasonalMultiplier) {  
 System.out.println("Seasonal Booking: " + (nights \* 2000 \* seasonalMultiplier));  
 }  
  
 void calculateCost(String roomType, int nights, double corporateDiscount, boolean mealPackage) {  
 double cost = nights \* 2000;  
 cost -= cost \* corporateDiscount;  
 if(mealPackage) cost += 1000;  
 System.out.println("Corporate Booking: " + cost);  
 }  
  
 void calculateCost(String roomType, int nights, int guestCount, double decorationFee, double cateringCost) {  
 double cost = nights \* 2000 + decorationFee + (guestCount \* cateringCost);  
 System.out.println("Wedding Package: " + cost);  
 }  
}

# Problem 2: Online Learning Platform (Method Overriding)

Here, method overriding is applied where different types of courses show progress in unique ways. A base class 'Course' defines showProgress(), which is overridden in child classes.

Example (Java):

class Course {  
 String title;  
 void showProgress() {  
 System.out.println("Generic course progress...");  
 }  
}  
  
class VideoCourse extends Course {  
 void showProgress() {  
 System.out.println("Video Course: 75% completed, 10 hrs watched");  
 }  
}  
  
class InteractiveCourse extends Course {  
 void showProgress() {  
 System.out.println("Interactive Course: Quiz avg 85%, 5 projects completed");  
 }  
}

# Problem 3: Transportation Fleet Management (Dynamic Method Dispatch)

Dynamic method dispatch allows runtime selection of methods. A base class Vehicle has dispatch(), and each subclass (Bus, Taxi, Train, Bike) overrides it. At runtime, the correct method is executed.

Example (Java):

class Vehicle {  
 void dispatch() {  
 System.out.println("Generic vehicle dispatch.");  
 }  
}  
  
class Bus extends Vehicle {  
 void dispatch() {  
 System.out.println("Bus on fixed route, 40 passenger capacity.");  
 }  
}  
  
class Taxi extends Vehicle {  
 void dispatch() {  
 System.out.println("Taxi available, fare by distance.");  
 }  
}

# Problem 4: Hospital Management System (Upcasting)

In upcasting, a subclass object is referenced by a superclass type. The system manages different medical staff like Doctor, Nurse, Technician, Administrator, but all are referred to as MedicalStaff for common operations.

Example (Java):

class MedicalStaff {  
 void commonOperation() {  
 System.out.println("Shift scheduling and payroll.");  
 }  
}  
  
class Doctor extends MedicalStaff {  
 void diagnose() {  
 System.out.println("Doctor diagnosing...");  
 }  
}  
  
public class Hospital {  
 public static void main(String[] args) {  
 MedicalStaff staff = new Doctor(); // Upcasting  
 staff.commonOperation();  
 }  
}

# Problem 5: Digital Art Gallery (Downcasting)

Downcasting allows accessing subclass-specific features from a superclass reference. For example, if a curator has an Artwork reference, it can be downcast to Painting, Sculpture, etc.

Example (Java):

class Artwork {}  
  
class Painting extends Artwork {  
 void showBrushTechniques() {  
 System.out.println("Painting brush techniques displayed.");  
 }  
}  
  
public class Gallery {  
 public static void main(String[] args) {  
 Artwork art = new Painting(); // Upcasting  
 Painting p = (Painting) art; // Downcasting  
 p.showBrushTechniques();  
 }  
}

# Problem 6: Smart Home Automation (Safe Downcasting with instanceof)

The 'instanceof' operator ensures safe downcasting. Mixed device collections can be processed by first checking the type and then casting to call device-specific features.

Example (Java):

class SmartDevice {}  
  
class SmartTV extends SmartDevice {  
 void controlTV() {  
 System.out.println("TV channel and volume controlled.");  
 }  
}  
  
public class HomeAutomation {  
 public static void main(String[] args) {  
 SmartDevice device = new SmartTV();  
 if (device instanceof SmartTV) {  
 SmartTV tv = (SmartTV) device;  
 tv.controlTV();  
 }  
 }  
}