Week 7 - S7 - Core OOP - Polymorphism - Practice Problem

Name:Ramesh Harisabapathi Chettiar

Submitted on:23/09/25

Qno1->

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

PROGRAM→

```java
public class GameBattle{
    /* TODO: Create an 'attack' method that takes damage (int) and
    prints "Basic attack for [damage] points!" */
    public void attack(int damage){
        System.out.println("Basic attack for " + damage + "points!");
    }

    // TODO: Overload 'attack' method to take damage and weapon name
    // Print "Attacking with [weapon] for [damage] points!"
    public void attack(int damage, String weapon) {
        System.out.println("Attacking with " + weapon + " for " + damage + "points!");
    }
    // TODO: Overload 'attack' method for critical hits (damage, weapon, isCritical)
    // If critical: "CRITICAL HIT! [weapon] deals [damage*2] points!"
    // Else: use the previous overloaded method
    public void attack(int damage, String weapon, boolean isCritical)
    {
        if(isCritical){
            System.out.println("CRITICAL HIT!" + weapon + "deals " + (damage * 2) + " points!");
        }
        attack(damage,weapon);
    }
    // TODO: Overload 'attack' for team attacks (damage, String[] teammates)
    // Print "Team attack with [teammate names] for [damage * team size] total damage!"
    public void attack(int damage,String []teammates){
        System.out.print("Team Attack with ");
        for(String teammate : teammates){
            System.out.print(teammate + " ");
        }
        System.out.println(" for " + (damage * teammates.length) + " total damage! ");
    }
```

```java
    public static void main(String[] args) {
        // TODO: Gaming Battle Simulation:
        // 1. Create a GameBattle object
        // 2. Test all overloaded attack methods:
        // - Basic attack with 50 damage
        // - Sword attack with 75 damage
        // - Critical bow attack with 60 damage
        // - Team attack with {"Alice", "Bob"} for 40 base damage
        // 3. Observe how the compiler chooses the correct method based on parameters
        GameBattle obj = new GameBattle();
        obj.attack(50);
        obj.attack(75,"Sword");
        obj.attack(60,"Bow",true);

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

        obj.attack(40,teammates);


    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise Problems\
ogram1> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise
oblems\Program1\" ; if ($?) { javac GameBattle.java } ; if ($?) { java GameBattle }
Basic attack for 50points!
Attacking with Sword for 75points!
CRITICAL HIT!Bowdeals 120 points!
Attacking with Bow for 60points!
Team Attack with Alice Bob  for 80 total damage!
```

Qno2->

Demonstrating runtime polymorphism through method overriding in social media

context.

PROGRAM→

```java
class SocialMediaPost {
    protected String content;
    protected String author;
    public SocialMediaPost(String content, String author) {
        this.content = content;
        this.author = author;
    }
    /*  TODO: Create 'share()' method that prints
    "Sharing: [content] by [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;
    }
    /*TODO: Override 'share()' to print " Instagram: [content]
    by @[author] - [likes] likes " */
    @Override
    public void share() {
        System.out.println("Instagram: " + content + " by @" + author + " - " + likes + " likes");
    }
}
```

```java
class TwitterPost extends SocialMediaPost {
    private int retweets;
    public TwitterPost(String content, String author, int retweets) {
        super(content, author);
        this.retweets = retweets;
    }
    // TODO: Override 'share()' to print "Tweet: [content] by @[author] - [retweets] retweets "
    @Override
    public void share() {
        System.out.println("Tweet: " + content + " by @" + author + " - " + retweets + " retweets");
    }
}

public class SocialMediaDemo {
    Run main | Debug main
    public static void main(String[] args) {
        // TODO: Social Media Feed Simulation:
        // 1. Create array of SocialMediaPost references
        // 2. Add InstagramPost("Sunset vibes!", "john_doe", 245)
        // 3. Add TwitterPost("Java is awesome!", "code_ninja", 89)
        // 4. Add regular SocialMediaPost("Hello world!", "beginner")
        // 5. Loop through and call share() on each - observe different behaviors!

        SocialMediaPost[] obj = new SocialMediaPost[3];
        obj[0] = new InstagramPost("Sunset vibes!", "john_doe", 245);
        obj[1] = new TwitterPost("Java is awesome!", "code_ninja", 89);
        obj[2] = new SocialMediaPost("Hello world!", "beginner");

        for (int i = 0; i < 3; i++) {
            obj[i].share();
        }
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise Problems\Program2> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise Problems\Program2\" ; if ($?) { javac SocialMediaDemo.java } ; if ($?) { java SocialMediaDemo }
Instagram: Sunset vibes! by @john_doe - 245 likes
Tweet: Java is awesome! by @code_ninja - 89 retweets
Sharing: Hello world! by beginner
```

QNO 3→

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

PROGRAM→

```java
class Restaurant{
    protected String name;
        public Restaurant(String name) {
        this.name = name;
    }

    // TODO: Create 'prepareFood()' method that prints "[name] is preparing generic food"
    public void prepareFood() {
        System.out.println(name + " is preparing generic food.");
    }

    // TODO: Create 'estimateTime()' method that prints "Estimated time: 30 minutes"
    public void estimateTime() {
        System.out.println("Estimated time: 30 minutes");
    }
}

class PizzaPlace extends Restaurant {
    public PizzaPlace(String name) {
        super(name);
    }
    // TODO: Override 'prepareFood()' to print " [name] is making delicious pizza with fresh toppings!"
    @Override
    public void prepareFood() {
        // ... (implementation)
        }
    // TODO: Override 'estimateTime()' to print "Pizza ready in 20 minutes! "
    @Override
    public void estimateTime() {
        System.out.println("Pizza ready on 20 minutes! ");
    }
}
```

```java
class SushiBar extends Restaurant {
    public SushiBar(String name) {
        super(name);
    }
// TODO: Override 'prepareFood()' to print "[name] is crafting fresh sushi with precision!"
    @Override
    public void prepareFood() {
        // ... (implementation)
        }
    // TODO: Override 'estimateTime()' to print "Sushi will be ready in 25 minutes! "
    @Override
    public void estimateTime() {
        System.out.println("Sushi will be ready in 25 minutes! ");
        }
}

public class FoodDelivery{
    Run main | Debug main
    public static void main(String args[]){
        // TODO: Dynamic Food Ordering System:
        // 1. Create a Restaurant reference variable
        // 2. Assign new PizzaPlace("Mario's Pizza") to it
        // 3. Call prepareFood() and estimateTime() - observe Pizza methods execute
        // 4. Reassign to new SushiBar("Tokyo Sushi")
        // 5. Call same methods again - observe Sushi methods execute
        // 6. Explain how JVM knows which method to call at runtime!
        Restaurant obj = new Restaurant("Mario's Pizza");
        obj.prepareFood();
        obj.estimateTime();
        obj = new SushiBar("Tokyo Sushi");
        obj.prepareFood();
        obj.estimateTime();
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise Problems\Program3> cd "c:\Users\Ram
esh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise Problems\Program3\" ; if ($?) { javac FoodDelivery
.java } ; if ($?) { java FoodDelivery }
Mario's Pizza is preparing generic food.
Estimated time: 30 minutes
Sushi will be ready in 25 minutes!
```

The JVM uses dynamic method dispatch to determine which method to call at runtime. It relies on the actual object's type, not the reference type, to invoke the correct overridden method—this enables polymorphism. So even if you upcast, the JVM still calls the subclass's version of the method.

QNO 4→

Learning safe upcasting and accessing inherited members in university context.

PROGRAM→

```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;
    }

    // TODO: Create 'introduce()' method that prints "Hi! I'm [name], [age] years old."
    public void introduce() {
        System.out.println("Hi! I'm " + name + ", " + age + " years old.");
    }

    // TODO: Create 'getContactInfo()' that prints "Email: [email]"
    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;
    }

    // TODO: Create 'attendLecture()' method that prints "[name] is attending [major] lecture"
    public void attendLecture() {
        System.out.println(name + " is attending " + major + " lecture");
    }

    // TODO: Create 'submitAssignment()' that prints "Assignment submitted by [studentId]"
    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;
    }

    // TODO: Create 'conductClass()' that prints "Prof. [name] is teaching [department]"
    public void conductClass() {
        System.out.println("Prof. " + name + " is teaching " + department);
    }
}
```

```java
public class UniversitySystem {
    Run main | Debug main
    public static void main(String[] args) {
        // TODO: University Registration Demo:
        // 1. Create Student("Alice", 20, "alice@uni.edu", "CS2021", "Computer Science")
        // 2. Upcast Student to Person reference: Person p = new Student(...)
        // 3. Call introduce() and getContactInfo() using Person reference
        // 4. Try calling attendLecture() with Person reference - observe compile error
        // 5. Access the 'name' field through Person reference
        // 6. Explain why upcasting is safe but limits access to subclass-specific methods
        Student obj = new Student("Alice", 20, "alice@uni.edu", "CS2021", "Computer Science");
        Person p = new Student("Alice", 20, "alice@uni.edu", "CS2021", "Computer Science");
        p.introduce();
        p.getContactInfo();
        //p.attendLecture();
        System.out.println(p.name);
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise Problems\Program4> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 7\Practise Problems\Program4\" ; if ($?) { javac UniversitySystem.java } ; if ($?) { java UniversitySystem }
Hi! I'm Alice, 20 years old.
Email: alice@uni.edu
Alice
```

Upcasting is safe because a subclass object always "is-a" superclass object, preserving type compatibility. However, it limits access to subclass-specific methods since the reference type only exposes the superclass's interface. You can regain access via downcasting if needed.