











Week 3_ Inheritance and Polymorphism

Table of Content

Week 3: Inheritance and Polymorphism

- Teaching Guide & Hands-on Project
-  Session Overview
-  Learning Objectives
-  Quick Review (5 minutes)
-  Detailed Lesson Plan (120 Minutes)
 - Part 1: Introduction to Inheritance (20 minutes)
 - Part 2: The super Keyword (10 minutes)
 - Part 3: Method Overriding vs Method Overloading (15 minutes)
 - Part 4: Polymorphism (15 minutes)
 - Part 5: Abstract Classes (15 minutes)
-  HANDS-ON PROJECT: Enhanced Student Management System (45 minutes)
 - Project Overview
 - Step-by-Step Implementation
-  Homework Assignment
 - Task 1: Add Graduate Student (Required)
 - Task 2: Method Overloading Challenge (Required)
 - Task 3: Department Class (Optional Challenge)
-  Assessment Checklist
-  Common Student Errors & Solutions
 - Error 1: Missing super() call
 - Error 2: Trying to instantiate abstract class
 - Error 3: Method signature mismatch
 - Error 4: Access modifier more restrictive
-  Additional Resources
-  Preview of Week 4

-  Key Takeaways

Week 3: Inheritance and Polymorphism

Teaching Guide & Hands-on Project



Session Overview

Duration: 2 - 3 Hours

Teaching: 75 minutes | **Hands-on Practice:** 45 minutes



Learning Objectives

By the end of this session, students will be able to: 1. Understand and implement inheritance using the extends keyword 2. Use the super keyword to access parent class members 3. Differentiate between method overriding and method overloading 4. Understand and apply polymorphism 5. Create and use abstract classes and methods 6. Build a hierarchical class structure for a Student Management System 7. Implement different user types (Student, Teacher, Admin)



Quick Review (5 minutes)

Start with questions: - Who completed the Student Management System? - Any challenges with encapsulation or constructors? - Quick poll: Who added the Student ID feature?

Quick Review Points: - Classes are blueprints, objects are instances - Encapsulation = private attributes + public getters/setters - Constructors initialize objects



Detailed Lesson Plan (120 Minutes)

Part 1: Introduction to Inheritance (20 minutes)

What is Inheritance? (5 minutes)

Real-world analogy:

Think of inheritance like family traits:

Parent (Animal)

- ├ has name
- ├ can eat()
- └ can sleep()

Children inherit from parent:

- ├ Dog (has name, can eat, sleep, AND bark)
- ├ Cat (has name, can eat, sleep, AND meow)
- └ Bird (has name, can eat, sleep, AND fly)

Why use inheritance? - Code reusability (Don't Repeat Yourself - DRY principle) - Establish relationships between classes - Create hierarchical class structures - Easier maintenance and updates

Basic Inheritance Syntax (15 minutes)

Example 1: Simple Inheritance - Animal Hierarchy

```
// Parent class (Base class / Superclass)
public class Animal {
    // Attributes
    protected String name; // 'protected' allows access by child classes
    protected int age;

    // Constructor
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Methods
    public void eat() {
        System.out.println(name + " is eating.");
    }
}
```

```

    }                public void sleep() {          System.out.println(name + " is sleeping.");
    }                public void displayInfo() {      System.out.println("Name: " + name);
                    System.out.println("Age: " + age);    }    }

```

```

// Child class (Derived class / Subclass)
public class Dog extends Animal {
    private String breed;

    // Constructor
    public Dog(String name, int age, String breed) {
        super(name, age); // Call parent constructor
        this.breed = breed;
    }

    // Dog-specific method
    public void bark() {
        System.out.println(name + " says: Woof! Woof!");
    }

    // Override parent method
    @Override
    public void displayInfo() {
        super.displayInfo(); // Call parent's displayInfo
        System.out.println("Breed: " + breed);
        System.out.println("Type: Dog");
    }
}

```

```

// Another child class
public class Cat extends Animal {
    private String color;

    public Cat(String name, int age, String color) {
        super(name, age);
        this.color = color;
    }

    public void meow() {
        System.out.println(name + " says: Meow!");
    }
}

```

```

        @Override    public void displayInfo() {        super.displayInfo();
        System.out.println("Color: " + color);        System.out.println("Type: Cat");    }
    }

```

Using the classes:

```

public class Main {
    public static void main(String[] args) {
        // Create a Dog object
        Dog dog = new Dog("Buddy", 3, "Golden Retriever");
        dog.displayInfo();
        dog.eat();        // Inherited from Animal
        dog.sleep();        // Inherited from Animal
        dog.bark();        // Dog-specific method

        System.out.println("\n" + "=".repeat(30) + "\n");

        // Create a Cat object
        Cat cat = new Cat("Whiskers", 2, "Orange");
        cat.displayInfo();
        cat.eat();        // Inherited from Animal
        cat.sleep();        // Inherited from Animal
        cat.meow();        // Cat-specific method
    }
}

```

Output:

```

Name: Buddy
Age: 3
Breed: Golden Retriever
Type: Dog
Buddy is eating.
Buddy is sleeping.
Buddy says: Woof! Woof!

=====

Name: Whiskers
Age: 2
Color: Orange

```

```
Type: Cat
Whiskers is eating.
Whiskers is sleeping.
Whiskers says: Meow!
```

Key Concepts: - extends keyword creates inheritance relationship - super() calls parent constructor (must be first line in child constructor) - super.methodName() calls parent's method - protected allows access by child classes but not outside - @Override annotation (optional but recommended)

Part 2: The `super` Keyword (10 minutes)

Understanding `super` (10 minutes)

The `super` keyword has three uses:

1. **Call parent constructor:** `super(parameters)`
2. **Access parent methods:** `super.methodName()`
3. **Access parent attributes:** `super.attributeName`

Example 2: Using `super` in different ways

```
public class Person {
    protected String name;
    protected int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void introduce() {
        System.out.println("Hi, I'm " + name + " and I'm " + age + " years old.");
    }
}
```

```
public class Student extends Person {
    private String studentId;
    private double gpa;

    public Student(String name, int age, String studentId, double gpa) {
```

```

        this.studentId = studentId;        this.gpa = gpa;    }                                @Override
public void introduce() {        super.introduce(); // 2. Call parent method
    System.out.println("I'm a student with ID: " + studentId);
    System.out.println("My GPA is: " + gpa);    }                                public void study() {
// 3. Access parent attribute
    System.out.println(super.name + " is studying hard!");    }    }

```

Important Rules: - super() must be the first statement in constructor - If you don't call super(), Java automatically calls parent's no-arg constructor - If parent has no no-arg constructor, you MUST explicitly call super()

Part 3: Method Overriding vs Method Overloading (15 minutes)

Clear Distinction (5 minutes)

Draw comparison table on screen:

METHOD OVERLOADING	METHOD OVERRIDING
Same class	Parent-child relationship
Same method name	Same method name
Different parameters	Same parameters
Compile-time	Runtime
Adds new behavior	Changes inherited behavior

Method Overriding in Detail (10 minutes)

Example 3: Method Overriding Rules

```

public class Shape {
    protected String color;

    public Shape(String color) {
        this.color = color;
    }

    // Method to be overridden
    public double calculateArea() {

```

```
        return 0.0;    }                public void display() {  
        System.out.println("Shape color: " + color);    }    }
```

```
public class Circle extends Shape {  
    private double radius;  
  
    public Circle(String color, double radius) {  
        super(color);  
        this.radius = radius;  
    }  
  
    // Override parent method  
    @Override  
    public double calculateArea() {  
        return Math.PI * radius * radius;  
    }  
  
    @Override  
    public void display() {  
        super.display(); // Call parent's display  
        System.out.println("Circle radius: " + radius);  
        System.out.println("Circle area: " + calculateArea());  
    }  
}
```

```
public class Rectangle extends Shape {  
    private double length;  
    private double width;  
  
    public Rectangle(String color, double length, double width) {  
        super(color);  
        this.length = length;  
        this.width = width;  
    }  
  
    @Override  
    public double calculateArea() {  
        return length * width;  
    }  
}
```



```

@Override    public void display() {        super.display();
    System.out.println("Rectangle dimensions: " + length + " x " + width);
    System.out.println("Rectangle area: " + calculateArea());    }    }

```

Testing:

```

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle("Red", 5.0);
        circle.display();

        System.out.println("\n" + "=".repeat(30) + "\n");

        Rectangle rectangle = new Rectangle("Blue", 4.0, 6.0);
        rectangle.display();
    }
}

```

Rules for Method Overriding: 1. Must have exact same method signature (name and parameters) 2. Return type must be same or subtype (covariant return type) 3. Access modifier cannot be more restrictive 4. Cannot override final methods 5. Cannot override static methods (they are hidden, not overridden)

Part 4: Polymorphism (15 minutes)

What is Polymorphism? (5 minutes)

Definition: - "Poly" = many, "morph" = forms - One interface, multiple implementations - Ability to treat objects of different classes uniformly

Types of Polymorphism: 1. **Compile-time (Static):** Method overloading 2. **Runtime (Dynamic):** Method overriding

Runtime Polymorphism in Action (10 minutes)

Example 4: Polymorphism Demonstration

```

public class Employee {
    protected String name;
    protected String id;
    protected double baseSalary;
}

```

```

public Employee(String name, String id, double baseSalary) {      this.name = name;
    this.id = id;          this.baseSalary = baseSalary;    }
// Method to be overridden    public double calculateSalary() {      return baseSalary;
}                               public void displayInfo() {      System.out.println("Name: " + name);
    System.out.println("ID: " + id);
    System.out.println("Salary: $" + calculateSalary());    }    }

```

```

public class FullTimeEmployee extends Employee {
    private double bonus;

    public FullTimeEmployee(String name, String id, double baseSalary, double bonus) {
        super(name, id, baseSalary);
        this.bonus = bonus;
    }

    @Override
    public double calculateSalary() {
        return baseSalary + bonus;
    }
}

```

```

public class PartTimeEmployee extends Employee {
    private int hoursWorked;
    private double hourlyRate;

    public PartTimeEmployee(String name, String id, int hoursWorked, double hourlyRate) {
        super(name, id, 0); // No base salary
        this.hoursWorked = hoursWorked;
        this.hourlyRate = hourlyRate;
    }

    @Override
    public double calculateSalary() {
        return hoursWorked * hourlyRate;
    }
}

```

```

public class Contractor extends Employee {
    private double projectFee;

```

```

        public Contractor(String name, String id, double projectFee) {
            super(name, id, 0);          this.projectFee = projectFee;    }                @Override
        public double calculateSalary() {          return projectFee;    }    }

```

Polymorphism in Action:

```

public class PayrollSystem {
    public static void main(String[] args) {
        // Polymorphism: Parent reference, child objects
        Employee emp1 = new FullTimeEmployee("Alice", "FT001", 50000, 5000);
        Employee emp2 = new PartTimeEmployee("Bob", "PT001", 120, 25);
        Employee emp3 = new Contractor("Charlie", "CT001", 75000);

        // Array of Employee references
        Employee[] employees = {emp1, emp2, emp3};

        System.out.println("=== PAYROLL REPORT ===\n");

        double totalPayroll = 0;
        for (Employee employee : employees) {
            employee.displayInfo(); // Calls appropriate version based on actual object type
            totalPayroll += employee.calculateSalary();
            System.out.println("-".repeat(30));
        }

        System.out.println("\nTotal Payroll: $" + totalPayroll);
    }
}

```

Output:

```

=== PAYROLL REPORT ===

```

```

Name: Alice

```

```

ID: FT001

```

```

Salary: $55000.0

```

```

Name: Bob

```

```

ID: PT001

```

```

Salary: $3000.0

```

Name: Charlie
ID: CT001
Salary: \$75000.0

Total Payroll: \$133000.0

Key Polymorphism Concepts: - Parent reference can hold child object: `Employee emp = new FullTimeEmployee(...)` - Method called is determined at runtime based on actual object type - Enables writing flexible, extensible code - Foundation of many design patterns

Part 5: Abstract Classes (15 minutes)

What are Abstract Classes? (5 minutes)

Concept: - Cannot be instantiated (cannot create objects directly) - Serves as a template for other classes - Can have both abstract methods (no implementation) and concrete methods (with implementation) - Used when you want to provide a common base with some shared implementation

When to use abstract classes: - When classes share common behavior but also have unique implementations - When you want to enforce certain methods in child classes - When you want to provide default implementations for some methods

Creating and Using Abstract Classes (10 minutes)

Example 5: Abstract Class Implementation

```
// Abstract class
public abstract class Vehicle {
    protected String brand;
    protected String model;
    protected int year;

    public Vehicle(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    // Abstract method (no implementation) - must be overridden
    public abstract void start();
}
```

```

        // Abstract method    public abstract void stop();
// Concrete method (with implementation) - can be inherited as-is
public void displayInfo() {        System.out.println("Brand: " + brand);
    System.out.println("Model: " + model);        System.out.println("Year: " + year);
}
        // Concrete method    public void honk() {
    System.out.println("Beep beep!");    }    }

```

```

public class Car extends Vehicle {
    private int numberOfDoors;

    public Car(String brand, String model, int year, int numberOfDoors) {
        super(brand, model, year);
        this.numberOfDoors = numberOfDoors;
    }

    @Override
    public void start() {
        System.out.println("Car engine starting... Vroom!");
    }

    @Override
    public void stop() {
        System.out.println("Car engine stopping... Engine off.");
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Number of doors: " + numberOfDoors);
        System.out.println("Type: Car");
    }
}

```

```

public class Motorcycle extends Vehicle {
    private boolean hasSidecar;

    public Motorcycle(String brand, String model, int year, boolean hasSidecar) {
        super(brand, model, year);
        this.hasSidecar = hasSidecar;
    }
}

```

```

        @Override    public void start() {
            System.out.println("Motorcycle engine starting... Vroom vroom!");    }
    @Override    public void stop() {
        System.out.println("Motorcycle engine stopping... Quiet now.");    }
    @Override    public void displayInfo() {        super.displayInfo();
        System.out.println("Has sidecar: " + (hasSidecar ? "Yes" : "No"));
        System.out.println("Type: Motorcycle");    }    }

```

Using Abstract Classes:

```

public class Main {
    public static void main(String[] args) {
        // Cannot do this: Vehicle v = new Vehicle(...); // Error!

        // Can use polymorphism with abstract class
        Vehicle car = new Car("Toyota", "Camry", 2023, 4);
        Vehicle motorcycle = new Motorcycle("Harley Davidson", "Street 750", 2022, false);

        System.out.println("=== Car ===");
        car.displayInfo();
        car.start();
        car.honk();
        car.stop();

        System.out.println("\n=== Motorcycle ===");
        motorcycle.displayInfo();
        motorcycle.start();
        motorcycle.honk();
        motorcycle.stop();
    }
}

```

Abstract Class Rules: - Use abstract keyword for class and methods - Cannot instantiate abstract classes - Child classes MUST implement all abstract methods (or be abstract themselves) - Can have constructors (called by child classes using `super()`) - Can have instance variables - Can mix abstract and concrete methods

HANDS-ON PROJECT: Enhanced Student Management System (45 minutes)

Project Overview

Extend the Week 2 Student Management System to support multiple user types: 1. **Person** (abstract base class) 2. **Student** (extends Person) 3. **Teacher** (extends Person) 4. **Admin** (extends Person)

Each user type has unique attributes and behaviors, but shares common person information.

Step-by-Step Implementation

Step 1: Abstract Person Class

```
public abstract class Person {
    protected String name;
    protected int age;
    protected String id;
    protected String email;

    public Person(String name, int age, String id, String email) {
        this.name = name;
        setAge(age);
        this.id = id;
        this.email = email;
    }

    // Abstract method - each person type displays info differently
    public abstract void displayInfo();

    // Abstract method - each person type has different role
    public abstract String getRole();

    // Concrete methods shared by all
    public String getName() {
        return name;
    }
}
```

```

public void setName(String name) {      this.name = name;    }
public int getAge() {      return age;    }      public void setAge(int age) {
    if (age >= 0 && age <= 120) {      this.age = age;    } else {
        System.out.println("Invalid age!");      this.age = 0;    }    }
    public String getId() {      return id;    }
public String getEmail() {      return email;    }
public void setEmail(String email) {      this.email = email;    }    }

```

Step 2: Student Class

```

public class Student extends Person {
    private String major;
    private double gpa;
    private int creditsCompleted;

    public Student(String name, int age, String id, String email, String major, double gpa) {
        super(name, age, id, email);
        this.major = major;
        setGpa(gpa);
        this.creditsCompleted = 0;
    }

    @Override
    public void displayInfo() {
        System.out.println("┌──────────────────────────────────┐");
        System.out.println("│          STUDENT INFORMATION          │");
        System.out.println("└──────────────────────────────────┘");
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Student ID: " + id);
        System.out.println("Email: " + email);
        System.out.println("Major: " + major);
        System.out.println("GPA: " + gpa);
        System.out.println("Credits Completed: " + creditsCompleted);
        System.out.println("Status: " + getAcademicStatus());
    }

    @Override
    public String getRole() {
        return "Student";
    }
}

```



```

        public double getGpa() { return gpa; }
    public void setGpa(double gpa) { if (gpa >= 0.0 && gpa <= 4.0) {
        this.gpa = gpa; } else {
        System.out.println("Invalid GPA! Must be 0.0-4.0"); this.gpa = 0.0;
    } }
        public String getMajor() { return major; }
        public void setMajor(String major) { this.major = major; }
    public int getCreditsCompleted() { return creditsCompleted; }
    public void addCredits(int credits) { if (credits > 0) {
        this.creditsCompleted += credits;
        System.out.println("Added " + credits + " credits. Total: " + creditsCompleted);
    } }
        public boolean isHonorRoll() { return gpa >= 3.5;
    }
        public String getAcademicStatus() {
    if (creditsCompleted < 30) return "Freshman";
    else if (creditsCompleted < 60) return "Sophomore";
    else if (creditsCompleted < 90) return "Junior"; else return "Senior";
    }
}

```

Step 3: Teacher Class

```

public class Teacher extends Person {
    private String department;
    private String subject;
    private double salary;
    private int yearsOfExperience;

    public Teacher(String name, int age, String id, String email,
        String department, String subject, double salary, int yearsOfExperience) {
        super(name, age, id, email);
        this.department = department;
        this.subject = subject;
        this.salary = salary;
        this.yearsOfExperience = yearsOfExperience;
    }

    @Override
    public void displayInfo() {
        System.out.println("┌───────────────────────────────────┐");
        System.out.println("│                                TEACHER INFORMATION                                │");
        System.out.println("└───────────────────────────────────┘");
        System.out.println("Name: " + name);
    }
}

```

```

        System.out.println("Teacher ID: " + id);          System.out.println("Email: " + email);
        System.out.println("Department: " + department);
        System.out.println("Subject: " + subject);
        System.out.println("Salary: $" + salary);
        System.out.println("Years of Experience: " + yearsOfExperience);
        System.out.println("Level: " + getTeacherLevel());    }           @Override
public String getRole() {          return "Teacher";    }
public String getDepartment() {          return department;    }
public String getSubject() {          return subject;    }
public double getSalary() {          return salary;    }
public void setSalary(double salary) {          if (salary > 0) {
            this.salary = salary;          }    }
public void giveRaise(double percentage) {
    double raise = salary * (percentage / 100);          salary += raise;
    System.out.println("Salary increased by " + percentage + "%. New salary: $" + salary);
}
    public int getYearsOfExperience() {          return yearsOfExperience;
}
    public String getTeacherLevel() {
    if (yearsOfExperience < 3) return "Junior Teacher";
    else if (yearsOfExperience < 10) return "Senior Teacher";
    else return "Master Teacher";    }    }

```

Step 4: Admin Class

```

public class Admin extends Person {
    private String position;
    private String department;
    private String[] permissions;

    public Admin(String name, int age, String id, String email,
        String position, String department, String[] permissions) {
        super(name, age, id, email);
        this.position = position;
        this.department = department;
        this.permissions = permissions;
    }

    @Override
    public void displayInfo() {
        System.out.println("=====");
        System.out.println("||          ADMIN INFORMATION          ||");
    }
}

```

```

        System.out.println("Name: " + name);           System.out.println("Age: " + age);
        System.out.println("Admin ID: " + id);         System.out.println("Email: " + email);
        System.out.println("Position: " + position);
        System.out.println("Department: " + department);
        System.out.print("Permissions: ");
        for (int i = 0; i < permissions.length; i++) {
            System.out.print(permissions[i]);
            if (i < permissions.length - 1) System.out.print(", ");
        }
        System.out.println();
        @Override public String getRole() {
        return "Admin";
        } public String getPosition() {
        return position;
        } public boolean hasPermission(String permission) {
        for (String perm : permissions) {
            if (perm.equalsIgnoreCase(permission)) {
                return true;
            }
        }
        return false;
        }
    public void listPermissions() {
        System.out.println("Permissions for " + name + ":");
        for (String permission : permissions) {
            System.out.println("    - " + permission);
        }
    }
}

```

Step 5: University Management System

```

import java.util.ArrayList;
import java.util.Scanner;

public class UniversityManagementSystem {
    private ArrayList<Person> people;
    private Scanner scanner;

    public UniversityManagementSystem() {
        people = new ArrayList<>();
        scanner = new Scanner(System.in);
    }

    public void displayMenu() {
        System.out.println("\n┌───────────────────────────────────────────┐");
        System.out.println("│ UNIVERSITY MANAGEMENT SYSTEM │");
        System.out.println("└───────────────────────────────────────────┘");
        System.out.println("1. Add Student");
        System.out.println("2. Add Teacher");
        System.out.println("3. Add Admin");
        System.out.println("4. Display All People");
        System.out.println("5. Search Person by ID");
    }
}

```

```

        System.out.println("7. Display Honor Roll Students");
        System.out.println("8. Exit");
        System.out.println("_____");
        System.out.print("Enter your choice (1-8): ");    }

    public void addStudent() {        System.out.println("\n--- Add New Student ---");
        System.out.print("Name: ");        String name = scanner.nextLine();
        System.out.print("Age: ");        int age = scanner.nextInt();
        scanner.nextLine();        System.out.print("Student ID: ");
        String id = scanner.nextLine();        System.out.print("Email: ");
        String email = scanner.nextLine();        System.out.print("Major: ");
        String major = scanner.nextLine();        System.out.print("GPA (0.0-4.0): ");
        double gpa = scanner.nextDouble();        scanner.nextLine();
        Student student = new Student(name, age, id, email, major, gpa);
        people.add(student);        System.out.println("✓ Student added successfully!");
    }

    public void addTeacher() {
        System.out.println("\n--- Add New Teacher ---");        System.out.print("Name: ");
        String name = scanner.nextLine();        System.out.print("Age: ");
        int age = scanner.nextInt();        scanner.nextLine();
        System.out.print("Teacher ID: ");        String id = scanner.nextLine();
        System.out.print("Email: ");        String email = scanner.nextLine();
        System.out.print("Department: ");        String department = scanner.nextLine();
        System.out.print("Subject: ");        String subject = scanner.nextLine();
        System.out.print("Salary: ");        double salary = scanner.nextDouble();
        System.out.print("Years of Experience: ");        int experience = scanner.nextInt();
        scanner.nextLine();
        Teacher teacher = new Teacher(name, age, id, email, department, subject, salary,
            experience);
        people.add(teacher);        System.out.println("✓ Teacher added successfully!");
    }

    public void addAdmin() {
        System.out.println("\n--- Add New Admin ---");        System.out.print("Name: ");
        String name = scanner.nextLine();        System.out.print("Age: ");
        int age = scanner.nextInt();        scanner.nextLine();
        System.out.print("Admin ID: ");        String id = scanner.nextLine();
        System.out.print("Email: ");        String email = scanner.nextLine();
        System.out.print("Position: ");        String position = scanner.nextLine();
        System.out.print("Department: ");        String department = scanner.nextLine();
        System.out.print("Number of permissions: ");        int numPerms = scanner.nextInt();
        scanner.nextLine();        String[] permissions = new String[numPerms];
        for (int i = 0; i < numPerms; i++) {
            System.out.print("Permission " + (i + 1) + ": ");
            permissions[i] = scanner.nextLine();        }
    }

```

```

Admin admin = new Admin(name, age, id, email, position, department, permissions);
people.add(admin);          System.out.println("✓ Admin added successfully!");    }

    public void displayAllPeople() {
System.out.println("\n--- All People in System ---");          if (people.isEmpty()) {
    System.out.println("No people in the system.");              return;              }
        for (int i = 0; i < people.size(); i++) {
            System.out.println("\nPerson #" + (i + 1) + " [" + people.get(i).getRole() + "]");
            people.get(i).displayInfo();          System.out.println("=".repeat(40));
        }          System.out.println("Total people: " + people.size());    }
public void searchById() {          System.out.println("\n--- Search Person by ID ---");
System.out.print("Enter ID: ");          String searchId = scanner.nextLine();
boolean found = false;          for (Person person : people) {
    if (person.getId().equalsIgnoreCase(searchId)) {
        System.out.println("\n✓ Person found:");                person.displayInfo();
        found = true;          break;          }          }
if (!found) {          System.out.println("X Person not found.");          }          }
    public void displayByRole() {
System.out.println("\n--- Filter by Role ---");
System.out.println("1. Students only");          System.out.println("2. Teachers only");
System.out.println("3. Admins only");          System.out.print("Choose (1-3): ");
int choice = scanner.nextInt();          scanner.nextLine();
String roleFilter = "";          switch (choice) {
    case 1: roleFilter = "Student"; break;
    case 2: roleFilter = "Teacher"; break;
    case 3: roleFilter = "Admin"; break;          default:
        System.out.println("Invalid choice!");                return;          }
System.out.println("\n--- " + roleFilter + "s ---");          int count = 0;
for (Person person : people) {          if (person.getRole().equals(roleFilter)) {
    person.displayInfo();          System.out.println("=".repeat(40));
    count++;          }          }          if (count == 0) {
    System.out.println("No " + roleFilter.toLowerCase() + "s found.");          } else {
    System.out.println("Total " + roleFilter.toLowerCase() + "s: " + count);          }
}          public void displayHonorRoll() {
System.out.println("\n--- Honor Roll Students (GPA >= 3.5) ---");          int count = 0;
    for (Person person : people) {
        // Use instanceof to check object type          if (person instanceof Student) {
            Student student = (Student) person; // Cast to Student
            if (student.isHonorRoll()) {                student.displayInfo();
                System.out.println("=".repeat(40));                count++;
            }          }          }          if (count == 0) {
                System.out.println("No students on honor roll.");          } else {

```

```

        System.out.println("Total honor roll students: " + count);    }    }

    public void run() {
        System.out.println("Welcome to University Management System!");
        int choice;        do {            displayMenu();
            choice = scanner.nextInt();            scanner.nextLine();
            switch (choice) {                case 1:                    addStudent();
                    break;                case 2:                    addTeacher();
                    break;                case 3:                    addAdmin();
                    break;                case 4:                    displayAllPeople();
                    break;                case 5:                    searchById();
                    break;                case 6:                    displayByRole();
                    break;                case 7:                    displayHonorRoll();
                    break;                case 8:
                        System.out.println("\nThank you for using University Management System!");
                        System.out.println("Goodbye!");                    break;
                    default:
                        System.out.println("\nX Invalid choice! Please enter 1-8.");            }
        } while (choice != 8);            scanner.close();    }

    public static void main(String[] args) {
        UniversityManagementSystem ums = new UniversityManagementSystem();        ums.run();
    }
}

```



Homework Assignment

Task 1: Add Graduate Student (Required)

Create a GraduateStudent class that extends Student: - Add attributes: thesisTitle, advisor (teacher name) - Override getAcademicStatus() to return "Graduate Student" - Add method: defendThesis() that displays thesis info - Update management system to handle graduate students

Task 2: Method Overloading Challenge (Required)

Add overloaded methods to the Student class:

```

// Add single credit
public void addCredits(int credits)

// Add credits with course name
public void addCredits(int credits, String courseName)

```

```
// Add credits with course name and grade
public void addCredits(int credits, String courseName, char grade)
```

Task 3: Department Class (Optional Challenge)

Create a Department class that: - Has department name and code - Contains ArrayList of Teachers - Has methods to add/remove teachers - Can calculate average salary of teachers - Can list all teachers in department

Assessment Checklist

Students should be able to: - [] Implement inheritance using extends - [] Use super keyword in constructors and methods - [] Differentiate between overriding and overloading - [] Override methods with @Override annotation - [] Understand and apply polymorphism - [] Create and use abstract classes - [] Use instanceof to check object types - [] Cast objects appropriately - [] Design class hierarchies - [] Apply access modifiers (protected)

Common Student Errors & Solutions

Error 1: Missing super() call

```
public Student(String name, int age) {
    // Error: no super() and Person has no default constructor
    this.studentId = "12345";
}
```

Solution: Add super(name, age, ...); as first line

Error 2: Trying to instantiate abstract class

```
Person p = new Person(...); // Error!
```

Solution: Use concrete subclass: Person p = new Student(...);

Error 3: Method signature mismatch

```
// In parent: public void display()
@Override
public void display(String name) // Not an override! Different signature
```

Solution: Match signature exactly for overriding

Error 4: Access modifier more restrictive

```
// Parent: public void method()
@Override
private void method() // Error! Cannot be more restrictive
```

Solution: Use same or less restrictive modifier



Additional Resources

- [Oracle Java Inheritance Tutorial](#)
- [Polymorphism Explained](#)
- [Abstract Classes vs Interfaces](#)



Preview of Week 4

Next week we'll cover: - **Interfaces:** Pure abstraction - **Differences:** Abstract classes vs Interfaces - **Collections Framework:** ArrayList, HashSet, HashMap - **When to use what:** Design decisions

Prepare: Complete the homework and review inheritance concepts!



Key Takeaways

"Inheritance = IS-A relationship"

A Dog IS-A Animal

“Polymorphism = One interface, many forms”

Same method call, different behavior based on object type

“Abstract Class = Partial blueprint”

Some things defined, some things left for children to decide

“super = Access to parent’s members”

Constructor, methods, and attributes