Table of Content

Week 2: Object-Oriented Programming Essentials

Teaching Guide & Hands-on Project

📋 Session Overview

Duration: 2 - 3 Hours
Teaching: 70 minutes | **Hands-on Practice:** 50 minutes

🎯 Learning Objectives

By the end of this session, students will be able to:

1. Understand the difference between classes and objects
2. Create classes with attributes and methods
3. Use constructors to initialize objects
4. Implement method overloading
5. Apply encapsulation using access modifiers
6. Use the `this` keyword appropriately
7. Build a Student Management System (console-based)

📚 Quick Review (5 minutes)

Start with quick questions:

- Who completed the calculator homework?
- Any challenges with methods or loops?
- Quick demo: Ask a volunteer to show their enhanced calculator

---

🕐 Detailed Lesson Plan (120 Minutes)

Part 1: Introduction to Classes and Objects (20 minutes)

What is Object-Oriented Programming? (5 minutes)

Explain the concept:

- Until now, we've written everything in the `main` method
- Real-world applications need to model real-world entities
- OOP helps organize code around "objects" that represent things

Real-world analogy:

```
Class = Blueprint (like a car blueprint)
Object = Actual instance (like your actual car)

Class "Car" defines:
- Properties: color, model, year, speed
- Behaviors: start(), stop(), accelerate()

Objects:
```

```
- myCar (red, Toyota, 2020)
- yourCar (blue, Honda, 2022)
```

Creating Your First Class (15 minutes)

Example 1: Simple Student Class

```java
// Student.java
public class Student {
    // Attributes (instance variables)
    String name;
    int age;
    String major;
    double gpa;

    // Method to display student information
    void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Major: " + major);
        System.out.println("GPA: " + gpa);
    }

    // Method to check if student is on honor roll
    boolean isHonorRoll() {
        return gpa >= 3.5;
    }
}
```

Using the Student Class:

```java
// Main.java
public class Main {
    public static void main(String[] args) {
        // Creating objects (instances) of Student class
        Student student1 = new Student();
        student1.name = "Alice Johnson";
        student1.age = 20;
        student1.major = "Computer Science";
        student1.gpa = 3.8;

        Student student2 = new Student();
        student2.name = "Bob Smith";
        student2.age = 22;
        student2.major = "Mathematics";
        student2.gpa = 3.2;

        // Using the objects
        System.out.println("=== Student 1 ===");
```

```
        student1.displayInfo();
        if (student1.isHonorRoll()) {
            System.out.println("Status: Honor Roll");
        }

        System.out.println("\n=== Student 2 ===");
        student2.displayInfo();
        if (student2.isHonorRoll()) {
            System.out.println("Status: Honor Roll");
        } else {
            System.out.println("Status: Regular");
        }
    }
}
```

Key Points to Emphasize:

- Class is defined outside of main (separate file or above main class)
- Use `new` keyword to create objects
- Each object has its own copy of attributes
- Access attributes and methods using dot notation: `object.attribute`

Part 2: Constructors (15 minutes)

What are Constructors? (3 minutes)

Explain:

- Special method that initializes objects
- Same name as the class
- No return type (not even void)
- Called automatically when object is created
- Makes object creation cleaner and ensures proper initialization

Default Constructor vs Parameterized Constructor (12 minutes)

Example 2: Student Class with Constructors

```
// Student.java
public class Student {
    // Attributes
    String name;
    int age;
    String major;
    double gpa;

    // Default constructor (no parameters)
    public Student() {
        name = "Unknown";
```

```
        age = 0;
        major = "Undeclared";
        gpa = 0.0;
        System.out.println("Default constructor called");
    }

    // Parameterized constructor (with parameters)
    public Student(String name, int age, String major, double gpa) {
        this.name = name;
        this.age = age;
        this.major = major;
        this.gpa = gpa;
        System.out.println("Parameterized constructor called");
    }

    // Another constructor (partial parameters)
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
        this.major = "Undeclared";
        this.gpa = 0.0;
    }

    void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Major: " + major);
        System.out.println("GPA: " + gpa);
    }

    boolean isHonorRoll() {
        return gpa >= 3.5;
    }
}
```

Using Different Constructors:

```
public class Main {
    public static void main(String[] args) {
        // Using default constructor
        Student student1 = new Student();
        System.out.println("=== Student 1 (Default) ===");
        student1.displayInfo();

        // Using parameterized constructor
        Student student2 = new Student("Alice Johnson", 20, "Computer Science",
3.8);
        System.out.println("\n=== Student 2 (Full Constructor) ===");
        student2.displayInfo();

        // Using partial constructor
        Student student3 = new Student("Bob Smith", 22);
```

```
        System.out.println("\n=== Student 3 (Partial Constructor) ===");
        student3.displayInfo();
    }
}
```

Important Concept: The `this` Keyword

Draw on screen or explain:

```
this.name = name;
  ↑           ↑
  |           |
  |           Parameter (local variable)
  |
  Instance variable (belongs to the object)
```

Key Points:

- `this` refers to the current object
- Used to distinguish between instance variables and parameters with same name
- Without `this`, parameter shadows instance variable

Part 3: Method Overloading (10 minutes)

Understanding Method Overloading (10 minutes)

Explain:

- Multiple methods with the same name but different parameters
- Compiler chooses which method to call based on arguments
- Makes code more intuitive and flexible

Example 3: Calculator with Method Overloading

```
public class Calculator {

    // Add two integers
    public int add(int a, int b) {
        System.out.println("Adding two integers");
        return a + b;
    }

    // Add three integers
    public int add(int a, int b, int c) {
        System.out.println("Adding three integers");
        return a + b + c;
    }
```

```
    // Add two doubles
    public double add(double a, double b) {
        System.out.println("Adding two doubles");
        return a + b;
    }

    // Add two strings (concatenation)
    public String add(String a, String b) {
        System.out.println("Concatenating two strings");
        return a + b;
    }

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

        System.out.println("Result: " + calc.add(5, 10));           // Calls int
version
        System.out.println("Result: " + calc.add(5, 10, 15));       // Calls three
int version
        System.out.println("Result: " + calc.add(5.5, 10.3));       // Calls
double version
        System.out.println("Result: " + calc.add("Hello ", "World")); // Calls
String version
    }
}
```

Output:

```
Adding two integers
Result: 15
Adding three integers
Result: 30
Adding two doubles
Result: 15.8
Concatenating two strings
Result: Hello World
```

Rules for Method Overloading:

1. Methods must have different parameter lists (number or type)
2. Return type alone is NOT enough
3. Parameter names don't matter, only types

Part 4: Encapsulation and Access Modifiers (20 minutes)

Why Encapsulation? (5 minutes)

Problem Scenario:

```
Student student = new Student("Alice", 20, "CS", 3.8);
student.gpa = -5.0;  // This shouldn't be allowed!
student.age = -10;   // This is invalid!
```

Solution: Encapsulation

- Hide internal details
- Provide controlled access through methods
- Validate data before setting

Access Modifiers (5 minutes)

Explain the four access levels:

```
public    - Accessible from anywhere
private   - Only within the same class
protected - Within same package and subclasses (cover in Week 3)
default   - Within same package (no modifier)
```

For now, focus on public and private:

- **public:** Use for methods that others should access
- **private:** Use for attributes to protect data

Getters and Setters (10 minutes)

Example 4: Encapsulated Student Class

```
public class Student {
    // Private attributes (cannot be accessed directly from outside)
    private String name;
    private int age;
    private String major;
    private double gpa;

    // Constructor
    public Student(String name, int age, String major, double gpa) {
        this.name = name;
        setAge(age);      // Use setter for validation
        this.major = major;
        setGpa(gpa);      // Use setter for validation
    }

    // Getter methods (read access)
    public String getName() {
        return name;
    }
```

```
    public int getAge() {
        return age;
    }

    public String getMajor() {
        return major;
    }

    public double getGpa() {
        return gpa;
    }

    // Setter methods (write access with validation)
    public void setName(String name) {
        if (name != null && !name.trim().isEmpty()) {
            this.name = name;
        } else {
            System.out.println("Invalid name!");
        }
    }

    public void setAge(int age) {
        if (age >= 16 && age <= 100) {
            this.age = age;
        } else {
            System.out.println("Invalid age! Must be between 16 and 100.");
            this.age = 18; // Default value
        }
    }

    public void setMajor(String major) {
        this.major = major;
    }

    public void setGpa(double gpa) {
        if (gpa >= 0.0 && gpa <= 4.0) {
            this.gpa = gpa;
        } else {
            System.out.println("Invalid GPA! Must be between 0.0 and 4.0.");
            this.gpa = 0.0; // Default value
        }
    }

    // Public method to display info
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Major: " + major);
        System.out.println("GPA: " + gpa);
    }

    public boolean isHonorRoll() {
        return gpa >= 3.5;
```

```
    }
}
```

Using the Encapsulated Class:

```
public class Main {
    public static void main(String[] args) {
        Student student = new Student("Alice", 20, "Computer Science", 3.8);

        // Cannot do this anymore (compilation error):
        // student.name = "Bob";     // Error: name has private access

        // Must use getters and setters
        System.out.println("Student Name: " + student.getName());

        // Try to set invalid values
        student.setGpa(5.0);    // Will show error message
        student.setAge(-5);     // Will show error message

        // Set valid values
        student.setGpa(3.9);
        student.setAge(21);

        System.out.println("\n=== Updated Student Info ===");
        student.displayInfo();
    }
}
```

Benefits of Encapsulation:

- Data validation
- Data protection
- Flexibility to change internal implementation
- Better control over data

💻 HANDS-ON PROJECT: Student Management System (50 minutes)

Project Overview

Build a console-based Student Management System that can:

1. Add new students
2. Display all students
3. Search for a student by name
4. Update student GPA
5. Delete a student
6. Display honor roll students (GPA >= 3.5)
7. Exit the program

Step-by-Step Implementation

Step 1: Enhanced Student Class (Already covered above)

Use the encapsulated Student class from Part 4.

Step 2: StudentManagementSystem Class

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

public class StudentManagementSystem {
    // ArrayList to store students
    private ArrayList<Student> students;
    private Scanner scanner;

    // Constructor
    public StudentManagementSystem() {
        students = new ArrayList<>();
        scanner = new Scanner(System.in);
    }

    // Method to display menu
    public void displayMenu() {
        System.out.println("\n╔═══════════════════════════════════════════╗");
        System.out.println("║   STUDENT MANAGEMENT SYSTEM               ║");
        System.out.println("╚═══════════════════════════════════════════╝");
        System.out.println("1. Add New Student");
        System.out.println("2. Display All Students");
        System.out.println("3. Search Student by Name");
        System.out.println("4. Update Student GPA");
        System.out.println("5. Delete Student");
        System.out.println("6. Display Honor Roll Students");
        System.out.println("7. Exit");
        System.out.println("─────────────────────────────────────────────");
        System.out.print("Enter your choice (1-7): ");
    }

    // Method to add a new student
    public void addStudent() {
        System.out.println("\n--- Add New Student ---");

        System.out.print("Enter name: ");
        String name = scanner.nextLine();

        System.out.print("Enter age: ");
        int age = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        System.out.print("Enter major: ");
```

```
        String major = scanner.nextLine();

        System.out.print("Enter GPA (0.0-4.0): ");
        double gpa = scanner.nextDouble();
        scanner.nextLine(); // Consume newline

        Student student = new Student(name, age, major, gpa);
        students.add(student);

        System.out.println("✓ Student added successfully!");
    }

    // Method to display all students
    public void displayAllStudents() {
        System.out.println("\n--- All Students ---");

        if (students.isEmpty()) {
            System.out.println("No students in the system.");
            return;
        }

        for (int i = 0; i < students.size(); i++) {
            System.out.println("\nStudent #" + (i + 1));
            students.get(i).displayInfo();
            System.out.println("──────────────────────────");
        }

        System.out.println("Total students: " + students.size());
    }

    // Method to search student by name
    public void searchStudent() {
        System.out.println("\n--- Search Student ---");
        System.out.print("Enter student name: ");
        String searchName = scanner.nextLine();

        boolean found = false;
        for (Student student : students) {
            if (student.getName().equalsIgnoreCase(searchName)) {
                System.out.println("\n✓ Student found:");
                student.displayInfo();
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("✗ Student not found.");
        }
    }

    // Method to update student GPA
    public void updateGPA() {
        System.out.println("\n--- Update Student GPA ---");
```

```
        System.out.print("Enter student name: ");
        String searchName = scanner.nextLine();

        boolean found = false;
        for (Student student : students) {
            if (student.getName().equalsIgnoreCase(searchName)) {
                System.out.println("\nCurrent GPA: " + student.getGpa());
                System.out.print("Enter new GPA (0.0-4.0): ");
                double newGpa = scanner.nextDouble();
                scanner.nextLine(); // Consume newline

                student.setGpa(newGpa);
                System.out.println("✓ GPA updated successfully!");
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("✗ Student not found.");
        }
    }

    // Method to delete a student
    public void deleteStudent() {
        System.out.println("\n--- Delete Student ---");
        System.out.print("Enter student name: ");
        String searchName = scanner.nextLine();

        boolean found = false;
        for (int i = 0; i < students.size(); i++) {
            if (students.get(i).getName().equalsIgnoreCase(searchName)) {
                System.out.print("Are you sure you want to delete " +
                                 students.get(i).getName() + "? (yes/no): ");
                String confirm = scanner.nextLine();

                if (confirm.equalsIgnoreCase("yes")) {
                    students.remove(i);
                    System.out.println("✓ Student deleted successfully!");
                } else {
                    System.out.println("Deletion cancelled.");
                }
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("✗ Student not found.");
        }
    }

    // Method to display honor roll students
    public void displayHonorRoll() {
```

```
        System.out.println("\n--- Honor Roll Students (GPA >= 3.5) ---");

        boolean hasHonorStudents = false;
        for (Student student : students) {
            if (student.isHonorRoll()) {
                student.displayInfo();
                System.out.println("————————————————————————");
                hasHonorStudents = true;
            }
        }

        if (!hasHonorStudents) {
            System.out.println("No students on honor roll.");
        }
    }

    // Method to run the system
    public void run() {
        System.out.println("Welcome to Student Management System!");

        int choice;
        do {
            displayMenu();
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    addStudent();
                    break;
                case 2:
                    displayAllStudents();
                    break;
                case 3:
                    searchStudent();
                    break;
                case 4:
                    updateGPA();
                    break;
                case 5:
                    deleteStudent();
                    break;
                case 6:
                    displayHonorRoll();
                    break;
                case 7:
                    System.out.println("\nThank you for using Student Management
System!");
                    System.out.println("Goodbye!");
                    break;
                default:
                    System.out.println("\n✗ Invalid choice! Please enter 1-7.");
            }
```

```
        } while (choice != 7);

        scanner.close();
    }

    // Main method
    public static void main(String[] args) {
        StudentManagementSystem sms = new StudentManagementSystem();
        sms.run();
    }
}
```

Sample Program Execution

```
Welcome to Student Management System!

╔═══════════════════════════════════════╗
║   STUDENT MANAGEMENT SYSTEM           ║
╚═══════════════════════════════════════╝
1. Add New Student
2. Display All Students
3. Search Student by Name
4. Update Student GPA
5. Delete Student
6. Display Honor Roll Students
7. Exit
─────────────────────────────────────────
Enter your choice (1-7): 1

--- Add New Student ---
Enter name: Alice Johnson
Enter age: 20
Enter major: Computer Science
Enter GPA (0.0-4.0): 3.8
✓ Student added successfully!

╔═══════════════════════════════════════╗
║   STUDENT MANAGEMENT SYSTEM           ║
╚═══════════════════════════════════════╝
1. Add New Student
2. Display All Students
3. Search Student by Name
4. Update Student GPA
5. Delete Student
6. Display Honor Roll Students
7. Exit
─────────────────────────────────────────
Enter your choice (1-7): 2

--- All Students ---
```

```
Student #1
Name: Alice Johnson
Age: 20
Major: Computer Science
GPA: 3.8
--------------------
Total students: 1
```

Homework Assignment

Task 1: Add Student ID (Required)

Enhance the Student class to include:

1. A private `studentId` attribute (String)
2. Modify constructor to accept student ID
3. Add getter and setter for student ID
4. Update the Student Management System to handle student IDs

Task 2: Add Course Management (Challenge)

Create a `Course` class with:

- Course code (e.g., "CS101")
- Course name (e.g., "Introduction to Programming")
- Credits (e.g., 3)
- Proper encapsulation

Add to Student class:

- ArrayList of enrolled courses
- Method to enroll in a course
- Method to drop a course
- Method to display enrolled courses
- Method to calculate total credits

Assessment Checklist

Students should be able to:

- [ ] Create classes with attributes and methods
- [ ] Understand the difference between class and object
- [ ] Write default and parameterized constructors
- [ ] Use the `this` keyword correctly
- [ ] Implement method overloading
- [ ] Apply access modifiers (public/private)
- [ ] Create getters and setters

- [ ] Validate data in setters
- [ ] Use ArrayList to store objects
- [ ] Build a functional console application with multiple classes

---

🔧 Common Student Errors & Solutions

Error 1: Constructor Not Found

```
Student s = new Student();  // Error if no default constructor exists
```

Solution: Either add default constructor or use parameterized one

Error 2: Forgetting `this`

```
public Student(String name) {
    name = name;  // Wrong! Just assigns parameter to itself
}
```

Solution: Use `this.name = name;`

Error 3: Scanner Buffer Issue

```
int age = scanner.nextInt();
String name = scanner.nextLine();  // Gets empty string!
```

Solution: Add `scanner.nextLine()` after `nextInt()`

📚 Additional Resources

- Oracle Java OOP Tutorial
- Encapsulation Explained
- Practice: Create a `BankAccount` class with deposit/withdraw methods

🔜 Preview of Week 3

Next week we'll cover:

- **Inheritance:** Creating class hierarchies
- **Polymorphism:** One interface, multiple implementations
- **Abstract classes:** Templates for other classes
- **The `super` keyword**

Prepare: Make sure your Student Management System is working perfectly!

📌 Key Takeaways

> **"A class is a blueprint, an object is the actual house built from that blueprint."**

> **"Encapsulation = Data hiding + Controlled access"**

> **"Constructor's job = Initialize the object properly"**