Table of Contents

Week 4: Interfaces and Java Collections Framework

Teaching Guide & Hands-on Project

📋 Session Overview

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

🎯 Learning Objectives

By the end of this session, students will be able to:
1. Understand and create interfaces in Java
2. Differentiate between abstract classes and interfaces
3. Implement multiple interfaces in a single class
4. Understand the Java Collections Framework hierarchy
5. Use ArrayList, LinkedList, HashSet, and HashMap effectively
6. Choose appropriate collection types for different scenarios
7. Iterate through collections using different methods
8. Build an enhanced Student Management System using collections

📚 Quick Review (5 minutes)

Quick Questions:
- Who completed the University Management System?
- What's the difference between method overriding and overloading?
- Can you instantiate an abstract class? Why or why not?
- What keyword do we use for inheritance?

Key Review Points:
- Inheritance creates IS-A relationships
- Abstract classes provide partial implementation
- Polymorphism allows treating different objects uniformly

🕐 Detailed Lesson Plan (120 Minutes)

Part 1: Introduction to Interfaces (20 minutes)

What is an Interface? (5 minutes)

Definition:
- A contract or blueprint that defines what a class should do, but not how
- 100% abstract (before Java 8)
- Cannot have instance variables (only constants)
- Cannot be instantiated
- Used to achieve multiple inheritance and loose coupling

Real-world analogy:

```
Think of a "Drivable" interface:

Interface Drivable:
  - start()
  - stop()
  - accelerate()
  - brake()

Classes that implement Drivable:
  - Car
  - Motorcycle
  - Truck
  - Bus

All must implement these methods, but each does it differently.
```

Creating and Implementing Interfaces (15 minutes)

Example 1: Basic Interface

```java
// Interface definition
public interface Printable {
    // Abstract method (public abstract by default)
    void print();

    // Constant (public static final by default)
    String DEFAULT_FORMAT = "PDF";
}
```

```java
// Implementing the interface
public class Document implements Printable {
    private String title;
    private String content;

    public Document(String title, String content) {
        this.title = title;
        this.content = content;
    }

    @Override
    public void print() {
        System.out.println("=== Printing Document ===");
        System.out.println("Title: " + title);
        System.out.println("Content: " + content);
        System.out.println("Format: " + DEFAULT_FORMAT);
    }
}
```

```java
public class Photo implements Printable {
    private String filename;
    private int width;
    private int height;

    public Photo(String filename, int width, int height) {
        this.filename = filename;
        this.width = width;
        this.height = height;
    }

    @Override
    public void print() {
        System.out.println("=== Printing Photo ===");
        System.out.println("Filename: " + filename);
        System.out.println("Dimensions: " + width + "x" + height);
        System.out.println("Format: " + DEFAULT_FORMAT);
    }
}
```

Using the interface:

```java
public class Main {
    public static void main(String[] args) {
        // Polymorphism with interfaces
        Printable doc = new Document("Java Guide", "This is a comprehensive guide...");
        Printable photo = new Photo("sunset.jpg", 1920, 1080);
```

```
        // Array of Printable items        Printable[] items = {doc, photo};
        for (Printable item : items) {            item.print();
            System.out.println();        }    }    }
```

Key Points:
- Interface methods are `public abstract` by default
- Interface variables are `public static final` by default
- Use `implements` keyword to implement an interface
- Must implement ALL methods from the interface

Part 2: Multiple Inheritance with Interfaces (15 minutes)

Why Multiple Inheritance? (5 minutes)

The Diamond Problem in Java:

```
Java doesn't allow:
class C extends A, B  // Not allowed!

But allows:
class C implements A, B  // Allowed!
```

Implementing Multiple Interfaces (10 minutes)

Example 2: Multiple Interface Implementation

```
// First interface
public interface Flyable {
    void fly();
    void land();
}
```

```
// Second interface
public interface Swimmable {
    void swim();
    void dive();
}
```

```
// Third interface
public interface Walkable {
    void walk();
```

```
    void run();}
```

```
// Duck can do all three!
public class Duck implements Flyable, Swimmable, Walkable {
    private String name;

    public Duck(String name) {
        this.name = name;
    }

    @Override
    public void fly() {
        System.out.println(name + " is flying in the sky!");
    }

    @Override
    public void land() {
        System.out.println(name + " is landing on the ground.");
    }

    @Override
    public void swim() {
        System.out.println(name + " is swimming in the water.");
    }

    @Override
    public void dive() {
        System.out.println(name + " is diving underwater!");
    }

    @Override
    public void walk() {
        System.out.println(name + " is walking on land.");
    }

    @Override
    public void run() {
        System.out.println(name + " is running quickly!");
    }
}
```

```
// Fish can only swim
public class Fish implements Swimmable {
    private String species;

    public Fish(String species) {
        this.species = species;
    }
```

```
    @Override    public void swim() {
        System.out.println(species + " is swimming gracefully.");    }
    @Override    public void dive() {        System.out.println(species + " is diving deep!");
    }    }
```

```
// Bird can fly and walk
public class Bird implements Flyable, Walkable {
    private String type;

    public Bird(String type) {
        this.type = type;
    }

    @Override
    public void fly() {
        System.out.println(type + " is soaring through the air!");
    }

    @Override
    public void land() {
        System.out.println(type + " is landing on a branch.");
    }

    @Override
    public void walk() {
        System.out.println(type + " is hopping around.");
    }

    @Override
    public void run() {
        System.out.println(type + " is running fast!");
    }
}
```

Testing:

```
public class Main {
    public static void main(String[] args) {
        Duck duck = new Duck("Donald");
        System.out.println("=== Duck ===");
        duck.swim();
        duck.fly();
        duck.walk();

        System.out.println("\n=== Fish ===");
        Fish fish = new Fish("Goldfish");
        fish.swim();
        fish.dive();
```

```
        System.out.println("\n=== Bird ===");        Bird bird = new Bird("Eagle");
        bird.fly();        bird.land();        bird.walk();    }    }
```

Part 3: Abstract Classes vs Interfaces (10 minutes)

Clear Comparison (10 minutes)

Draw comparison table:

ABSTRACT CLASS	INTERFACE
Can have constructors	Cannot have constructors
Can have instance variables	Only constants (static final)
Can have concrete methods	Only abstract methods*
Single inheritance (extends)	Multiple inheritance (impl.)
Can have any access modifier	Methods are public by default
Use: IS-A relationship	Use: CAN-DO capability
Example: Animal, Vehicle	Example: Flyable, Printable

* Java 8+ allows default and static methods in interfaces

When to Use Which?

Use Abstract Class when:
- Classes share common code
- You want to provide default implementations
- You need instance variables
- You need constructors
- Related classes in hierarchy (IS-A relationship)

Use Interface when:
- Unrelated classes need same behavior
- You want multiple inheritance
- Defining a contract/capability (CAN-DO)
- Want to achieve loose coupling

Example 3: Combined Use

```
// Abstract class for common animal properties
public abstract class Animal {
    protected String name;
    protected int age;
```

```
    public Animal(String name, int age) {        this.name = name;        this.age = age;
    }                // Concrete method    public void sleep() {
        System.out.println(name + " is sleeping.");    }                // Abstract method
    public abstract void makeSound();}
```

```
// Interface for flying capability
public interface Flyable {
    void fly();
}
```

```
// Interface for swimming capability
public interface Swimmable {
    void swim();
}
```

```
// Parrot: IS-A Animal, CAN-DO Flyable
public class Parrot extends Animal implements Flyable {

    public Parrot(String name, int age) {
        super(name, age);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " says: Squawk!");
    }

    @Override
    public void fly() {
        System.out.println(name + " is flying with colorful wings!");
    }
}
```

```
// Penguin: IS-A Animal, CAN-DO Swimmable (but cannot fly!)
public class Penguin extends Animal implements Swimmable {

    public Penguin(String name, int age) {
        super(name, age);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " says: Honk honk!");
    }

    @Override
```

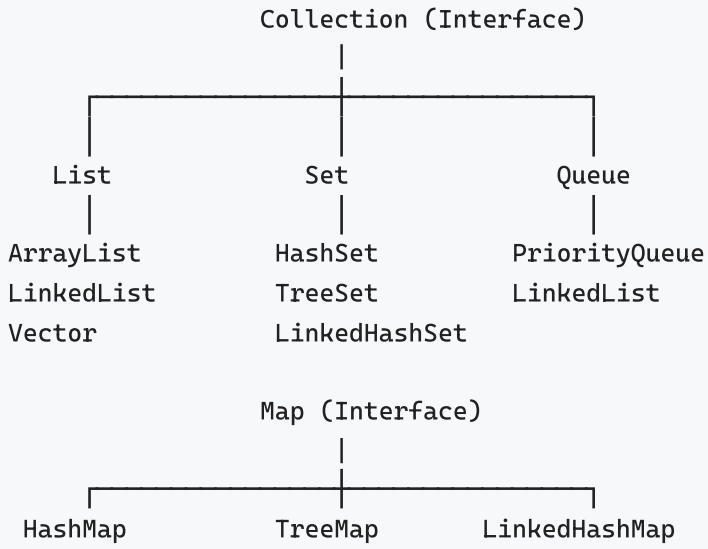
```
    public void swim() {        System.out.println(name + " is swimming underwater!");
    }    }
```

Part 4: Java Collections Framework Introduction (25 minutes)

What is Collections Framework? (5 minutes)

Explain:
- Set of classes and interfaces for storing and manipulating groups of objects
- Provides ready-to-use data structures
- More powerful and flexible than arrays
- Part of `java.util` package

Collections Hierarchy Overview:



ArrayList - Dynamic Arrays (8 minutes)

Example 4: ArrayList Basics

```
import java.util.ArrayList;

public class ArrayListDemo {
    public static void main(String[] args) {
        // Creating ArrayList
        ArrayList<String> fruits = new ArrayList<>();

        // Adding elements
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");
        fruits.add("Mango");
```

```
                System.out.println("Fruits: " + fruits);
        System.out.println("Size: " + fruits.size());                // Accessing elements
        System.out.println("First fruit: " + fruits.get(0));
        System.out.println("Last fruit: " + fruits.get(fruits.size() - 1));
        // Adding at specific position        fruits.add(1, "Grapes");
        System.out.println("After insertion: " + fruits);
        // Checking if element exists        if (fruits.contains("Banana")) {
            System.out.println("We have bananas!");        }
        // Removing elements        fruits.remove("Orange");  // Remove by value
        fruits.remove(0);          // Remove by index
        System.out.println("After removal: " + fruits);
        // Iterating through ArrayList        System.out.println("\n=== All Fruits ===");
        for (String fruit : fruits) {            System.out.println("- " + fruit);        }
                // Clear all elements        fruits.clear();
        System.out.println("After clear, size: " + fruits.size());
        System.out.println("Is empty? " + fruits.isEmpty());    }    }
```

ArrayList of Objects:

```
import java.util.ArrayList;

class Student {
    String name;
    double gpa;

    public Student(String name, double gpa) {
        this.name = name;
        this.gpa = gpa;
    }

    @Override
    public String toString() {
        return name + " (GPA: " + gpa + ")";
    }
}

public class StudentListDemo {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();

        students.add(new Student("Alice", 3.8));
        students.add(new Student("Bob", 3.2));
        students.add(new Student("Charlie", 3.9));

        System.out.println("All students:");
        for (Student student : students) {
            System.out.println(student);
        }

        // Find student with highest GPA
```

```
        Student topStudent = students.get(0);        for (Student student : students) {
            if (student.gpa > topStudent.gpa) {                topStudent = student;
            }        }        System.out.println("\nTop student: " + topStudent);    }
}
```

Key ArrayList Methods:

- `add(element)` - Add element
- `add(index, element)` - Insert at position
- `get(index)` - Get element at index
- `set(index, element)` - Replace element
- `remove(index)` or `remove(object)` - Remove element
- `size()` - Get number of elements
- `contains(element)` - Check if exists
- `clear()` - Remove all elements
- `isEmpty()` - Check if empty

HashSet - Unique Elements (6 minutes)

Example 5: HashSet Basics

```java
import java.util.HashSet;

public class HashSetDemo {
    public static void main(String[] args) {
        // Creating HashSet
        HashSet<String> cities = new HashSet<>();

        // Adding elements
        cities.add("New York");
        cities.add("London");
        cities.add("Tokyo");
        cities.add("Paris");
        cities.add("London");  // Duplicate - won't be added!

        System.out.println("Cities: " + cities);
        System.out.println("Size: " + cities.size());  // Still 4, not 5

        // Check if element exists
        if (cities.contains("Tokyo")) {
            System.out.println("Tokyo is in the set!");
        }

        // Remove element
        cities.remove("Paris");
        System.out.println("After removal: " + cities);

        // Iterate through HashSet
```

```
            System.out.println("- " + city);        }
        // NOTE: Order is not guaranteed in HashSet!    }    }
```

Removing Duplicates from ArrayList:

```
import java.util.ArrayList;
import java.util.HashSet;

public class RemoveDuplicatesDemo {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(5);
        numbers.add(3);
        numbers.add(7);
        numbers.add(3);  // duplicate
        numbers.add(9);
        numbers.add(5);  // duplicate

        System.out.println("Original list: " + numbers);
        System.out.println("Size: " + numbers.size());

        // Remove duplicates using HashSet
        HashSet<Integer> uniqueNumbers = new HashSet<>(numbers);

        System.out.println("Unique numbers: " + uniqueNumbers);
        System.out.println("Size: " + uniqueNumbers.size());

        // Convert back to ArrayList if needed
        ArrayList<Integer> cleanList = new ArrayList<>(uniqueNumbers);
        System.out.println("Clean list: " + cleanList);
    }
}
```

Key HashSet Features:
- No duplicate elements
- No guaranteed order
- Fast lookup/add/remove operations
- Allows one `null` value

HashMap - Key-Value Pairs (6 minutes)

Example 6: HashMap Basics

```
import java.util.HashMap;

public class HashMapDemo {
    public static void main(String[] args) {
        // Creating HashMap (Key → Value)
```

```
        HashMap<String, Integer> ages = new HashMap<>();
        // Adding key-value pairs        ages.put("Alice", 25);        ages.put("Bob", 30);
        ages.put("Charlie", 28);        ages.put("Diana", 22);
        System.out.println("Ages: " + ages);                // Getting value by key
        System.out.println("Alice's age: " + ages.get("Alice"));
        // Check if key exists        if (ages.containsKey("Bob")) {
            System.out.println("Bob is in the map!");        }
        // Check if value exists        if (ages.containsValue(28)) {
            System.out.println("Someone is 28 years old!");        }
        // Update value        ages.put("Alice", 26);  // Updates existing key
        System.out.println("Updated age: " + ages.get("Alice"));
        // Remove entry        ages.remove("Charlie");
        System.out.println("After removal: " + ages);
        // Iterate through HashMap        System.out.println("\n=== All Entries ===");
        for (String name : ages.keySet()) {
            System.out.println(name + " is " + ages.get(name) + " years old");        }
                // Alternative iteration
        System.out.println("\n=== Using entrySet ===");
        for (var entry : ages.entrySet()) {
            System.out.println(entry.getKey() + " → " + entry.getValue());        }    }
}
```

Practical HashMap Example:

```
import java.util.HashMap;
import java.util.Scanner;

public class PhonebookDemo {
    public static void main(String[] args) {
        HashMap<String, String> phonebook = new HashMap<>();
        Scanner scanner = new Scanner(System.in);

        // Add some contacts
        phonebook.put("Alice", "555-1234");
        phonebook.put("Bob", "555-5678");
        phonebook.put("Charlie", "555-9012");

        System.out.println("=== Simple Phonebook ===");
        System.out.print("Enter name to lookup: ");
        String name = scanner.nextLine();

        if (phonebook.containsKey(name)) {
            System.out.println(name + "'s number: " + phonebook.get(name));
        } else {
            System.out.println(name + " not found in phonebook.");
        }

        scanner.close();
    }
}
```

Key HashMap Methods:
- `put(key, value)` - Add or update entry
- `get(key)` - Get value for key
- `containsKey(key)` - Check if key exists
- `containsValue(value)` - Check if value exists
- `remove(key)` - Remove entry
- `keySet()` - Get all keys
- `values()` - Get all values
- `entrySet()` - Get all key-value pairs

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

Project Overview

Refactor the Week 3 University Management System using:
1. **Interfaces** for common behaviors
2. **Collections** (ArrayList, HashMap) for efficient data storage
3. **Enhanced features** using collection operations

New Features:

- Search students by various criteria
- Sort students by different attributes
- Statistical reports
- Course enrollment system
- Performance tracking

Step-by-Step Implementation

Step 1: Define Interfaces

```
// Interface for entities that can be searched
public interface Searchable {
    boolean matches(String keyword);
}
```

```java
// Interface for entities that can be displayed
public interface Displayable {
    void displayInfo();
    String getRole();
}
```

```java
// Interface for comparable entities (for sorting)
public interface Comparable<T> {
    int compareTo(T other);
}
```

Step 2: Updated Person Class

```java
public abstract class Person implements Searchable, Displayable {
    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;
        this.age = age;
        this.id = id;
        this.email = email;
    }

    // Implementing Searchable interface
    @Override
    public boolean matches(String keyword) {
        String lowerKeyword = keyword.toLowerCase();
        return name.toLowerCase().contains(lowerKeyword) ||
               id.toLowerCase().contains(lowerKeyword) ||
               email.toLowerCase().contains(lowerKeyword);
    }

    // Getters
    public String getName() { return name; }
    public int getAge() { return age; }
    public String getId() { return id; }
    public String getEmail() { return email; }

    // Setters
    public void setName(String name) { this.name = name; }
    public void setAge(int age) { this.age = age; }
```

```
    public void setEmail(String email) { this.email = email; }}
```

Step 3: Enhanced Student Class with Course Management

```
import java.util.ArrayList;
import java.util.HashMap;

public class Student extends Person {
    private String major;
    private double gpa;
    private ArrayList<String> enrolledCourses;
    private HashMap<String, String> courseGrades;  // Course → Grade

    public Student(String name, int age, String id, String email, String major, double gpa) {
        super(name, age, id, email);
        this.major = major;
        this.gpa = gpa;
        this.enrolledCourses = new ArrayList<>();
        this.courseGrades = new HashMap<>();
    }

    @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("Enrolled Courses: " + enrolledCourses.size());

        if (!enrolledCourses.isEmpty()) {
            System.out.println("\nCourses:");
            for (String course : enrolledCourses) {
                String grade = courseGrades.get(course);
                System.out.println("  - " + course +
                    (grade != null ? " (Grade: " + grade + ")" : " (In Progress)"));
            }
        }
    }

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

```
    // Course management methods    public void enrollCourse(String course) {
        if (!enrolledCourses.contains(course)) {            enrolledCourses.add(course);
            System.out.println("✓ Enrolled in " + course);        } else {
            System.out.println("Already enrolled in " + course);        }    }
    public void dropCourse(String course) {        if (enrolledCourses.remove(course)) {
            courseGrades.remove(course);            System.out.println("✓ Dropped " + course);
        } else {            System.out.println("Not enrolled in " + course);        }    }
         public void assignGrade(String course, String grade) {
        if (enrolledCourses.contains(course)) {            courseGrades.put(course, grade);
            System.out.println("✓ Grade assigned: " + course + " → " + grade);
        } else {            System.out.println("Not enrolled in " + course);        }    }
         public ArrayList<String> getEnrolledCourses() {
        return new ArrayList<>(enrolledCourses);    }
    public double getGpa() { return gpa; }
    public void setGpa(double gpa) { this.gpa = gpa; }
    public String getMajor() { return major; }
    public void setMajor(String major) { this.major = major; }
    public boolean isHonorRoll() {        return gpa ≥ 3.5;    }                @Override
    public boolean matches(String keyword) {
        return super.matches(keyword) || major.toLowerCase().contains(keyword.toLowerCase());
    }    }
```

Step 4: Enhanced Management System with Collections

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

public class EnhancedUniversitySystem {
    private HashMap<String, Person> peopleById;  // ID → Person
    private ArrayList<Student> students;
    private ArrayList<String> availableCourses;
    private Scanner scanner;

    public EnhancedUniversitySystem() {
        peopleById = new HashMap<>();
        students = new ArrayList<>();
        availableCourses = new ArrayList<>();
        scanner = new Scanner(System.in);
        initializeCourses();
    }

    private void initializeCourses() {
        availableCourses.add("CS101 - Intro to Programming");
        availableCourses.add("CS201 - Data Structures");
        availableCourses.add("CS301 - Algorithms");
        availableCourses.add("MATH101 - Calculus I");
```

```
        availableCourses.add("MATH201 - Linear Algebra");
        availableCourses.add("ENG101 - English Composition");    }
    public void displayMenu() {
        System.out.println("\n╔═══════════════════════════════════════════╗");
        System.out.println("║   ENHANCED UNIVERSITY SYSTEM              ║");
        System.out.println("╚═══════════════════════════════════════════╝");
        System.out.println("1.  Add Student");
        System.out.println("2.  Display All Students");
        System.out.println("3.  Search by Keyword");
        System.out.println("4.  Find Student by ID");
        System.out.println("5.  Display Honor Roll");
        System.out.println("6.  Enroll Student in Course");
        System.out.println("7.  Drop Student from Course");
        System.out.println("8.  Assign Grade");
        System.out.println("9.  Display Available Courses");
        System.out.println("10. Show Statistics");
        System.out.println("11. List Students by Major");
        System.out.println("12. Exit");
        System.out.println("─────────────────────────────────────────────");
        System.out.print("Enter choice (1-12): ");    }
    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();                if (peopleById.containsKey(id)) {
            System.out.println("✗ ID already exists!");            return;        }
        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);
        peopleById.put(id, student);        students.add(student);
        System.out.println("✓ Student added successfully!");    }
    public void displayAllStudents() {        System.out.println("\n--- All Students ---");
        if (students.isEmpty()) {            System.out.println("No students in system.");
            return;        }                for (int i = 0; i < students.size(); i++) {
            System.out.println("\nStudent #" + (i + 1));
            students.get(i).displayInfo();            System.out.println("=".repeat(40));
        }        System.out.println("Total students: " + students.size());    }
    public void searchByKeyword() {        System.out.println("\n--- Search by Keyword ---");
        System.out.print("Enter keyword: ");        String keyword = scanner.nextLine();
                ArrayList<Person> results = new ArrayList<>();
        for (Person person : peopleById.values()) {            if (person.matches(keyword)) {
                results.add(person);            }        }
        if (results.isEmpty()) {            System.out.println("No results found.");
        } else {            System.out.println("\nFound " + results.size() + " result(s):");
            for (Person person : results) {
                System.out.println("\n" + "-".repeat(40));
                person.displayInfo();            }        }    }
    public void findById() {        System.out.println("\n--- Find by ID ---");
```

```
        System.out.print("Enter ID: ");        String id = scanner.nextLine();
        Person person = peopleById.get(id);        if (person ≠ null) {
            System.out.println("\n✓ Found:");            person.displayInfo();        } else {
            System.out.println("✗ Person not found.");        }    }
    public void displayHonorRoll() {
        System.out.println("\n--- Honor Roll (GPA ≥ 3.5) ---");
        ArrayList<Student> honorStudents = new ArrayList<>();
        for (Student student : students) {            if (student.isHonorRoll()) {
                honorStudents.add(student);            }        }
        if (honorStudents.isEmpty()) {
            System.out.println("No students on honor roll.");        } else {
            for (Student student : honorStudents) {                student.displayInfo();
                System.out.println("─".repeat(40));            }
            System.out.println("Total: " + honorStudents.size());        }    }
    public void enrollInCourse() {        System.out.println("\n--- Enroll in Course ---");
        System.out.print("Enter Student ID: ");        String id = scanner.nextLine();
                Person person = peopleById.get(id);
        if (person == null || !(person instanceof Student)) {
            System.out.println("✗ Student not found.");            return;        }
        Student student = (Student) person;
        System.out.println("\nAvailable Courses:");
        for (int i = 0; i < availableCourses.size(); i++) {
            System.out.println((i + 1) + ". " + availableCourses.get(i));        }
        System.out.print("Enter course number: ");        int courseNum = scanner.nextInt();
        scanner.nextLine();
        if (courseNum ≥ 1 && courseNum ≤ availableCourses.size()) {
            student.enrollCourse(availableCourses.get(courseNum - 1));        } else {
            System.out.println("✗ Invalid course number.");        }    }
    public void dropCourse() {        System.out.println("\n--- Drop Course ---");
        System.out.print("Enter Student ID: ");        String id = scanner.nextLine();
                Person person = peopleById.get(id);
        if (person == null || !(person instanceof Student)) {
            System.out.println("✗ Student not found.");            return;        }
        Student student = (Student) person;
        ArrayList<String> enrolled = student.getEnrolledCourses();
        if (enrolled.isEmpty()) {
            System.out.println("Student not enrolled in any courses.");            return;
        }                System.out.println("\nEnrolled Courses:");
        for (int i = 0; i < enrolled.size(); i++) {
            System.out.println((i + 1) + ". " + enrolled.get(i));        }
        System.out.print("Enter course number to drop: ");
        int courseNum = scanner.nextInt();        scanner.nextLine();
        if (courseNum ≥ 1 && courseNum ≤ enrolled.size()) {
            student.dropCourse(enrolled.get(courseNum - 1));        } else {
            System.out.println("✗ Invalid course number.");        }    }
    public void assignGrade() {        System.out.println("\n--- Assign Grade ---");
        System.out.print("Enter Student ID: ");        String id = scanner.nextLine();
                Person person = peopleById.get(id);
        if (person == null || !(person instanceof Student)) {
            System.out.println("✗ Student not found.");            return;        }
```

```
        Student student = (Student) person;
        ArrayList<String> enrolled = student.getEnrolledCourses();
        if (enrolled.isEmpty()) {
            System.out.println("Student not enrolled in any courses.");            return;
        }                System.out.println("\nEnrolled Courses:");
        for (int i = 0; i < enrolled.size(); i++) {
            System.out.println((i + 1) + ". " + enrolled.get(i));        }
        System.out.print("Enter course number: ");        int courseNum = scanner.nextInt();
        scanner.nextLine();
        if (courseNum ≥ 1 && courseNum ≤ enrolled.size()) {
            System.out.print("Enter grade (A, B, C, D, F): ");
            String grade = scanner.nextLine().toUpperCase();
            student.assignGrade(enrolled.get(courseNum - 1), grade);        } else {
            System.out.println("✗ Invalid course number.");        }    }
    public void displayAvailableCourses() {
        System.out.println("\n--- Available Courses ---");
        for (String course : availableCourses) {            System.out.println("• " + course);
        }        System.out.println("\nTotal courses: " + availableCourses.size());    }
         public void showStatistics() {
        System.out.println("\n╔═══════════════════════════════════════╗");
        System.out.println("║          SYSTEM STATISTICS            ║");
        System.out.println("╚═══════════════════════════════════════╝");
        // Total students        System.out.println("Total Students: " + students.size());
             if (students.isEmpty()) {
            System.out.println("No data to display.");            return;        }
        // Average GPA        double totalGpa = 0;        for (Student student : students) {
            totalGpa += student.getGpa();        }
        double avgGpa = totalGpa / students.size();
        System.out.println("Average GPA: " + String.format("%.2f", avgGpa));
        // Honor roll count        int honorCount = 0;
        for (Student student : students) {            if (student.isHonorRoll()) {
                honorCount++;            }        }
        System.out.println("Honor Roll Students: " + honorCount);
        // Students by major        HashMap<String, Integer> majorCount = new HashMap<>();
        for (Student student : students) {            String major = student.getMajor();
            majorCount.put(major, majorCount.getOrDefault(major, 0) + 1);        }
        System.out.println("\nStudents by Major:");
        for (String major : majorCount.keySet()) {
            System.out.println("  " + major + ": " + majorCount.get(major));        }
        // Most popular courses
        HashMap<String, Integer> courseEnrollment = new HashMap<>();
        for (Student student : students) {
            for (String course : student.getEnrolledCourses()) {
                courseEnrollment.put(course, courseEnrollment.getOrDefault(course, 0) + 1);
            }        }                if (!courseEnrollment.isEmpty()) {
            System.out.println("\nCourse Enrollments:");
            for (String course : courseEnrollment.keySet()) {
                System.out.println("  " + course + ": " + courseEnrollment.get(course) + "
     students");
            }        }    }                public void listByMajor() {
```

```
        System.out.println("\n--- Students by Major ---");
        // Group students by major
        HashMap<String, ArrayList<Student>> studentsByMajor = new HashMap<>();
        for (Student student : students) {            String major = student.getMajor();
            if (!studentsByMajor.containsKey(major)) {
                studentsByMajor.put(major, new ArrayList<>());            }
            studentsByMajor.get(major).add(student);        }
        if (studentsByMajor.isEmpty()) {
            System.out.println("No students in system.");            return;        }
        for (String major : studentsByMajor.keySet()) {
            System.out.println("\n=== " + major + " ===");
            ArrayList<Student> majorStudents = studentsByMajor.get(major);
            for (Student student : majorStudents) {
                System.out.println("  • " + student.getName() + " (GPA: " + student.getGpa() +
     ")");
            }            System.out.println("Total: " + majorStudents.size());        }
    }               public void run() {
        System.out.println("Welcome to Enhanced University Management System!");
        int choice;        do {            displayMenu();
            choice = scanner.nextInt();            scanner.nextLine();
            switch (choice) {               case 1: addStudent(); break;
                case 2: displayAllStudents(); break;
                case 3: searchByKeyword(); break;               case 4: findById(); break;
                case 5: displayHonorRoll(); break;
                case 6: enrollInCourse(); break;               case 7: dropCourse(); break;
                case 8: assignGrade(); break;
                case 9: displayAvailableCourses(); break;
                case 10: showStatistics(); break;
                case 11: listByMajor(); break;               case 12:
                    System.out.println("\nThank you for using the system!");
                    System.out.println("Goodbye!");                    break;
                default:                    System.out.println("X Invalid choice!");
            }        } while (choice != 12);               scanner.close();    }
    public static void main(String[] args) {
        EnhancedUniversitySystem system = new EnhancedUniversitySystem();        system.run();
    }    }
```

📝 Homework Assignment

Task 1: Add LinkedList Feature (Required)

Add a feature to track student attendance using LinkedList:

```
private LinkedList<String> attendanceDates;  // In Student class
```

- Add methods: `markAttendance()`, `getAttendanceCount()`, `displayAttendance()`

- Show last 5 attendance dates

Task 2: TreeSet for Sorted Names (Required)

Modify the system to display students in alphabetical order:
- Use TreeSet to store student names
- Display sorted student list
- Implement Comparable interface in Student class

Task 3: Advanced Statistics (Challenge)

Add the following statistical features:
1. Find student with highest/lowest GPA
2. Calculate grade distribution (A, B, C, D, F count)
3. Find most enrolled course
4. Show students with no course enrollments
5. Calculate average courses per student

🎯 Assessment Checklist

Students should be able to:
- [ ] Create and implement interfaces
- [ ] Differentiate between abstract classes and interfaces
- [ ] Implement multiple interfaces in a class
- [ ] Use ArrayList for dynamic lists
- [ ] Use HashSet to store unique elements
- [ ] Use HashMap for key-value mappings
- [ ] Iterate through collections using different methods
- [ ] Choose appropriate collection type for scenarios
- [ ] Use collections with custom objects
- [ ] Apply collections methods (add, remove, contains, etc.)

🔧 Common Student Errors & Solutions

Error 1: Wrong Collection Type

```
// Wrong: Using ArrayList when uniqueness matters
ArrayList<String> studentIds = new ArrayList<>();
```

```
studentIds.add("S001");studentIds.add("S001");  // Duplicate allowed!
```

Solution: Use HashSet for unique elements

Error 2: Null Key in HashMap

```
HashMap<String, Integer> map = new HashMap<>();
String name = null;
map.put(name, 25);  // Works but not recommended
```

Solution: Always validate keys before adding

Error 3: ConcurrentModificationException

```
for (String item : list) {
    if (condition) {
        list.remove(item);  // Error!
    }
}
```

Solution: Use Iterator or collect items to remove first

Error 4: Comparing Objects

```
Student s1 = new Student("Alice", ...);
Student s2 = new Student("Alice", ...);
list.contains(s1);  // Might not work as expected
```

Solution: Override equals() and hashCode() methods

📑 Additional Resources

- Java Collections Tutorial
- Collections Framework Overview
- When to Use Each Collection
- Practice: Implement your own simplified ArrayList

Preview of Week 5

Next week we'll cover:
- **Exception Handling:** Try-catch-finally blocks
- **Custom Exceptions:** Creating your own exception classes
- **File I/O:** Reading and writing files
- **CSV Processing:** Working with comma-separated values
- **Data Persistence:** Saving and loading application data

Prepare: Your student system will get file-based persistence!

Key Takeaways

> ***"Interface = Contract"***
> *Defines WHAT, not HOW*

> ***"Abstract Class vs Interface"***
> *Abstract Class = Partial implementation, Interface = Pure contract*

> ***"Collection Selection"***
> *ArrayList = Ordered list, HashSet = Unique set, HashMap = Key-value pairs*

> ***"Use the Right Tool"***
> *Choose collections based on what you need: order, uniqueness, or lookup speed*