Table of Contents

Week 5: Exception Handling and File I/O

- Teaching Guide & Hands-on Project
- 📋 Session Overview
- 🎯 Learning Objectives
- 📚 Quick Review (5 minutes)
- 🕐 Detailed Lesson Plan (120 Minutes)
  - Part 1: Understanding Exceptions (15 minutes)
  - Part 2: Try-Catch-Finally Blocks (20 minutes)
  - Part 3: Throwing Exceptions and Custom Exceptions (15 minutes)
  - Part 4: File I/O Basics (20 minutes)
  - Part 5: Try-With-Resources (10 minutes)
- 💻 HANDS-ON PROJECT: Student Management System with File Persistence (50 minutes)
  - Project Overview
  - Step-by-Step Implementation
- 📝 Homework Assignment
  - Task 1: Add Try-Catch to User Input (Required)
  - Task 2: Implement Auto-Save (Required)
  - Task 3: Advanced File Operations (Challenge)
- 🎯 Assessment Checklist
- 🔧 Common Student Errors & Solutions
  - Error 1: File Not Found
  - Error 2: Resource Leak
  - Error 3: Buffer Not Flushed
  - Error 4: Incorrect Exception Handling Order
- 📚 Additional Resources
- 🔜 Preview of Week 6
- 📌 Key Takeaways

Week 5: Exception Handling and File I/O

Teaching Guide & Hands-on Project

📋 Session Overview

Duration: 2 - 3 Hours
Teaching: 90 minutes | **Hands-on Practice:** 50 minutes

🎯 Learning Objectives

By the end of this session, students will be able to:
1. Understand exceptions and error handling concepts
2. Use try-catch-finally blocks effectively
3. Handle multiple exception types
4. Throw and create custom exceptions
5. Read and write text files in Java
6. Work with CSV (Comma-Separated Values) files
7. Use try-with-resources for automatic resource management
8. Implement data persistence in applications using files

📚 Quick Review (5 minutes)

Quick Questions:
- What's the difference between ArrayList and HashSet?
- When would you use HashMap instead of ArrayList?
- What does an interface define?
- Can a class implement multiple interfaces?

Key Review Points:
- Collections provide dynamic data structures
- Choose collections based on needs: order, uniqueness, lookup
- Interfaces define contracts for behavior

🕐 Detailed Lesson Plan (120 Minutes)

Part 1: Understanding Exceptions (15 minutes)

What are Exceptions? (5 minutes)

Definition:
- Exceptions are events that disrupt normal program flow
- Represent errors or unexpected situations
- Allow graceful error handling instead of program crashes

Real-world analogy:

```
Ordering food at a restaurant:

Normal Flow:
1. You order food
2. Kitchen prepares it
3. You receive your meal
4. You eat

Exception Scenarios:
- Item not available → MenuItemException
- Kitchen is closed → KitchenClosedException
- Payment declined → PaymentException
- You're allergic → AllergyException

Each exception is handled differently!
```

Exception Hierarchy (5 minutes)

Draw the hierarchy:

```
                    Throwable
                        |
        ┌───────────────┴────────────────┐
        |                                |
    Exception                          Error
        |                                |
        ├── IOException              StackOverflowError
        ├── SQLException             OutOfMemoryError
        ├── FileNotFoundException    (Don't catch these!)
        └── RuntimeException
                |
                ├── NullPointerException
```

```
├── ArithmeticException
                └── IllegalArgumentException
```

Two Main Categories:

1. **Checked Exceptions** (must handle)
    - IOException, FileNotFoundException, SQLException
    - Compiler forces you to handle them
2. **Unchecked Exceptions** (Runtime Exceptions)
    - NullPointerException, ArrayIndexOutOfBoundsException
    - Can occur anywhere, not required to handle

Common Exceptions Demonstration (5 minutes)

Example 1: Common Runtime Exceptions

```java
public class CommonExceptionsDemo {
    public static void main(String[] args) {
        // 1. ArithmeticException
        try {
            int result = 10 / 0;  // Division by zero
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        }

        // 2. NullPointerException
        try {
            String text = null;
            System.out.println(text.length());  // Calling method on null
        } catch (NullPointerException e) {
            System.out.println("Error: String is null!");
        }

        // 3. ArrayIndexOutOfBoundsException
        try {
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[5]);  // Index doesn't exist
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Array index out of bounds!");
        }

        // 4. NumberFormatException
        try {
            int num = Integer.parseInt("abc");  // Not a valid number
        } catch (NumberFormatException e) {
            System.out.println("Error: Cannot convert to number!");
        }
```

```
                System.out.println("Program continues running!");    }    }
```

Output:

```
Error: Cannot divide by zero!
Error: String is null!
Error: Array index out of bounds!
Error: Cannot convert to number!
Program continues running!
```

Part 2: Try-Catch-Finally Blocks (20 minutes)

Basic Try-Catch (8 minutes)

Example 2: Simple Try-Catch

```java
import java.util.Scanner;

public class TryCatchDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first number: ");
        int num1 = scanner.nextInt();

        System.out.print("Enter second number: ");
        int num2 = scanner.nextInt();

        try {
            int result = num1 / num2;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
            System.out.println("Please enter a non-zero divisor.");
        }

        System.out.println("Thank you for using the calculator!");
        scanner.close();
    }
}
```

Key Points:
- Code that might fail goes in `try` block
- Exception handling goes in `catch` block
- Program continues after catch block

Multiple Catch Blocks (7 minutes)

Example 3: Handling Multiple Exceptions

```java
import java.util.Scanner;

public class MultipleCatchDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Enter array size: ");
            int size = scanner.nextInt();

            int[] numbers = new int[size];

            System.out.print("Enter index to access: ");
            int index = scanner.nextInt();

            System.out.print("Enter value: ");
            int value = scanner.nextInt();

            numbers[index] = value;
            System.out.println("Value stored successfully!");

            // Division operation
            System.out.print("Divide by: ");
            int divisor = scanner.nextInt();
            int result = value / divisor;
            System.out.println("Result: " + result);

        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Index out of array bounds!");
            System.out.println("Index must be between 0 and " + (e.getMessage()));
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        } catch (NegativeArraySizeException e) {
            System.out.println("Error: Array size cannot be negative!");
        } catch (Exception e) {
            // Catch-all for any other exceptions
            System.out.println("Error: Something went wrong!");
            System.out.println("Details: " + e.getMessage());
        }

        scanner.close();
    }
}
```

Important: Order matters! More specific exceptions before general ones.

Finally Block (5 minutes)

Example 4: Try-Catch-Finally

```java
import java.util.Scanner;

public class FinallyDemo {
    public static void main(String[] args) {
        Scanner scanner = null;

        try {
            scanner = new Scanner(System.in);
            System.out.print("Enter a number: ");
            int number = scanner.nextInt();

            int result = 100 / number;
            System.out.println("Result: " + result);

        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero!");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            // This ALWAYS runs, even if exception occurs
            System.out.println("Cleaning up resources...");
            if (scanner ≠ null) {
                scanner.close();
                System.out.println("Scanner closed.");
            }
        }

        System.out.println("Program ended.");
    }
}
```

Finally Block Rules:

- ALWAYS executes (even if exception occurs)
- Used for cleanup (closing files, connections, etc.)
- Runs even if there's a return statement in try/catch

Part 3: Throwing Exceptions and Custom Exceptions (15 minutes)

Throwing Exceptions (7 minutes)

Example 5: Throwing Exceptions

```java
public class ThrowExceptionDemo {

    public static void validateAge(int age) {
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be negative!");
        }
        if (age < 18) {
            throw new IllegalArgumentException("Must be 18 or older!");
        }
        System.out.println("Age is valid: " + age);
    }

    public static double calculateDiscount(double price, double discount) {
        if (price < 0) {
            throw new IllegalArgumentException("Price cannot be negative!");
        }
        if (discount < 0 || discount > 100) {
            throw new IllegalArgumentException("Discount must be between 0 and 100!");
        }
        return price * (1 - discount / 100);
    }

    public static void main(String[] args) {
        try {
            validateAge(25);
            validateAge(-5);  // This will throw exception
        } catch (IllegalArgumentException e) {
            System.out.println("Validation Error: " + e.getMessage());
        }

        try {
            double finalPrice = calculateDiscount(100, 20);
            System.out.println("Final price: $" + finalPrice);

            calculateDiscount(100, 150);  // Invalid discount
        } catch (IllegalArgumentException e) {
            System.out.println("Calculation Error: " + e.getMessage());
        }
    }
}
```

Custom Exceptions (8 minutes)

Example 6: Creating Custom Exceptions

```java
// Custom exception class
public class InvalidGradeException extends Exception {
    public InvalidGradeException(String message) {
        super(message);
```

```
    }    }
```

```
public class InsufficientBalanceException extends Exception {
    private double balance;
    private double amount;

    public InsufficientBalanceException(double balance, double amount) {
        super("Insufficient balance. Balance: $" + balance + ", Requested: $" + amount);
        this.balance = balance;
        this.amount = amount;
    }

    public double getBalance() {
        return balance;
    }

    public double getShortfall() {
        return amount - balance;
    }
}
```

```
// Using custom exceptions
public class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    public void withdraw(double amount) throws InsufficientBalanceException {
        if (amount > balance) {
            throw new InsufficientBalanceException(balance, amount);
        }
        balance -= amount;
        System.out.println("Withdrawal successful. New balance: $" + balance);
    }

    public void deposit(double amount) {
        if (amount ≤ 0) {
            throw new IllegalArgumentException("Deposit amount must be positive!");
        }
        balance += amount;
        System.out.println("Deposit successful. New balance: $" + balance);
    }

    public double getBalance() {
        return balance;
```

```
    }    }
```

```
public class BankDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("ACC001", 1000);

        try {
            account.deposit(500);
            account.withdraw(300);
            account.withdraw(2000);  // This will throw exception
        } catch (InsufficientBalanceException e) {
            System.out.println("Transaction Failed: " + e.getMessage());
            System.out.println("You need $" + e.getShortfall() + " more.");
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid Operation: " + e.getMessage());
        }

        System.out.println("Final balance: $" + account.getBalance());
    }
}
```

Part 4: File I/O Basics (20 minutes)

Reading Files (10 minutes)

Example 7: Reading Text Files

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileReadingDemo {
    public static void main(String[] args) {
        // Method 1: Using Scanner
        try {
            File file = new File("data.txt");
            Scanner fileScanner = new Scanner(file);

            System.out.println("=== Reading File ===");
            while (fileScanner.hasNextLine()) {
                String line = fileScanner.nextLine();
                System.out.println(line);
            }

            fileScanner.close();
```

```
            System.out.println("Error: File not found!");
            System.out.println("Please create 'data.txt' in the project directory.");        }
    }    }
```

Example 8: Reading with BufferedReader

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class BufferedReaderDemo {
    public static void main(String[] args) {
        BufferedReader reader = null;

        try {
            reader = new BufferedReader(new FileReader("students.txt"));
            String line;
            int lineNumber = 1;

            System.out.println("=== File Contents ===");
            while ((line = reader.readLine()) != null) {
                System.out.println(lineNumber + ": " + line);
                lineNumber++;
            }

        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found!");
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                System.out.println("Error closing file: " + e.getMessage());
            }
        }
    }
}
```

Writing Files (10 minutes)

Example 9: Writing to Files

```
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;
```

```
public class FileWritingDemo {    public static void main(String[] args) {
        // Method 1: Simple FileWriter        try {
            FileWriter writer = new FileWriter("output.txt");
            writer.write("Hello, File I/O!\n");            writer.write("This is line 2.\n");
            writer.write("This is line 3.\n");            writer.close();
            System.out.println("File written successfully!");
        } catch (IOException e) {
            System.out.println("Error writing file: " + e.getMessage());        }
        // Method 2: BufferedWriter (more efficient)        try {
            BufferedWriter bw = new BufferedWriter(new FileWriter("students.txt"));
                        bw.write("Alice,20,Computer Science,3.8");            bw.newLine();
            bw.write("Bob,22,Mathematics,3.5");            bw.newLine();
            bw.write("Charlie,21,Physics,3.9");            bw.newLine();
            bw.close();            System.out.println("Student data written successfully!");
                    } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());        }    }    }
```

Example 10: Appending to Files

```
import java.io.FileWriter;
import java.io.IOException;

public class AppendFileDemo {
    public static void main(String[] args) {
        try {
            // true parameter means append mode
            FileWriter writer = new FileWriter("log.txt", true);

            writer.write("New log entry at " + new java.util.Date() + "\n");
            writer.write("User logged in\n");
            writer.write("---\n");

            writer.close();
            System.out.println("Log appended successfully!");

        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Part 5: Try-With-Resources (10 minutes)

Modern Resource Management (10 minutes)

Example 11: Try-With-Resources

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class TryWithResourcesDemo {

    // Old way (verbose)
    public static void readFileOldWay(String filename) {
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader(filename));
            String line;
            while ((line = reader.readLine()) ≠ null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            try {
                if (reader ≠ null) {
                    reader.close();
                }
            } catch (IOException e) {
                System.out.println("Error closing: " + e.getMessage());
            }
        }
    }

    // New way (cleaner with try-with-resources)
    public static void readFileNewWay(String filename) {
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) ≠ null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
        // Resource is automatically closed!
    }

    public static void main(String[] args) {
        System.out.println("≡≡≡ Reading with try-with-resources ≡≡≡");
        readFileNewWay("data.txt");
    }
}
```

Benefits of Try-With-Resources:

- Automatic resource closing

- Cleaner, more readable code
- Reduces resource leaks
- Handles multiple resources easily

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

Project Overview

Enhance the Week 4 Student Management System to:

1. **Save data to files** when exiting
2. **Load data from files** when starting
3. **Export to CSV** for spreadsheet compatibility
4. **Import from CSV** to bulk add students
5. **Handle all file operations** with proper exception handling
6. **Create backup files** automatically

Step-by-Step Implementation

Step 1: Add File Persistence to Student Class

```
public class Student extends Person {
    private String major;
    private double gpa;
    private ArrayList<String> enrolledCourses;

    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<>();
    }

    // Convert student to CSV format
    public String toCSV() {
        // Format: name,age,id,email,major,gpa,course1;course2;course3
        String courses = String.join(";", enrolledCourses);
        return String.format("%s,%d,%s,%s,%s,%.2f,%s",
            name, age, id, email, major, gpa, courses);
    }

    // Create student from CSV line
    public static Student fromCSV(String csvLine) throws InvalidDataException {
```

```
        try {            String[] parts = csvLine.split(",");
            if (parts.length < 6) {
                throw new InvalidDataException("Invalid CSV format: insufficient fields");
            }                        String name = parts[0];
            int age = Integer.parseInt(parts[1]);            String id = parts[2];
            String email = parts[3];            String major = parts[4];
            double gpa = Double.parseDouble(parts[5]);
            Student student = new Student(name, age, id, email, major, gpa);
            // Add courses if present
            if (parts.length > 6 && !parts[6].isEmpty()) {
                String[] courses = parts[6].split(";");
                for (String course : courses) {
                    student.enrollCourse(course);                }            }
            return student;                    } catch (NumberFormatException e) {
            throw new InvalidDataException("Invalid number format in CSV: " + csvLine);
        }    }                // Other methods from previous weeks...    @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);
             if (!enrolledCourses.isEmpty()) {
            System.out.println("\nEnrolled Courses:");
            for (String course : enrolledCourses) {
                System.out.println("  - " + course);            }        }    }
    @Override    public String getRole() {        return "Student";    }
    public void enrollCourse(String course) {        if (!enrolledCourses.contains(course)) {
            enrolledCourses.add(course);        }    }
    public void dropCourse(String course) {        enrolledCourses.remove(course);    }
             public ArrayList<String> getEnrolledCourses() {
        return new ArrayList<>(enrolledCourses);    }                // Getters
    public double getGpa() { return gpa; }    public String getMajor() { return major; }
             // Setters    public void setGpa(double gpa) { this.gpa = gpa; }
    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 2: Custom Exception for Invalid Data

```
public class InvalidDataException extends Exception {
    public InvalidDataException(String message) {
        super(message);
    }

    public InvalidDataException(String message, Throwable cause) {
```

```
        super(message, cause);    }    }
```

Step 3: File Manager Class

```
import java.io.*;
import java.util.ArrayList;
import java.text.SimpleDateFormat;
import java.util.Date;

public class FileManager {
    private static final String DATA_FILE = "students.csv";
    private static final String BACKUP_DIR = "backups/";

    // Save all students to file
    public static void saveStudents(ArrayList<Student> students) throws IOException {
        // Create backup first
        createBackup();

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(DATA_FILE))) {
            // Write header
            writer.write("Name,Age,ID,Email,Major,GPA,Courses");
            writer.newLine();

            // Write student data
            for (Student student : students) {
                writer.write(student.toCSV());
                writer.newLine();
            }

            System.out.println("✓ Data saved successfully to " + DATA_FILE);

        } catch (IOException e) {
            System.out.println("✗ Error saving data: " + e.getMessage());
            throw e;
        }
    }

    // Load all students from file
    public static ArrayList<Student> loadStudents() {
        ArrayList<Student> students = new ArrayList<>();

        File file = new File(DATA_FILE);
        if (!file.exists()) {
            System.out.println("No existing data file found. Starting fresh.");
            return students;
        }

        try (BufferedReader reader = new BufferedReader(new FileReader(DATA_FILE))) {
```

```
            String line;            boolean firstLine = true;            int lineNumber = 1;
                        while ((line = reader.readLine()) != null) {
                lineNumber++;                                    // Skip header
                if (firstLine) {                        firstLine = false;
                    continue;                    }
                // Skip empty lines                    if (line.trim().isEmpty()) {
                    continue;                    }                                    try {
                    Student student = Student.fromCSV(line);
                    students.add(student);                    } catch (InvalidDataException e) {
                    System.out.println("Warning: Skipping invalid data at line " +
        lineNumber);
                    System.out.println("  Reason: " + e.getMessage());                    }
            }
            System.out.println("✓ Loaded " + students.size() + " students from file");
                    } catch (FileNotFoundException e) {
            System.out.println("Data file not found. Starting with empty database.");
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());        }
        return students;    }                // Create backup of current data
    private static void createBackup() {        File dataFile = new File(DATA_FILE);
        if (!dataFile.exists()) {            return;  // Nothing to backup        }
        // Create backup directory if it doesn't exist
        File backupDir = new File(BACKUP_DIR);        if (!backupDir.exists()) {
            backupDir.mkdir();        }
        // Generate backup filename with timestamp
        SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd_HHmmss");
        String timestamp = sdf.format(new Date());
        String backupFilename = BACKUP_DIR + "students_" + timestamp + ".csv";
        try (BufferedReader reader = new BufferedReader(new FileReader(DATA_FILE));
             BufferedWriter writer = new BufferedWriter(new FileWriter(backupFilename))) {
                        String line;            while ((line = reader.readLine()) != null) {
                writer.write(line);                writer.newLine();            }
            System.out.println("✓ Backup created: " + backupFilename);
        } catch (IOException e) {
            System.out.println("Warning: Could not create backup: " + e.getMessage());
        }    }                // Export students to CSV (user-friendly format)
    public static void exportToCSV(ArrayList<Student> students, String filename) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
            // Write header
            writer.write("Student Name,Age,Student ID,Email,Major,GPA,Number of Courses");
            writer.newLine();                        // Write data
            for (Student student : students) {
                writer.write(String.format("%s,%d,%s,%s,%s,%.2f,%d",
                    student.getName(),                    student.getAge(),
                    student.getId(),                    student.getEmail(),
                    student.getMajor(),                    student.getGpa(),
                    student.getEnrolledCourses().size()));                writer.newLine();
            }                        System.out.println("✓ Data exported to " + filename);
                    } catch (IOException e) {
            System.out.println("✗ Export failed: " + e.getMessage());        }    }
```

```
        // Import students from CSV
    public static ArrayList<Student> importFromCSV(String filename) throws IOException,
        InvalidDataException {
        ArrayList<Student> importedStudents = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;            boolean firstLine = true;            int lineNumber = 1;
            int successCount = 0;            int errorCount = 0;
            while ((line = reader.readLine()) != null) {                lineNumber++;
                            if (firstLine) {                    firstLine = false;
                    continue;                }
                if (line.trim().isEmpty()) {                    continue;                }
                            try {
                    Student student = Student.fromCSV(line);
                    importedStudents.add(student);                    successCount++;
                } catch (InvalidDataException e) {
                    System.out.println("Line " + lineNumber + " error: " + e.getMessage());
                    errorCount++;                }            }
            System.out.println("\n=== Import Summary ===");
            System.out.println("Successfully imported: " + successCount);
            System.out.println("Errors: " + errorCount);
        } catch (FileNotFoundException e) {
            throw new IOException("Import file not found: " + filename);        }
        return importedStudents;    }    }
```

Step 4: Enhanced Management System with File Operations

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;
import java.io.IOException;

public class PersistentUniversitySystem {
    private HashMap<String, Student> studentsById;
    private ArrayList<Student> students;
    private Scanner scanner;

    public PersistentUniversitySystem() {
        studentsById = new HashMap<>();
        students = new ArrayList<>();
        scanner = new Scanner(System.in);
        loadData();
    }

    private void loadData() {
        System.out.println("Loading data from file...");
        students = FileManager.loadStudents();
```

```
            studentsById.put(student.getId(), student);        }    }
    private void saveData() {        try {            FileManager.saveStudents(students);
        } catch (IOException e) {            System.out.println("Failed to save data!");
        }    }                public void displayMenu() {
        System.out.println("\n╔═══════════════════════════════════════╗");
        System.out.println("║   STUDENT MANAGEMENT SYSTEM v2.0      ║");
        System.out.println("╚═══════════════════════════════════════╝");
        System.out.println("1.  Add Student");
        System.out.println("2.  Display All Students");
        System.out.println("3.  Search Student");
        System.out.println("4.  Update Student GPA");
        System.out.println("5.  Delete Student");
        System.out.println("6.  Display Honor Roll");
        System.out.println("7.  Save Data");        System.out.println("8.  Export to CSV");
        System.out.println("9.  Import from CSV");        System.out.println("10. Exit");
        System.out.println("─────────────────────────────────────────");
        System.out.print("Enter choice (1-10): ");    }
    public void addStudent() {        System.out.println("\n--- Add New Student ---");
            try {            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 (studentsById.containsKey(id)) {
                System.out.println("✗ Student 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();
            if (gpa < 0.0 || gpa > 4.0) {
                throw new IllegalArgumentException("GPA must be between 0.0 and 4.0");
            }
            Student student = new Student(name, age, id, email, major, gpa);
            students.add(student);            studentsById.put(id, student);
            System.out.println("✓ Student added successfully!");
        } catch (IllegalArgumentException e) {
            System.out.println("✗ Invalid input: " + e.getMessage());
            scanner.nextLine(); // Clear buffer        } catch (Exception e) {
            System.out.println("✗ Error adding student: " + e.getMessage());
            scanner.nextLine(); // Clear buffer        }    }
    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("=".repeat(40));
        }                System.out.println("Total students: " + students.size());    }
         public void searchStudent() {
        System.out.println("\n--- Search Student ---");
```

```
        System.out.print("Enter search keyword: ");
        String keyword = scanner.nextLine();
        ArrayList<Student> results = new ArrayList<>();
        for (Student student : students) {            if (student.matches(keyword)) {
                results.add(student);            }        }
        if (results.isEmpty()) {            System.out.println("No students found.");
        } else {            System.out.println("\nFound " + results.size() + " student(s):");
            for (Student student : results) {                System.out.println();
                student.displayInfo();                System.out.println("─".repeat(40));
            }        }    }                public void updateGPA() {
        System.out.println("\n--- Update Student GPA ---");
        System.out.print("Enter Student ID: ");        String id = scanner.nextLine();
                Student student = studentsById.get(id);        if (student == null) {
            System.out.println("✗ Student not found.");            return;        }
        try {            System.out.println("Current GPA: " + student.getGpa());
            System.out.print("Enter new GPA (0.0-4.0): ");
            double newGpa = scanner.nextDouble();            scanner.nextLine();
            if (newGpa < 0.0 || newGpa > 4.0) {
                throw new IllegalArgumentException("GPA must be between 0.0 and 4.0");
            }                        student.setGpa(newGpa);
            System.out.println("✓ GPA updated successfully!");
        } catch (IllegalArgumentException e) {
            System.out.println("✗ " + e.getMessage());            scanner.nextLine();
        } catch (Exception e) {            System.out.println("✗ Invalid input!");
            scanner.nextLine();        }    }                public void deleteStudent() {
        System.out.println("\n--- Delete Student ---");
        System.out.print("Enter Student ID: ");        String id = scanner.nextLine();
                Student student = studentsById.get(id);        if (student == null) {
            System.out.println("✗ Student not found.");            return;        }
        System.out.println("\nStudent to delete:");        student.displayInfo();
        System.out.print("\nAre you sure? (yes/no): ");
        String confirm = scanner.nextLine();
        if (confirm.equalsIgnoreCase("yes")) {            students.remove(student);
            studentsById.remove(id);
            System.out.println("✓ Student deleted successfully!");        } else {
            System.out.println("Deletion cancelled.");        }    }
    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 exportData() {        System.out.println("\n--- Export Data ---");
        System.out.print("Enter filename (e.g., export.csv): ");
        String filename = scanner.nextLine();
        if (!filename.endsWith(".csv")) {            filename += ".csv";        }
```

```
        FileManager.exportToCSV(students, filename);    }
    public void importData() {        System.out.println("\n--- Import Data ---");
        System.out.print("Enter filename to import: ");
        String filename = scanner.nextLine();                try {
            ArrayList<Student> importedStudents = FileManager.importFromCSV(filename);
                        if (importedStudents.isEmpty()) {
                System.out.println("No valid students to import.");                return;
            }
            System.out.print("\nImport " + importedStudents.size() + " students? (yes/no): ");
            String confirm = scanner.nextLine();
            if (confirm.equalsIgnoreCase("yes")) {                int addedCount = 0;
                int skippedCount = 0;
                for (Student student : importedStudents) {
                    if (studentsById.containsKey(student.getId())) {
                        System.out.println("Skipping duplicate ID: " + student.getId());
                        skippedCount++;                    } else {
                        students.add(student);
                        studentsById.put(student.getId(), student);
                        addedCount++;                    }                }
                System.out.println("\n✓ Import complete!");
                System.out.println("Added: " + addedCount);
                System.out.println("Skipped (duplicates): " + skippedCount);
            } else {                System.out.println("Import cancelled.");            }
                    } catch (IOException e) {
            System.out.println("✗ Import failed: " + e.getMessage());
        } catch (InvalidDataException e) {
            System.out.println("✗ Data error: " + e.getMessage());        }    }
    public void run() {
        System.out.println("╔═══════════════════════════════════════════╗");
        System.out.println("║  Welcome to Student Management System     ║");
        System.out.println("║           with File Persistence           ║");
        System.out.println("╚═══════════════════════════════════════════╝");
        int choice;        do {            displayMenu();                        try {
                choice = scanner.nextInt();                scanner.nextLine();
                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: saveData(); break;                    case 8: exportData(); break;
                    case 9: importData(); break;                    case 10:
                        System.out.println("\nSaving data before exit...");
                        saveData();
                        System.out.println("Thank you for using the system!");
                        System.out.println("Goodbye!");                        break;
                    default:
                        System.out.println("✗ Invalid choice! Please enter 1-10.");
                }                            } catch (Exception e) {
                System.out.println("✗ Invalid input! Please enter a number.");
```

```
                scanner.nextLine(); // Clear invalid input
                choice = 0; // Continue loop            }
        } while (choice != 10);                scanner.close();    }
    public static void main(String[] args) {
        PersistentUniversitySystem system = new PersistentUniversitySystem();
        system.run();    }    }
```

📝 Homework Assignment

Task 1: Add Try-Catch to User Input (Required)

Enhance the system to handle all user input errors gracefully:
- Catch `InputMismatchException` for invalid number inputs
- Validate email format
- Validate age range (16-100)
- Show user-friendly error messages

Task 2: Implement Auto-Save (Required)

Add automatic saving feature:
- Auto-save after every 5 operations
- Show "Auto-saving..." message
- Track number of unsaved changes

Task 3: Advanced File Operations (Challenge)

Implement the following features:
1. **Search in files:** Search without loading entire file into memory
2. **Restore from backup:** Allow user to restore from a backup file
3. **Export filtered data:** Export only honor roll students or students by major
4. **Log file:** Create a log file that records all operations with timestamps

---

🎯 Assessment Checklist

Students should be able to:
- [ ] Understand exception types and hierarchy
- [ ] Use try-catch-finally blocks correctly
- [ ] Handle multiple exception types
- [ ] Throw exceptions when appropriate
- [ ] Create custom exception classes

- [ ] Read text files using Scanner and BufferedReader
- [ ] Write text files using FileWriter and BufferedWriter
- [ ] Use try-with-resources for automatic cleanup
- [ ] Parse CSV files
- [ ] Implement data persistence in applications
- [ ] Handle file I/O errors gracefully

🔧 Common Student Errors & Solutions

Error 1: File Not Found

```
File file = new File("data.txt");  // Wrong path!
```

Solution: Check file location, use absolute path if needed, or create file first

Error 2: Resource Leak

```
FileWriter writer = new FileWriter("file.txt");
writer.write("text");
// Forgot to close!
```

Solution: Use try-with-resources or ensure close() in finally block

Error 3: Buffer Not Flushed

```
BufferedWriter writer = new BufferedWriter(new FileWriter("file.txt"));
writer.write("text");
// Data might not be written!
```

Solution: Call flush() or close() to ensure data is written

Error 4: Incorrect Exception Handling Order

```
try {
    // code
} catch (Exception e) {
    // Too general!
} catch (IOException e) {  // Unreachable!
```

```
    // Never executed}
```

Solution: More specific exceptions before general ones

📚 Additional Resources

- Java Exception Handling
- Java I/O Tutorial
- Try-With-Resources
- Working with Files

🔜 Preview of Week 6

Next week we'll cover:
- **Database Concepts:** Tables, rows, columns, relationships
- **SQL Fundamentals:** SELECT, INSERT, UPDATE, DELETE
- **PostgreSQL:** Installation and setup
- **Database Design:** Creating schemas and relationships
- **SQL Joins:** Combining data from multiple tables

Prepare: Install PostgreSQL before next class!

📌 Key Takeaways

> ***"Exceptions = Graceful Error Handling"***
> *Don't let your program crash - handle errors elegantly*

> ***"Try-With-Resources = Modern Best Practice"***
> *Automatic resource management prevents leaks*

> ***"Files = Data Persistence"***
> *Your data lives beyond program execution*

> ***"Always Validate Input"***
> *Never trust user input - validate and handle errors*

> ***"Backups Save Lives"***
> *Always create backups before modifying data*