**Assignment1: Track and Persist Customer Feedback using File I/O**

**Objective:**

To develop a Java console-based application that allows users to enter customer feedback and store it into a file. Later, the application should read and display all stored feedback using Java File I/O (FileOutputStream and FileInputStream).

**Assignment Tasks:**

**Task 1: Create the Feedback class**

* Define a class Feedback with the following fields:
  + int feedbackId
  + String customerName
  + String feedbackText
* Provide constructors, getter/setter methods, and a toString() method.

**Task 2: Write feedback to a file**

* Accept multiple feedback entries from the user (use Scanner).
* Use FileOutputStream to save each feedback as a new line in a file named feedback.txt.
* Convert each feedback object into a string like:

101|John Doe|The product is great!

**Task 3: Read and display all feedback**

* Use FileInputStream to read data from the same file.
* Display each feedback in a readable format:

Feedback ID: 101

Name: John Doe

Feedback: The product is great!

**Task 4: Menu-driven program**

Implement the following menu:

1. Add New Feedback

2. View All Feedback

3. Exit

**Assignment2 : "Student Result Processing System"**

**Objective:**

Design and implement a Java program that:

* Manages student records with marks
* Calculates results (Pass/Fail)
* Validates data using **custom exceptions**
* Uses **File I/O** to persist and retrieve student information

**Scenario:**

You are building a Student Result Processing System for a college. The system should:

* Accept student data
* Validate marks using **custom exceptions**
* Store and retrieve student records from a file
* Display a menu-based interface

**Data Model**

**Student class**

int studentId;

String name;

int[] marks = new int[3]; // 3 subjects

float average;

String result; // "Pass" or "Fail"

**Custom Exceptions**

**InvalidMarksException**

* Thrown if any mark is < 0 or > 100
* Message: "Marks should be between 0 and 100"

**StudentNotFoundException**

* Thrown if a student ID is not found during search
* Message: "Student with ID X not found"

**Functional Requirements**

**Task 1: Create Model**

* Define Student class with:
  + Constructor
  + Getters, setters
  + toString()
  + Implements Serializable

**Task 2: Add Student**

* Read student info from user
* Read 3 subject marks
* Validate marks using InvalidMarksException
* Calculate average and set result:
  + Pass: All marks >= 35
  + Fail: Any mark < 35
* Serialize student object to file: students.dat

**Task 3: View All Students**

* Deserialize student list and display formatted output

**Task 4: Search by ID**

* Ask for student ID
* If not found, throw StudentNotFoundException

**Task 5: Menu-Driven Program**

======= Student Result Menu =======

1. Add New Student

2. View All Students

3. Search by Student ID

4. Exit

**File I/O Notes**

* Use FileOutputStream and ObjectOutputStream to save data
* Use FileInputStream and ObjectInputStream to read data
* Avoid overwriting file each time (implement append logic or load all, add new, and save again)

**Case Study: Library Book Lending System – With Multithreading**

**🎯 Overview**

Design and develop a **Library Book Lending System** using Core Java that supports:

* Member registration
* Book creation
* Book lending with borrowing limits
* Persistent storage using File I/O
* Exception handling via custom exceptions
* **Simulated concurrent access using multithreading**

**🧩 Task Breakdown**

**🔹 Task 1: Create Core Classes for Book and Member**

* **Create Book class** with fields:  
  bookId, title, author, isAvailable
* **Create Member class** with fields:  
  memberId, name, List<Book> borrowedBooks
* Include constructors, getters/setters, and toString().

**🔹 Task 2: Implement Custom Exception**

Define a **custom exception class** BookLimitExceededException:

* This should be thrown when a member tries to borrow more than 3 books.

java

CopyEdit

public class BookLimitExceededException extends Exception {

public BookLimitExceededException(String message) {

super(message);

}

}

**Task 3: Add File I/O for Persistence**

**Serialization:**

* Write a method to save the list of Member objects to a file (members.dat)
* Use ObjectOutputStream with FileOutputStream.

**Deserialization:**

* Read member data from file on application startup.
* Use ObjectInputStream.

**Task 4: Menu-Driven Console Application**

Provide a menu-driven interface with options:

* Register new member
* Add new book
* Borrow a book
* View member and their borrowed books
* Save and Exit

Ensure exception handling and validations.

**Task 5: Introduce Multithreading – Simulate Multiple Members Borrowing Books**

1. **Create a class BorrowBookTask that implements Runnable**.
   * It should accept a Member and a Book and try to borrow the book.
   * It should handle BookLimitExceededException.
2. **Use Thread class** to run multiple members borrowing books **simultaneously**.
3. **Synchronize the borrowBook(Book book) method** in Member class to ensure thread safety.

public synchronized void borrowBook(Book book) throws BookLimitExceededException {

if (borrowedBooks.size() >= 3) {

throw new BookLimitExceededException("Limit exceeded!");

}

borrowedBooks.add(book);

}

1. Demonstrate multithreaded borrowing:

Thread t1 = new Thread(new BorrowBookTask(member1, book1));

Thread t2 = new Thread(new BorrowBookTask(member2, book2));

t1.start();

t2.start();

**Task 6: (Optional Advanced) Auto-Save Member Data Using a Background Thread**

* Create a thread that runs in the background and **periodically auto-saves the list of members** to the file every 15 seconds.

class AutoSaveTask extends Thread {

private List<Member> members;

public void run() {

while (true) {

try {

Thread.sleep(15000);

saveMembers(members); // implement saveMembers using ObjectOutputStream

System.out.println("Auto-saved members.");

} catch (Exception e) {

e.printStackTrace();

}

}

}

}