

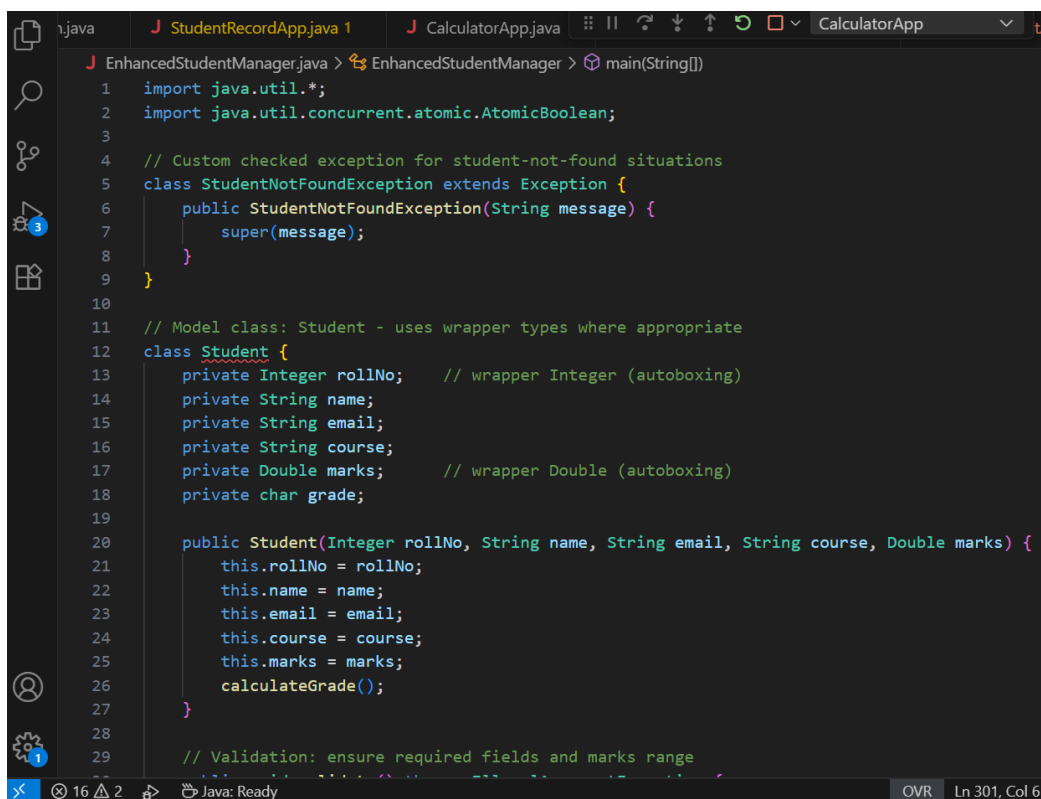
# Lab Assignment – 3

Name: Aman

Roll no: 2401201115

Course: BCA (AI&DS)

INPUT:

A screenshot of an IDE window showing a Java file named EnhancedStudentManager.java. The code defines a custom exception StudentNotFoundException, a Student model class with attributes rollNo, name, email, course, marks, and grade, and a main method. The IDE interface includes a toolbar at the top with icons for file operations, a left sidebar with icons for Explorer, Search, Source Explorer, Run and Debug, and a bottom status bar showing 'Java: Ready' and 'Ln 301, Col 6'.

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
1  import java.util.*;
2  import java.util.concurrent.atomic.AtomicBoolean;
3
4  // Custom checked exception for student-not-found situations
5  class StudentNotFoundException extends Exception {
6      public StudentNotFoundException(String message) {
7          super(message);
8      }
9  }
10
11 // Model class: Student - uses wrapper types where appropriate
12 class Student {
13     private Integer rollNo;    // wrapper Integer (autoboxing)
14     private String name;
15     private String email;
16     private String course;
17     private Double marks;     // wrapper Double (autoboxing)
18     private char grade;
19
20     public Student(Integer rollNo, String name, String email, String course, Double marks) {
21         this.rollNo = rollNo;
22         this.name = name;
23         this.email = email;
24         this.course = course;
25         this.marks = marks;
26         calculateGrade();
27     }
28
29     // Validation: ensure required fields and marks range
30     ...
31 }
```

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
12 class Student {
30     public void validate() throws IllegalArgumentException {
31         if (rollNo == null) throw new IllegalArgumentException(s: "Roll number cannot be null.");
32         if (name == null || name.trim().isEmpty()) throw new IllegalArgumentException(s: "Name cannot be empty.");
33         if (email == null || email.trim().isEmpty()) throw new IllegalArgumentException(s: "Email cannot be empty.");
34         if (course == null || course.trim().isEmpty()) throw new IllegalArgumentException(s: "Course cannot be empty.");
35         if (marks == null) throw new IllegalArgumentException(s: "Marks cannot be null.");
36         if (marks < 0.0 || marks > 100.0) throw new IllegalArgumentException(s: "Marks must be between 0 and 100.");
37     }
38
39     public void calculateGrade() {
40         if (marks == null) {
41             grade = 'F';
42             return;
43         }
44         double m = marks.doubleValue(); // unboxing
45         if (m >= 90) grade = 'A';
46         else if (m >= 75) grade = 'B';
47         else if (m >= 50) grade = 'C';
48         else if (m >= 33) grade = 'D';
49         else grade = 'F';
50     }
51
52     public Integer getRollNo() { return rollNo; }
53     public String getName() { return name; }
54     public String getEmail() { return email; }
55     public String getCourse() { return course; }
56     public Double getMarks() { return marks; }
57     public char getGrade() { return grade; }
}
```

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
12 class Student {
59     @Override
60     public String toString() {
61         return "Roll No: " + rollNo + "\n" +
62             "Name: " + name + "\n" +
63             "Email: " + email + "\n" +
64             "Course: " + course + "\n" +
65             "Marks: " + marks + "\n" +
66             "Grade: " + grade;
67     }
68 }
69
70 // Interface defining record actions
71 interface RecordActions {
72     boolean addStudent(Student s) throws IllegalArgumentException;
73     boolean deleteStudent(Integer rollNo) throws StudentNotFoundException;
74     Student searchStudent(Integer rollNo) throws StudentNotFoundException;
75     List<Student> viewAllStudents();
76 }
77
78 // Loader Runnable: simulates loading/saving with console progress
79 class Loader implements Runnable {
80     private final String message;
81     private final int steps;
82     private final long delayMillis;
83     private final AtomicBoolean running;
84
85     public Loader(String message, int steps, long delayMillis, AtomicBoolean running) {
86         this.message = message;
87     }
88 }
89
90 // Main method
91 public static void main(String[] args) {
92     // Create a manager instance
93     EnhancedStudentManager manager = new EnhancedStudentManager();
94
95     // Add a student
96     Student s1 = new Student(1, "John Doe", "john.doe@example.com", "Computer Science", 85.5);
97     manager.addStudent(s1);
98
99     // Search for a student
100    Student s2 = manager.searchStudent(1);
101    System.out.println(s2);
102
103    // View all students
104    List<Student> students = manager.viewAllStudents();
105    for (Student student : students) {
106        System.out.println(student);
107    }
108
109    // Delete a student
110    manager.deleteStudent(1);
111
112    // Loader example
113    AtomicBoolean running = new AtomicBoolean(false);
114    Loader loader = new Loader("Loading data...", 10, 500, running);
115    Thread thread = new Thread(loader);
116    thread.start();
117
118    // Wait for the loader to finish
119    while (running.get()) {
120        // Do something while waiting
121    }
122
123    // Print a message after the loader finishes
124    System.out.println("Data loaded successfully!");
125
126    // Close the application
127    System.exit(0);
128 }
```

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
79 class Loader implements Runnable {
85     public Loader(String message, int steps, long delayMillis, AtomicBoolean running) {
87         this.steps = steps;
88         this.delayMillis = delayMillis;
89         this.running = running;
90     }
91
92     @Override
93     public void run() {
94         try {
95             System.out.print(message);
96             for (int i = 0; i < steps && running.get(); i++) {
97                 Thread.sleep(delayMillis);
98                 System.out.print(s: ".");
99             }
100             System.out.println();
101         } catch (InterruptedException e) {
102             Thread.currentThread().interrupt();
103             System.out.println(x: "\nLoading interrupted.");
104         }
105     }
106 }
107
108 // StudentManager implements RecordActions; thread-safe using synchronization
109 class StudentManager implements RecordActions {
110     private final Map<Integer, Student> studentMap = new HashMap<>();
111     private final List<Student> studentList = new ArrayList<>();
112
113     @Override
```

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
109 class StudentManager implements RecordActions {
114     public synchronized boolean addStudent(Student s) throws IllegalArgumentException {
115         s.validate(); // may throw IllegalArgumentException for invalid data
116         Integer key = s.getRollNo(); // autoboxing already done
117         if (studentMap.containsKey(key)) {
118             return false; // duplicate roll prevented
119         }
120         studentMap.put(key, s);
121         studentList.add(s);
122         return true;
123     }
124
125     @Override
126     public synchronized boolean deleteStudent(Integer rollNo) throws StudentNotFoundException {
127         if (!studentMap.containsKey(rollNo)) throw new StudentNotFoundException("Student with roll " + rollNo + " not found.");
128         Student removed = studentMap.remove(rollNo);
129         studentList.removeIf(st -> st.getRollNo().equals(rollNo));
130         return removed != null;
131     }
132
133     @Override
134     public synchronized Student searchStudent(Integer rollNo) throws StudentNotFoundException {
135         Student s = studentMap.get(rollNo);
136         if (s == null) throw new StudentNotFoundException("Student with roll " + rollNo + " not found.");
137         return s;
138     }
139
140     @Override
141     public synchronized List<Student> viewAllStudents() {
```

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
109 class StudentManager implements RecordActions {
141     public synchronized List<Student> viewAllStudents() {
142         return new ArrayList<>(studentList); // return copy
143     }
144 }
145
146 // Main application with menu, input parsing, threading, and exception handling
147 public class EnhancedStudentManager {
148     private final Scanner scanner = new Scanner(System.in);
149     private final StudentManager manager = new StudentManager();
150
151     // helper to read Integer using wrapper class parsing and autoboxing
152     private Integer readInteger(String prompt) {
153         while (true) {
154             System.out.print(prompt);
155             String line = scanner.nextLine().trim();
156             try {
157                 return Integer.valueOf(line); // wrapper Integer
158             } catch (NumberFormatException e) {
159                 System.out.println(x: "Invalid integer. Please enter a valid whole number.");
160             }
161         }
162     }
163
164     // helper to read Double using wrapper class parsing
165     private Double readDouble(String prompt) {
166         while (true) {
167             System.out.print(prompt);
168             String line = scanner.nextLine().trim();
```

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
147 public class EnhancedStudentManager {
165     private Double readDouble(String prompt) {
168         String line = scanner.nextLine().trim();
169         try {
170             return Double.valueOf(line); // wrapper Double
171         } catch (NumberFormatException e) {
172             System.out.println(x: "Invalid number. Please enter a valid decimal number (e.g., 77.5).");
173         }
174     }
175
176     private String readNonEmpty(String prompt) {
177         while (true) {
178             System.out.print(prompt);
179             String s = scanner.nextLine().trim();
180             if (!s.isEmpty()) return s;
181             System.out.println(x: "Input cannot be empty.");
182         }
183     }
184
185     // Start loader thread and optionally wait for it
186     private void simulateLoading(String message, int steps, long delayMillis) {
187         AtomicBoolean running = new AtomicBoolean(initialValue: true);
188         Thread loaderThread = new Thread(new Loader(message, steps, delayMillis, running));
189         loaderThread.start();
190         try {
191             loaderThread.join(); // wait for completion (simulate synchronous save)
192         } catch (InterruptedException e) {
193             Thread.currentThread().interrupt();
194         }
```

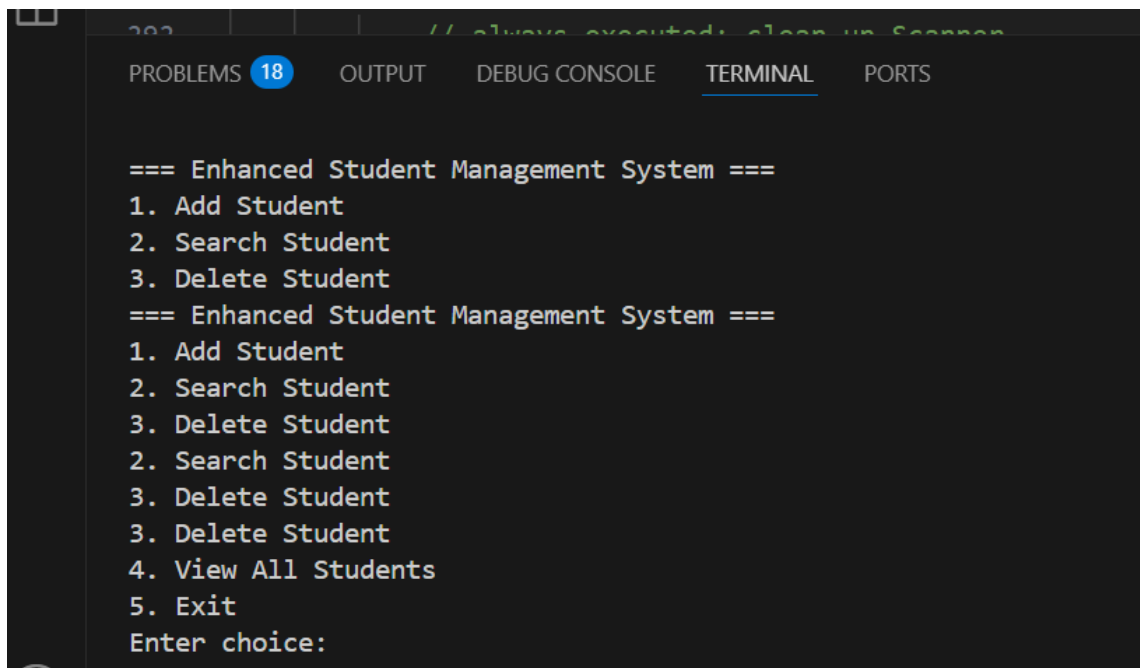
```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
147 public class EnhancedStudentManager {
187     private void simulateLoading(String message, int steps, long delayMillis) {
195         running.set(new Value<Boolean>() {
196             public boolean get() {
197                 return false;
198             }
199         });
200     }
201     private void addStudentFlow() {
202         try {
203             Integer roll = readInteger(prompt: "Enter Roll No (Integer): ");
204             String name = readNonEmpty(prompt: "Enter Name: ");
205             String email = readNonEmpty(prompt: "Enter Email: ");
206             String course = readNonEmpty(prompt: "Enter Course: ");
207             Double marks = readDouble(prompt: "Enter Marks (0-100): ");
208
209             // build student and validate inside manager.addStudent
210             Student s = new Student(roll, name, email, course, marks);
211
212             // simulate loading/saving via thread
213             simulateLoading(message: "Loading", steps: 5, delayMillis: 300);
214
215             boolean added = manager.addStudent(s); // may throw IllegalArgumentException
216             if (!added) {
217                 System.out.println("Error: Student with roll number " + roll + " already exists.");
218             } else {
219                 System.out.println(x: "Student added successfully.\n");
220                 System.out.println(s);
221             }
222         } catch (IllegalArgumentException iae) {
```

```
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
222         System.out.println("Validation Error: " + iae.getMessage());
223     } catch (Exception ex) {
224         System.out.println("Unexpected error while adding student: " + ex.getMessage());
225     }
226 }
227
228 private void searchStudentFlow() {
229     try {
230         Integer roll = readInteger(prompt: "Enter Roll No to search: ");
231         // simulate quick loading during search
232         simulateLoading(message: "Searching", steps: 3, delayMillis: 200);
233         Student s = manager.searchStudent(roll); // may throw StudentNotFoundException
234         System.out.println("Student found:\n" + s);
235     } catch (StudentNotFoundException snfe) {
236         System.out.println("Error: " + snfe.getMessage());
237     } catch (Exception ex) {
238         System.out.println("Unexpected error while searching: " + ex.getMessage());
239     }
240 }
241
242 private void deleteStudentFlow() {
243     try {
244         Integer roll = readInteger(prompt: "Enter Roll No to delete: ");
245         simulateLoading(message: "Deleting", steps: 3, delayMillis: 200);
246         boolean deleted = manager.deleteStudent(roll);
247         if (deleted) System.out.println("Student with roll " + roll + " deleted.");
248     } catch (StudentNotFoundException snfe) {
```

```
CalculatorApp
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
147 public class EnhancedStudentManager {
242 private void deleteStudentFlow() {
249     System.out.println("Error: " + snfe.getMessage());
250 } catch (Exception ex) {
251     System.out.println("Unexpected error while deleting: " + ex.getMessage());
252 }
253 }
254
255 private void viewAllFlow() {
256     List<Student> all = manager.viewAllStudents();
257     if (all.isEmpty()) {
258         System.out.println(x: "No students to display.");
259         return;
260     }
261     System.out.println(x: "All Students:");
262     for (Student s : all) {
263         System.out.println(x: "-----");
264         System.out.println(s);
265     }
266 }
267
268 public void mainMenu() {
269     try {
270         boolean running = true;
271         while (running) {
272             System.out.println(x: "\n=== Enhanced Student Management System ===");
273             System.out.println(x: "1. Add Student");
274             System.out.println(x: "2. Search Student");
275             System.out.println(x: "3. Delete Student");
```

```
CalculatorApp
EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
147 public class EnhancedStudentManager {
268     public void mainMenu() {
276         System.out.println(x: "4. View All Students");
277         System.out.println(x: "5. Exit");
278         Integer choice = readInteger(prompt: "Enter choice: ");
279         switch (choice) {
280             case 1 -> addStudentFlow();
281             case 2 -> searchStudentFlow();
282             case 3 -> deleteStudentFlow();
283             case 4 -> viewAllFlow();
284             case 5 -> {
285                 System.out.println(x: "Exiting... Program execution completed.");
286                 running = false;
287             }
288             default -> System.out.println(x: "Invalid option. Choose 1-5.");
289         }
290     } finally {
291         // always executed: clean up Scanner
292         scanner.close();
293         System.out.println(x: "Scanner closed. Goodbye!");
294     }
295 }
296
297
298 public static void main(String[] args) {
299     EnhancedStudentManager app = new EnhancedStudentManager();
300     app.mainMenu();
301 }
```

## INPUT:

A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS 18', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The terminal content shows a menu for the 'Enhanced Student Management System'. The menu is displayed twice. The first instance shows options 1. Add Student, 2. Search Student, and 3. Delete Student. The second instance shows options 1. Add Student, 2. Search Student, 3. Delete Student, 2. Search Student, 3. Delete Student, 3. Delete Student, 4. View All Students, and 5. Exit. Below the menu, it says 'Enter choice:'.

```
// always executed: clean up Scanner

PROBLEMS 18 OUTPUT DEBUG CONSOLE TERMINAL PORTS

=== Enhanced Student Management System ===
1. Add Student
2. Search Student
3. Delete Student
=== Enhanced Student Management System ===
1. Add Student
2. Search Student
3. Delete Student
2. Search Student
3. Delete Student
3. Delete Student
4. View All Students
5. Exit
Enter choice:
```

# Explanation —

## 1. High-level overview

This is an **enhanced Student Management System** that demonstrates:

- **Exception handling** (built-in and custom),
- **Multithreading** (simulated loading),
- **Wrapper classes** (`Integer`, `Double`) and autoboxing,
- **Thread-safety** for shared data access.

It provides a menu-driven console UI to **add**, **search**, **delete**, and **view** students.

---

## 2. Main classes & roles

### Student

- Model class that stores student data using wrapper types:
  - `Integer rollNo`, `String name`, `String email`, `String course`, `Double marks`, `char grade`.
- `validate()` checks for null/empty fields and marks range (0–100). Throws `IllegalArgumentException` on invalid data.

- `calculateGrade()` computes grade (A/B/C/D/F) from marks.
- `toString()` formats student details for display.

### StudentNotFoundException

- Custom checked exception extending `Exception`.
- Thrown by `StudentManager` when a requested roll number is missing.

### RecordActions

- Interface declaring CRUD-like methods (`addStudent`, `deleteStudent`, `searchStudent`, `viewAllStudents`).

### StudentManager

- Implements `RecordActions`.
- Stores students in:
  - `Map<Integer, Student> studentMap` for O(1) lookup and duplicate prevention.
  - `List<Student> studentList` to preserve insertion order.
- Methods are `synchronized` to ensure thread-safety when modifying shared collections.
- `addStudent()` validates and prevents duplicate roll numbers.
- `searchStudent()` and `deleteStudent()` throw `StudentNotFoundException` if roll missing.
- `viewAllStudents()` returns a copy of the list.

### Loader (implements Runnable)

- Simulates a loading/saving process by printing a message and dots with delays.
- Controlled by an `AtomicBoolean` running flag and `Thread.sleep()` for delays.
- Intended to show how background work can be performed with threads.

### EnhancedStudentManager (main application)

- Handles console I/O (`Scanner`) and menu logic.
- Helper methods:
  - `readInteger()`, `readDouble()` — parse input into wrapper types (`Integer.valueOf`, `Double.valueOf`) with validation loops.
  - `readNonEmpty()` — enforces non-empty strings.
  - `simulateLoading()` — starts a `Loader` thread and `join()`s it (waits for completion) to simulate synchronous loading.
- Flow methods:
  - `addStudentFlow()` — reads input, constructs `Student`, simulates loading, calls `manager.addStudent()`, and prints feedback. Catches `IllegalArgumentException` for validation errors.
  - `searchStudentFlow()` — reads roll, simulates searching, calls `manager.searchStudent()`, catches `StudentNotFoundException`.

- `deleteStudentFlow()` — deletes student and handles `StudentNotFoundException`.
    - `viewAllFlow()` — prints all students.
  - `mainMenu()` loops until exit; finally closes the Scanner.
- 

### 3. Exception handling — where and why

- **Validation exceptions:** `Student.validate()` uses `IllegalArgumentException` to signal invalid/missing fields or out-of-range marks.
  - **Domain exception:** `StudentNotFoundException` (checked) is thrown when search/delete cannot find a student. Check vs unchecked: checked forces callers to handle or declare the exception.
  - **Input parsing:** `readInteger()` and `readDouble()` catch `NumberFormatException` and re-prompt the user.
  - **Top-level safety:** `addStudentFlow`, `searchStudentFlow`, and `deleteStudentFlow` wrap operations in try-catch to prevent program crashes and give friendly messages.
  - **Resource cleanup:** `mainMenu()` uses a finally block to close the Scanner.
- 

### 4. Multithreading & Loader

- `Loader` implements `Runnable` and prints a message followed by a series of dots with delays, simulating progress.
  - `simulateLoading()` creates a `Loader` thread and `join()`s it, producing synchronous visible loading. (The design can be changed to non-blocking if desired.)
  - `AtomicBoolean` running lets the loader detect interruptions safely.
  - `StudentManager` methods are synchronized so concurrent threads don't corrupt `studentMap/studentList` if you later make loader asynchronous.
- 

### 5. Wrapper classes & autoboxing

- Numeric inputs are parsed into **wrapper** types: `Integer.valueOf(line)` and `Double.valueOf(line)`. These are used everywhere as `Integer/Double`.
  - Autoboxing/unboxing occurs when:
    - storing into collections (`Map<Integer, Student>`)
    - performing numeric comparisons (`marks.doubleValue()` or auto-unbox)
  - Using wrappers demonstrates conversion and how Java treats primitives vs objects.
-

## 6. Example program flow (Add Student)

1. User selects **Add Student**.
  2. `readInteger()` obtains roll; non-integer input is re-prompted.
  3. `readNonEmpty()` obtains name, email, course.
  4. `readDouble()` obtains marks; invalid inputs re-prompted.
  5. Student **constructed**; `manager.addStudent(s)` calls `s.validate()` and adds to maps/lists if valid and not duplicate.
  6. `simulateLoading()` runs the `Loader` thread showing `Loading.....`
  7. On success, student details are printed; on validation error the error message is shown and the menu returns.
- 

## 7. Thread-safety & design notes

- `synchronized` methods prevent concurrent modification issues in this simple design.
  - `simulateLoading()` uses `join()` so the main thread waits — this is simple but blocks the main thread; to make UI more responsive, you could let the loader run asynchronously and not wait, using callbacks or futures.
  - `AtomicBoolean` and proper interruption handling make loader robust to interrupts.
- 

## 8. Limitations & possible improvements

- Currently `simulateLoading()` blocks (`join()`); change to non-blocking for true responsiveness.
- Persistence (file or DB) is not implemented — could be added inside loader threads.
- Email format and more complex validation could be added (regex).
- More fine-grained exception types could be used instead of `IllegalArgumentException`.
- Split classes into packages/files (`model`, `service`, `util`) for production structure.

