

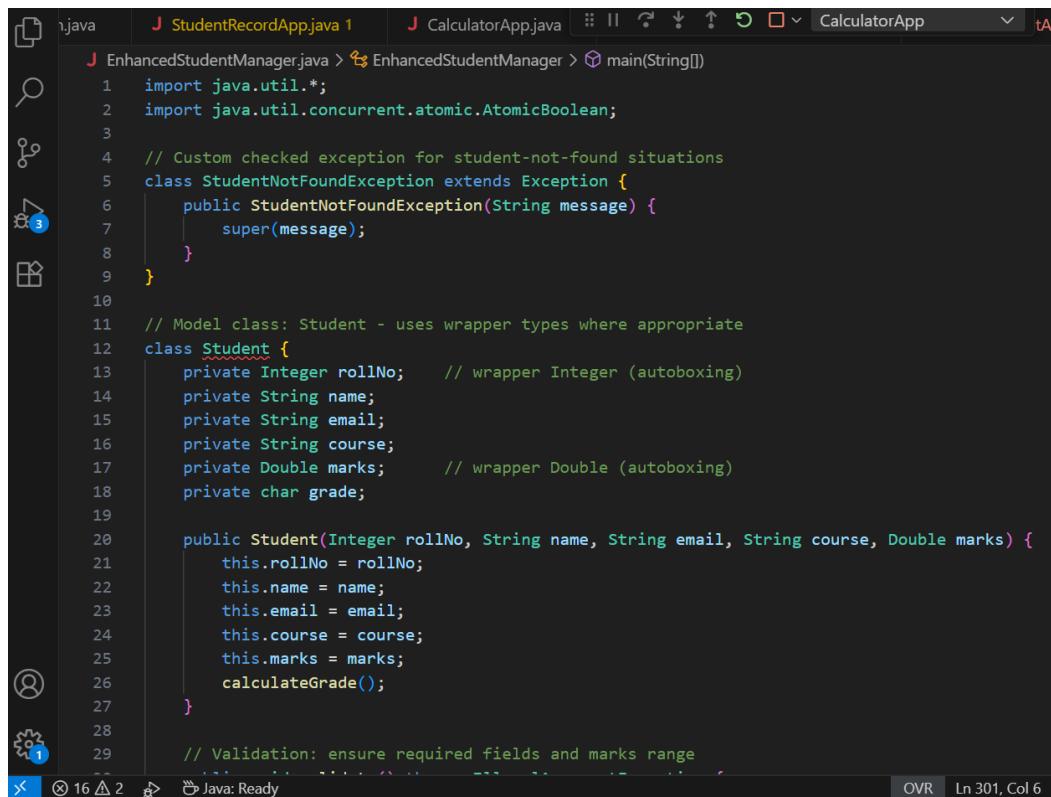
Lab Assignment – 3

Name: Aman

Roll no: 2401201115

Course: BCA (AI&DS)

INPUT:



The screenshot shows a Java code editor interface with two tabs open: EnhancedStudentManager.java and StudentRecordApp.java. The EnhancedStudentManager.java tab is active, displaying the following code:

```
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 }
```

The code editor has various status icons on the left and a toolbar at the top. The status bar at the bottom shows "Java: Ready" and "OVR Ln 301, Col 6".

The screenshot shows an IDE interface with multiple tabs open. The active tab is `StudentRecordApp.java`, which contains the following Java code:

```
1  package com.simplilearn;
2
3  public class StudentRecordApp {
4      public static void main(String[] args) {
5          Student student = new Student("John Doe", "john.doe@example.com", "Computer Science", 95);
6          student.validate();
7          student.calculateGrade();
8
9          System.out.println("Roll No.: " + student.getRollNo());
10         System.out.println("Name: " + student.getName());
11         System.out.println("Email: " + student.getEmail());
12         System.out.println("Course: " + student.getCourse());
13         System.out.println("Marks: " + student.getMarks());
14         System.out.println("Grade: " + student.getGrade());
15     }
16 }

```

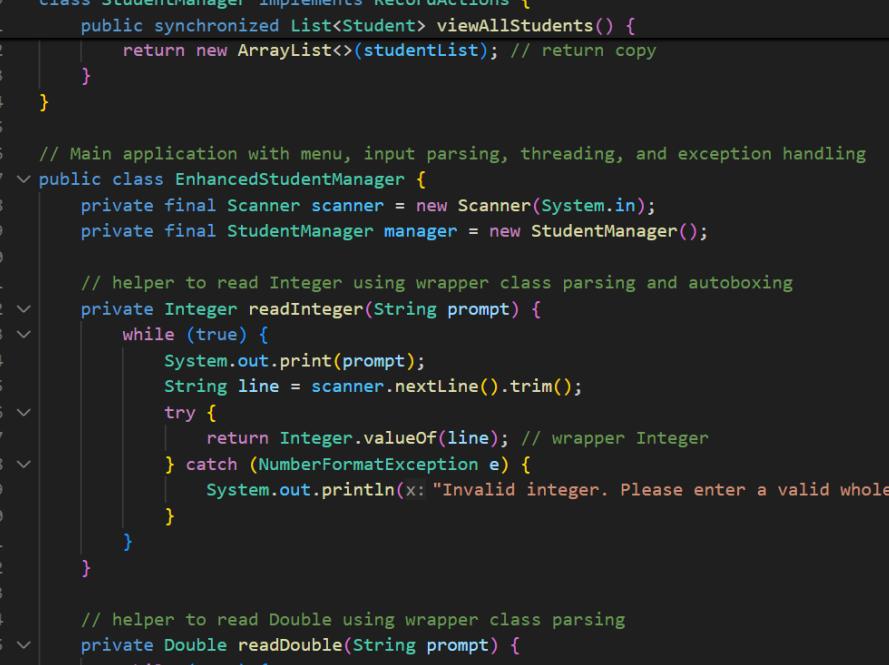
The code defines a `Student` class with methods for validating input, calculating grade based on marks, and retrieving student details. The `main` method creates a `Student` object and prints its details to the console.

The screenshot shows an IDE interface with several toolbars and icons on the left. The main area displays the code for EnhancedStudentManager.java:

```
12 class Student {
13     @Override
14     public String toString() {
15         return "Roll No: " + rollNo + "\n" +
16                "Name: " + name + "\n" +
17                "Email: " + email + "\n" +
18                "Course: " + course + "\n" +
19                "Marks: " + marks + "\n" +
20                "Grade: " + grade;
21     }
22 }
23
24 // Interface defining record actions
25 interface RecordActions {
26     boolean addStudent(Student s) throws IllegalArgumentException;
27     boolean deleteStudent(Integer rollNo) throws StudentNotFoundException;
28     Student searchStudent(Integer rollNo) throws StudentNotFoundException;
29     List<Student> viewAllStudents();
30 }
31
32 // Loader Runnable: simulates loading/saving with console progress
33 class Loader implements Runnable {
34     private final String message;
35     private final int steps;
36     private final long delayMillis;
37     private final AtomicBoolean running;
38
39     public Loader(String message, int steps, long delayMillis, AtomicBoolean running) {
40         this.message = message;
41     }
42 }
```

```
java J StudentRecordApp.java 1 J CalculatorApp.java 1 II ⌂ ↴ ↵ ↑ ⌂ □ v CalculatorApp
J EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
79  class Loader implements Runnable {
80      public Loader(String message, int steps, long delayMillis, AtomicBoolean running) {
81          this.message = message;
82          this.steps = steps;
83          this.delayMillis = delayMillis;
84          this.running = running;
85      }
86
87      @Override
88      public void run() {
89          try {
90              System.out.print(message);
91              for (int i = 0; i < steps && running.get(); i++) {
92                  Thread.sleep(delayMillis);
93                  System.out.print(".");
94              }
95              System.out.println();
96          } catch (InterruptedException e) {
97              Thread.currentThread().interrupt();
98              System.out.println("Loading interrupted.");
99          }
100     }
101 }
102 // StudentManager implements RecordActions; thread-safe using synchronization
103 class StudentManager implements RecordActions {
104     private final Map<Integer, Student> studentMap = new HashMap<>();
105     private final List<Student> studentList = new ArrayList<>();
106
107     @Override
```

```
java J StudentRecordApp.java 1 J CalculatorApp.java 1 II ⌂ ↴ ↵ ↑ ⌂ □ v CalculatorApp J EnhancedStudentManager
J EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
109  class EnhancedStudentManager implements RecordActions {
110
111     public synchronized boolean addStudent(Student s) throws IllegalArgumentException {
112         s.validate(); // may throw IllegalArgumentException for invalid data
113         Integer key = s.getRollNo(); // autoboxing already done
114         if (studentMap.containsKey(key)) {
115             return false; // duplicate roll prevented
116         }
117         studentMap.put(key, s);
118         studentList.add(s);
119         return true;
120     }
121
122     @Override
123     public synchronized boolean deleteStudent(Integer rollNo) throws StudentNotFoundException {
124         if (!studentMap.containsKey(rollNo)) throw new StudentNotFoundException("Student with roll " + rollNo + " not found.");
125         Student removed = studentMap.remove(rollNo);
126         studentList.removeIf(st -> st.getRollNo().equals(rollNo));
127         return removed != null;
128     }
129
130     @Override
131     public synchronized Student searchStudent(Integer rollNo) throws StudentNotFoundException {
132         Student s = studentMap.get(rollNo);
133         if (s == null) throw new StudentNotFoundException("Student with roll " + rollNo + " not found.");
134         return s;
135     }
136
137     @Override
138     public synchronized List<Student> viewAllStudents() {
139
140
141     }
142
143     @Override
144     public synchronized List<Student> viewAllStudents() {
```



The screenshot shows a Java code editor with the following details:

- Project Bar:** Shows projects: StudentRecordApp.java 1, CalculatorApp.java, and CalculatorApp.
- Code Area:** Displays the content of EnhancedStudentManager.java. The code implements RecordActions and provides methods for viewing all students and reading input from the scanner. It includes helper methods for reading Integer and Double values.
- Sidebar:** Features icons for file operations (New, Open, Save, Find, Replace, Cut, Copy, Paste, Delete), a search bar, and a help section.
- Status Bar:** Shows the status "Java: Ready" and the current line and column information "Ln 301, Col 6".

```
J StudentRecordApp.java 1 J CalculatorApp.java :: || ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋ CalculatorApp
J EnhancedStudentManager.java > EnhancedStudentManager > main(String[])
109 class StudentManager implements RecordActions {
110     public synchronized List<Student> viewAllStudents() {
111         return new ArrayList<>(studentList); // return copy
112     }
113 }
114
115
116 // Main application with menu, input parsing, threading, and exception handling
117 public class EnhancedStudentManager {
118     private final Scanner scanner = new Scanner(System.in);
119     private final StudentManager manager = new StudentManager();
120
121     // helper to read Integer using wrapper class parsing and autoboxing
122     private Integer readInteger(String prompt) {
123         while (true) {
124             System.out.print(prompt);
125             String line = scanner.nextLine().trim();
126             try {
127                 return Integer.valueOf(line); // wrapper Integer
128             } catch (NumberFormatException e) {
129                 System.out.println("x: Invalid integer. Please enter a valid whole number.");
130             }
131         }
132     }
133
134     // helper to read Double using wrapper class parsing
135     private Double readDouble(String prompt) {
136         while (true) {
137             System.out.print(prompt);
138             String line = scanner.nextLine().trim();
139             try {
140                 return Double.valueOf(line); // wrapper Double
141             } catch (NumberFormatException e) {
142                 System.out.println("x: Invalid double. Please enter a valid decimal number.");
143             }
144         }
145     }
146 }
```

StudentRecordApp.java 1

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

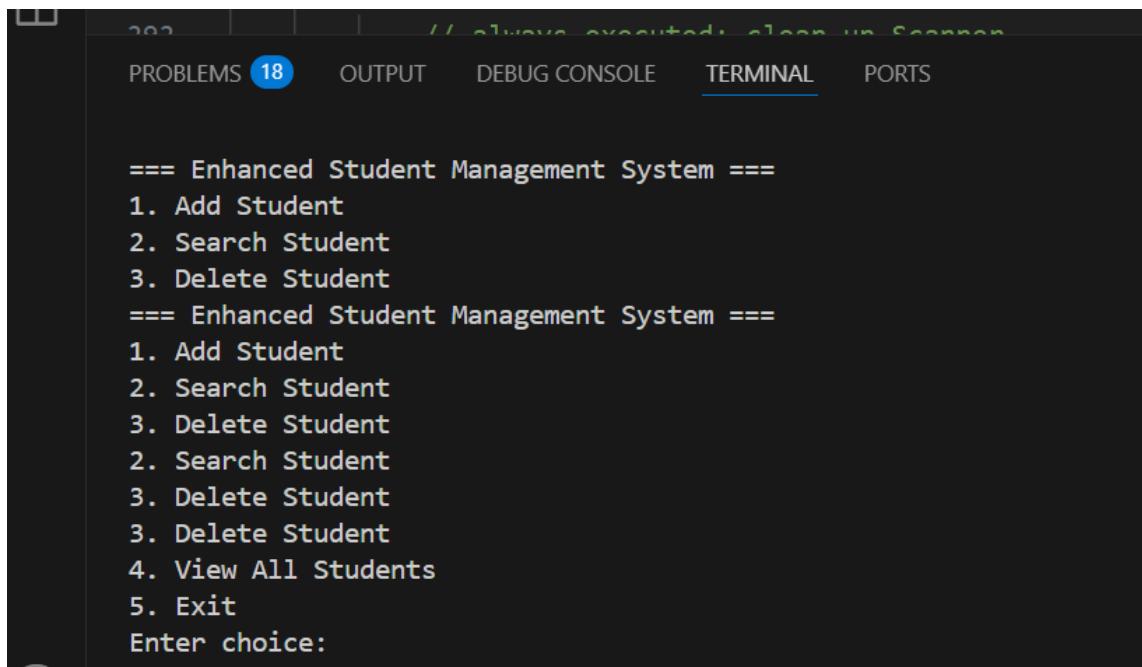
StudentRecordApp.java 1

```
147 public class EnhancedStudentManager {
200     private void addStudentFlow() {
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) {
```

```
147 public class EnhancedStudentManager {
148     private void deleteStudentFlow() {
149         System.out.println("Error: " + snfe.getMessage());
150     } catch (Exception ex) {
151         System.out.println("Unexpected error while deleting: " + ex.getMessage());
152     }
153 }
154
155 private void viewAllFlow() {
156     List<Student> all = manager.viewAllStudents();
157     if (all.isEmpty()) {
158         System.out.println(x: "No students to display.");
159         return;
160     }
161     System.out.println(x: "All Students:");
162     for (Student s : all) {
163         System.out.println(x: "-----");
164         System.out.println(s);
165     }
166 }
167
168 public void mainMenu() {
169     try {
170         boolean running = true;
171         while (running) {
172             System.out.println(x: "\n==== Enhanced Student Management System ====");
173             System.out.println(x: "1. Add Student");
174             System.out.println(x: "2. Search Student");
175             System.out.println(x: "3. Delete Student");
176             System.out.println(x: "4. View All Students");
177             System.out.println(x: "5. Exit");
178             Integer choice = readInteger(prompt: "Enter choice: ");
179             switch (choice) {
180                 case 1 -> addStudentFlow();
181                 case 2 -> searchStudentFlow();
182                 case 3 -> deleteStudentFlow();
183                 case 4 -> viewAllFlow();
184                 case 5 -> {
185                     System.out.println(x: "Exiting... Program execution completed.");
186                     running = false;
187                 }
188                 default -> System.out.println(x: "Invalid option. Choose 1-5.");
189             }
190         }
191     } finally {
192         // always executed: clean up Scanner
193         scanner.close();
194         System.out.println(x: "Scanner closed. Goodbye!");
195     }
196 }
197
198 Run | Debug
199 public static void main(String[] args) [
200     EnhancedStudentManager app = new EnhancedStudentManager();
201     app.mainMenu();
202 ]
```

```
147 public class EnhancedStudentManager {
148     public void mainMenu() {
149         System.out.println(x: "4. View All Students");
150         System.out.println(x: "5. Exit");
151         Integer choice = readInteger(prompt: "Enter choice: ");
152         switch (choice) {
153             case 1 -> addStudentFlow();
154             case 2 -> searchStudentFlow();
155             case 3 -> deleteStudentFlow();
156             case 4 -> viewAllFlow();
157             case 5 -> {
158                 System.out.println(x: "Exiting... Program execution completed.");
159                 running = false;
160             }
161             default -> System.out.println(x: "Invalid option. Choose 1-5.");
162         }
163     }
164
165     } finally {
166         // always executed: clean up Scanner
167         scanner.close();
168         System.out.println(x: "Scanner closed. Goodbye!");
169     }
170 }
171
172 Run | Debug
173 public static void main(String[] args) [
174     EnhancedStudentManager app = new EnhancedStudentManager();
175     app.mainMenu();
176 ]
```

INPUT:



The screenshot shows a terminal window with the following content:

```
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`.

- o `deleteStudentFlow()` — deletes student and handles `StudentNotFoundException`.
 - o `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:
 - o storing into collections (`Map<Integer, Student>`)
 - o 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.

