July 2025 CSE 214: Software Engineering Sessional Assignment 2: Behavioral Design Patterns

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

January 2026

Overview

This assignment provides hands-on experience with identifying design problems in existing code and applying behavioral design patterns to improve software architecture. You are given a functional but poorly structured Java codebase for a university course registration system. Your task is to analyze the existing implementation, identify appropriate locations where behavioral design patterns can address structural issues, and refactor the code accordingly. The goal is not merely to apply patterns mechanically, but to understand *why* certain patterns are appropriate for specific problems and how they improve maintainability, extensibility, and separation of concerns.

Change Log

Description	Updated By	Timestamp
Assignment declared	NTD	9 JAN, 2026, 06:30 PM

1 Overview

The provided codebase implements a console-based university course registration system in Java (no external libraries). The system supports the following key functionalities:

- Student operations: Enrollment, waitlisting, dropping courses, viewing schedules
- Admin operations: Managing courses, adjusting capacity, viewing rosters

While the system is fully functional, it contains intentional structural issues that make the code difficult to maintain and extend. Your task is to identify these issues and apply appropriate behavioral design patterns to address them.

2 Codebase

The complete source code, along with detailed documentation and test scenarios is available at:

https://github.com/tahmid-404-20/CSE214-SWE-Designed-Assignments/tree/main/july2025/behavioral

The repository contains the following files:

- `CourseStatus.java`
- `Student.java`
- `Course.java`
- `RegistrarSystem.java`
- `ConsoleUI.java`
- `Main.java`
- `TestScenarios.java`

Refer to the repository's `README.md` for compilation instructions, execution modes, and comprehensive documentation of system behavior.

3 Design Issues

You must analyze and address the following two design issues using appropriate behavioral design patterns.

Issue 1: Communication and Coordination Problems

Observations:
The code exhibits scattered communication between objects with no central coordination:

- `Student` directly calls `Course` methods (`tryEnroll`, `addToWaitlist`, `dropStudent`)
- `Course` directly manipulates `Student` objects (e.g., when promoting from waitlist)
- `ConsoleUI` directly calls methods on both `Student` and `Course`
- Enrollment business logic is split between `Student` and `Course` classes
- Both classes maintain duplicate state (lists of enrolled/waitlisted courses)
- No single component validates or coordinates complete operations

Question: *What if "someone who already knows all" was responsible for coordinating all operations between `Student` and `Course`?*

Issue 2: Conditional Complexity for Course States

Observations:
The `Course` class contains extensive conditional logic based on `CourseStatus`:

- `tryEnroll()` uses a large `switch` statement checking current status
- `addToWaitlist()` has conditional logic checking if state allows waitlisting
- `setStatusAdmin()` has nested conditionals validating transitions from each state
- `setCapacity()` checks status to decide response behavior
- Adding new states or changing transition rules requires modifying multiple methods
- The same status checks appear repeatedly across different methods

Question: *Could each course status have its own encapsulation that knows how to handle `tryEnroll()`, `addToWaitlist()`, `canTransitionTo()`, etc.?*

4 Key tasks

1. **Pattern Identification:** For each issue, identify the behavioral design pattern that best address the problem. You have to justify your choices during viva.
2. **Implementation:** Submit the refactored Java codebase with all original functionality preserved.
3. **Testing:** Ensure all 25 test scenarios in `TestScenarios.java` pass after refactoring.

5 Evaluation Criteria

Your submission will be evaluated on:

- Correctness of pattern identification and justification
- Code quality and adherence to design principles
- Preservation of original system functionality
- No mark will be given on code that you cannot explain.

6 Grading Rubric

Criterion	Points	Details
Correctness & Functionality	20	Code compiles, all commands work, outputs match baseline, no regressions
Design Issue 1 Resolution	40	Addressing the issue with the correct design pattern
Design Issue 2 Resolution	40	Addressing the issue with the correct design pattern
TOTAL	**100**	

7 Submission Protocol

- Create a directory named with your 7-digit student ID (2105XXX)
- Place all source code within this directory
- Compress the folder and rename to `2105XXX.zip`
- Submit the compressed archive

Deadline: January 23, 2025 (Friday), 11:59 PM

8 Warning

1. Don't copy! We regularly use copy checkers. Do not copy codes from online resources and LLMs.
2. First time copier and copyee will receive negative marking because of dishonesty. Their default is bigger than those who will not submit.
3. Repeated occurrence will lead to severe departmental action and jeopardize your academic career. We expect fairness and honesty from you. Don't disappoint us!