Practice Assignment 1

Problem 1. (50%) Singleton Design Pattern

Scenario 1. You're given the task of building a course registration system. Due to the time limitation of the project, you decided to apply Singleton pattern for its simplicity, your goal is to ensure that there is a single point of control for managing courses and student registrations. More detailed description of the scenario as follows:

Components:

1. **Course**:
    - Each course is identified by a course ID and has a name.
2. **Student**:
    - Each student is identified by a student ID and has a name.
3. **Registration**:
    - A registration represents a student's enrollment in a specific course.
    - The Registration class includes references to the associated Student and Course.
4. **CourseManager** (Singleton):
    - The CourseManager class is implemented as a Singleton to ensure there is only one instance responsible for managing courses, students, and registrations.
    - It includes methods for **adding** and **removing** courses and students, **registering** and **unregistering** students for courses, and **retrieving** lists of courses, students, and registrations.

a) Implement *course registration system* with Lazy initialization. **(25%)**
b) Implement *course registration system* with Eager Initialization. **(25%)**

Problem 2. (50%) Facade Design Pattern

Scenario 2. You're developing an application that needs to write data to files in various formats (CSV, JSON, and XML), optionally compressing and encrypting the data for security and storage efficiency. You want to provide a simple interface for client code, hiding the complexities of format conversions, compression, and encryption.

Components:

1. **FileWriterFacade:** Acts as the main interface for client code. It exposes a simplified method writeData that takes the data, format, compression, and encryption options as arguments.
2. **CompressionCodec:** Handles data compression using different algorithms (Gzip, or Bzip2).
3. **EncryptionCodec:** Encrypts data using various encryption algorithms (AES, DES).
4. **FileWriterFactory:** Creates specific file writers based on the chosen format (CSVFileWriter, JSONFileWriter, XMLFileWriter).
5. **Concrete File Writers:** Implements the writeData method for each supported format, handling specific formatting and file writing logic.

Implement the Facade pattern to create a simplified interface for file operations. **(50%)**

~The end~

Your project/ solution should be submitted to Moodle before the deadline.