PROG-2200 Assignment 2

Title

Refactoring code to meet the Single-Responsibility Principle

Learning Outcome

Develop an application by selecting and utilizing appropriate design patterns to support object-oriented design principles.

Value

10%

Related Modules

SOLID

Learning Outcomes Evaluated

Develop an application by selecting and utilizing appropriate design patterns to support object-oriented design principles.

Instructions

Refactor each of the **three** code examples given in class, breaking out existing methods (functions) into new classes as appropriate.

- **Order Processing System:** A class that manages customer orders. Initially, it includes methods for adding items to an order, calculating the total order cost, applying discounts, processing payments, and generating order confirmation emails. Students will need to refactor this class to separate these responsibilities.
- **Employee Management System:** A class that handles various employee-related functionalities such as adding new employees, calculating salaries, managing employee benefits, and generating performance reports. The assignment would involve breaking down this class into smaller classes, each handling a specific aspect of employee management.
- **School Management System:** A class designed to manage different aspects of a school system. It includes methods for enrolling students, scheduling classes, recording grades, and handling school events. Students will refactor this class to ensure each functionality is managed by its dedicated class.

For each class, students should identify the multiple responsibilities it currently holds and then create separate classes to handle these responsibilities individually. This exercise will help them understand and apply the SRP, enhancing their skills in creating maintainable and scalable software.

Include your .java classes in your submission, as well as a Main.java class for each that sets up a runnable example.

Deliverables

Code in a ZIP file or GitHub URL in a .txt file submitted to Brightspace

Evaluation

Learning Outcomes	**Components**	**Points**
Develop an application by selecting and utilizing appropriate design patterns to support object-oriented design principles.	Single Responsibility Principle #1	/4
	#2	/4
	#3	/4
	Clean code	/4
Outcome Mark Value		/16
Total Mark Value		/16

Grading Rubric

Single Responsibility Principle Example #1: Order Processing System (4 points)

- 4 points: The class is refactored to separate responsibilities into distinct classes (e.g., Order, Item, Discount, Payment, EmailService). Each new class has a single responsibility and is well-encapsulated.
- 3 points: Most responsibilities are separated into distinct classes, but there is minor overlap or one responsibility is not fully encapsulated.
- 2 points: The class is partially refactored, but several responsibilities are still combined in some classes.
- 1 point: Minimal refactoring; the majority of responsibilities are still within the original class.
- 0 points: No attempt to refactor; the class remains unchanged.

Single Responsibility Principle Example #2: Employee Management System (4 points)

- 4 points: The class is refactored to separate responsibilities into distinct classes (e.g., Employee, SalaryCalculator, BenefitsManager, PerformanceReport). Each new class has a single responsibility and is well-encapsulated.
- 3 points: Most responsibilities are separated into distinct classes, but there is minor overlap or one responsibility is not fully encapsulated.
- 2 points: The class is partially refactored, but several responsibilities are still combined in some classes.
- 1 point: Minimal refactoring; the majority of responsibilities are still within the original class.
- 0 points: No attempt to refactor; the class remains unchanged.

Single Responsibility Principle Example #3: School Management System (4 points)

- 4 points: The class is refactored to separate responsibilities into distinct classes (e.g., Student, Enrollment, ClassSchedule, GradeRecord, EventManager). Each new class has a single responsibility and is well-encapsulated.
- 3 points: Most responsibilities are separated into distinct classes, but there is minor overlap or one responsibility is not fully encapsulated.
- 2 points: The class is partially refactored, but several responsibilities are still combined in some classes.
- 1 point: Minimal refactoring; the majority of responsibilities are still within the original class.
- 0 points: No attempt to refactor; the class remains unchanged.

Clean Code (4 points)

- 4 points: The code is well-organized, properly commented, uses meaningful variable names, and follows consistent formatting. All new classes and methods are clearly defined and documented.
- 3 points: The code is mostly clean with minor issues in organization, comments, or formatting. New classes and methods are mostly well-defined.
- 2 points: The code is somewhat clean but has several issues with organization, comments, or formatting. Some new classes or methods are not clearly defined.
- 1 point: The code has significant issues with organization, comments, or formatting. Many new classes or methods are poorly defined.
- 0 points: The code is unorganized, poorly commented, and inconsistently formatted. New classes and methods are not clearly defined.

Total: 16 points