Лабораторна робота №2
Проектування та розробка програм з використанням патернів проектування
Варіант 1

Для виконання цієї лабораторної роботи я взяла вже існуючий проект за основу (що і в попередній лабораторній) .

Патерни програмування поділяються на :

- Породжуючі/ твірні (Creational)
- Структурні (Structural)
- Поведінкові (Behavioural) .

При виконанні даної лабораторної роботи я використала 7 патернів проектування, які підвищують гнучкість проекту , масштабованість і підтримуваність , а саме :

Creational	***Structural***	***Behavioural***
Singleton	Adapter	Strategy
Factory Method		Observer
		Command
		Template Method

Конкретніше про кожен паттерн та його застосування:

Singleton *– належежить до породжуючих патернів .*

Забезпечує єдиний доступний екземпляр класу для управління задачами протягом всього життєвого циклу програми. Це гарантує, що всі частини програми працюють з одним набором даних задач.

Плюси	***Мінуси***
Забезпечує єдиний доступ до екземпляра класу, що може бути корисно для керування доступом до спільного ресурсу	Ускладнюють юніт-тестування
Гарантує створення лише одного екземпляра класу	Можуть мати більше одного обов'язку, що суперечить принципам чистої архітектури
Зменшує необхідність використання глобальних змінних, які можуть бути важкими для відстеження і контролю	Може призвести до жорсткого зв'язування компонентів, що ускладнює їхню заміну або розширення
	Потребує додаткової синхронізації для забезпечення безпеки, що може

	ускладнити реалізацію і вплинути на продуктивність

Використання у коді : «TaskManager»

Factory Method *– належежить до породжуючих патернів* **.**

Відповідає за створення задач різних типів (простих та складних). Це забезпечує гнучкість у створенні нових типів задач без необхідності змінювати код, що використовує ці задачі.

Плюси	***Мінуси***
Дозволяє змінювати тип створюваних об'єктів без зміни клієнтського коду	Додавання фабрик додає складності , що може ускладнити розуміння коду
Легко додавати нові типи задач	Патерн не дуже легкий для сприйняття та розуміння
Спрощує підміну конкретних реалізацій на підставні (mock) об'єкти	Інколи простіше створити об'єкт напряму, ніж використовувати патерн

Використання у коді : «TaskFactory»

Adapter *– належежить до структурних патернів .*

Діє як адаптер між системою збереження задач і конкретними реалізаціями стратегій збереження, що дозволяє легко додавати нові способи збереження без зміни існуючого коду.

Плюси	***Мінуси***
Дозволяє використовувати сторонні або застарілі компоненти без зміни їх коду	Додавання адаптерів може ускладнює розуміння коду
Легко додавати нові адаптери для нових способів збереження	Використання адаптерів може додати накладні витрати на продуктивність
Зміни в адаптованих класах не впливають на клієнтський код	

Використання у коді : «ConcreteFileTaskSaver»

Strategy *– належежить до поведінкових патернів .*

Інтерфейс для визначення алгоритму збереження задач. Реалізовано дві конкретні стратегії - TextFileSaveStrategy та BinaryFileSaveStrategy, що дозволяє зберігати задачі у текстовий або бінарний файл відповідно.

Плюси	***Мінуси***

Можна легко змінювати алгоритми збереження без зміни клієнтського коду.	Для кожної наступної стратегії треба створювати новий клас.
Окремо зберігає алгоритми, роблячи їх більш читабельними та керованими.	Патерн не дуже легкий для сприйняття та розуміння
Легко додавати нові стратегії без зміни існуючого коду.	

Використання у коді : «TaskSaveStrategy», «TextFileSaveStrategy», «BinaryFileSaveStrategy»

Observer *– належежить до поведінкових патернів .*

Використовується для підписки на події інтерфейсу користувача (натискання кнопок), що дозволяє динамічно реагувати на дії користувача і оновлювати інтерфейс.

Плюси	***Мінуси***
Дозволяє різним частинам системи реагувати на події без жорсткого зв'язування	Важко відстежити потік управління при великій кількості спостерігачів
Легко додавати або видаляти спостерігачів без зміни суб'єкта	Може спричинити затримки через велике навантаження на обробку подій
Зменшує залежності між компонентами системи	

Використання у коді : «MainFrame»

Command *– належежить до поведінкових патернів .*

Ізолює код, що виконує конкретні дії, від коду, що ініціює ці дії. В обробці подій користувача наприклад: додавання, видалення, очищення задач.

Плюси	***Мінуси***
Дозволяє інкапсулювати запити як об'єкти, що спрощує управління ним	Кожна нова команда вимагає створення нового класу
Легко додавати нові команди без зміни існуючого коду	Може ускладнити архітектуру додатку
Легко реалізувати функції відміни та повтору дій	

Використання у коді : «MainFrame»

Template Method *– належежить до поведінкових патернів .*

Визначає шаблон для перевірки задач, де базовий клас визначає структуру перевірки, а похідні класи реалізують конкретні перевірки.

Плюси	***Мінуси***
Визначає основну структуру алгоритму в базовому класі, що дозволяє використовувати спільний код	Зміна основної структури алгоритму може бути важкою через наявність багатьох похідних класів
Легко реалізувати різні варіанти алгоритму, змінивши лише частини коду в похідних класах	Збільшена залежність похідних класів від базового класу
абезпечує контроль за порядком виконання етапів алгоритму	

Використання у коді : «TaskValidator», «BasicTaskValidator», «TaskTimeValidator»