Основи об'єктно-орієнтованого програмування
Лабораторна робота № 2
Проектування та розробка програм з використанням патернів проектування

Розробити програму для демонстрації роботи певного класу алгоритмів. Програма має реалізовувати графічний інтерфейс користувача (наприклад, з використанням Qt або інших фреймворків), збереження даних (наприклад, в файли чи базу даних).

Цю лабораторну роботу бажано виконувати в командах з 3-4 студентів (команди можуть бути такими ж, як для проектів, або відрізнятись.) Програма має бути спроектована таким чином, щоб був поділ на відносно незалежні компоненти, і можливість різним учасникам команди працювати над різними компонентами незалежно. В репозиторії має бути видно, ким був написаний кожен фрагмент коду (варіант «один з команди комітить за всіх» не приймається).

Можливі ідеї для компонентів:

1. Власне реалізації алгоритмів, структур даних.
2. Механізм виміру продуктивності алгоритмів (час виконання, обсяг пам’яті)
3. Механізм оцінки теоретичної складності алгоритмів.
4. Механізм підбору параметрів алгоритмів з метою підвищення продуктивності.
5. Візуалізація поведінки алгоритмів
6. Візуалізація теоретичної та практичної продуктивності алгоритмів (таблиці, графіки, …)
7. Документація щодо алгоритмів
8. Інші можливі компоненти, які мають сенс для цієї задачі

Можна розбити програму на дві частини: 1) бібліотеку з реалізацією алгоритмів чи структур даних та 2) застосунок з графічним інтерфейсом, який буде використовувати написану бібліотеку.

Необхідно підготувати документацію реалізованої програми. Зокрема, треба описати заплановану архітектуру програми (використовуючи текстові описи та/або UML діаграми). Також має бути опис того, як використовувати можливості програми – як через графічний інтерфейс, так і з коду.

Необхідно реалізувати юніт-тести для перевірки реалізованих алгоритмів. Зокрема, необхідно реалізувати перевірку того, що різні версії алгоритмів (тобто різні версії одного алгоритму) повертають однакові значення, якщо вони так мають працювати. Або перевірити властивості чи інваріанти структур даних чи алгоритмів (наприклад, що збалансоване дерево має приблизно однакову висоту у всіх гілках).

Юніт-тести мають більш-менш покривати реалізовану функціональність алгоритмів. Робити тести для інтерфейсу користувача, збереження даних в БД і т.д. не обов’язково (хоча можна, для отримання ***додаткових балів***)

Використання патернів проектування

Мета цієї лабораторної роботи – навчитись реалізовувати і використовувати на практиці патерни проектування (design patterns). Тому бажано реалізувати хоча б основні і найбільш популярні патерни.

Ідеї щодо використання патернів:

1. Патерн **Strategy** – варто використати для реалізації алгоритмів, з можливістю заміни реалізації без зміни клієнтського коду
2. Патерн **Template Method** – варто використати для реалізації варіантів алгоритмів, коли загальна структура лишається незмінною, але якісь аспекти реалізуються по-

різному в підкласах

3. Патерн **Composite** – варто використати для реалізації складних алгоритмів, які містять під-алгоритми. Можна також використати для реалізації структур даних.
4. Патерн **Decorator** – варто використати для модифікацій алгоритмів, наприклад алгоритми з додатковою перевіркою правильності параметрів. Також варто використати для вимірів часу виконання. Можна використати і для інших модифікацій.
5. Патерн **Iterator**– варто використати для обходу структур даних.
6. Патерн **Adapter**– варто використати з метою використання бібліотечних реалізацій алгоритмів (зі стандартної бібліотеки або сторонніх бібліотек)
7. Патерн **Abstract Factory** / **Factory Method**– варто використати для побудови стандартних реалізацій певних алгоритмів (залежно від налаштувань)
8. Патерн **Builder**– варто використати для побудови складних алгоритмів, які складаються з багатьох частин
9. Патерн **Singleton**– варто використати для підтримки класів, які мають існувати в одному екземплярі і бути доступними іншим класам (наприклад, класи для побудови алгоритмів)
10. Патерн **Visitor**– варто використати для розрахунку певних властивостей складних алгоритмів/структур даних, наприклад теоретичної складності чи обсягу пам’яті
11. Патерн **Bridge**– варто використати для створення кількох версій інтерфейсу алгоритму чи структури даних, і незалежного розвитку їх реалізацій
12. Патерн **Command**– варто використати для реалізації інтерактивного режиму роботи з алгоритмами, зокрема можливість виконувати послідовність операцій, а також режим «машини часу» з можливістю повернути назад операції
13. Патерн **Memento**– варто використати для збереження та відновлення стану алгоритмів та структур даних (можливість завершити програму посередині інтерактивного режиму і потім відновитись з тої ж позиції)
14. Патерн **Facade**– варто використати для загального інтерфейсу компонентів та взаємодії компонентів між собою.
15. Інші патерни як з книги GoF, так і з інших джерел – варто використовувати там, де це доцільно.

Зазвичай є рекомендація «не зловживати» використанням патернів, тобто не намагатись втиснути в одну програму усі патерни, які відомі розробнику. Для цієї лабораторної з демонстраційною метою можна послабити цю рекомендацію: тобто якщо в деякій ситуації можна реалізувати щось або з патернами, або без патернів – то варто про це написати в документації, і варто реалізувати варіант з патернами. (Можна також реалізувати обидва варіанти і додати механізм вибору між ними; за якісну реалізацію будуть виставлені ***додаткові бали***)

Можливі варіанти

Варіанти цієї лабораторної будуються навколо певного класу близьких алгоритмів. Необхідно реалізувати різні алгоритми з обраного класу, починаючи від простих і до досить складних.

Можливі ідеї варіантів

1. Списки і схожі структури
2. Дерева (в тому числі балансовані дерева пошуку)
3. Індексовані таблиці (реалізації на основі хеш-таблиць та дерев)
4. Сортування
5. Алгоритми на графах
6. Комбінаторні алгоритми (зокрема, з використанням динамічного програмування, backtracking, методу гілок і границь, …)

7. Алгоритми дискретної оптимізації (пошук максимуму/мінімуму)
8. Алгоритми теорії чисел
9. Алгоритми лінійної алгебри (вектори, матриці, лінійні рівняння, …)
10. Алгоритми математичного аналізу (похідні, інтеграли, як символьні, так і чисельні алгоритми)
11. Чисельні методи
12. Алгоритми математичної статистики

В рамках однієї теми можна фокусуватись на різних аспектах алгоритмів. Можна також запропонувати свою тему, але треба погодити її з викладачем.