**Основи об'єктно-орієнтованого програмування**

**Лабораторна робота № 1a**

**Моделювання з використанням UML**

# Загальні рекомендації

Метою виконання лабораторної роботи №1 є вивчення засобів проєктування та моделювання програмних систем, зокрема з використанням мови UML. Лабораторна складається з основного завдання 1a (описаного в цьому файлі), а також чотирьох додаткових завдань (додаткові завдання 1b, 1c, 1d, 1e). Виконання додаткових завдань не є обов’язковим, проте дозволить отримати ***додаткові бали***.

Для цієї лабораторної треба викладати в репозиторій не лише код, а й діаграми UML. Необхідно викладати як файли проєктів (відповідно до інструменту, який використовується для розробки діаграм), так і згенеровані з них зображення.

# Design/Implementation Modeling

Використати UML діаграми для опису структури існуючого коду та його рефакторінгу. Виконання цього завдання складається з наступних кроків:

1. Обрати існуючу програму/проєкт, з якою планується працювати. Це може бути навчальний проєкт з минулого семестру, чи одна з лабораторних робіт, чи код з інших предметів, чи з якихось онлайн курсів, чи щось подібне. Також це може бути невеличкий open-source проєкт. Можна взяти кілька невеликих програм з метою їх подальшого об’єднання. Код має бути досить складним – тобто не рівня Hello world чи реалізації одного нескладного алгоритму (хоча це може бути кілька схожих чи якось пов’язаних алгоритмів, і на подальших кроках можна буде створити для них спільний програмний інтерфейс).
2. Реалізувати unit tests, шо описують функціональність обраної програми. (Якщо такі тести вже існують – їх можна доповнити, або залишити як є)
3. Побудувати UML діаграми, що описують обрану програму. Варто описати сценарії використання (**Use Case**), структуру коду (**Class, Component, Object, Composite Structure, Deployment, Package**), логіку та поведінку програми (**Sequence, Communication, Timing, Activity, Interaction Overview, State**). Окрім діаграм, можна додатково реалізувати текстові описи. Зокрема, обов’язково має бути присутнім **глосарій** проєкту. Для побудови деяких діаграм можна використати автоматичну генерацію діаграм з коду; але при цьому діаграми мають бути зрозумілими. Наприклад, взяти 100 класів і кинути їх усі на одну діаграму класів – мабуть, не найкращий варіант ☺
4. Запропонувати якісь зміни в структурі/інтерфейсі/реалізації програми. Це може бути кращий object-oriented design, кращий поділ на компоненти чи відокремлення різних аспектів (наприклад, логіки програми від графічного інтерфейсу), використання якихось патернів проєктування, можливість вибору різних варіантів реалізації і т.д. Бажано використовувати побудовану модель програми для опису запропонованих змін. Запропоновані зміни треба узгодити з викладачем.
5. Реалізувати запропоновані зміни.
6. Перевірити, що нова версія програми не вносить зміни в логіку/алгоритми (якщо це не було заплановано). Використати для цього реалізовані раніше unit tests і аналогічні тести, які будуть реалізовані для нової версії.
7. Порівняти попередню та оновлену версії програми за часом виконання окремих алгоритмів/функцій, обсягом коду і т.д. Можна використовувати автоматизовані інструменти для вимірювання продуктивності, отримання метрик коду тощо (за доцільне використання таких інструментів будуть виставлені ***додаткові бали***).
8. Проаналізувати код (попередній, змінений, самі зміни) з точки зору дотримання принципів архітектури та проєктування взагалі, та ООП зокрема. Принципи, на які варто звернути увагу: основи ООП (інкапсуляція, успадкування, поліморфізм), більш загальні принципи (простота – KISS, можливість розвитку, YAGNI, відсутність дублювання – DRY, абстракція, розподіл функціональності, повторне використання, можливість підтримки), більш конкретні об’єктно-орієнтовані принципи (SOLID, coupling/cohesion, Law of Demeter, Principle of Least Astonishment). Якщо в коді використано якісь патерни – варто це також описати. Аналогічно попередньому пункту, можна використати автоматизовані інструменти для отримання певних метрик коду – проте необхідно пояснити, що саме означають отримані інструментом дані.

Мета цього завдання – покращити структуру коду, зробити його більш гнучким та розширюваним. Як частину перетворень, можна реалізувати нову функціональність чи виправити недоліки в попередній – але це має бути додатково до покращень object-oriented design, а не замість нього.

У виконаному завданні має бути чітко зрозуміло, яким був попередній код (до змін/рефакторінгу), та які зміни було зроблено. Наприклад, можна викласти існуючий код в репозиторій першим комітом, помітити його тегом, і потім робити зміни.

Моделі можуть складатись з текстових описів та/або UML діаграм. Також має бути зрозуміло, де модель попереднього коду, а де запропоновані зміни, описані на рівні моделі. Не обов’язково будувати модель/діаграми двічі (для попереднього та зміненого коду) – краще описати попередній код, і потім лише ті зміни, які пропонується зробити. Особливо це стосується тих моделей чи діаграм, де відносно невелика частка змін (тому на оновленій діаграмі важко зрозуміти, що саме змінилося). З іншого боку, якщо буде зрозуміліше зробити дві різні моделі чи набори діаграм – так теж можна. Варто виділити зміни в моделях, використовуючи засоби UML (notes, stereotypes, різні способи групування такі як packages, swimlanes) та/або засоби обраного редактора діаграм (кольори, шрифти тощо).

Необхідно реалізувати юніт-тести для перевірки реалізованої функціональності, алгоритмів, логіки роботи програми тощо. Юніт-тести мають більш-менш покривати реалізовану функціональність/логіку. Робити тести для інтерфейсу користувача, збереження даних в БД і т.д. не обов’язково (хоча можна, для отримання ***додаткових балів***)

В цьому семестрі обов’язковим є створення документації коду виконаних завдань (наприклад, з використанням Doxygen). Бажано викласти згенеровану документацію в репозиторій та налаштувати GitHub Pages ( github.io ) для можливості перегляду документації без клонування репозиторію. Можна також налаштувати генерацію документації на сервері (наприклад, з використанням GitHub Actions) – за правильну реалізацію можна отримати ***додаткові бали***. Можна додавати посилання між згенерованою документацією та створеними моделями/описами/діаграмами UML.

Для отримання максимальної кількості балів, варто в зміненій версії програми реалізувати розбиття на окремі компоненти. Наприклад, винести частину коду в бібліотеки і підключити їх до основної програми. Або розбити на клієнтську та серверну частину із взаємодією по мережі.