**Звіт. Лабораторна робота №1**

**Refactoring (попередній)**

1. Застосування принципу YANGI. Спочатку задумувалося, що зростання номерів в індексах сетів та папках повинно відбуватися повільно й щоб досягти цього було реалізовано Garbage Collector. Це був список, що запам’ятовував видалені номери та підставляв їх, коли створювався новий індекс (папка). Але такий підхід себе не виправдав: дуже швидко зросла кількість помилок під час збереження на зчитування інформації. Наприклад, якщо видалити та створити новий сет, то туди автоматично могла завантажитися інформація його попередника. Замість того, щоб виправляти всі помилки, я вирішив спростити підхід – і це дозволило програмі продовжити стабільно працювати (+ зрозуміліше + ефективніше)
2. Застосування поліморфізму (+template method). Його застосування також стосується абстрактного класу ActivePageController. Спочатку він мав лише метод onClose() для закриття сцени за допомогою Workspace Controller. Але один цей метод не рятував від повторень коду. В деякий момент стало зрозуміло, що вихід із сцени для багатьох сцен має схожі та відмінні властивості. Наприклад, усі вони зберігають інформацію користувача, але деяким для виходу необхідне підтвердження «Ви точно хочете вийти?». Тому для операції виходу було реалізовано методи onCloseRequest(), onClose(), saveUserData(), де останній є темплейтним (спільним), а інші можуть бути перевизначені залежно від потреб класу.
3. Також були моменти незначних покращень коду, зокрема дублікат методу saveUserData(), який був у класах-наслідниках ActivePaneController було перенесено у батьківський клас.
4. «Чистка» коду. Наприклад, деякі поля в класі SetOpenController було винесено в методи, оскільки вони не мали постійного використання. Це суперечить «чистоті» класів.
5. Поділ відповідальності: для невеликих задач, зокрема роботи з датами / завантаження іконки програми – було створено окремі невеликі класи, що дозволяє уникнути дублікатів коду та покращити читабельність.

**Пропозиції Refactoring**

1. Дотримання принципів SOLID:

***«S»:***

***Проблема:*** Клас UserLearnStyle має забагато відповідальності, зокремо контроль над опціями користувача, збереження інформації про індекси та папки, а також регулювання серії користувача.

***Рішення:***

Розділити клас на менші класи з однією відповідальністю.

***Проблема:*** У Workspace Controller є методи як loadPage(), loadAttachedToIndex(), loadAttachedToFolder(). Відповідно, якщо з’явиться якийсь новий тип сторінок, то доведеться створювати новий метод, що суперечить Open / Closed.

***Проблема:*** SetIndex та Folder у WorkspaceController в методах WorkspaceController є взаємодією між класами високого та низького рівня

***Рішення:***  Заміна на інтерфейс, спільний для SetIndex та Folder / відмова від взаємодії цих класів.

1. ***Проблема:*** TermList має наступника – StudyTermList, який призначений для вивчення сету у режимі «Письмо». Але разом з списком термінів цьому класу й передаються методи calculateMasteredCount(), insert(), refresh(), які не мають практичного застосування, адже основна задача класу – виводити терміни у звичайному чи перемішаному порядку, а не вносити зміни в них.

***Рішення:*** Розбити клас.

1. Пов’язати AdditionalWindows успадкуванням від спільного абстрактного класу.
2. StudyTermList успадковується від TermList, тоді як ці класи мають зовсім різні застосування (й можуть працювати по-різному). Пропонується використати успадкування від спільного абстрактного класу.
3. Поділ на Client Server – поки не реалізовано.

**Реалізація запропонованих змін**

Реалізація змін, пов’язаних з відкриттям різних сторінок та додатковими вікнами не викликала труднощів. Основною перевагою було те, що проект попередньо вдало був поділений на компоненти з дотриманням принципів об’єктно орієнтованого програмування.

Наведені зміни сприяли кращому дотриманню принципу інкапусляції та SOLID, оскільки більшість класів не поглиблюються в реалізацію суміжних класів. Зменшилася залежність коду: класи вищого рівня менше залежать від класів нижчого рівня.

Окрім того, оскільки в клас Workspace Controller передаються уже проініціалізовані контролери сцен, відбувається застосування досить розповсюдженого в Java принципу Dependency Injection.

Було прийнято рішення розділити TermList на звичайний TermList та StudyTermList, які успадковуються від AbstractTermList. Відповідно TermList має методи, потрібні для зберігання інформації, а StudyTermList – методи, необхідні в режимі вивчення термінів.

Реалізація додаткових вікон, окрім того, значно скоротила розмір коду (та покращила його читабельність). Замість трьох незалежних конструкторів додаткових вікон тепер існує один спільний. На мою думку, це чудова демонстрація застосування принципу DRY.

Генерацію було згенеровано за допомогою Java Doc та вбудованих інструментів в IDE. Генерація відбулася успішно, жодних труднощів не виникало. Для збереження документації було створено директорію docs. Документацію було викладено на [GitHub pages](https://mousecreator.github.io/Labs2course-UML1a/docs/com.example.lab23a/module-summary.html).

Було реалізовано мікробенчмаркінг для перевірки ефективності завантаження / вивантаження інформації в середньому випадку. Обидва процеси відбуваються приблизно за 100 мікросекунд.

Розбиття класу UserLearnStyle на три незалежні класи UserSavedInfo, UserStreak та UserOptions була нескладною і, на мою думку, доцільною. Зокрема, StudyTermList тепер не має отримувати як параметр весь style з непотрібною додатковою інформацією, а може отримати тільки UserOptions, а звідти значення shuffleIsOn (цю зміну також було реалізовано). Помилкою було те, що я намагався спростити запис у файл (справді, один клас легко передати як параметр у FileBuilder), але не врахував, що це додаткова функція і насправді основним є правильно реалізувати логіку програми з дотриманням відповідних принципів. Можливість запису через FileBuilder теж була адаптована.

**Аналіз результатів**

Змін у застосуванні програми майже не виявилося. Виняток: тепер усі додаткові вікна пріоритет за основне вікно, оскільки вони мають на меті отримання якоїсь інформації від користувача, щоб продовжити основний процес: експорт файлу, відкриття папки, перехід до наступного етапу вивчення термінів. За необхідності, це легко змінюється у методі load відповідного Additional Window. Зміна є запланованою.

Натомість код став значно гнучкішим, менш залежним та відкритим для розширення. З ним стало легше працювати та легше читати.

Як висновок можна відзначити, що рефакторинг є важливою і невід’ємною частиною написання програм. До того ж, що більш вчасно він зроблений, то меншою є ймовірність ускладнень чи виникнення помилок при додаванні нових функцій чи покращені коду.

Як було продемонстровано вище, код значно поліпшився у розумінні дотримання принципів ООП, а також принципів SOLID, KISS, DRY. У майбутньому до даного застосунку можна застосувати і патерни проектування.