Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 4

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до паттернів проектування»

Виконала:
студентка групи ІА-33
Самойленко Анастасія

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

Київ 2025

Тема: Вступ до паттернів проектування
Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.
Тема роботи:
27. Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA)
Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

Вступ

У сучасному світі розроблення програмного забезпечення потребує використання перевірених підходів, які забезпечують гнучкість, масштабованість та легкість у супроводі програмних систем. Одним із ключових інструментів, що дозволяють досягти цих цілей, є патерни (шаблони) проектування. Вони описують типові способи вирішення поширених задач, які виникають під час створення програмних продуктів, і допомагають уникнути дублювання коду, покращити його структуру та підвищити рівень повторного використання компонентів.

Метою даної лабораторної роботи є вивчення структури та принципів реалізації шаблонів “Singleton”, “Iterator”, “Proxy”, “State”, “Strategy” і застосування їх у розробці програмної системи. Зокрема, під час виконання роботи реалізовується програмний продукт «Особиста бухгалтерія», який є наочним засобом для ведення особистих фінансів — обліку витрат, прибутків, періодичних платежів, аналізу фінансової статистики, а також керування банківськими рахунками, вкладами та кредитами.

Теоретичні відомості

Шаблон (патерн) проєктування — це формалізований опис типового рішення, яке багаторазово зустрічається під час розроблення інформаційних систем. Він містить узагальнений спосіб вирішення певної задачі проєктування, рекомендації щодо його застосування та має загальновживану назву.

Головна мета використання шаблонів — повторне застосування вже знайдених ефективних рішень у нових ситуаціях. Важливою умовою є коректне моделювання предметної області, що дозволяє правильно визначити задачу та обрати відповідний патерн.

Застосування шаблонів проєктування забезпечує низку переваг:

- створення структурованої та логічної моделі системи;
- підвищення наочності та спрощення вивчення архітектури;
- глибше опрацювання структури програмного продукту;
- підвищення стійкості системи до змін вимог;
- спрощення подальшого доопрацювання та розширення;
- полегшення інтеграції систем і взаєморозуміння між розробниками завдяки спільному “словнику проєктування”.

Шаблон «Singleton»

Патерн Singleton забезпечує існування лише одного екземпляра певного класу та надає глобальну точку доступу до нього. Це означає, що створення нового об’єкта цього класу можливе лише один раз, а всі інші частини програми використовують вже наявний екземпляр.

Проблема:

Використання шаблону «Одинак» виправдане у випадках, коли:

- має існувати не більше N фізичних об’єктів певного типу;
- необхідно контролювати всі операції, що проходять через певний клас (наприклад, робота з єдиним файлом налаштувань або керування сеансом зв’язку).

Прикладом може бути файл конфігурації програми, який є унікальним і не потребує створення декількох об’єктів для читання чи запису даних. Також шаблон зручно застосовувати для керування єдиним каналом зв’язку, де потрібно гарантувати, що одночасно надсилається лише один запит.

Переваги:

- Гарантує наявність єдиного екземпляра класу.
- Забезпечує глобальну точку доступу до цього екземпляра.

Недоліки:

- Порушує принцип єдиної відповідальності, оскільки клас одночасно відповідає і за створення об’єкта, і за його логіку.
- Може маскувати поганий дизайн та ускладнювати супровід програми.
- Є анти-патерном у сучасному проєктуванні, оскільки фактично виступає аналогом глобальної змінної з власним станом, що ускладнює тестування та відстеження змін.

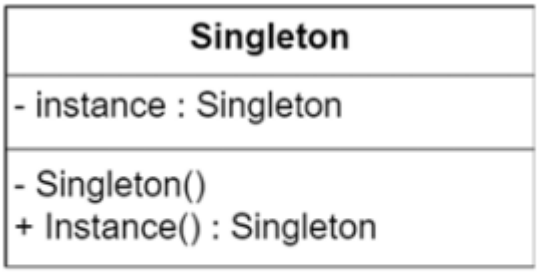


Рисунок 1.1. Структура патерну Одинак

Шаблон «Iterator»

Патерн Iterator забезпечує послідовний доступ до елементів колекції (агрегату) без розкриття її внутрішньої структури. Завдяки цьому шаблону відбувається розподіл обов’язків: колекція відповідає за зберігання даних, а ітератор — за їх послідовне проходження.

Ітератор дозволяє реалізувати різні способи обходу колекції — у прямому чи зворотному порядку, по парних або непарних індексах тощо — незалежно від способу представлення даних.

Проблема:

Колекції можуть мати різні структури — від простих списків до складних дерев чи графів. Проте користувач повинен мати єдиний інтерфейс доступу до елементів будь-якої колекції.

Якщо додавати алгоритми обходу безпосередньо в код колекції, це ускладнює її реалізацію і порушує принцип єдиної відповідальності, оскільки колекція має лише зберігати дані, а не визначати способи їх обходу.

Рішення:

Шаблон «Ітератор» пропонує винести логіку обходу колекції в окремий клас — ітератор.

Об’єкт-ітератор відстежує поточну позицію, стан проходження та кількість елементів, що залишилося обійти.

Кілька ітераторів можуть одночасно працювати з однією колекцією, не заважаючи один одному.

При необхідності додати новий спосіб обходу достатньо створити новий клас ітератора, не змінюючи код самої колекції.

Типова структура ітератора:

- First() – встановлює покажчик на перший елемент колекції;
- Next() – переходить до наступного елемента;
- IsDone – булеве поле, що сигналізує про завершення обходу;
- CurrentItem – повертає поточний елемент колекції.

Переваги:

- Уніфікує процес доступу до елементів різних типів колекцій.
- Дозволяє легко додавати нові способи обходу даних.
- Спрощує класи колекцій, розвантажуючи їх від логіки перебору.

Недоліки:

- Використання ітератора може бути надмірним, якщо достатньо звичайного циклу.

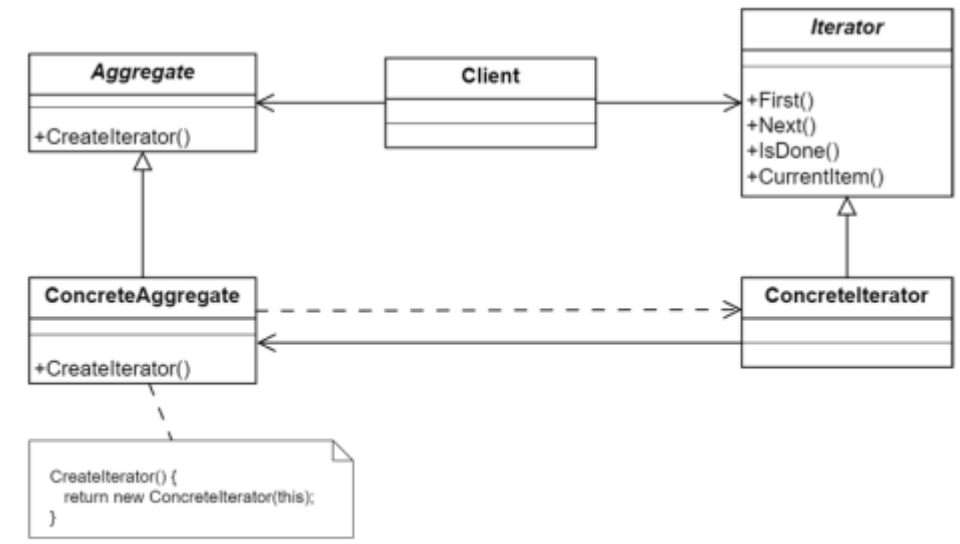


Рисунок 1.2. - Структура патерну Ітератор

Шаблон «Proxy»

Патерн Proxy передбачає створення об’єкта-замісника (заглушки), який контролює доступ до іншого, реального об’єкта. Такий підхід дозволяє додавати додаткову логіку — наприклад, кешування, контроль доступу, відкладене створення об’єкта або оптимізацію запитів — без зміни коду клієнта, який працює

із системою.

Іншими словами, проксі «перехоплює» звернення до об’єкта й може змінювати або відкладати виконання запитів.

Проблема:

Уявімо систему, яка працює з зовнішнім сервісом — наприклад, DocuSign для електронного підписання документів.

При зростанні кількості користувачів збільшилася кількість запитів до сервісу, що спричинило високі витрати. Аналіз показав, що немає потреби надсилати документи на підписання миттєво — достатньо робити це, наприклад, раз на годину. Так само можна періодично перевіряти статус підписання.

Рішення:

Використовується шаблон «Proxy» для створення проміжного класу, наприклад DocSignManagerProxy, який реалізує той самий інтерфейс IDocSignManager, що й оригінальний клас DocSignManager.

Проксі накопичує запити клієнтів у черзі, відправляє їх до сервісу пакетно (раз на годину), а відповіді зберігає в базі даних. Клієнтський код при цьому не змінюється — він продовжує працювати з інтерфейсом IDocSignManager, не знаючи, що звертається до проксі, а не до справжнього менеджера.

У результаті система зменшує кількість запитів до зовнішнього сервісу, економить ресурси і забезпечує стабільнішу роботу.

Переваги:

- Можливість додати проміжний рівень логіки без змін клієнтського коду.
- Забезпечує контроль доступу та керування життєвим циклом об’єкта.
- Дає змогу оптимізувати роботу з дорогими або віддаленими ресурсами (наприклад, через кешування або групування запитів).

Недоліки:

- Може знизити швидкість роботи через додаткові проміжні операції.
- Існує ризик некоректної або застарілої відповіді при роботі через замісник.

Таким чином, шаблон Proxy є ефективним інструментом для оптимізації взаємодії між клієнтом і реальним об’єктом, дозволяючи додавати нові можливості без втручання в основну архітектуру системи.

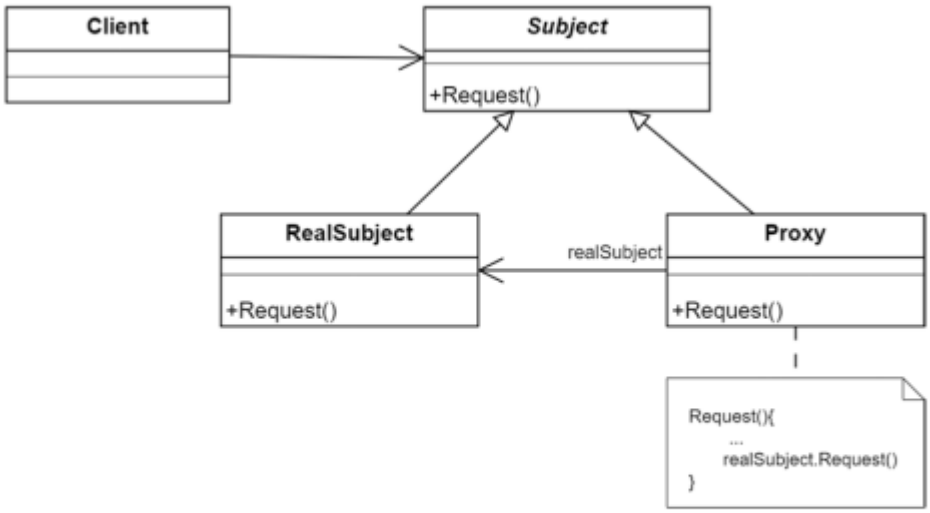


Рисунок 1.3. - Структура патерну Proxy

Шаблон «State»

Шаблон «State» (Стан) — це поведінковий шаблон проєктування, який дозволяє об’єкту змінювати свою поведінку залежно від внутрішнього стану. Зовні здається, що змінюється сам клас об’єкта є. Використовується для того, щоб уникнути великої кількості умовних операторів, що перевіряють стан об’єкта, та замінити їх динамічною зміною об’єкта стану.

Ідея реалізації:

- Поведінку, що залежить від стану, виділяють у загальний інтерфейс (State).
- Кожен конкретний стан реалізується у вигляді окремого класу (ConcreteStateA, ConcreteStateB), що імплементує цей інтерфейс.
- Клас, який має стан (Context), містить посилання на поточний об’єкт стану і делегує йому виконання дій.
- Зміна стану здійснюється шляхом заміни об’єкта стану у контексті, що змінює поведінку всієї системи без зміни її структури.

Приклад застосування:

У системі з модулем Listener, який обробляє запити клієнтів, можна виділити три стани:

- InitializingState — сервер запускається, запити ігноруються;
- OpenState — сервер працює у звичайному режимі, приймає та обробляє запити;

- ClosingState — сервер завершує роботу, нові запити не приймає, але може відправляти відповіді.

Кожен стан реалізує інтерфейс IListenerState, а Listener делегує виклики поточному стану.

Переваги:

- Код для кожного стану ізольований у власному класі.
- Система стає гнучкішою, оскільки можна легко додавати нові стани.
- Код контексту спрощується, адже логіка станів винесена в окремі класи.
- Можливе повторне використання станів у різних контекстах.

Недоліки:

– Ускладнюється структура програми через більшу кількість класів і необхідність керування переходами між станами.

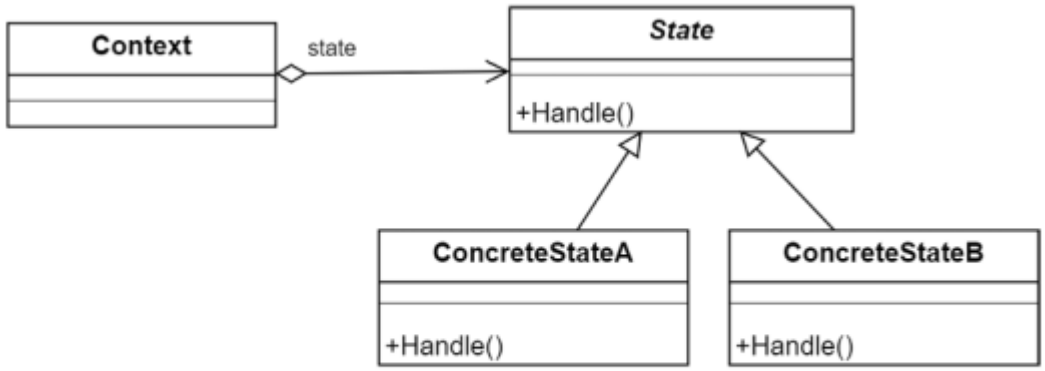


Рисунок 4.4. Структура патерну Стан

Шаблон «Strategy»

Шаблон «Strategy» (Стратегія) — це поведінковий шаблон проєктування, який дає змогу визначати родину алгоритмів, інкапсулювати кожен із них у власному класі та робити їх взаємозамінними. Це дозволяє змінювати спосіб виконання певної дії під час роботи програми без зміни її структури.

Патерн використовується для того, щоб об’єкт міг змінювати свою поведінку, вибираючи одну з кількох можливих стратегій (алгоритмів), які досягають однакової мети різними способами. Наприклад, різні способи сортування або обробки даних.

Ідея реалізації:

- Створюється загальний інтерфейс Strategy, який визначає метод виконання алгоритму.
- Для кожної конкретної поведінки реалізується окремий клас (ConcreteStrategyA, ConcreteStrategyB), що реалізує цей інтерфейс.
- Клас Context зберігає посилання на об’єкт стратегії й делегує йому виконання алгоритму.
- Заміна стратегії виконується простою підстановкою іншого об’єкта в полі Context.strategy.

Приклад з життя:

Якщо потрібно дістатися на роботу, можна обрати різні способи — автомобіль, метро або пішки. Це і є різні стратегії досягнення однієї мети. Вибір конкретної стратегії залежить від ситуації (пробки, несправність транспорту, близькість до місця роботи).

Переваги:

- Можна змінювати алгоритми під час виконання програми.
- Алгоритми ізольовані від основного коду, що підвищує зрозумілість і підтримуваність.
- Зменшується кількість умовних конструкцій (if, switch) у контексті.
- Стратегії можна повторно використовувати з різними контекстами.

Недоліки:

– Збільшується кількість класів у системі.

– Якщо алгоритмів небагато, застосування шаблону може ускладнити код без суттєвої користі.

– Клієнтський код повинен враховувати відмінності між стратегіями під час вибору алгоритму.

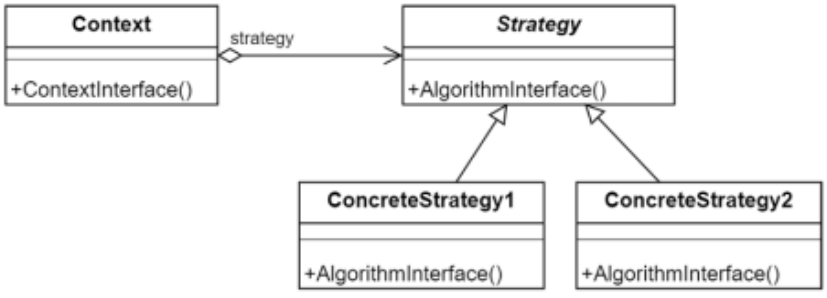


Рисунок 1.5. - Структура патерну Стратегія.

Хід роботи

1) Діаграма класів частини функціоналу робочої програми. (рис. 1)

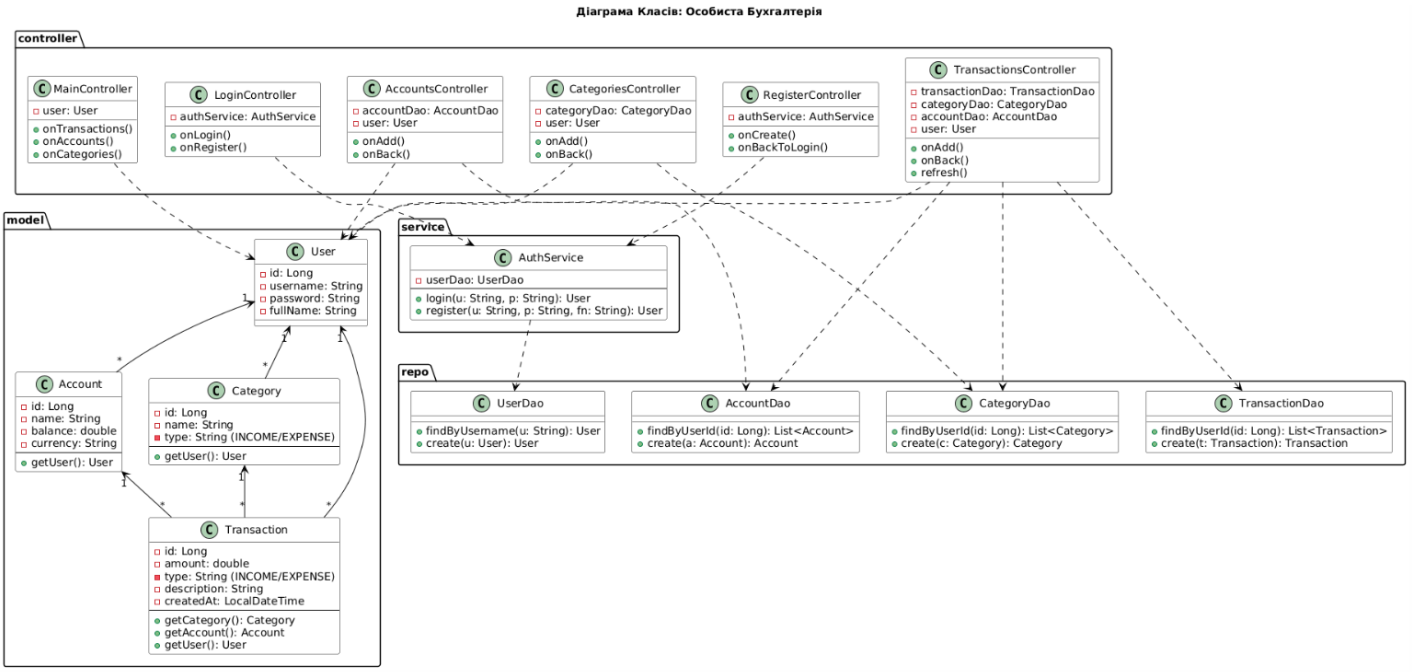


Рисунок 1 – Діаграма класів

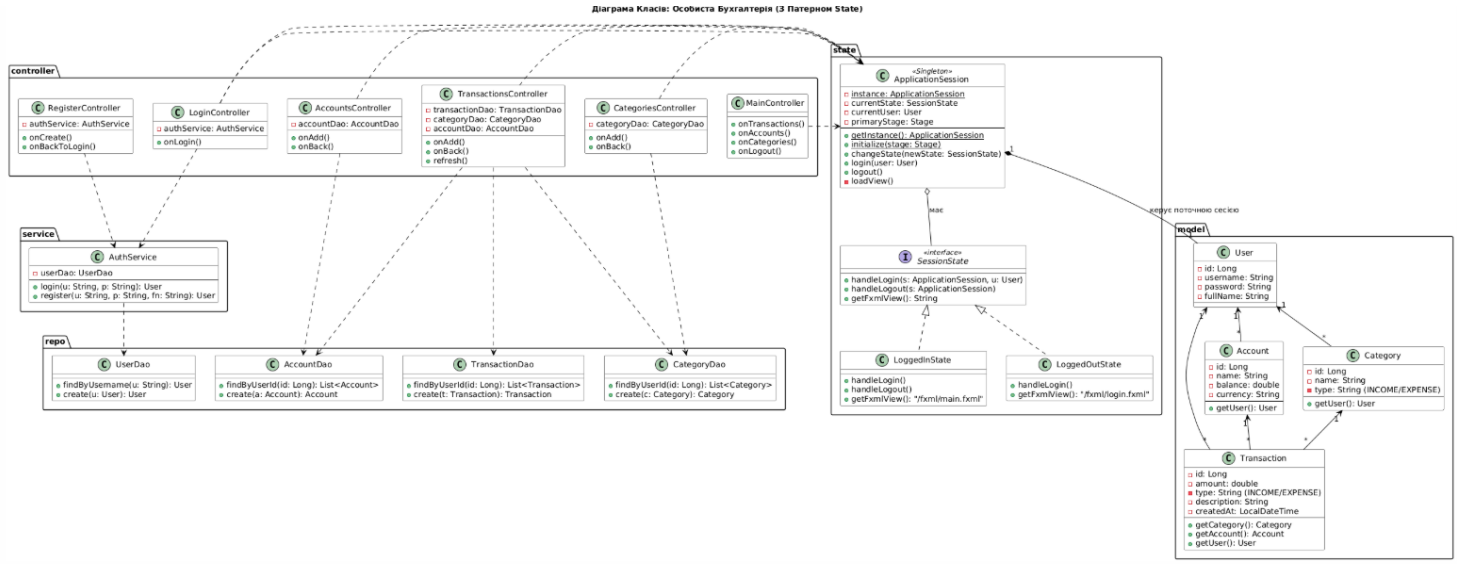


Рисунок 2 – Діаграма класів з патерном State

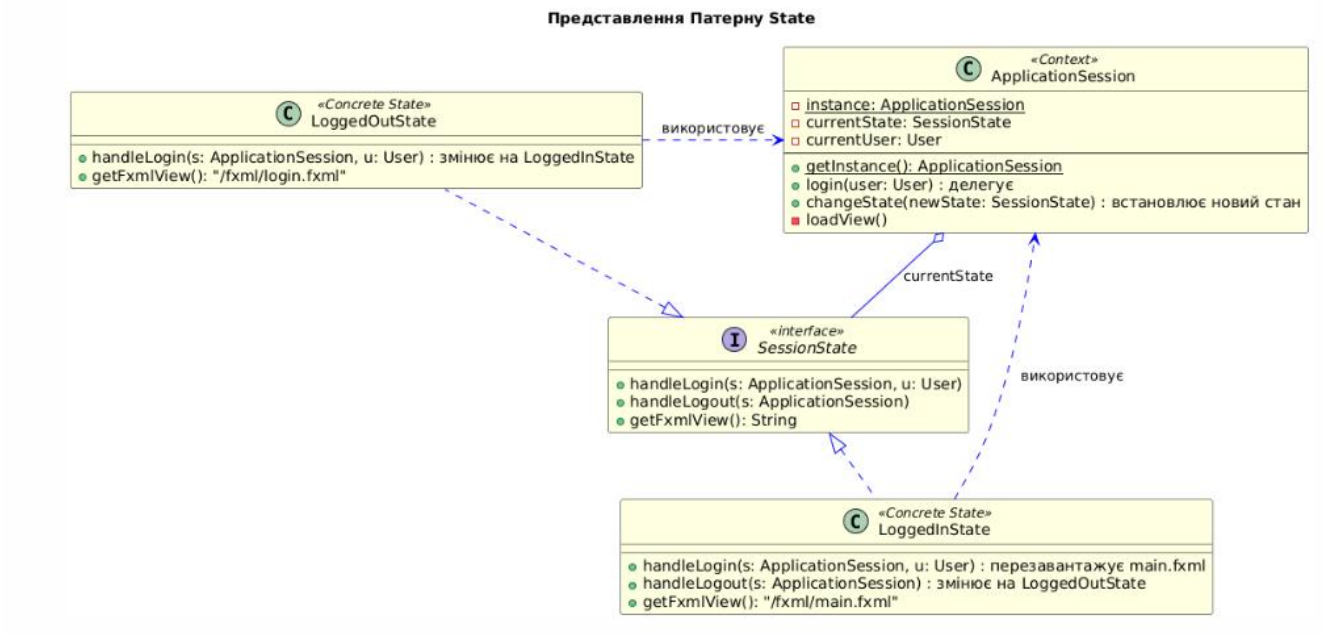


Рисунок 3 – Представлення самого шаблону на діаграмі

2) Реалізація Шаблону «State» для Управління Сесією

Шаблон State (Стан) був використаний для управління життєвим циклом користувацької сесії (вхід/вихід) та динамічного визначення того, який екран (FXML View) має бути відображений у головному вікні (Stage) програми. Це забезпечує чисту архітектуру, де поведінка системи змінюється залежно від її внутрішнього стану.

Шаблон реалізовано через три основні елементи:

1) Центральний Контекст (ApplicationSession.java)

Клас ApplicationSession виконує роль Контексту в шаблоні State. Це також Singleton, що гарантує єдиний централізований механізм управління станом у всій програмі.

- Призначення: Зберігати поточний об'єкт стану (currentState) та об'єкт поточного користувача (currentUser).
- Механізм делегування: Замість самостійної реалізації логіки, ApplicationSession делегує всі запити на зміну стану (наприклад, виклик login(User user)) поточному об'єкту стану.
- Функція навігації: Містить метод changeState(), який не лише замінює currentState на новий об'єкт, але й викликає приватний метод loadView(). Це гарантує, що при кожній зміні стану автоматично завантажується відповідний FXML-інтерфейс, шлях до якого надає новий об'єкт стану.

2) Стан «Вийшов із системи» (LoggedOutState.java)

Клас LoggedOutState є Конкретним Станом, який описує поведінку системи, коли користувач не аутентифікований.

- FMXL View: Метод getFxmlView() цього стану завжди повертає шлях до екрана входу/реєстрації (/fxml/login.fxml).
- Обробка подій:
    - handleLogout(): Нічого не робить, оскільки неможливо вийти зі стану, у якому ви вже перебуваєте.
    - handleLogin(): Містить ключову логіку: якщо користувач успішно пройшов аутентифікацію, цей метод встановлює currentUser в сесії та ініціює перехід до стану LoggedInState через виклик session.changeState(new LoggedInState()).

3) Стан «Увійшов» (LoggedInState.java)

Клас LoggedInState є Конкретним Станом, який описує поведінку системи, коли користувач успішно аутентифікований.

- FMXL View: Метод getFxmlView() цього стану завжди повертає шлях до головного меню програми (/fxml/main.fxml).
- Обробка подій:
    - handleLogout(): Обробляє вихід із системи. Він очищає currentUser та ініціює перехід до LoggedOutState.
    - handleLogin(): Цей метод використовується для реалізації логіки кнопки "Назад" (onBack()) з внутрішніх екранів (транзакцій, рахунків). Замість реального повторного входу, він викликає session.changeState(this). Це змушує ApplicationSession перезавантажити FXML для поточного стану, що ефективно повертає користувача на головний екран (main.fxml) без зміни стану сесії.

Завдяки застосуванню шаблону State, контролери в системі перестають виконувати складну логіку, пов’язану з обробкою станів користувача або системи. Раніше контролер сам би перевіряв, в якому стані зараз знаходиться об’єкт (наприклад, користувач може бути незареєстрований, авторизований або заблокований) і на основі цих перевірок виконував різні дії. Це ускладнювало код, зменшувало його читабельність та збільшувало ймовірність помилок при зміні логіки.

Після впровадження State:

- Контролер більше не приймає рішення про те, що робити в кожному конкретному стані.
- Контролер делегує дію контексту, наприклад:

ApplicationSession.getInstance().login(user);

- Контекст сам визначає, яку поведінку виконати, виходячи з поточного стану (Initializing, LoggedIn, LoggedOut, Blocked тощо).
- Кожен стан реалізує свою власну поведінку через інтерфейс State, тому додавання нового стану не потребує зміни контролера.

Переваги такого підходу:

- Поділ відповідальності: контролер відповідає лише за делегування дій, а логіка станів винесена в окремі класи.
- Вища зв’язність: логіка, що залежить від стану, зосереджена у відповідному класі стану.
- Структурована архітектура: код більш організований, зрозумілий і легко підтримується.
- Масштабованість: легко додавати нові стани та змінювати логіку без порушення існуючого коду.

Відповіді на контрольні запитання:

1. Що таке шаблон проєктування?

Патерн проєктування — це перевірене часом архітектурне рішення для типових проблем програмного забезпечення. Він описує структуру класів, об’єктів і зв’язки між ними, не прив’язуючись до конкретної мови програмування. Шаблони дозволяють стандартизувати підхід до проєктування та забезпечують повторне використання рішення.

2. Навіщо використовувати шаблони проєктування?

Використання патернів дозволяє:

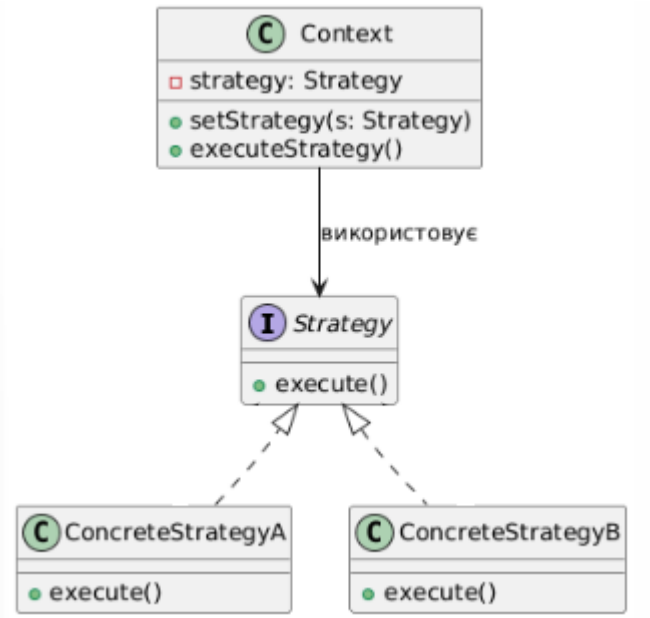
- підвищити гнучкість і масштабованість системи;
- спростити супровід і розвиток коду;
- зменшити кількість дублювання та помилок;
- стандартизувати рішення, щоб архітектура була зрозумілою для всіх розробників.

3. Призначення шаблону «Стратегія»

Шаблон «Strategy» дозволяє змінювати алгоритм поведінки об’єкта динамічно, не змінюючи клас, який його використовує.

Приклад: алгоритми сортування (QuickSort, MergeSort, BubbleSort) реалізовані у окремих класах і взаємозамінні. Контекст просто викликає метод execute() конкретної стратегії, не знаючи деталей її реалізації.

4. Структура шаблону «Стратегія»



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

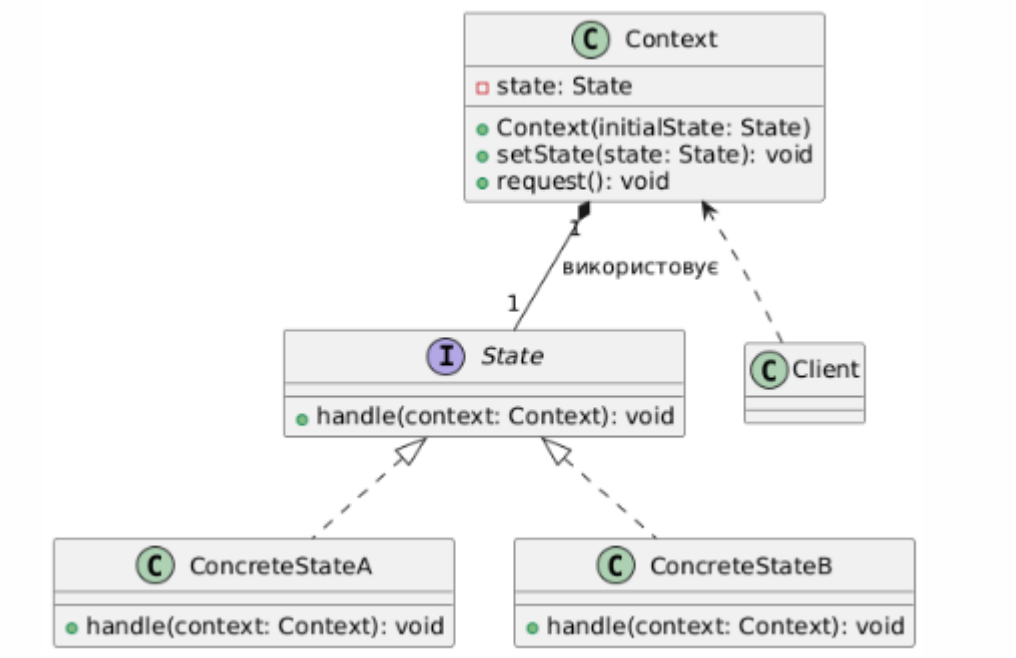
- Context — містить посилання на Strategy і викликає метод execute().
- Strategy — інтерфейс для всіх стратегій.
- ConcreteStrategyA / ConcreteStrategyB — конкретні реалізації алгоритмів.

Context отримує конкретну стратегію через setStrategy() і викликає execute(), завдяки чому можна змінювати поведінку «на льоту» без змін коду Context.

6. Яке призначення шаблону «Стан»?

Шаблон «State» дозволяє об’єкту змінювати свою поведінку залежно від внутрішнього стану. Зовні виглядає так, ніби об’єкт змінює свій клас. Це дозволяє уникнути великих блоків if або switch, що перевіряють стан.

7. Нарисуйте структуру шаблону «Стан».



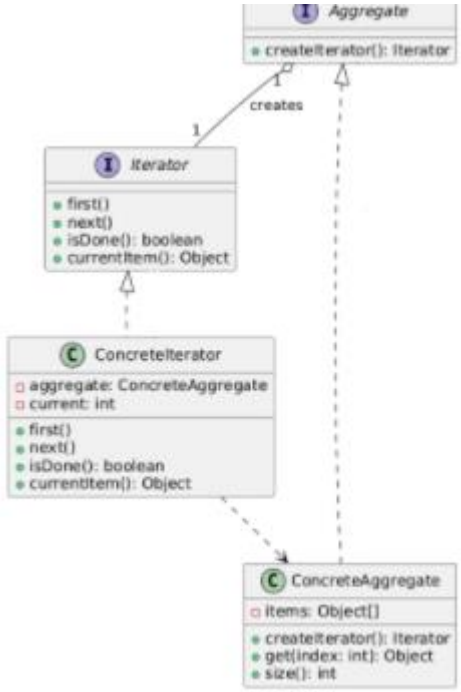
8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

У шаблоні «Стан» пов'язані зі станом поля, властивості та методи виділяються в окремий загальний інтерфейс State. Кожен конкретний стан реалізується як окремий клас (ConcreteStateA, ConcreteStateB), що відповідає цьому інтерфейсу. Об'єкти, які мають стан (Context), при його зміні просто оновлюють своє поле state новим об'єктом стану, що призводить до повної зміни поведінки об'єкта.

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» надає спосіб послідовного доступу до елементів колекції (або агрегату) без розкриття її внутрішньої структури. Він виносить логіку перебору елементів з самої колекції, розділяючи відповідальність: колекція відповідає за зберігання даних, а ітератор – за навігацію по них.

10. Намалюйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

 Шаблон «Ітератор» складається з таких класів:

- Iterator (Ітератор): Інтерфейс, який визначає методи для навігації по колекції (наприклад, next() для переходу до наступного елемента, isDone() для перевірки кінця колекції, first() для початку, currentItem() для отримання поточного елемента).
- ConcreteIterator (Конкретний Ітератор): Реалізує інтерфейс Iterator і відстежує поточну позицію в конкретній колекції.
- Aggregate (Агрегат): Інтерфейс, який визначає метод для створення ітератора.
- ConcreteAggregate (Конкретний Агрегат): Конкретна реалізація колекції, яка повертає об'єкт ConcreteIterator.

Клієнт отримує об'єкт Iterator від Aggregate і використовує його методи для послідовного доступу до елементів колекції, не знаючи про її внутрішню реалізацію.

12. В чому полягає ідея шаблона «Одинак»?

Ідея шаблону «Одинак» (Singleton) полягає в тому, щоб гарантувати, що певний клас матиме лише один екземпляр (об'єкт) у всій системі. Цей єдиний об'єкт зазвичай зберігається як статичне поле в самому класі і надається через статичний метод доступу.

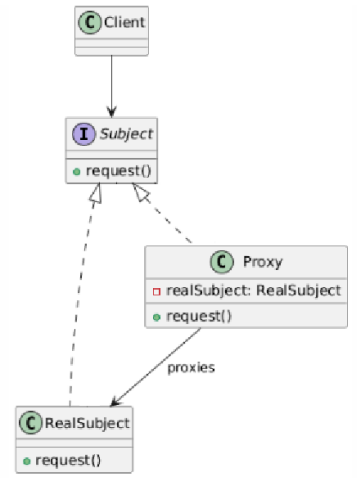
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Шаблон «Одинак» часто вважають "анти-шаблоном", оскільки він створює глобальний стан у програмі. Це може ускладнити тестування (оскільки об'єкти стають сильно пов'язаними), знизити гнучкість системи та призвести до прихованих залежностей між класами. Зміна єдиного екземпляра може непередбачувано вплинути на інші частини програми, що ускладнює підтримку та розробку.

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» (Proxy) створює об'єкти-заглушки або двійники, які діють як замінники для реальних об'єктів. Проксі-об'єкти зазвичай додають додатковий функціонал або спрощують взаємодію з базовим (реальним) об'єктом, не змінюючи його логіки.

15. Намалюйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

Шаблон «Проксі» включає такі класи:

- Subject (Суб'єкт): Спільний інтерфейс, який реалізують як реальний об'єкт, так і його проксі. Це гарантує, що проксі може замінити реальний об'єкт.
- RealSubject (Реальний Суб'єкт): Власне об'єкт, до якого здійснюється доступ. Він містить основну бізнес-логіку.
- Proxy (Проксі): Обгортка, яка контролює доступ до RealSubject. Вона може виконувати додаткові дії перед або після виклику методів реального об'єкта.

Клієнт взаємодіє з Proxy так, ніби це RealSubject. Proxy вирішує, чи виконати дію самостійно, чи делегувати її RealSubject. Це дозволяє додавати такі функції, як

перевірка доступу, кешування, логування або ліниве завантаження, без зміни логіки RealSubject.

Висновки

У ході виконання лабораторної роботи ми ознайомилися з основними патернами проектування, такими як Singleton, Iterator, Proxy, State та Strategy, та вивчили їх структуру і призначення. У практичній частині на прикладі програми «Особиста бухгалтерія» ми реалізували патерн State, що дозволив об’єкту змінювати поведінку залежно від стану, спростивши логіку програми та підвищивши її гнучкість. Робота сприяла набуттю навичок застосування патернів для побудови масштабованих і зрозумілих програмних систем.