МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

КУРСОВА РОБОТА

з дисципліни «Об’єктно-орієнтоване програмування»

на тему

Шаблони проектування в ООП. Система керування станціями метрополітену

Виконав студент
2 курсу групи КП-12
Мальцев Микита Ігорович

Керівник роботи
доцент, к.т.н. Заболотня Т.М.

Оцінка

_____
(дата, підпис)

КИЇВ 2023

ЗМІСТ

ВСТУП

Дана курсова робота присвячена розробці програмного забезпечення системи керування станціями метрополітену за допомогою використання шаблонів проектування. І зрозуміло, що сучасна система кервуання станціями має керувати як інструментами забезпечення безпечної подорожі для пасажирів, так і має підтримувати можливість керувати розміщенням рекламних банерів на станціях, розміщення яких стає дедалі більш значною частиною отримання прибутку системами метрополітену. Завдяки тому, що розроблена програма використовує мову програмування Java, він може бути встановлений та може використовуватися на більшості відомих платформ, починаючи із популярних операційних систем, таких як Windows та Linux, до портативних мікрокомпютерів, що підтримують ввід даних із клавіатури та вивід їх на екран. Тому використання моєї програми є обґрунтованим вибором для метрополітену, оскільки це дозволило б використовувати програму як на вже наявному старому обладнанні, так і на новій обчислювальній техніці. Дана тематика обрана для виконання курсової роботи тому, що результати абстрагування об’єктів у цій предметній галузі дозволяють застосувати вивчені принципи та методи об’єктно-орієнтованого програмування для створення програмного забезпечення, зокрема шаблони проектування.

Об’єктом дослідження є система керування станціями метрополітену, яка може функціонувати на основі мого додатку.

Метою роботи є розроблення програмного забезпечення системи керування станціями метрополітену з використанням шаблонів проектування.

Для досягнення визначеної мети необхідно виконати такі *завдання*:

- абстрагувати об’єкти предметної галузі;
- розробити структурну організацію ПЗ за допомогою застосування основних принципів ООП та шаблонів проектування;
- визначити та описати функціональні характеристики програми;
- обґрунтувати вибір шаблонів проектування, використаних для побудови програми;

- розробити інтерфейс роботи програми та його логіку;
- виконати реалізацію програмного забезпечення відповідно до вимог технічного завдання;
- виконати тестування розробленої програми;
- оформити документацію з курсової роботи.

Розроблене ПЗ системи керування станціями метрополітену складається з декількох модулів: контролю версій, який надається для створення снапшотов системи, модуль контролю системами оповіщення, модуль керування безпосередньо станціями, модуль керування рекламними банерами на станціях, модуль керування блоками оповіщення (голосове сповіщення, оповіщення повідомлення на телефон), модулі роботи з квитками.

Реалізовані шаблони проектування: Знімок, Стан, Вільний Будівельник, Спостерігач, Прототип, Композит.

До функціональних можливостей програми належать: створення відображень станцій, налаштування рекламних банерів та засобів оповіщення на станціях, створення та контроль загальних систем оповіщення (систем контролю над окремими сповіщувачами), змінення стану станцій, можливість контролю обслуговуючого персоналу.

Для функціонування розробленої програми необхідно забезпечити наявність на комп’ютері 30 Кб вільного дискового простору.

Розроблене програмне забезпечення може бути використане як і безпосередньо компаніями, що керують метрополітеном, так і учбовими закладами задля моделювання та дослідження процесів метрополітену.

Пояснювальна записка складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел (3-х найменувань, з них 3 – іноземною мовою). Робота містить 21 рисунок. Загальний обсяг роботи – 30 друкованих сторінок, з них 24 сторінок основного тексту та 1 сторінка списку використаних джерел.

1. ОПИС СТРУКТУРНО-ЛОГІЧНОЇ СХЕМИ ПРОГРАМИ

1.1. Модульна організація програми

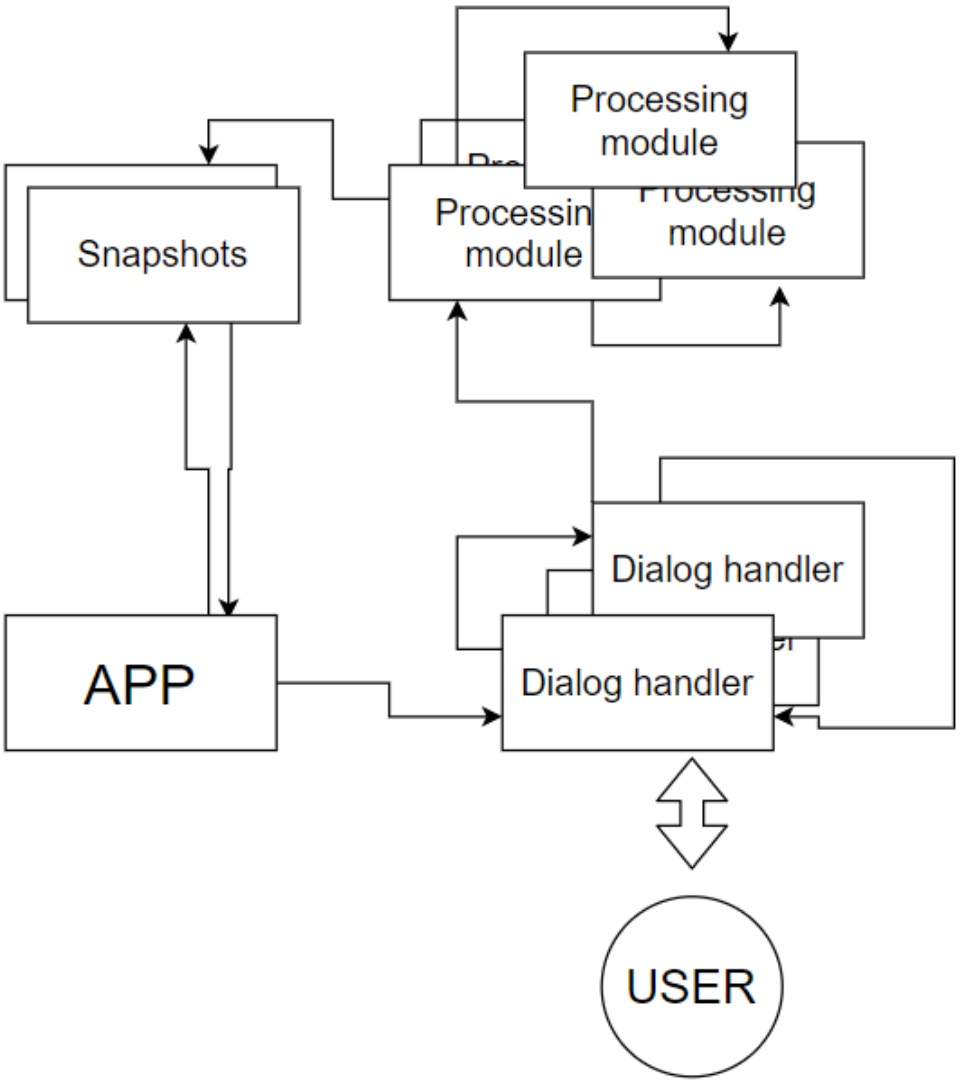

Рис. 1.1.1. Модульна організація програми

Модуль **APP** реалізує головні функції програми. Він зосереджує необхідні дані та посилання для роботи інших модулів.

За взаємодію користувача та іншими модулями програми відповідають модулі типу **Dialog handler**, які надають користувачу інструкції для конкретної взаємодії із програмним застосунком, в тому числі і інструкції по внесенню даних у програмний застосунок. Відповідно цей модуль відповідає і за отримання даних та команд від користувача, і саме він на їх основі буде

задіювати або інші модулі **Dialog handler,** або ж вже на основі отриманих команд та даних звертатиметься до модулів **Processing Module.**

Модулі **Processing module** відповідають за обробку отриманих від інших модулів даних та обробку отриманих команд. За допомогою них також здійснюється збереження стану програми на диск та відновлення стану програми за снапшотами. Окрім цього за допомогою цього модуля здійснюється перевірка відповідності введених даних.

Модуль **Snapshots** відповідає безпосередньо системі послідовного відображення стану програми та її модулів у текстовий формат, та подальше його збереження на диск. Та очевидно, що цей же модуль відповідає і за зчитування раніше збережених даних.

1.2. Функціональні характеристики

Основними функціональними характеристиками мого програмного продукту є: здатність збереження стану програми до файлу та відновлення програми із файлу. Слід зазначити що цей процес відбувається завдяки окремо створений системі перетворення стану програми у текст.

Також програма дозволяє працювати навіть без наперед збереженого файлу стану тобто існують можливості для додавання або створення нових нових станцій метро та нових систем оповіщення. Відповідно після створення нових станцій та систем оповіщення існує достатній набір функцій та можливостей для створення рекламних оголошень, додавання рекламних оголошень та видалення їх видалення. Аналогічно для блоків системи оповіщення теж передбачені можливості створення нових та блоків видалення нових блоків. Також передбачено можливість підписати блок оповіщення до будь-якого до будь-якої кількості систем оповіщення.

Окремо передбачені можливості для створення квитків як одноразових квитків багаторазових передбачена можливість поповнення багаторазових квитків та можливість їх використати.

1.3 Опис реалізованих класів

Клас AdvertHandler

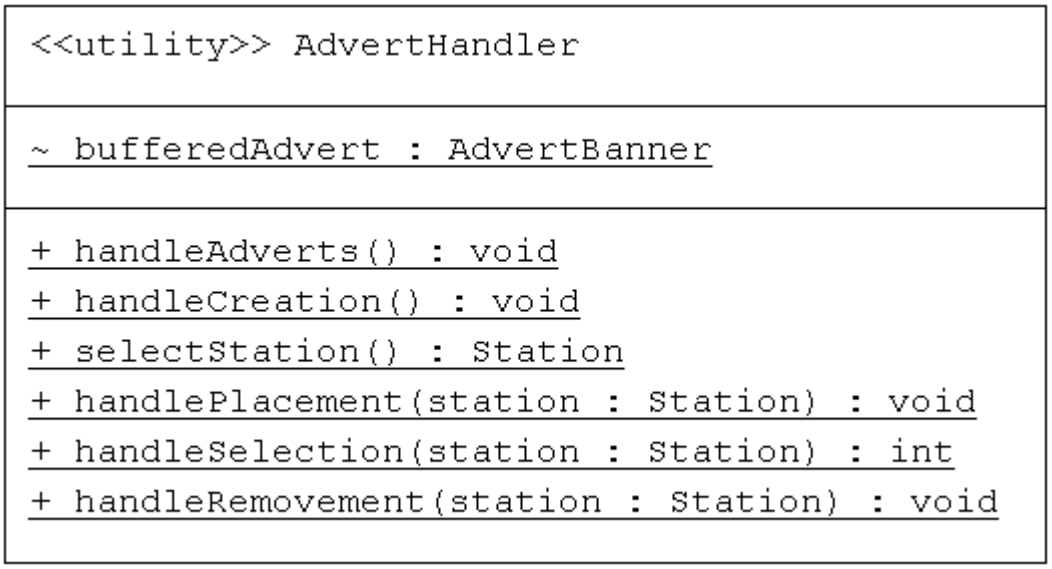

Рис. 1.2.1. AdvertHandler class

Цей клас є елементом модуля опрацювання користувацької взаємодії. Він має поле, що містить останній створений рекламний банер, який може бути клонований для повторного розміщення на різних станціях.

Методи:

- handleAdverts – виводить головне меню взаємодії із секцією управління рекламними оголошеннями.
- handleCreation, handleSelection, handleRemovement, selectStation – ці методи утворюють допоміжні меню взаємодії, які безпосередньо забезпечують керування оголошеннями.

На основі дії цього класу можна зрозуміти структуру роботи і інших класів що забезпечують користувацьку взаємодію.

Клас MetroConverter

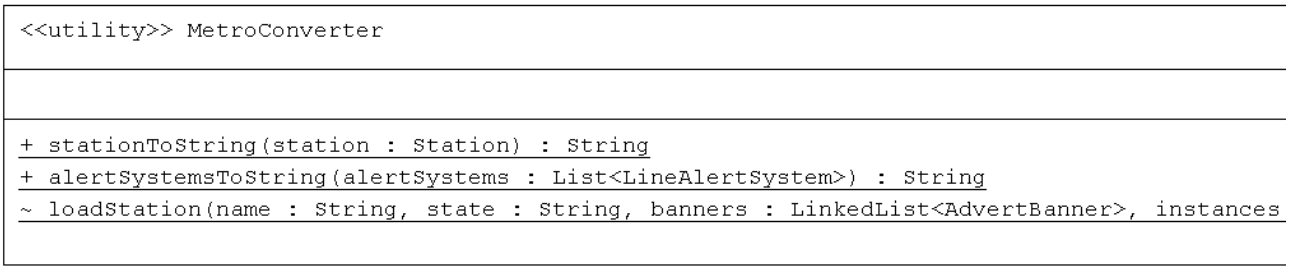

Рис. 1.2.2. MetroConverter class

Цей клас є допоміжним класом для MetroLineMemento, створений для зменшення кількості методів у основному класі, та для спрощення коду у ньому.

Методи:

- stationToString – утворює повне текстове подання безпосередньо станції метро, і, відповідно, усіх рекламних банерів та блоків оповіщення які вона містить.
- stationToString – утворює повне текстове подання системи керування оповіщеннями, що включає останнє збережене повідомлення та повний список блоків оповіщення, які були підписані на це сповіщення.
- loadStation - створює новий екземпляр типу Station на основі отриманого подання її компонентів.

Клас MetroLineMemento

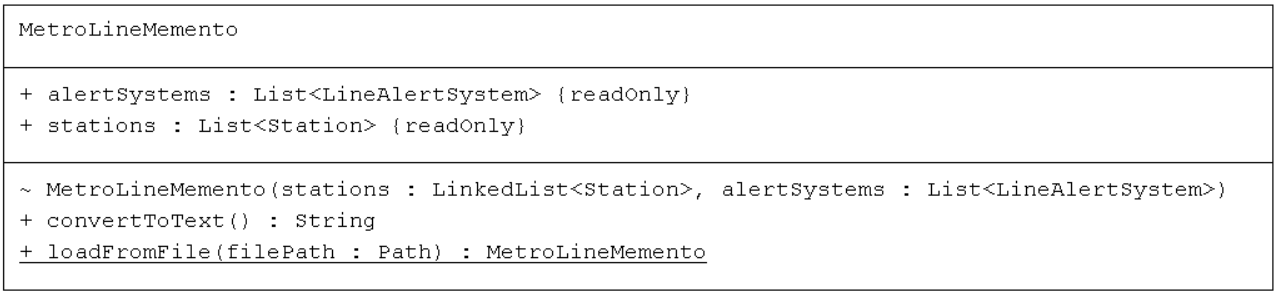

Рис. 1.2.3. Модульна організація програми

Цей клас є закритим для модифікації прототипом гілки метро. Він містить усі наявні на лінії станції, та усі системи керування сповіщеннями, які наявні на гілці метро.

Методи:

- convertToString – утворює повне текстове подання гілки метро, а отже і стану кожної станції, кожного блока оповіщення, а містить інформацію про підписки блоків оповіщення до систем контролю.
- loadFromFile – повертає новий екземпляр класу MetroLineMemento, завантажений із файлу на диску.

Клас AdvertBanner

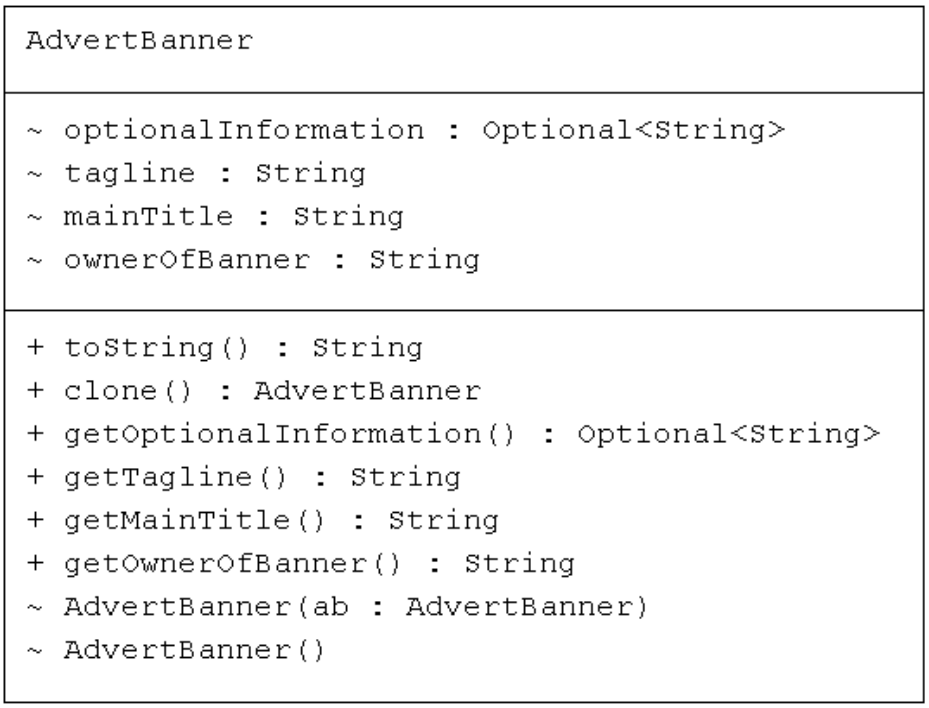

Рис. 1.2.4. Модульна організація програми

Цей клас є прототипом рекламного оголошення на станції метро. Він містить як інформацію про замовника, так і основні деталі, які будуть відображені на банері. Також, у ньому передбачено поле для необов'язкової інформації.

Методи:

- getOptionalInformation, getTagline, getMainTitle, getOwnerOfBanner – отримання інформації з інкапсульованих полів.
- clone – клонує прототип, створюючи новий екземпляр класу із такими самими даними.

- toString – відображає клас у текстовому форматі, подіному до XML.

Клас AlertInstance

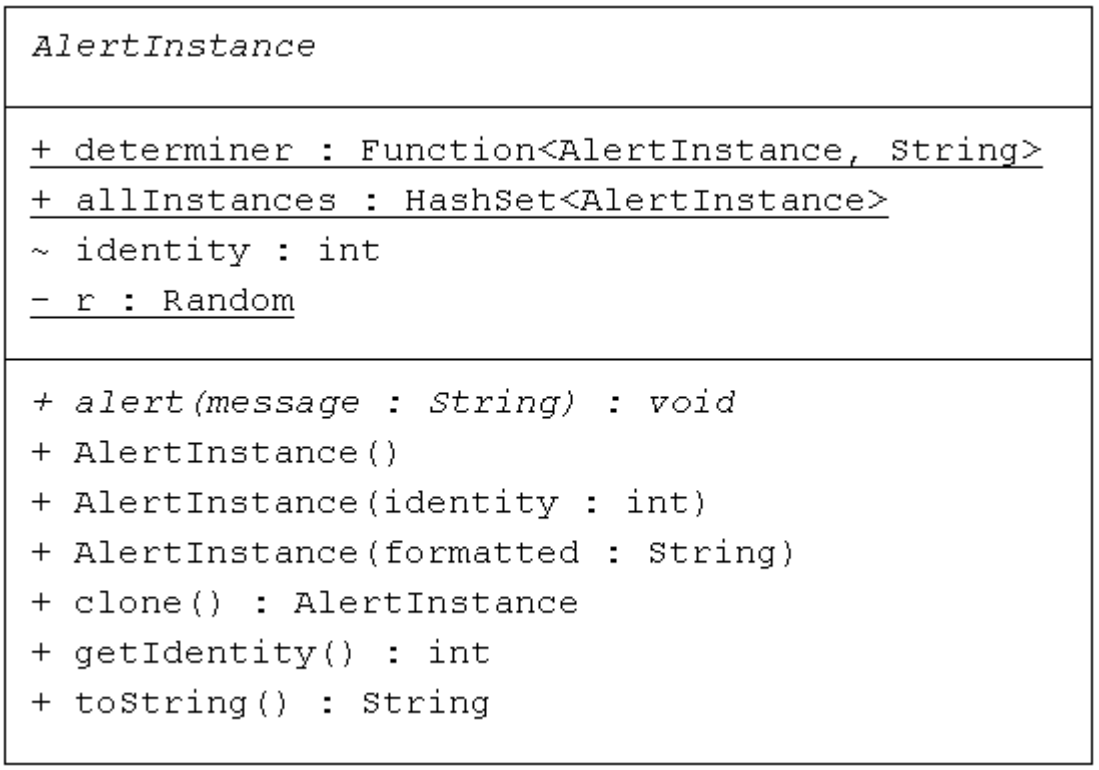

Рис. 1.2.5. Модульна організація програми

Це абстрактний клас який описує блок системи сповіщення. В ньому передбачено поле індивідуального ідентифікатора яке випадковим чином створюється при створенні об'єкту. А також в класі передбачений функціональний інтерфейс який містить частину реалізації методу toString. Тобто при зміні структури програми або її модифікації, наявність такого поля значно спростить зміну способу дії методу.

Методи:

- getOptionalInformation, getTagline, getMainTitle, getOwnerOfBanner – отримання інформації з інкапсульованих полів.
- alert – метод задіюється для сповіщення про тривогу
- clone - дозволяє створити копію класу.
- getIdentity, toString - методи, що надають необхідну інформацію.

2. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ

2.1. Обґрунтування вибору та опис шаблонів проектування для програмної реалізації системи керування станціями метрополітену

1) Знімок

Шаблон проектування "Знімок" - це спосіб збереження і відновлення стану об'єкта без порушення його інкапсуляції.

Його найбільш доцільно використовувати за наявності об'єктів, що постійно змінюють свій стан, але при цьому може виникнути потреба повернутися до минулого стану.

Тобто, головна ідея шаблону полягає у виділенні стану об'єкту в окремий об'єкт, який називається знімок. Цей знімок може зберігати в собі певну інформацію про стан об'єкту, наприклад, значення полів чи стан пам'яті. Оригінальний об'єкт, може створювати знімки свого стану, а також відновлювати свій стан з "знімка".

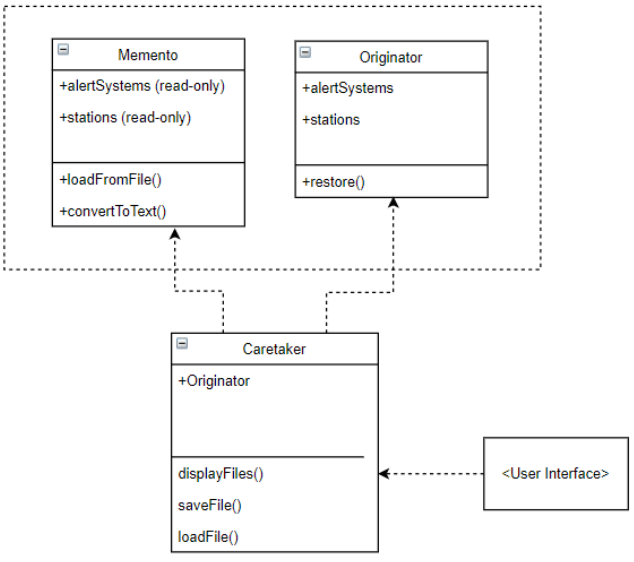

Рис. 2.1.1. Діаграма класів, які входять до шаблону «Знімок»

Учасники шаблону:

- **Originator:** об'єкт, стан якого потрібно зберегти або відновити**.**
- **Memento:** об'єкт, який зберігає внутрішній стан Originator і надає йому доступ тільки до себе**.**
- **Caretaker:** об'єкт, який відповідає за збереження та відновлення Memento**.**

Результати використання конструкції:

- При запуску додатку створюється екземпляр класу **Originator**. І далі усі дії програми, пов'язані із гілкою метрополітену діють саме на нього;
- При спробі запису стану програми у файл, створюється новий екземпляр класу **Momento** який не можна модифікувати вже він за допомогою окремих методів буде записаний на файл; Цей же клас буде створюватися кожний раз при читанні стану програми із файлу лише після створення цього класу буде можливе оновлення загального класу Originator .
- Клас **Caretaker** у цілому надає користувачеві усі доступні можливості для роботи із гілкою метро. Клас **Caretaker** практична перенаправлять запити користувача іншим класом які зображення умовним блоком на діаграмі.

Обґрунтування використання шаблону:

Використання заданого шаблону у моєму додатку обумовлено можливістю скористатися перевагами цього шаблони із різних сторін. По-перше, оскільки снапшот моєї програми характеризує лінію метро, то завантаження різних снапшотов дозволить використовувати програму для керування будь якими станціями метро. По-друге, завдяки можливості зберігати різні стани лінії метро, адміністратор системи матиме інструмент.

2) Стан

Шаблон проектування "Стан" - це поведінковий шаблон, який дозволяє

об'єкту змінювати свою поведінку в залежності від свого внутрішнього стану. Тобто він дає об'єктові можливість вести себе по-різному при одних і тих самих діях.

Шаблон проектування "Стан" слід використовувати, якщо виконуються наступні передумови:

- Об'єкт має багато можливих станів, які впливають на його поведінку.

- Логіка переходу між станами складна і залежить від багатьох умов.

- Код об'єкта стає занадто складним і нечитабельним через велику кількість умовних операторів.

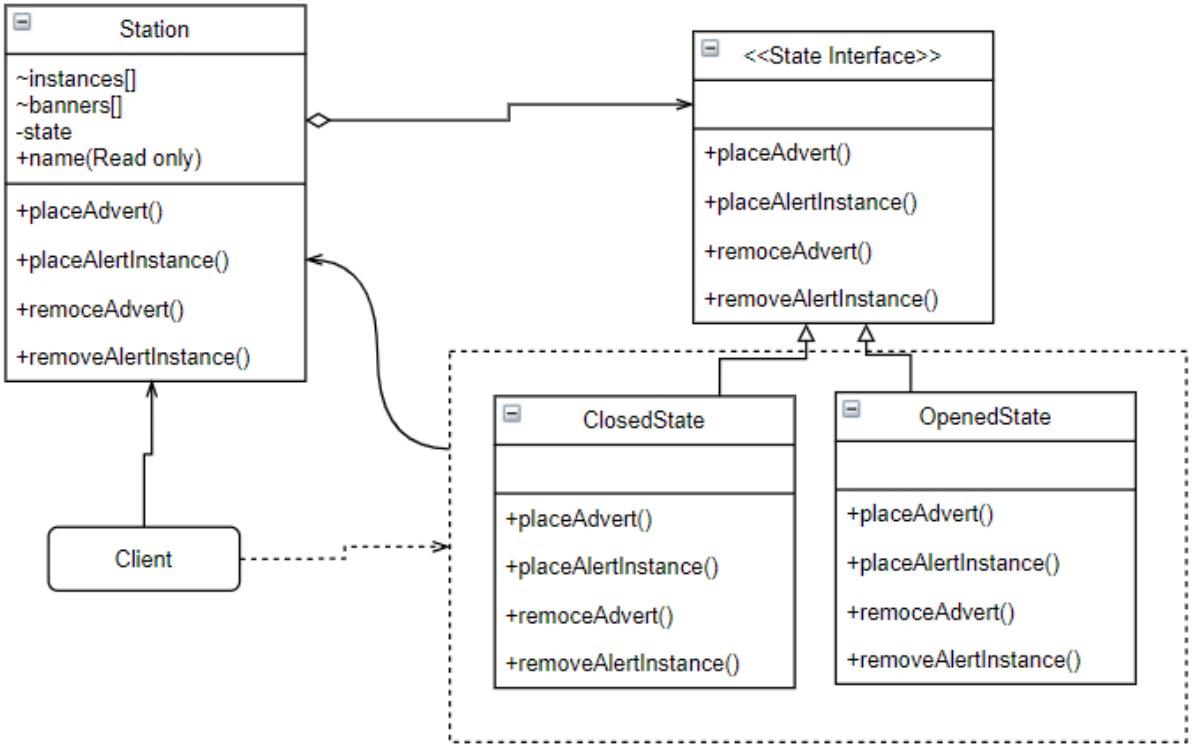

Рис. 2.1.2. Діаграма класів, які входять до шаблону «Стан»

Результати використання конструкції:

Учасники шаблону:

- Інтерфейс(**State Interface**) для опису інтерфейсу стану. Він містить методи для взаємодії з контекстом і може мати деякі загальні реалізації.

- Конкретні класи стану(**Closed state, Opened state**), які наслідують або

реалізують абстрактний клас або інтерфейс. Вони містять логіку поведінки для кожного стану і можуть переводити контекст в інший стан.

- Клас контексту (**Station**), який має посилання на поточний стан і надає йому доступ до своїх даних і методів. Він також може змінювати своє посилання на стан за допомогою спеціального методу.

Використання такого шаблону може полегшити розробку і покращити читабельність коду, який залежить від багатьох станів об'єкта. До того ж, за допомогою цього шаблону можна ізолювати логіку кожного стану в окремих класах, що сприяє принципу єдиної відповідальності (**S**OLID). Також це забезпечує простий спосіб додавати нові стани або модифікувати існуючі без зміни коду контексту або інших станів.

Результати використання конструкції:

- При спробі змінити наявні на станції об'єкти, запити на зміну за допомогою класу Station надходять до збереженого класу-наслідка від інтерфейсу State. І відповідно вже у цих класах відбувається обробка запиту користувача відповідно до стану станції.
- Методи класу State приймають в якості параметра об'єкт класу Station над яким відбувається усі необхідні дії .
- За допомогою користувацького інтерфейсу користувача має можливість змінювати стан станції наразі він може обрати один із двох запропонованих .

Обґрунтування використання шаблону:

Використання цього шаблону дозволяє ефективно та інтуїтивно зрозуміло контролювати стан станцій які може відповідати дійсного стану станцій в метрополітені. З його допомогою я можу обмежити доступ до певних частин станції під час того коли станція відкрита, і разом з тим я наголошую таким чином на необхідності закривати станцію для пасажирів при необхідності модифікувати певні її системи.

3) **Прототип**

Даний шаблон задає види створюваних об’єктів за допомогою екземпляра-прототипу і створює нові об’єкти шляхом копіювання цього прототипу. Початкові екземпляри класу можуть створюватися за допомогою інших патернів проектування як-от Factory, або, як це виконано в моїй роботі за допомогою патерну Builder.

Учасники шаблону:

- **AdvertBannerBuilder** – створює прототипи рекламних банерів для станцій метрополітену;
- **AdvertBanner -** безпосередньо прототип, який зберігає всю пов'язану з рекламним банером інформацію, а також наслідує інтерфейс **Cloneable** і може створювати нові об'єкти **-** точні копії себе;
- **Station** – зберігає обмежену кількість об'єктів типу AdvertBanner;

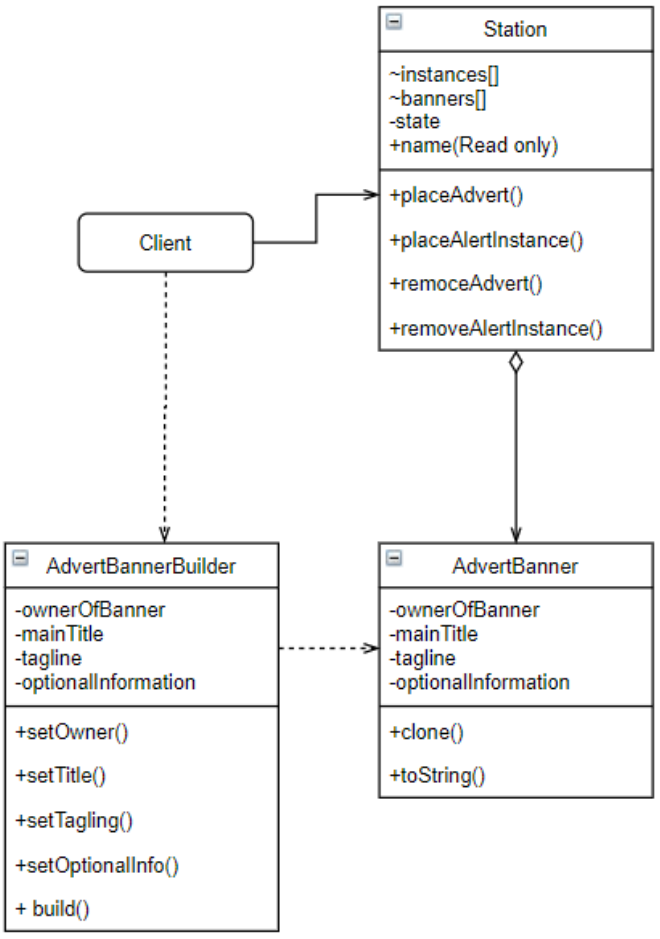

Рис. 2.1.3. Діаграма класів, які входять до шаблону «Прототип»

Результати використання конструкції:

Під час виконання команди "create" отриманої від користувацького інтерфейсу видається меню для роботи для взаємодії із класом **AdvertBannerBuilder.** Після створення прототипу він зберігається у окремій комірці задля можливості додати його копії на всі необхідні станції.

Обґрунтування використання шаблону:

Використання шаблону прототип у контексті і програми дозволяє значно спростити процес масового додавання рекламних оголошень на станції метрополітену, оскільки для додавання однакових рекламних оголошень достатньо створити лише 1 прототип який потім може бути доданий будь-яку кількість разів і розміщений на будь якій кількості станції .

4) **Вільний Будівельник**

Шаблон проектування Fluent Builder (Вільний Будівельник) - це спосіб створення складних об'єктів за допомогою послідовності методів, які повертають той самий об'єкт будівельник з різними параметрами. Цей шаблон дозволяє писати читабельний і гнучкий код, який можна легко модифікувати і розширювати.

Передумовою для використання шаблону Fluent Builder є наявність класу, який має багато полів або конструктор з багатьма параметрами. Такий клас може бути важким для розуміння і створення екземплярів. За допомогою шаблону Fluent Builder можна створити окремий клас-будівельник, який надає методи для встановлення кожного поля або параметра окремо. Кожен метод повертає той самий об'єкт-будівельник, що дозволяє ланцюжково викликати методи.

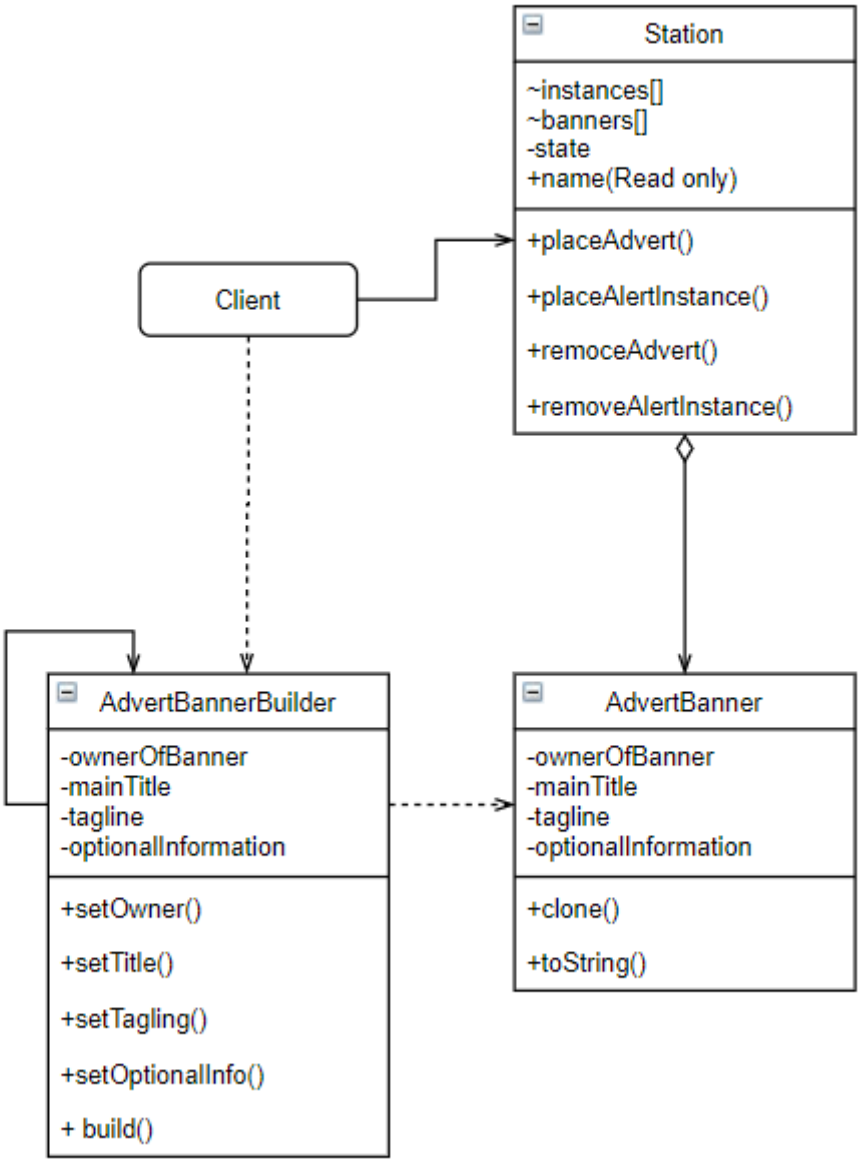

Рис. 2.1.4. Діаграма класів, які входять до шаблонів «Вільний будівельник»

Учасники шаблону:

– **AdvertBannerBuilder** – Клас якій надає можливість різними підходами провести створення об'єктів класу AdvertBanner у одному із сценаріїв використання він буде використовуватися сам на себе заради можливості викликати методи створення ланцюгом;

– **AdvertBanner -** клас, що містить всю необхідну інформацію про рекламне оголошення с процесутанції метрополітену. Конструюється за допомогою класу- будівельника. Має package-private конструктор, який не дозволить

створення об'єкта окрім як через AdvertBannerBuilder або за допомогою методу clone();

– **Station** – зберігає обмежену кількість об'єктів типу AdvertBanner;

Результати використання конструкції:

Створений клас AdvertBannerBuilder, методи якого повертати клас який їх викликав (самі себе). І завдяки цьому можна буде за допомогою мою ланцюжків викликів методів одне після одного створювати об'єкти AdvertBanner. Тобто це стає можливим саме завдяки повернення самого себе.

Обґрунтування використання шаблону:

Застосування патерну проектування "Вільний Будівельник" дозволяє поетапно створювати об'єкти рекламних банерів, при цьому надаючи можливість розробникам при розширенні програми користуватися різними підходами у застосуванні класів, що реалізують цей шаблон. І у поєднанні із патерном програмування "прототип", програмна реалізація створення нових рекламних банерів а також користувацький інтерфейс які викликатиме методи будуть значною мірою більш інтуїтивно зрозумілими.

5) Спостерігач

Шаблон проектування Observer дозволяє створити зв'язок "один до багатьох" між об'єктами, так що при зміні стану одного об'єкта всі його спостерігачі отримують повідомлення про це. Цей шаблон корисний, коли потрібно синхронізувати різні компоненти системи, не залежачи від їх внутрішньої реалізації.

Шаблон Observer має такі переваги: він знижує зв'язність між компонентами системи; він дозволяє динамічно додавати і видаляти спостерігачів; він покращує модульність і розширюваність коду.

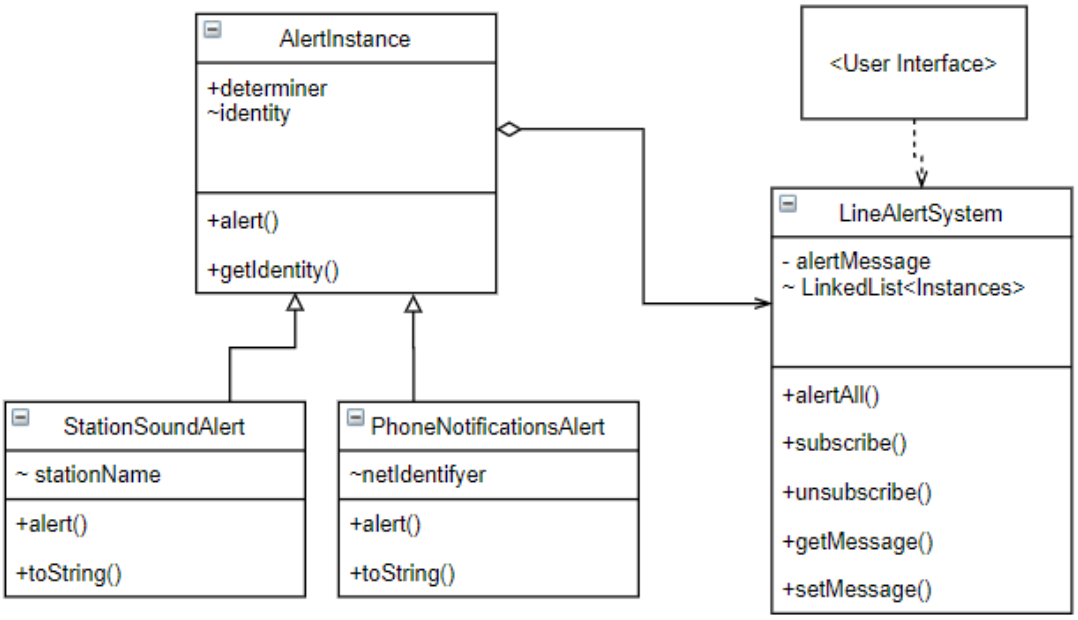

Рис. 2.1.5. Діаграма класів, які входять до шаблонів «Спостерігач»

Учасники шаблону:

Шаблон Observer складається з трьох основних елементів: суб'єкта(**LineAlertSystem**) та спостерігача(**StationSoundAlert & PhoneNotificationAlert**) і узагальнення - абстрактного класу(**AlertInstance**).

Суб'єкт - це загальна система оповіщення, що здатна передавати повідомлення про тривогу усім блокам оповіщення які до неї підписані. Тобто це клас який містить посилання на всі підписані об'єкти та повідомлення яке він може їм передати.

Спостерігач - це блок система оповіщення такий як під'єднано до мережі гучномовець або блок розсилки повідомлень на мобільні пристрої. Окрім цього, класи мають унікальний ідентифікатор які дозволяють підписати їх до систем оповіщення у разі відновлення стану програми із файлу.

Узагальнення - це абстрактний клас які описує основні характеристики які має мати система оповіщення метрополітену та дозволяє підписувати до системи оповіщення будь-які блоки які наслідують цей клас.

Обґрунтування використання шаблону:

Цей шаблон дуже природньо відображає системи оповіщення, в тому числі і системи оповіщення пасажирів метрополітену. Використання цього шаблону проектування дозволяє на початку створити систему оповіщення до якої згодом, при розширенні системи можна буде додати інші типи сповіщувачів, причому це можна буде зробити не міняючи код уже існуючих блоків.

6) Композит

Шаблон проектування Composite дозволяє створювати ієрархічні структури об'єктів, які можна опрацьовувати однаковим чином. Цей шаблон використовується, коли потрібно представити групу об'єктів або дерево об'єктів як один об'єкт. Шаблон Composite складається з трьох типів об'єктів: Component, Composite і Leaf. Component - це абстрактний клас або інтерфейс, який визначає спільний інтерфейс для всіх об'єктів у структурі. Composite - це клас, який реалізує Component і містить колекцію дочірніх об'єктів типу Component. Composite може додавати, видаляти і отримувати доступ до своїх дочірніх об'єктів. Leaf - це клас, який реалізує Component і представляє листковий вузол у структурі. Leaf не має дочірніх об'єктів і виконує певну функцію.

Передумови для застосування шаблону Composite - це наявність складної структури об'єктів, яка може бути представлена у вигляді дерева або графа, і необхідність опрацьовувати всю структуру або її частини однаковим чином. Засоби для реалізації шаблону Composite - це використання абстракції Component для опису спільного інтерфейсу для всіх об'єктів у структурі, використання класу Composite для збереження та керування дочірніми об'єктами типу Component, використання класу Leaf для реалізації конкретної функціональності листкових вузлів.

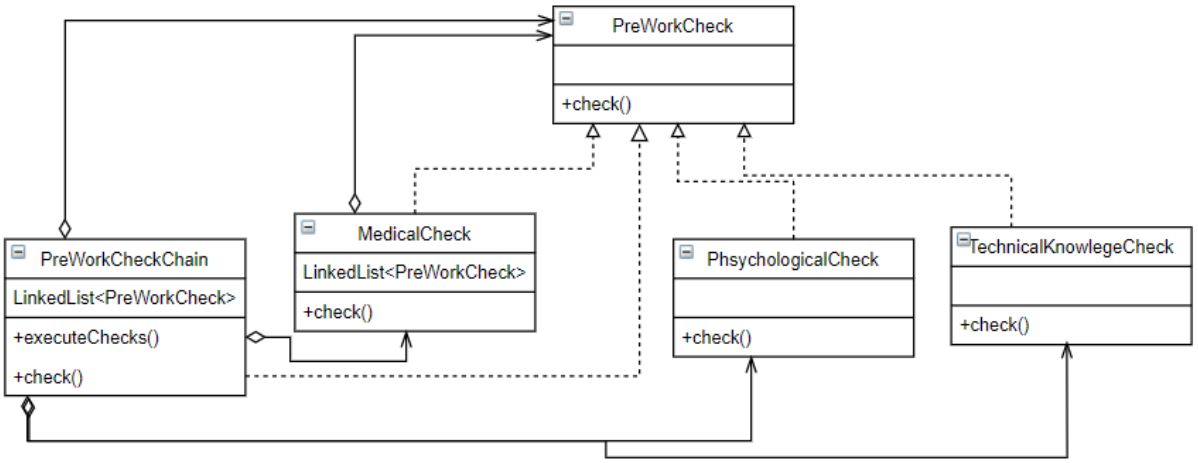

Рис. 2.1.6. Діаграма класів, які входять до шаблону «Композит»

Обґрунтування використання шаблону:

Учасники шаблону:

- **PreWorkCheckChain** – Це клас типо композит який місяць в собі посилання на базові перевірки які робітник має пройти Перед застосуванням на зміну у метрополітені. Він представляє Найвищий рівень композитор
- **MedicalCheck** - цей класс знаходиться на більш низькому рівні композитору. Він містить певну свою логіку та все ще може містити інші класи які наслідують інтерфейс. Тобто до цього класу все ще можна додати нові елементи перевірки, без зміни його коду.
- **PsychologicalCheck & TechnicalKnowlegeCheck** - це класи типу листя. Вони мають виключно свою логіку та не можуть містити посилання на інші перевірки;

Результати використання конструкції:

- Був реалізований клас який містить в собі посилання на всі перевірки середнього рівня він використовується кожен раз коли працівник має розпочати роботу на станції і коли він має бути перевірений за усіма необхідними процедурами.
- Був створений інший клас, який теж може містити в собі сторонні

поглиблюючі перевірки на випадок розширення необхідного переліку перевірок працівників.

– І також були створені класи, поглиблення яких іншими видами перевірок не передбачатиметься.

Обґрунтування використання шаблону

Завдяки цьому шаблону була створена система перевірки працездатності робітників метрополітену. Ця система може бути розширена новими перевірками без необхідності змінювати код вже існуючих. До того що вона достатньо природно відображає порядок перевірки працівників у реальному житті .

7) **Фабричний метод**

Шаблон проектування Factory - це породжуючий шаблон, який дозволяє створювати об'єкти різних типів за одним інтерфейсом. Цей шаблон використовується, коли класу не відомо заздалегідь, які саме об'єкти йому потрібно створювати, або коли він хоче делегувати цю відповідальність своїм підкласам.

Передумови для використання шаблону проектування Factory - це наявність декількох типів продуктів, які мають спільний інтерфейс та поведінку, але різну реалізацію.

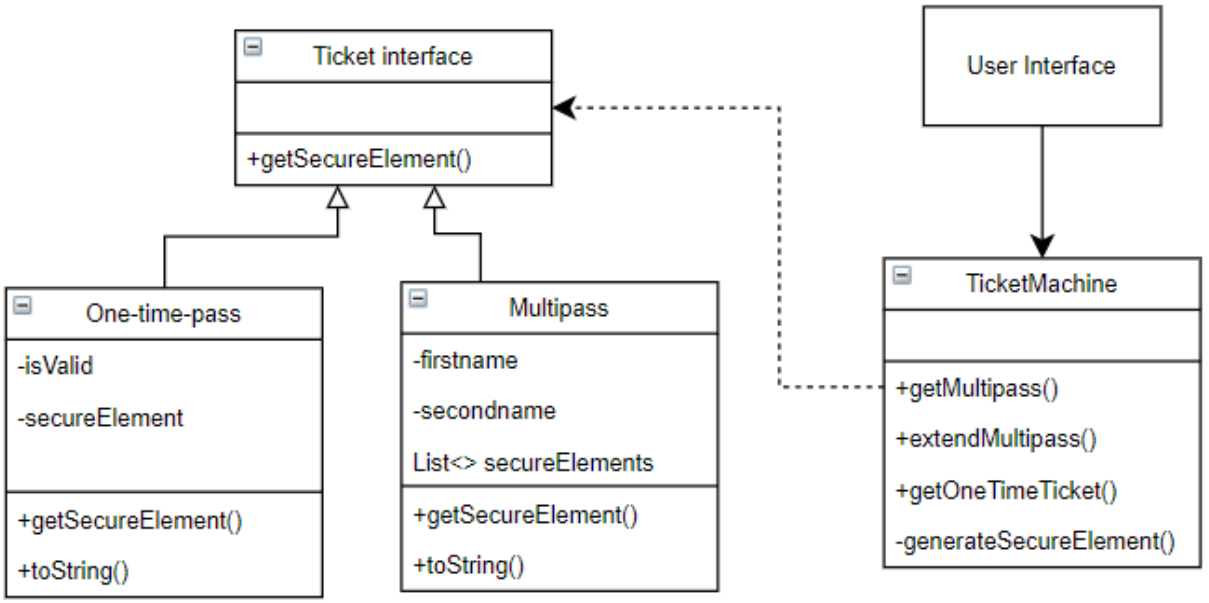

Рис. 2.1.7. Діаграма класів, які входять до шаблону «Фабрика»

Результати використання конструкції:

Створено:

- Product (**Ticket**) - це інтерфейс, який описує об'єкти, що створюються фабрикою, тобто квитки. Він визначає спільну поведінку та інтерфейс для всіх типів квитків та проїзних документів.

- ConcreteProduct (**OneTimeTicket || Multipass**) - це конкретні класи, які мають свої дані, та реалізовують різні методи надання на перевірку захисних елементів.

- Creator (**TicketMachine**) - це безпосередньо фабрика, яка створює різні об'єкти типу Ticket, яка окрім необхідних властивостей, має також і засіб для подовження багаторазових квитків.

Обґрунтування використання шаблону:

Даний шаблон проектування було обрано щоб отримати відносно простий та ефективний засіб створення квитків, які користувач може використати у програмі. Квитки ці - однотипні, конструктори не міститимуть великої кількості полів, а тому інші шаблони використовувати тут менш доцільно.

2.2. Діаграма класів

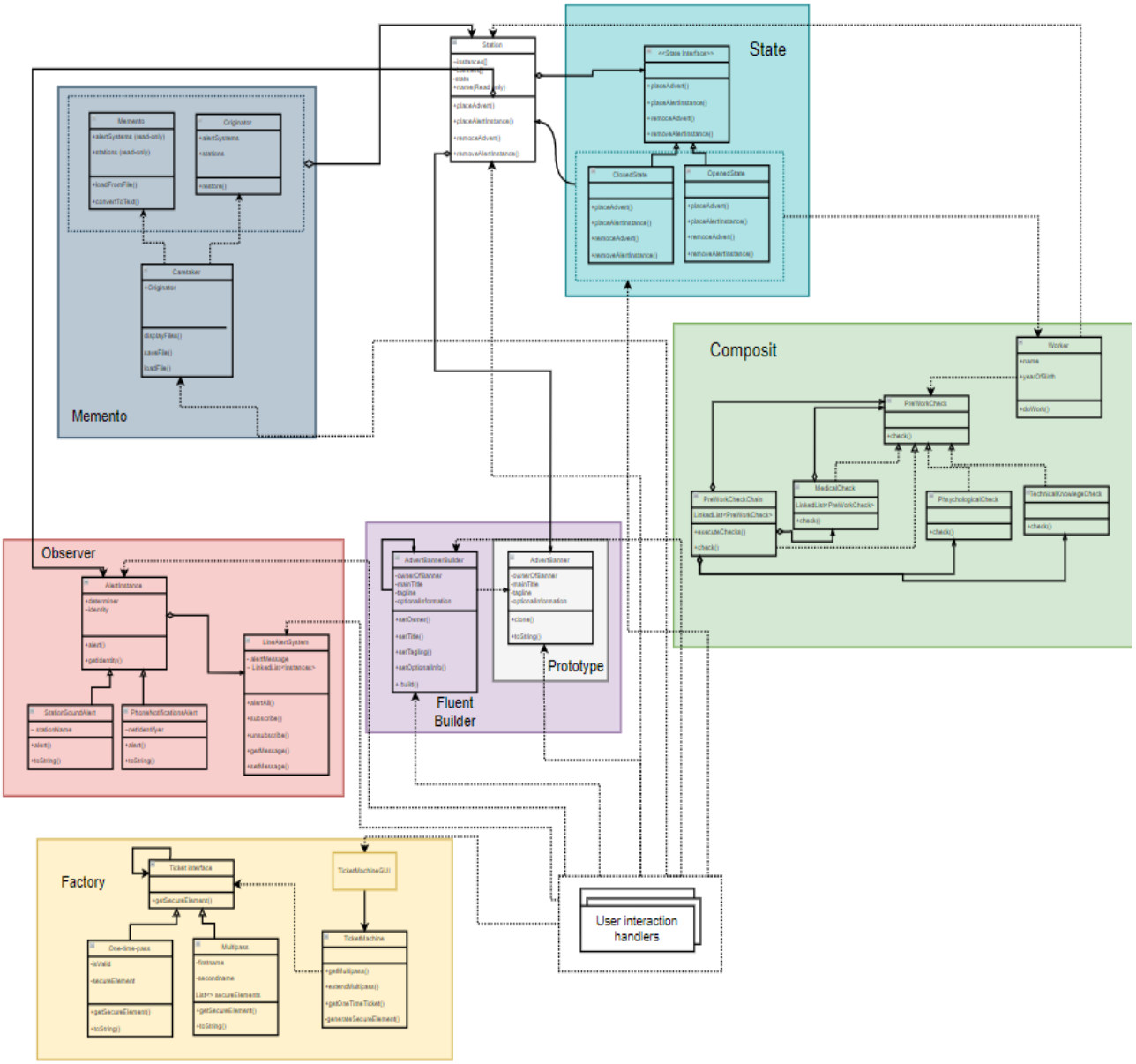

Рис. 2.2, . Діаграма класів системи керування станціями метрополітену

2.3. Опис результатів роботи програми

Продемонструю основні можливості програми за допомогою опису основних користувацьких меню:

```
====================
Type "0" to get into Version control menu
Type "1" to get into Alert systems menu
Type "2" to get into Station control
Type "3" to get into Tickets menu
Type "exit" to close the program
```

Рис. 2.3.1. Головне меню

У головному меню ми можемо обрати до якого наступного меню нас буде перенаправлено:

- це може бути меню контролю версій, тобто збереження стану лінії метро або його завантаження
- або ж це може бути система керування системи сповіщення метро
- або це може бути системи керування властивостями станції або це може бути робота із автоматом квитків

```
Type "select" to load version file
Type "save" to save current state
Type "back" to go back
```

Рис. 2.3.2. Робота з файлами версій

У модулі контролю версій буде запропоновані пункти меню такі дозволяють завантажити стан програми із наявного файлу або зберегти стан програми у файл. Причому при переході на обрати буде запропонована інше меню яка продемонструє список файлів із відповідним розширеннями. При переході на вкладку "зберегти" буде запропоновано ввести назву файлу.

```
Type "new-system" to make new alert system
Type "set-alert-message" to make change the message of alert system
Type "subscribe" to subscribe alert instance to system
Type "unsubscribe" to unsubscribe alert instance from system
Type "launch-alert" to use the ticket
Type "back" to go back
```

Рис. 2.3.3. Меню систем оповіщення

У меню керування системами сповіщення э пункти переходу на інші діалогові вікна, кожне з яких матиме наступний зміст:

- можна створити нову систему оповіщення і одразу встановити відповідне повідомлення
- можна встановити нове повідомлення для існуючої системи сповіщення
- можна підписати блок сповіщення до системи
- можна відписати блок сповіщення від системи
- можна запустити оповіщення на будь-якій системі

```
Type "advertising" to manege adverts
Type "alert-sys" to manage alert systems
Type "tickets" buy/use tickets
Type "make-station" to use add new station
Type "set-state" to use add new station
Type "back" to go back
```

Рис. 2.3.4. Меню керування станціями

У меню керування станцією можна перейти до інших діалогових вікон:

- Меню реклами
- Меню блоків сповіщення
- Меню квитків

Окрім цього можна встановити інший стан для існуючої станції або додати нову станцію то системи.

```
advertising
Type "create" to create a new advert
Type "place" to place advert to station
Type "remove" to remove advert from station
Type "watch" to look threw adverts
Type "back" to go back
```

Рис. 2.3.5. Робота з рекламними банерами

У меню для реклами можна:

- створити новий рекламний банер, який одразу буде збережений у буферний клітинці
- додати до станції банер який попередньо був збережений у буферні клітинці
- зняти рекламний банер зі станції
- подивитися наявні банери

```
alert-sys
Type "place" to place alert instance to station
Type "remove" to remove alert instance from station
Type "watch" to look threw alert instance on stations
Type "back" to go back
```

Рис. 2.3.6. Робота з блоками системи оповіщення

У меню блоків системи оповіщення можна:

- розмістити блок оповіщення на станції, при цьому процесі буде відкрите ще одне діалогове вікно для обрання та подальшого наповнення блоку сповіщення
- видалити блог сповіщення при цьому він автоматично буде від писаний від усіх систем оповіщення
- передивитися доступні встановлені блоки оповіщення на станції

```
Type "one-time-pass" to make one time pass
Type "multipass" to make multi-time ticket
Type "top-up" to add charges on multipass
Type "use-ticket" to use the ticket
Type "back" to go back
```

Рис. 2.3.7. Робота з квитками

Меню роботи із квитками буде запропоновані наступні опції:

- додати одноразовий квиток
- створити багаторазовий квиток
- поповнити багаторазовий квиток
- використати будь-який квиток

ВИСНОВКИ

Метою даної курсової роботи було розроблення системи керування станциями метрополітену з використанням шаблонів проектування. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» студентами ІІ курсу кафедри ПЗКС НТУУ «КПІ ім. І.Сікорського».

Для досягнення поставленої мети у повному обсязі виконано завдання, визначені у аркуші завдання на курсову роботу; розроблено графічні матеріали; реалізовано всі вимоги до програмного продукту, програмного та апаратного забезпечення, наведені у технічному завданні; створено відповідну документацію.

Розроблене програмне забезпечення дозволяє відображати стан станцій метрополітену у програмі, моделюючи їх. Створювати рекламні банери із інформацією про них та їх наповнення, розміщувати банери на станціях та переглядати їх. До того ж, розроблене програмне забезпечення моделює процес перевірки придатності працівників метрополітену, перед їх відправленням на встановку компонентів станції. Крім цього, програма дозволяє встановлювати модулі оповіщення пасажирів на станціях, підписувати ці модулі до різних систем оповіщення та викликати їх. Мій програмний застосунок також надає можливість певним чином моделювати процес роботи із квитками метро.

Програму створено на основі використання шаблонів проектування: зокрема до структури розробленого програмного забезпечення входить реалізація семи шаблонів, які належать до різних груп шаблонів проектування.

Для розроблення програмного забезпечення використано мову програмування Java (версія Java 20).

Перспективним напрямом подальшого дослідження даної тематики є розширення функціональності програми, а також додавання методів взаємодії систем станції з рухомим складом метро.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Refactoring Guru [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://refactoring.guru/design-patterns/catalog (дата звернення 10.05.2023) - Назва з екрана
2. Vertex Academy [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://vertex-academy.com/tutorials/en/ (дата звернення 19.05.2023) - Назва з екрана
3. Vertex Academy [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://www.freecodecamp.org/news/object-oriented-programming-concepts-java/ (дата звернення 23.03.2023) - Назва з екрана