Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря
Сікорського” Факультет інформатики та
обчислювальної техніки

Лабораторна робота №6
«Технології розроблення програмного забезпечення»
Тема: « E-mail клієнт»

Виконав:
Студент групи
ІА-34
Марченко А.О.
Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

15. **E-mail клієнт** (singleton, builder, decorator, template method, interpreter, client-server)

Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Теоритичні відомості

Шаблон «Abstract Factory»

Призначення патерну: Шаблон «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів [6]. Для цього виноситься загальний інтерфейс фабрики (AbstractFactory) і створюються його реалізації для різних сімейств продуктів.

Переваги та недоліки:

+ Спрощує створення об’єктів і код стає легшим для розуміння.

+ Об’єкти створені однією фабрикою добре узгоджуються один з одним і зменшується кількість помилок взаємодії між ними.

+ Відокремлення створення об’єктів від їх використання, за рахунок чого, код стає біль структурованим.

+ Додавання нових сімейств продуктів виконується без зміни існуючого коду.

- Збільшується складність коду, особливо для простих проєктів.

- Додавання нового типу продукту є складним і вимагає змін коду в багатьох місцях.

Шаблон «Factory Method»

Призначення: Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу [6]. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів не є інтерфейсною (AnOperation) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон «Фабричний метод» носить ще назву «Віртуальний конструктор».

Переваги та недоліки:

+ Позбавляє клас від прив’язки до конкретних класів продуктів.

+ Виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.

+ Спрощує додавання нових продуктів до програми.

- Може призвести до створення великих паралельних ієрархій класів.

Шаблон «Memento»

Призначення: Шаблон використовується для збереження і відновлення

стану об'єктів без порушення інкапсуляції [6]. Об'єкт «Memento» служить виключно для збереження змін над початковим об'єктом (Originator). Лише початковий об'єкт має можливість зберігати і отримувати стан об'єкту «Memento» для власних цілей, цей об'єкт є «порожнім» для кого-небудь ще. Об'єкт «Caretaker» використовується для передачі і зберігання ментo об'єктів в системі.

Переваги та недоліки:

+ Не порушує інкапсуляцію вихідного об’єкта.

+ Спрощує структуру вихідного об’єкта. Не потрібно зберігати історію версій свого стану.

- Вимагає багато пам’яті, якщо клієнти дуже часто створюють знімки.

- Може спричинити додаткові витрати пам’яті, якщо об’єкти, що зберігають історію, не звільняють ресурси, зайняті застарілими знімками.

Шаблон «Observer»

Призначення: Шаблон визначає залежність «один-до-багатьох» таким чином, що коли один об'єкт змінює власний стан, усі інші об'єкти отримують про це сповіщення і мають можливість змінити власний стан також.

Переваги та недоліки:

+ Можливість паралельної та асинхронної обробки повідомлень про оновлення.

+ Спостерігачів можна добавляти та видаляти в будь-який момент часу.

+ Спостерігач і суб’єкт можуть працювати в різних потоках.

+ Реалізує принцип слабкого зв’язку між об’єктами.

- Послідовність розсилки повідомлень підписникам не підтримується

Шаблон «Decorator»

Призначення: Шаблон призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми . Декоратор

деяким чином «обертає» (за рахунок агрегації) початковий об'єкт зі збереженням його функцій, проте дозволяє додати додаткові дії. Такий шаблон надає гнучкіший спосіб зміни поведінки об'єкту чим просте спадкоємство, оскільки початкова функціональність зберігається в повному об'ємі. Більше того, таку поведінку можна застосовувати до окремих об'єктів, а не до усієї системи в цілому.

Переваги та недоліки:

+ Дозволяє мати кілька дрібних об’єктів, замість одного об’єкта «на всі випадки життя».

+ Дозволяє додавати обов’язки «на льоту».

+ Більша гнучкість, ніж у спадкування.

- Велика кількість крихітних класів.

- Важко конфігурувати об’єкти, які загорнуто в декілька обгорток одночасно.

Хід роботи

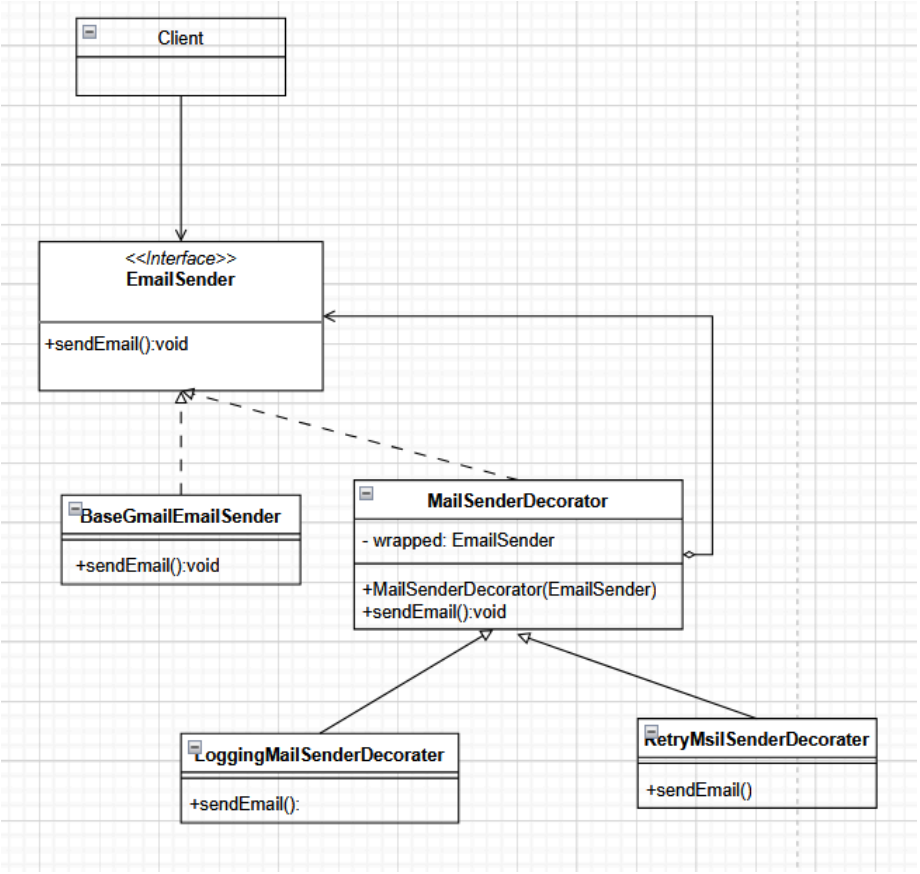


Рис. 6.1 - Патерн Decorater для відправки повідомлення

У центрі архітектури знаходиться інтерфейс EmailSender, який визначає базову операцію sendEmail(), що повинна виконуватися будь-яким класом, здатним надсилати повідомлення. Базовий функціонал забезпечує клас BaseGmailEmailSender, який є конкретною реалізацією інтерфейсу EmailSender. Він містить реальну логіку надсилання листів через SMTP Gmail і є основним відправником, який може використовуватися самостійно або бути обгорнутим декораторами.

Для додавання додаткових можливостей над базовим відправником використовується абстрактний клас MailSenderDecorator. Він також реалізує інтерфейс EmailSender, але містить посилання на інший об’єкт EmailSender, якого він декорує. У конструктор декоратора передається цей внутрішній об’єкт, а метод sendEmail() за замовчуванням делегує виклик вкладеному відправнику.

Від абстрактного декоратора успадковуються два конкретні декоратори.

Перший - LoggingMailSenderDecorator, який перевизначає sendEmail(), додаючи логування дій перед або після фактичного надсилання листа. Такий декоратор дозволяє прозоро доповнити роботу будь-якого відправника можливістю запису дій у лог.

Другий - RetryMailSenderDecorator, що додає механізм повторних спроб відправлення. Його метод sendEmail() виконує делегований виклик у циклі, повторюючи спроби до встановленої кількості разів у разі помилки. Це покращує надійність механізму відправлення, особливо при тимчасових збоях мережі або SMTP-сервера.

Посилання на код:

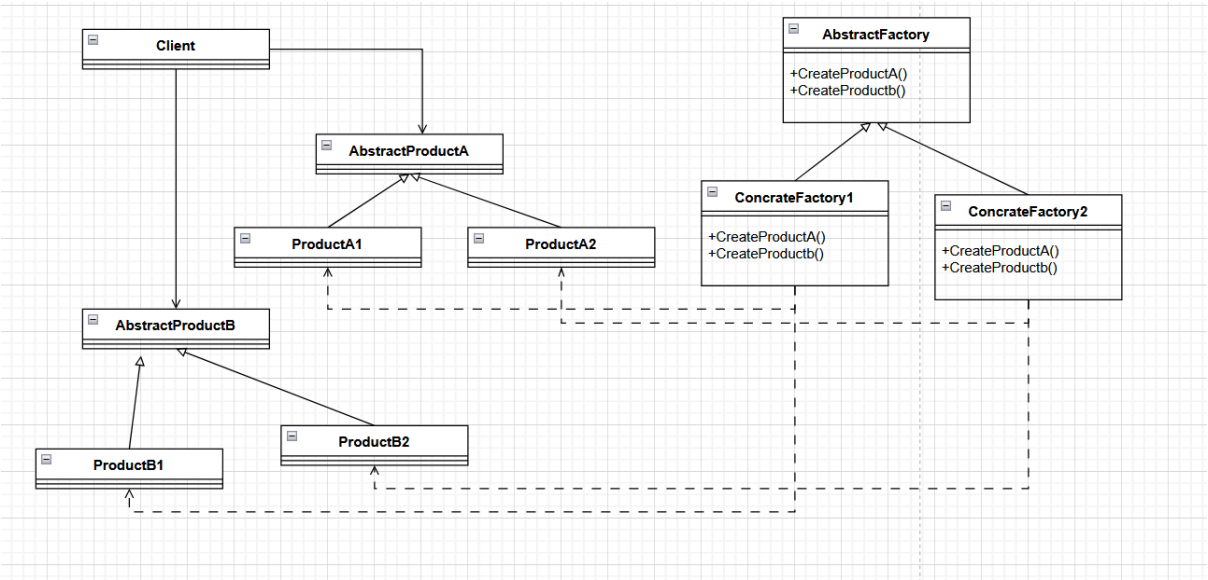
https://github.com/BeautifulBublik/TRPZ/tree/master/EmailClient/src/main/java/com/example/emailclient

Питання до лабораторної роботи

1. Яке призначення шаблону «Абстрактна фабрика»?

Абстрактна фабрика - це породжувальний патерн проектування, що дає змогу створювати сімейства пов'язаних об'єктів, не прив'язуючись до конкретних класів створюваних об'єктів.

2.Нарисуйте структуру шаблону «Абстрактна фабрика».



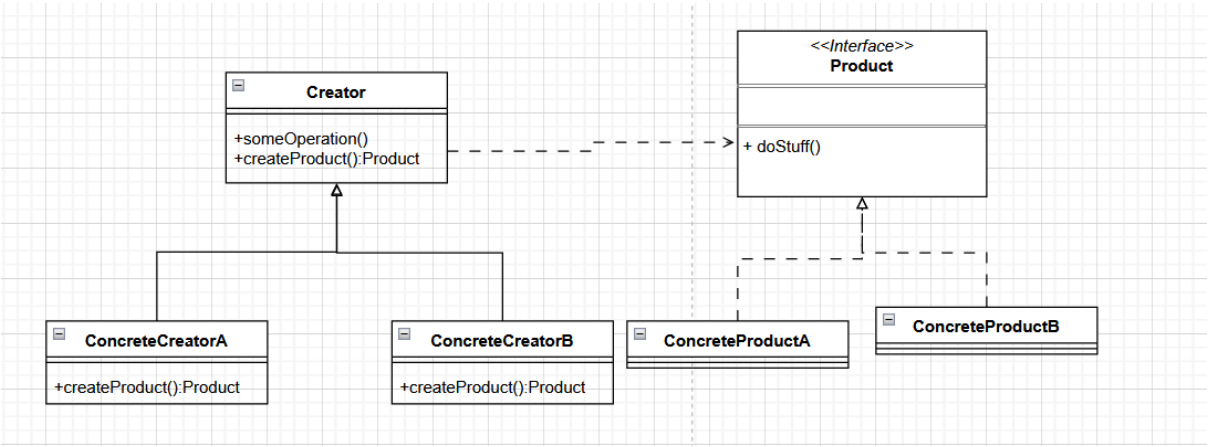
3.Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

В шаблон «Абстрактна фабрика» входять такі класи: Абстрактна фабрика, Конкретні фабрики, Абстрактні продукти та Конкретні продукти. Взаємодія відбувається наступним чином: абстрактна фабрика визначає інтерфейс для створення сімейства об'єктів-продуктів, а конкретні фабрики реалізують цей інтерфейс для створення продуктів конкретних сімейств. Клієнт використовує абстрактну фабрику для створення конкретного сімейства продуктів, не знаючи про їх конкретний клас.

4.Яке призначення шаблону «Фабричний метод»?

Призначення шаблону «Фабричний метод» - визначати інтерфейс для створення об'єктів у суперкласі, але дозволяти підкласам самостійно вирішувати, який саме клас інстанціювати. Цей шаблон дозволяє уникнути прив'язки коду до конкретних класів-продуктів, роблячи систему гнучкішою та полегшуючи її розширення

5.Нарисуйте структуру шаблону «Фабричний метод».



6.Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

В шаблон "Фабричний метод" входять класи Product (інтерфейс продукту), ConcreteProduct (конкретний продукт), Creator (творець, що містить фабричний метод) та ConcreteCreator (конкретний творець, який реалізує фабричний метод). Взаємодія полягає в тому, що Creator оголошує фабричний метод, а ConcreteCreator реалізує його, створюючи екземпляр конкретного продукту, який відповідає інтерфейсу Product.

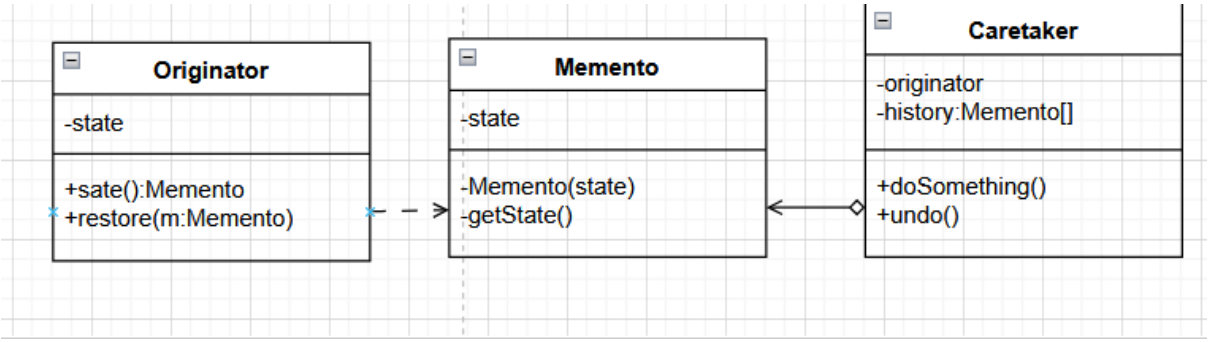
7.Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Фабричний метод — це шаблон, який дозволяє підкласам самостійно вирішувати, який конкретний об’єкт створювати. Він працює з одним типом продукту, а створення відбувається через перевизначення методу у нащадках. Натомість абстрактна фабрика створює цілу сім’ю пов’язаних між собою об’єктів, забезпечуючи їхню сумісність і приховуючи конкретні класи від клієнта. Тобто фабричний метод обирає конкретний продукт, а абстрактна фабрика — набір взаємопов’язаних продуктів.

8.Яке призначення шаблону «Знімок»?

Не порушуючи інкапсуляції, фіксує та виносить за межі об'єкта його внутрішній стан так, щоб пізніше можна було відновити з нього об'єкт.

9.Нарисуйте структуру шаблону «Знімок».



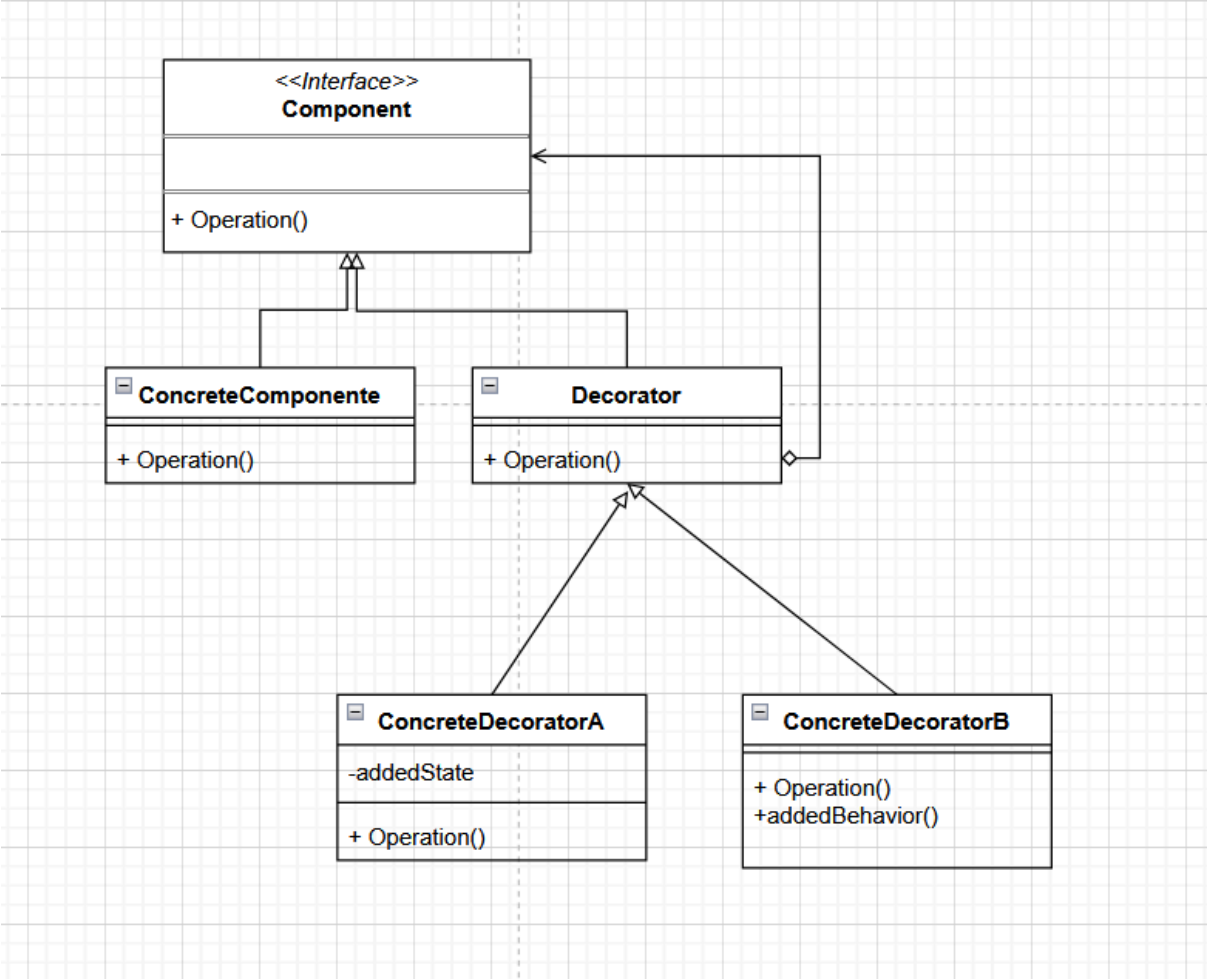
10.Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

Шаблон «Знімок» (Memento) складається з трьох основних класів: Originator (Творець), Caretaker (Опікун) та Memento (Знімок). Originator (Творець) створює і зберігає стан об'єкта. Він відповідає за створення об'єкта Memento, який містить потрібний стан. Caretaker (Опікун) зберігає об'єкт Memento в безпеці, не дивлячись у його внутрішню структуру. Caretaker також може запитувати у Originator створити новий знімок або відновити попередній стан за допомогою об'єкта Memento. Memento (Знімок) це об'єкт, що містить інформацію про стан об'єкта Originator на певний момент часу. Memento ділиться на дві частини: зовнішню (доступну Caretaker) і внутрішню (доступну тільки Originator).

11.Яке призначення шаблону «Декоратор»?

Призначення шаблону «Декоратор» — динамічно додавати нову поведінку до об'єкта, обгортаючи його в спеціальні об'єкти-декоратори. Це гнучка альтернатива класичному наслідуванню, яка дозволяє розширювати функціональність об'єктів, не змінюючи їхній вихідний клас. Шаблон забезпечує можливість додавати або видаляти функціональність під час виконання програми, просто приєднуючи або від'єднуючи декоратори.

12.Нарисуйте структуру шаблону «Декоратор».



13.Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

У шаблоні "Декоратор" беруть участь Компонент (абстрактний або конкретний клас), Конкретний компонент (клас, що реалізує базову функціональність) та Декоратор (абстрактний клас, який обгортає Компонент), а також Конкретний декоратор, який додає нову функціональність. Декоратори обгортають об'єкт компонента, передаючи виклики методів йому, і додають власну функціональність до або після них

14.Які є обмеження використання шаблону «декоратор»?

Декоратор може суттєво ускладнити структуру програми, бо велика кількість дрібних обгорток робить код важчим для читання та відлагодження. Порядок накладання декораторів має значення, тому

неправильна послідовність може привести до неочікуваної поведінки. Декоратор не підходить, якщо потрібно змінювати поведінку об’єкта цілком, а не поступово — тоді краще використати композицію або успадкування. Також не всі інтерфейси можуть бути зручно декоровані: якщо об’єкт має багато методів, кожен декоратор змушений їх повторювати, що створює зайву складність і дублювання.

Висновок: Я використав варіант шаблону «Декоратор», тому що він дозволяє розширити функціональність відправки електронних листів без зміни початкового класу відправника. Замість того щоб вносити зміни в базовий код, ми обгорнули основний сервіс відправки пошти у додаткові декоратори — для логування та повторного надсилання. Це дало можливість додати нову поведінку вибірково, гнучко комбінувати її та легко масштабувати систему. Такий підхід забезпечує чисту архітектуру, мінімізує ризик помилок і робить код набагато зручнішим для супроводу та розширення.Та вивчив шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчився застосовувати їх в реалізації програмної системи.