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

Лабораторна робота №2

із дисципліни «**Технології розроблення програмного забезпечення**»

Діаграма варіантів використання. Сценарії варіантів використання. Діаграми uml. Діаграми класів. Концептуальна модель системи.

Аудіоредактор

Виконав
студент групи ІА–24
Криворучек В.С.

Перевірив
викладач
Мягкий М.Ю.

Київ 2024

Зміст

Мета: Вивчення основних принципів побудови та проектування програмних систем з використанням схем прецедентів, діаграм класів та структури бази даних.

Теоретичні відомості

Діаграма прецедентів (Use Case Diagram) — це тип діаграми, що використовується для візуалізації функціональних вимог до системи у вигляді прецедентів використання (сценаріїв). Вона допомагає зрозуміти, як користувачі взаємодіють із системою, що дозволяє визначити функціональність, яка повинна бути реалізована. Діаграма прецедентів є частиною уніфікованої мови моделювання (UML) і широко використовується для проектування та аналізу систем у процесі розробки програмного забезпечення.

Основні елементи діаграми прецедентів

1. Актори (Actors):
    - Актор — це будь-який зовнішній елемент, що взаємодіє із системою. Зазвичай це користувачі, але може бути й інша система чи зовнішній процес.
    - Актори поділяються на основних (тих, хто ініціює взаємодію) і допоміжних (тих, хто реагує на дії).
    - Приклад акторів: користувач, адміністратор, система обробки платежів тощо.
    - На діаграмі актори зображуються у вигляді фігурок людей або простих символів з підписами.
2. Прецеденти (Use Cases):
    - Прецедент — це конкретна функціональність або завдання, яке виконується системою на запит актора.
    - Прецеденти представляються у вигляді овалів із назвою усередині. Назва повинна описувати дію, наприклад: "Зареєструватися", "Зробити замовлення", "Згенерувати звіт".

- Прецедент є описом послідовності дій, що виконується для досягнення певної мети користувача.

3. Зв’язки (Relationships):
    - Асоціація (Association) — зв’язок між актором і прецедентом, що показує, як актор взаємодіє з системою.
    - Залежність (Include) — коли один прецедент завжди виконує інший як частину своєї функції. Це допомагає уникати дублювання функціоналу. Вказується за допомогою стрілки з написом <<include>>.
    - Розширення (Extend) — коли прецедент може додатково виконати інший прецедент за певних умов. Це корисно для опису необов’язкових або варіативних дій. Вказується за допомогою стрілки з написом <<extend>>.
    - Загальний батьківський елемент (Generalization) — коли один актор або прецедент є узагальненням іншого (аналогія з успадкуванням у класах).

Основне призначення діаграми прецедентів

- Визначення, хто саме буде взаємодіяти із системою (всі типи користувачів або інші системи).
- Визначення основних функцій системи з точки зору користувачів.
- Допомога у визначенні функціональних вимог.
- Візуалізація взаємодії між користувачами та системою у простій і зрозумілій формі.
- Служить основою для створення більш деталізованих діаграм і специфікацій для подальшої розробки.

Діаграма класів (Class Diagram) — це тип діаграми в **уніфікованій мові моделювання (UML)**, який використовується для моделювання статичної структури системи. Вона відображає класи, їх атрибути, методи (операції), а також зв’язки між класами, такі як асоціації, агрегації, композиції та успадкування. Діаграма класів є важливим інструментом у процесі

проєктування та розробки програмного забезпечення, оскільки вона дозволяє побудувати загальну модель об'єктів, які утворюють систему.

Основні елементи діаграми класів

1. Клас (Class):
    - Клас представляє об'єктну структуру та описує властивості (атрибути) і поведінку (методи), які мають об'єкти цього класу.
    - Клас зображається у вигляді прямокутника, розділеного на три частини: ім'я класу (у верхній частині), атрибути (у середній частині) і методи (у нижній частині).
    - Приклад: клас "Користувач" може мати атрибути "ім'я" і "електронна пошта" та методи "увійти в систему" і "вийти".
2. Атрибути (Attributes):
    - Це характеристики або властивості класу, які описують стан об'єктів цього класу.
    - Наприклад, клас "Автомобіль" може мати атрибути "марка", "модель" і "рік випуску".
3. Методи (Operations):
    - Це функції або дії, які можуть виконувати об'єкти класу.
    - Наприклад, метод "запустити двигун" для класу "Автомобіль".
4. Зв'язки між класами (Relationships):
    - Асоціація (Association): показує, як один клас взаємодіє з іншим. Зазвичай це лінія між двома класами, яка може мати додаткові позначення, наприклад, множинність (один-до-одного, один-до-багатьох тощо).
    - Агрегація (Aggregation): показує відношення "частина-ціле", коли один клас є частиною іншого, але може існувати незалежно. Це позначається порожнім ромбом біля "цілого".
    - Композиція (Composition): це сильніша форма агрегації, яка означає, що "частина" не може існувати без "цілого". Позначається зафарбованим ромбом.

- Успадкування (Inheritance): зв'язок, що вказує на відношення "батьківський-клас" і "дочірній-клас". Клас-спадкоємець успадковує атрибути та методи від базового класу. Це позначається стрілкою з трикутником.

5. Інтерфейси (Interfaces):
    - Інтерфейс описує набір методів, які клас повинен реалізувати. Інтерфейси використовуються для створення контрактів у проєктуванні, забезпечуючи абстрактність і гнучкість.

Основне призначення діаграми класів

- Моделювання структури системи: Діаграма класів показує, як організовані та взаємодіють об'єкти в системі.
- Розробка програмного забезпечення: Вона є основою для створення коду, оскільки визначає структуру класів і їхні взаємозв'язки.
- Спрощення розуміння системи: Діаграма дозволяє наочно зрозуміти, як різні частини системи пов'язані між собою.
- Аналіз об'єктів і їх відносин: Вона дає змогу визначити атрибути і методи класів, побачити їх зв'язки, що дозволяє ефективніше розробляти функціонал системи.

База даних (БД) — це організований набір даних, що зберігаються та обробляються комп'ютерною системою. Вона забезпечує спосіб організації, збереження та керування великими обсягами даних, а також дозволяє здійснювати швидкий пошук, оновлення, вилучення та аналіз інформації. Бази даних зазвичай використовуються для зберігання структурованої інформації, такої як записи клієнтів, фінансові дані, інвентар товарів, статистика користувачів тощо.

Основні елементи структури бази даних:

1. Таблиці (Tables):
    - Таблиця — це основний елемент структури бази даних, яка складається з рядків (записів) і стовпців (полів).
    - Кожен рядок (або запис) представляє одну одиницю даних.

- Кожен стовпець (або поле) містить атрибути цих даних.
- Наприклад, таблиця "Клієнти" може містити стовпці "Ім'я", "Прізвище", "Електронна пошта" і "Телефон".

2. Рядки (Записи) і стовпці (Поля):
    - Рядок (Record) містить повний набір даних для однієї сутності або об’єкта (наприклад, один клієнт).
    - Стовпець (Field) представляє атрибут даних (наприклад, "Ім'я клієнта").
3. Первинний ключ (Primary Key):
    - Унікальний ідентифікатор для кожного запису в таблиці.
    - Він гарантує, що жоден запис у таблиці не буде повторюватися.
    - Наприклад, у таблиці "Клієнти" первинним ключем може бути "Ідентифікатор клієнта" (унікальне значення для кожного клієнта).
4. Зовнішній ключ (Foreign Key):
    - Поле або набір полів, яке встановлює зв’язок між таблицями.
    - Він дозволяє пов'язувати дані між різними таблицями, що забезпечує цілісність даних.
    - Наприклад, якщо у таблиці "Замовлення" є поле "Ідентифікатор клієнта", яке посилається на таблицю "Клієнти", це буде зовнішнім ключем.
5. Зв’язки (Relationships):
    - Визначають, як таблиці в базі даних взаємодіють між собою.
    - Основні типи зв’язків: Один-до-одного (1:1): кожен запис однієї таблиці відповідає тільки одному запису іншої таблиці; Один-до-багатьох (1:M): один запис у першій таблиці може мати кілька відповідних записів у другій таблиці; Багато-до-багатьох (M:M): кожен запис першої таблиці може відповідати багатьом записам другої таблиці і навпаки. Для реалізації цього типу зв’язку зазвичай використовується проміжна таблиця.
6. Індекси (Indexes):

- Індекси використовуються для прискорення пошуку та сортування даних у таблицях.
- Вони створюють спеціальну структуру, яка дозволяє швидше отримувати дані.

7. Запити (Queries):
    - Запити дозволяють отримувати, змінювати, додавати або видаляти дані в базі даних за допомогою спеціальних команд.
    - Запити часто створюються за допомогою мови SQL (Structured Query Language).
8. Вид (View):
    - Віртуальна таблиця, створена на основі результатів запиту.
    - Види дозволяють відображати певні дані з однієї або кількох таблиць у зручному для користувача вигляді, не зберігаючи їх фізично.
9. Схема (Schema):
    - Структурний опис бази даних, який визначає, як дані організовані.
    - Схема включає таблиці, їх поля, типи даних, зв’язки між таблицями та інші об'єкти бази даних.

Призначення баз даних:

- Зберігання даних: БД дозволяють ефективно зберігати дані та забезпечують їх структурованість.
- Обробка даних: Вони надають інструменти для обробки, фільтрації та аналізу даних.
- Забезпечення доступу: Користувачі можуть отримувати доступ до необхідної інформації швидко й безпечно.
- Цілісність і послідовність: Забезпечується контроль над дублюванням даних і забезпечується їхня правильність та актуальність.

Бази даних є невід'ємною частиною багатьох програмних систем, що допомагає зберігати й обробляти великі обсяги даних з ефективним керуванням і доступом до них.

Шаблон Репозиторію (Repository Pattern) — це шаблон проєктування, який забезпечує абстрактний шар для доступу до даних у додатках. Він дозволяє ізолювати бізнес-логіку від логіки доступу до даних, забезпечуючи чітку структуру коду. Завдяки цьому шаблону можна працювати з джерелом даних так, ніби це проста колекція об'єктів, не переймаючись тим, як саме ці дані зберігаються або отримуються.

Основні характеристики шаблону Репозиторію:

1. Інкапсуляція логіки доступу до даних:
    - Репозиторій виступає як прошарок між бізнес-логікою та джерелом даних, який інкапсулює логіку взаємодії з базою даних або іншим джерелом даних.
    - Це дозволяє змінювати або оновлювати спосіб збереження даних без змін у бізнес-логіці.
2. Методи CRUD:
    - Типовий репозиторій реалізує основні методи CRUD (Create, Read, Update, Delete) для роботи з об'єктами.
    - Наприклад, методи можуть включати Add(), Remove(), GetById(), Find() тощо.
3. Розділення відповідальності:
    - Репозиторій зосереджується виключно на взаємодії з даними, а бізнес-логіка залишається окремо.
    - Це забезпечує кращу підтримуваність коду та спрощує тестування.
4. Абстракція джерела даних:
    - Репозиторій приховує джерело даних. Незалежно від того, використовуємо ми базу даних, веб-службу або локальні файли, клієнт працює лише з методами репозиторію.

Переваги шаблону Репозиторію:

1. Зменшення дублювання коду: Логіка доступу до даних зосереджена в одному місці, що дозволяє уникнути дублювання.

2. Тестованість: Бізнес-логіку можна легко тестувати, оскільки репозиторій можна замінити мок-об'єктом (імітацією).
3. Гнучкість: Якщо зміниться джерело даних (наприклад, зміна з SQL на NoSQL базу), вам потрібно лише змінити реалізацію репозиторію, не змінюючи бізнес-логіку.

Вибір прецедентів та аналіз є важливим етапом у процесі моделювання системи, що дозволяє чітко визначити її функціональні можливості та взаємодії користувачів з системою. Прецеденти, або сценарії використання (use cases), визначають, як різні типи користувачів (актори) можуть взаємодіяти з системою через певні функції або дії.

Процес вибору та аналізу прецедентів:

1. Ідентифікація акторів:
    - Спочатку визначають, хто є користувачами або зовнішніми системами, які будуть взаємодіяти з вашою системою. Це можуть бути кінцеві користувачі, адміністративний персонал, інші програмні модулі тощо.
    - Актори можуть бути внутрішніми (користувачі, які взаємодіють безпосередньо з системою) або зовнішніми (системи або люди, які взаємодіють із системою через посередництво).
2. Визначення прецедентів (сценаріїв використання):
    - Для кожного актора визначають можливі дії або функції, які він може виконувати. Ці дії визначаються як прецеденти.
    - Наприклад, для актору "Користувач" в системі управління обліковими записами прецедентами можуть бути "Увійти в систему", "Змінити пароль", "Створити новий обліковий запис" тощо.
3. Визначення зв’язків між акторами і прецедентами:
    - Прецеденти пов’язуються з актором, що їх ініціює або використовує.

- Діаграма прецедентів візуально показує, які дії виконує кожен актор.

4. Аналіз прецедентів:
    - Кожен прецедент аналізується для визначення його взаємодій, кроків виконання та очікуваних результатів.
    - Це дозволяє зрозуміти, які функціональні вимоги має реалізовувати система та які можливі виклики виникатимуть під час їх реалізації.
5. Деталізація обраних прецедентів:
    - Після ідентифікації прецедентів необхідно деталізувати їх, описуючи кроки взаємодії, передумови виконання, очікувані результати та виняткові ситуації.
    - Для прикладу, прецедент "Вхід в систему" може містити кроки: введення імені користувача та пароля, перевірка даних у базі даних, повернення результату про успішний або невдалий вхід.
6. Пріоритезація:
    - Після вибору та аналізу прецедентів, їх пріоритизують відповідно до важливості, частоти використання або складності реалізації. Це дозволяє сфокусувати розробку на найважливіших функціях на ранніх етапах проекту.

Мета вибору та аналізу прецедентів:

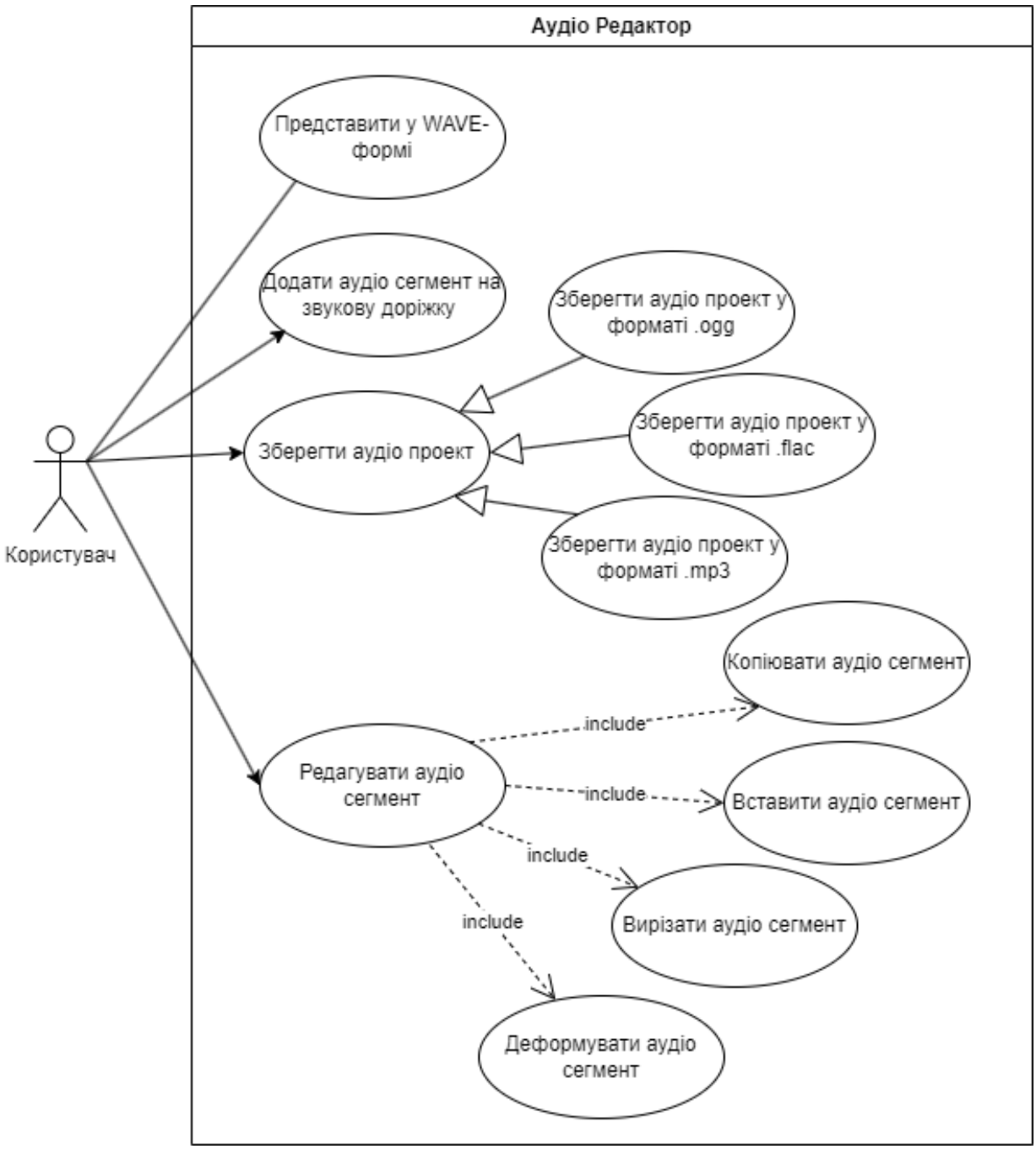
- Забезпечити розуміння функціональних вимог системи на високому рівні.
- Визначити сценарії, які описують очікувану поведінку системи для користувачів.
- Створити базу для подальшого проектування системи (діаграм класів, моделювання взаємодій).
- Полегшити комунікацію між розробниками, замовниками та користувачами системи.

Таким чином, вибір і аналіз прецедентів допомагає побудувати чітке уявлення про те, як система повинна функціонувати, і забезпечує структуру для її проектування та розробки.

Хід роботи

1. Ознайомитися з короткими теоретичними відомостями.
2. Проаналізуйте тему та намалюйте схему прецеденту, що відповідає обраній темі лабораторії.
3. Намалюйте діаграму класів для реалізованої частини системи.
4. Виберіть 3 прецеденти і напишіть на їх основі прецеденти.
5. Розробити основні класи і структуру системи баз даних.
6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.

Діаграма прецедентів



Прямокутною рамкою позначено систему.

Операції з сегментами аудіозапису.

- Копіювання
- Вставка
- Вирізання
- Деформація

Робота з декількома звуковими доріжками. Дана функція дозволяє користувачеві працювати з кількома звуковими доріжками.

Кодування в поширених форматах.

- ogg
- flac
- mp3

Можна відзначити, що є саме чотири головних сценаріїв використання: Представити у WAVE-формі, Додати аудіо сегмент на звукову доріжку, Зберегти аудіо проект, Редагувати аудіо сегмент. У свою чергу сценарій Зберегти аудіо проект, розширюють ще 3 сценарії: Зберегти аудіо проєкт у форматі .ogg, Зберегти аудіо проект у форматі .flac, Зберегти аудіо проєкт у форматі .mp3. Крім цього, ще можна побачити, що сценарій Редагувати аудіо сегмент містить в собі 4 різних сценаріїв редагування аудіо, а саме: Копіювати аудіо сегмент, Вставити аудіо сегмент, Вирізати аудіо сегмент, Деформувати аудіо сегмент.

Оберемо 3 основні прецеденти для аудіо редактора та опишемо їх:

1. Прецедент: "Додавання аудіо сегменту на звукову доріжку"

Опис: Користувач може додати новий аудіо сегмент на вибрану звукову доріжку в аудіо редакторі.

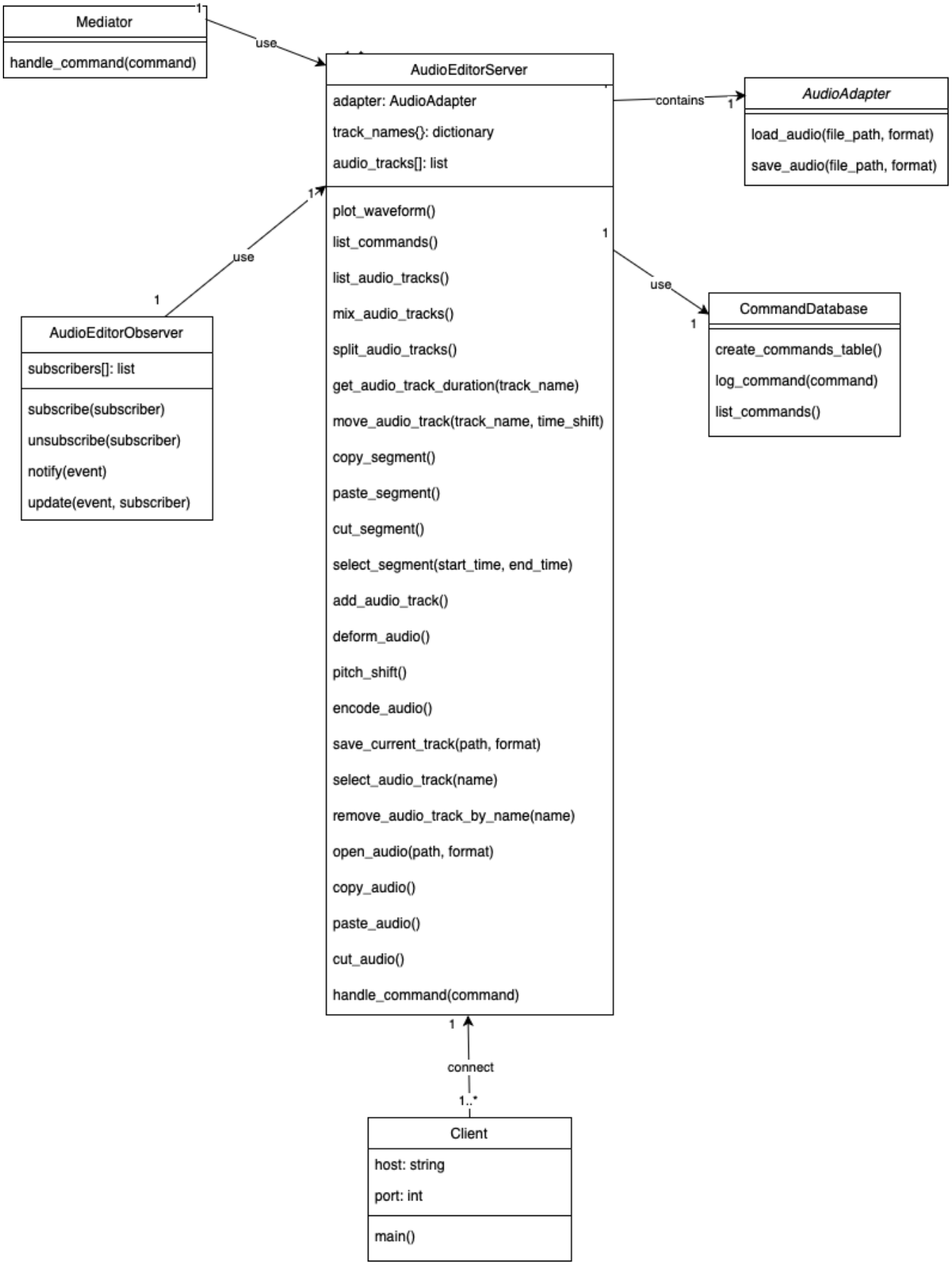
2. Прецедент: "Збереження аудіо проєкту у вказаному форматі"

Опис: Користувач може зберегти свій поточний аудіо проєкт у вказаному аудіоформаті (наприклад, ogg, flac, mp3).

3. Прецедент: "Редагування аудіо сегменту"

Опис: Користувач може вибрати аудіо сегмент та виконати на ньому операції редагування, такі як зміна гучності, зміна швидкості тощо.

Діаграма класів



Як можна побачити на діаграмі:

1. AudioEditorServer використовує AudioAdapter для завантаження та збереження аудіо. Це означає, що AudioEditorServer володіє екземпляром AudioAdapter та викликає його методи load_audio() і save_audio().
2. Mediator використовує AudioEditorServer для обробки команд від клієнтів. Це означає, що Mediator має посилання на AudioEditorServer та викликає його метод handle_command().
3. AudioEditorObserver повідомляє про події в класі AudioEditorServer та інших частинах програми. Це означає, що AudioEditorObserver може викликати методи класу AudioEditorServer для сповіщення про певні події.
4. CommandDatabase використовується AudioEditorServer для логування команд. Це означає, що AudioEditorServer може викликати методи класу CommandDatabase для реєстрації логів.

Структура бази даних

Для системи аудіо-редактора база даних буде використовуватися для збереження журналу команд. Для цього потрібно буде створити одну таблицю, у якій будуть Ось загальна структура бази даних:

Таблиця "commands":

Ця таблиця буде використовуватися для збереження історії команд, що виконувались в системі.

id: INT - Унікальний ідентифікатор команди

Стовпець id є ідентифікатором, отже для його створення було використано такі команди:

PRIMARY KEY – показник, що id - ідентифікатор

AUTOINCREMENT – показує, що при додаванні значень до таблиці, ідентифікатор автоматично буде збільшуватись.

command_text: TEXT - Текст команди, що виконувалась

timestamp: TIMESTAMP - Мітка часу виконання команди

При створенні стовпців command_text та timestamp було використано команду NOT NULL, що означає, що при додаванні значень у таблицю, ці стовпці не можуть бути пустими(обов'язково повинно бути значення). В іншому випадку дані до таблиці не додадуться.

Дана структура таблиці може бути використана для:

1. Аналізу дій і безпеки: Вона дозволяє відстежувати виконані команди. Це може бути корисно для загального аналізу паттернів використання системи та виявлення проблем.
2. Відновлення стану системи: Таблиця може служити для відновлення стану системи до певного моменту часу.
3. Журналу та аудиту дій: Вона фіксує дії, що виконуються в системі, що може бути корисним для аудиту дій та виявлення неполадок.

Висновок

У ході виконання першої лабораторної роботи на тему створення аудіоредактора ми ознайомилися з основними принципами побудови та проектування програмних систем, використовуючи схеми прецедентів, діаграми класів і структуру бази даних. Робота дозволила отримати теоретичне розуміння та практичний досвід у моделюванні ключових компонентів системи на основі обраного завдання.

На початку ми розглянули короткі теоретичні відомості, що дало змогу зрозуміти основні принципи побудови програмних систем, а також значення діаграм прецедентів, які описують взаємодію користувачів із системою. Після цього було створено схему прецедентів, яка відображала основні функціональні можливості аудіоредактора, такі як завантаження аудіо, редагування та експорт у різні формати.

Наступним етапом була побудова діаграми класів для реалізованої частини системи. Діаграма класів допомогла сформувати базову структуру системи, що включає основні класи, їх атрибути та методи, а також зв'язки між

ними. Це дало можливість чітко визначити відповідальність кожного компонента.

Було обрано три прецеденти, які описували базові сценарії взаємодії з аудіоредактором, а саме: завантаження аудіофайлу, редагування сегменту (копіювання, вирізання та вставка) і збереження результату в бажаному форматі. Кожен із прецедентів було детально описано, що дало змогу краще зрозуміти процеси, що відбуваються всередині системи.

Ми також розробили основні класи системи та структуру бази даних для зберігання інформації про аудіофайли та виконані операції. Це стало основою для подальшого використання шаблону Репозиторію, який забезпечує ефективну взаємодію з базою даних та ізоляцію рівня збереження даних від бізнес-логіки системи.

Таким чином, перша лабораторна робота забезпечила початкову структуризацію проєкту аудіоредактора, дала змогу закласти основу для подальшої реалізації функціоналу та дала чітке розуміння ключових етапів проектування програмної системи.