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
Технології розроблення програмного забезпечення
«Основи проектування.»

Тема: Редактор зображень

Виконала:

студентка групи ІА-32

Ейсмонт А.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Тема: Основи проектування.

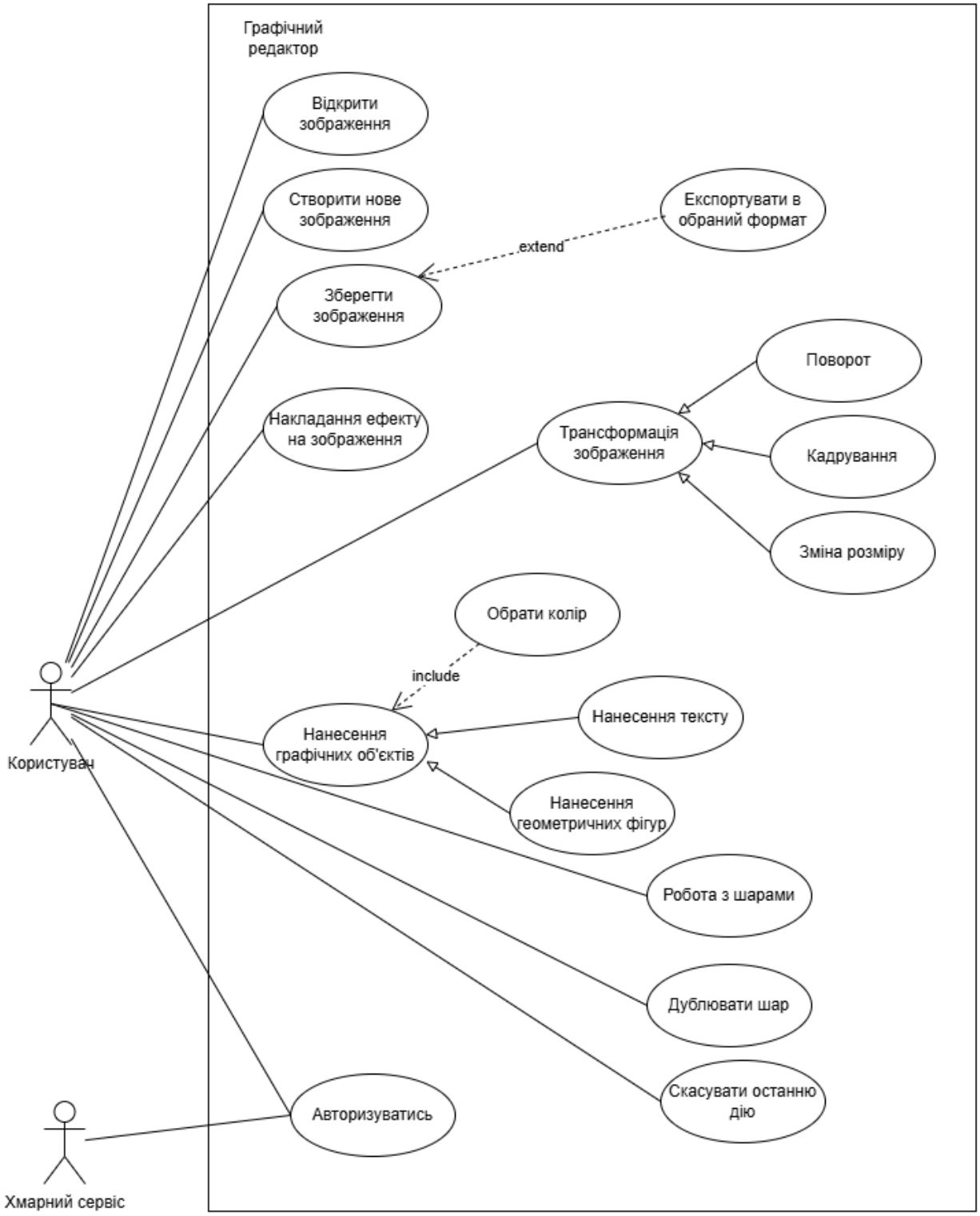
Мета: Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

Теоретичні відомості:

UML (Unified Modeling Language) — універсальна мова візуального моделювання для специфікації, візуалізації, проєктування й документування програмних систем та бізнес-процесів. UML дозволяє створювати концептуальні, логічні та фізичні моделі складних систем на різних рівнях абстракції. В ООАП (об’єктно-орієнтований аналіз і проєктування) модель системи складається з різних уявлень, які відображають її поведінку або структуру. Діаграма варіантів використання (Use Case Diagram) – це тип діаграми UML, що описує функціональність системи з точки зору її користувачів (акторів) і взаємодії між ними та системою. Вона показує, які дії (варіанти використання) можуть виконуватися користувачами, але не вдається у внутрішні механізми їх реалізації. Сценарії варіантів використання (Use Case Scenarios) – це текстовий опис варіантів використання, де детально викладається, як система повинна реагувати на дії користувачів у кожній конкретній ситуації. Включає в себе основний потік подій та альтернативні шляхи розвитку сценарію. Діаграма класів (Class Diagram) – це структура, яка моделює класи системи, їх властивості, методи, а також зв'язки між ними. Класи представляють основні об'єкти системи, які мають атрибути та операції, а також відображають взаємодію між різними компонентами. Концептуальна модель системи – це абстрактне представлення об'єктів та зв'язків між ними, що відображає ключові аспекти системи з точки зору бізнесу або 4 предметної області. Вона описує основні компоненти, їх взаємодію та структуру, але не деталізує технічну реалізацію. Ці діаграми дозволяють аналізувати вимоги до системи та планувати її розробку

Хід роботи:

1. **Діаграма використання**



Користувач запускає графічний редактор та може:

- Відкривати та створювати нові зображення.
- Редагувати зображення (трансформація, малювання, накладання ефектів).
- Керувати шарами (створювати, дублювати, видаляти).

- Експортувати результат у популярні формати (JPG, PNG, BMP, TIFF, GIF).
- Зберігати/Завантажувати проєкт у хмару (через авторизацію).

Прецедент 1: Відкрити зображення

Передумови: Редактор запущений; користувач має доступ до файлової системи; файл зображення існує.

Постумови: Зображення завантажено у робочу область (canvas); створено базовий шар із цим зображенням. У разі помилки — показано повідомлення, робоча область залишається без змін.

Сторони взаємодії: Користувач, Система графічного редактора.

Короткий опис: Завантаження графічного файлу з диска для подальшого редагування.

Основний потік подій:

1. Користувач обирає дію «Відкрити зображення».
2. Система відкриває стандартний діалог вибору файлу.
3. Користувач обирає файл підтримуваного формату та підтверджує вибір.
4. Система зчитує дані файлу, створює новий проєкт (або шар) та візуалізує зображення в робочій області.
5. Система відображає параметри зображення (розмір, колірна модель).

Винятки:

Виняток №1: Невідомий формат файлу / файл пошкоджений — система показує повідомлення про помилку декодування та пропонує обрати інший файл.

Виняток №2: Файл занадто великий (нестача оперативної пам’яті) — система попереджає про обмеження ресурсів і пропонує відкрити файл у зменшеному масштабі або скасувати дію.

Виняток №3: Відмова у доступі (файл захищений/заблокований системою) — система просить надати права доступу.

Прецедент 2: Нанесення графічних об'єктів (Малювання)

Передумови: Проєкт відкритий; існує активний шар для малювання; обрано інструмент (Пензлик, Текст або Фігура).

Постумови: На активному шарі з’являються нові пікселі або векторні об'єкти відповідно до дій користувача.

Сторони взаємодії: Користувач, Система.

Короткий опис: Користувач обирає параметри (колір, товщина) та наносить елементи на полотно.

Основний потік подій:

1. Користувач обирає інструмент малювання (наприклад, "Пензлик" або "Геометрична фігура").
2. Користувач обирає колір (*include "Обрати колір"*) та налаштовує параметри (товщина лінії, прозорість).
3. Користувач взаємодіє з робочою областю (натискає, тягне курсор).
4. Система в реальному часі відмальовує об'єкт на активному шарі.
5. Користувач завершує дію (відпускає кнопку миші), система фіксує зміни в історії дій (для можливості Undo).

Винятки:

Виняток №1: Шар заблокований або прихований — система показує індикатор заборони редагування і не виконує малювання.

Виняток №2: Спроба малювання за межами полотна — система ігнорує координати поза межами або автоматично розширює полотно (залежно від налаштувань).

Прецедент 3: Трансформація зображення

Передумови: У проєкті є активний об'єкт або виділена область; інструмент трансформації доступний.

Постумови: Геометрія зображення/шару змінена (повернута, обрізана, змінена в розмірі).

Сторони взаємодії: Користувач, Система.

Короткий опис: Застосування геометричних перетворень до всього зображення або окремого шару.

Основний потік подій:

1. Користувач обирає тип трансформації (Поворот, Кадрування, Зміна розміру).
2. Система відображає елементи керування (рамку кадрування, поле введення кута повороту або нових розмірів).
3. Користувач задає нові параметри (перетягує рамку або вводить значення вручну).
4. Користувач підтверджує дію («Застосувати»).

5. Система виконує перерахунок пікселів (інтерполяцію) та оновлює відображення.

Винятки:

Виняток №1: Некоректні параметри (наприклад, від'ємний розмір або нульова ширина) — система блокує кнопку «Застосувати» та підсвічує помилку.

Виняток №2: Область кадрування порожня — система попереджає, що результат буде пустим зображенням.

Прецедент 4: Експорт зображення

Передумови: Користувач завершив редагування; проєкт містить дані для збереження.

Постумови: На диску створено графічний файл у вибраному форматі (JPG, PNG тощо), який містить зведене зображення всіх видимих шарів.

Сторони взаємодії: Користувач, Система.

Короткий опис: Конвертація внутрішнього формату проєкту у загальнодоступний графічний файл.

Основний потік подій:

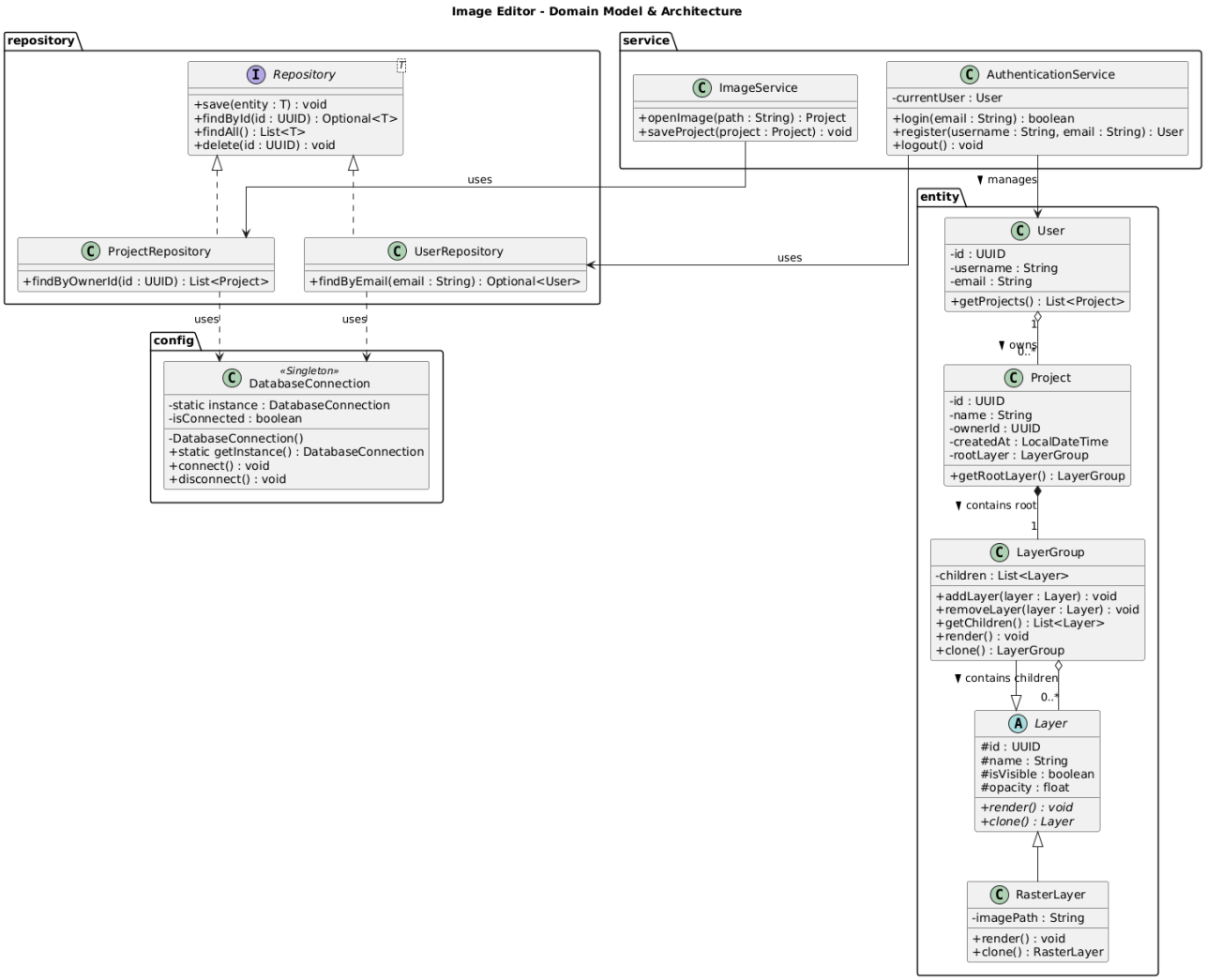
1. Користувач обирає опцію «Експортувати» (як розширення дії "Зберегти").
2. Система пропонує обрати формат (один із 5 підтримуваних), ім'я файлу та шлях.
3. Система пропонує специфічні налаштування формату (якість для JPG, прозорість для PNG).
4. Користувач підтверджує експорт.
5. Система об'єднує шари (flattening), кодує зображення та записує файл на диск.
6. Система повідомляє про успішне завершення.

Винятки:

Виняток №1: Недостатньо місця на диску — система зупиняє запис і повідомляє про помилку.

Виняток №2: Помилка запису (файл зайнятий іншим процесом) — система пропонує зберегти під іншим ім'ям.

2. Діаграма класів



Repository Pattern

- `Repository<T>`: базовий інтерфейс, що визначає методи `save(T entity)`, `findById(UUID id)`, `findAll()`, `delete(UUID id)`.

Моделі (предметна область графічного редактора)

- **User** `{ id, username, email, projects }` — сутність користувача системи.
- **Project** `{ id, name, ownerId, createdAt, rootLayer }` — головний клас проєкту, що містить посилання на кореневу структуру шарів.
- **Layer** (Abstract) `{ id, name, isVisible, opacity }` — базовий абстрактний клас для всіх типів шарів. Реалізує патерн **Prototype** (метод `clone`).
- **RasterLayer** `{ imagePath }` — конкретна реалізація шару, що зберігає шлях до растрового зображення.
- **LayerGroup** `{ children }` — контейнер для інших шарів. Реалізує патерн **Composite** (може містити як звичайні шари, так і інші групи).

Репозиторії

- **UserRepository** (`findByEmail`) — для доступу до даних користувачів.
- **ProjectRepository** (`findByOwnerId`) — для пошуку проєктів конкретного користувача.
- Усі репозиторії імплементують базовий інтерфейс `Repository<T>`.

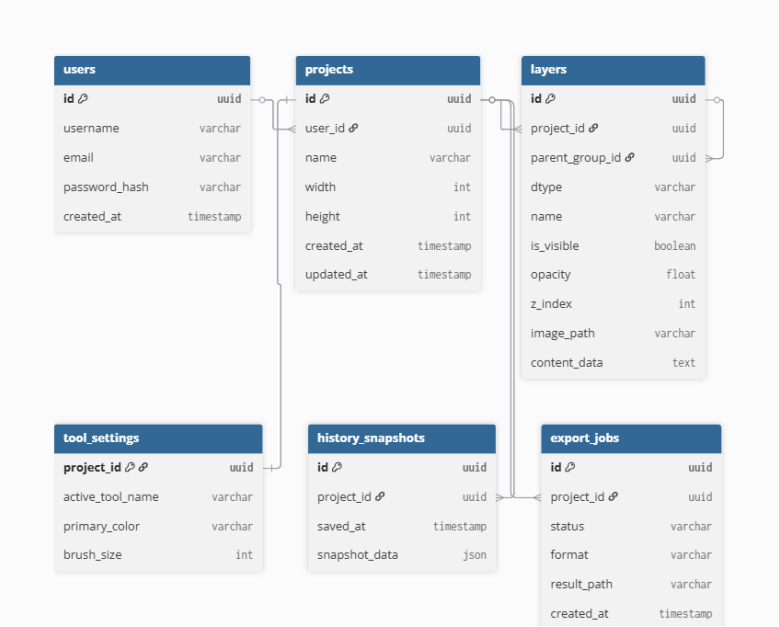
Зв’язки між класами

- *User 1 — 0.. Project**: (Асоціація) Один користувач може бути власником багатьох проєктів.
- **Project 1 — 1 LayerGroup**: (Композиція) Проєкт обов'язково містить одну кореневу групу шарів (`rootLayer`).
- *LayerGroup 1 — 0.. Layer**: (Агрегація/Композиція) Група шарів містить список дочірніх елементів, якими можуть бути як растрові шари, так і вкладені групи.
- **Layer <|— RasterLayer, LayerGroup**: (Успадкування) `RasterLayer` та `LayerGroup` є нащадками базового класу `Layer`.

Сервіси та Utility-класи

- **AuthenticationService**: `login(email)`, `register(username, email)`, `logout()`. Відповідає за бізнес-логіку автентифікації та використовує `UserRepository`.
- **ImageService**: `openImage(path)`, `saveProject(project)`. Відповідає за високорівневі операції з файлами та проєктами.
- **DatabaseConnection**: Клас, реалізований за патерном **Singleton**, що відповідає за єдине підключення до бази даних та ізолює технічні деталі з'єднання.

3. Зображення структури бази даних



4. **Коди класів**

DatabaseConnection.java

```
package ia32.eismont.config;

public class DatabaseConnection {
    private static DatabaseConnection instance;
    private boolean isConnected;



    private DatabaseConnection() {
        this.isConnected = false;
    }

    public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }

    public void connect() {
        if (!isConnected) {
            System.out.println(">>> [DB] Підключення до бази даних встановлено.");
            isConnected = true;
        }
    }

    public void disconnect() {
        if (isConnected) {
            System.out.println(">>> [DB] З'єднання розірвано.");
```

```
                isConnected = false;
            }
        }
    }
```

Layer.java

```
    package ia32.eismont.entity;

    import java.util.UUID;

    public abstract class Layer implements Cloneable {
        protected UUID id;
        protected String name;
        protected boolean isVisible;

        public Layer(String name) {
            this.id = UUID.randomUUID();
            this.name = name;
            this.isVisible = true;
        }

        public String getName() { return name; }
        public void setName(String name) { this.name = name; }

        public boolean isVisible() { return isVisible; }
        public void setVisible(boolean visible) { isVisible = visible; }

        public abstract void render();

        @Override
        public abstract Layer clone();
    }
```

LayerGroup.java

```
package ia32.eismont.entity;

import java.util.ArrayList;
import java.util.List;

public class LayerGroup extends Layer {
   private List<Layer> children;

   public LayerGroup(String name) {
      super(name);
      this.children = new ArrayList<>();
   }

   public void addLayer(Layer layer) {
      children.add(layer);
   }

   public void removeLayer(Layer layer) {
      children.remove(layer);
   }

   public List<Layer> getChildren() { return children; }

   @Override
   public void render() {
      System.out.println("Group: " + name);
      for (Layer child : children) {
         child.render();
      }
   }
```

```
    @Override
    public LayerGroup clone() {
        LayerGroup copy = new LayerGroup(this.name + "_copy");
        for (Layer child : children) {
            copy.addLayer(child.clone());
        }
        return copy;
    }
}
```

Project.java

```
package ia32.eismont.entity;

import java.time.LocalDateTime;
import java.util.UUID;

public class Project {
    private UUID id;
    private String name;
    private UUID ownerId;
    private LocalDateTime createdAt;

    private LayerGroup rootLayer;

    public Project(String name, User owner) {
        this.id = UUID.randomUUID();
        this.name = name;
        this.ownerId = owner.getId();
        this.createdAt = LocalDateTime.now();
        this.rootLayer = new LayerGroup("Root");
    }
```

```
    public UUID getId() { return id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public UUID getOwnerId() { return ownerId; }

    public LayerGroup getRootLayer() { return rootLayer; }
      public void setRootLayer(LayerGroup rootLayer) { this.rootLayer = rootLayer; }
    }
```

ProjectState.java

```
    package ia32.eismont.entity;

    import java.time.LocalDateTime;

    public class ProjectState {
      private final LayerGroup rootLayerState;
      private final LocalDateTime savedAt;

      public ProjectState(LayerGroup rootLayer) {
        this.rootLayerState = rootLayer.clone();
        this.savedAt = LocalDateTime.now();
      }

      public LayerGroup getState() {
        return rootLayerState;
      }
    }
```

User.java

```
package ia32.eismont.entity;

import java.util.UUID;
import java.util.List;
import java.util.ArrayList;

public class User {
    private UUID id;
    private String username;
    private String email;
    private List<Project> projects;

    public User(String username, String email) {
        this.id = UUID.randomUUID();
        this.username = username;
        this.email = email;
        this.projects = new ArrayList<>();
    }

    public UUID getId() { return id; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public List<Project> getProjects() { return projects; }
}
```

ProjectRepository.java

```
package ia32.eismont.repository;

import ia32.eismont.entity.Project;
import java.util.*;
import java.util.stream.Collectors;

public class ProjectRepository implements Repository<Project> {
   private Map<UUID, Project> database = new HashMap<>();

   @Override
   public void save(Project entity) {
      database.put(entity.getId(), entity);
      System.out.println("Project saved: " + entity.getName());
   }

   @Override
   public Optional<Project> findById(UUID id) {
      return Optional.ofNullable(database.get(id));
   }

   @Override
   public List<Project> findAll() {
      return new ArrayList<>(database.values());
   }

   @Override
   public void delete(UUID id) {
      database.remove(id);
   }
```

```
    public List<Project> findByOwnerId(UUID ownerId) {
        return database.values().stream()
                .filter(project -> project.getOwnerId().equals(ownerId))
                .collect(Collectors.toList());
    }
}
```

Repository.java

```
package ia32.eismont.repository;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

public interface Repository<T> {
    void save(T entity);
    Optional<T> findById(UUID id);
    List<T> findAll();
    void delete(UUID id);
}
```

UserRepository.java

```
package ia32.eismont.repository;

import ia32.eismont.entity.User;
import java.util.*;

public class UserRepository implements Repository<User> {
    private Map<UUID, User> database = new HashMap<>();

    @Override
    public void save(User entity) {
```

```
            database.put(entity.getId(), entity);
            System.out.println("User saved to DB: " + entity.getUsername());
        }

        @Override
        public Optional<User> findById(UUID id) {
            return Optional.ofNullable(database.get(id));
        }

        @Override
        public List<User> findAll() {
            return new ArrayList<>(database.values());
        }

        @Override
        public void delete(UUID id) {
            database.remove(id);
            System.out.println("User deleted: " + id);
        }

        public Optional<User> findByEmail(String email) {
            return database.values().stream()
                    .filter(user -> user.getEmail().equals(email))
                    .findFirst();
        }
    }
```

Authentification.java

```
    package ia32.eismont.service;

    import ia32.eismont.entity.User;
    import ia32.eismont.repository.UserRepository;
```

```
import java.util.Optional;

public class AuthenticationService {
    private UserRepository userRepository;
    private User currentUser;
    public AuthenticationService() {
        this.userRepository = new UserRepository();
    }

    public User register(String username, String email) {
        User newUser = new User(username, email);
        userRepository.save(newUser);
        return newUser;
    }

    public boolean login(String email) {
        Optional<User> user = userRepository.findByEmail(email);

        if (user.isPresent()) {
            this.currentUser = user.get();
              System.out.println("Welcome back, " + currentUser.getUsername() +
"!");
            return true;
        } else {
            System.out.println("Login failed: User not found.");
            return false;
        }
    }

    public User getCurrentUser() {
```

```
        return currentUser;
    }

    public void logout() {
                System.out.println("User " + (currentUser != null ?
currentUser.getUsername() : "") + " logged out.");
        this.currentUser = null;
    }
}
```

ImageFileUtils.java

```
package ia32.eismont.utils;

import java.io.File;

public class ImageFileUtils {

    public static boolean fileExists(String path) {
        File file = new File(path);
        return file.exists();
    }

    public static String readImageBytes(String path) {
        if (fileExists(path)) {
            System.out.println("--- Reading bytes from: " + path);
            return "HEX_DATA_OF_IMAGE"; // Повертаємо фейкові дані
        } else {
            System.out.println("--- Error: File not found: " + path);
            return null;
        }
    }
```

```
    public static void saveImage(String path, String format) {
        System.out.println("--- Saving image to [" + path + "] in format [" + format
+ "]");
    }
}
```

5. Питання до лабораторної роботи:

1. Що таке UML?

UML — це уніфікована мова візуального моделювання, призначена для специфікації, візуалізації, проєктування та документування програмних систем.

2. Що таке діаграма класів UML?

Діаграма класів UML — це структурна діаграма, що моделює класи системи, їхні атрибути, методи, а також статичні зв'язки між ними.

3. Які діаграми UML називають канонічними?

Канонічними діаграмами називають основний набір із дев'яти типів діаграм UML (прецедентів, класів, об'єктів, послідовності, кооперації, станів, діяльності, компонентів, розгортання), що складають ядро мови.

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання — це тип діаграми, що описує функціональність системи з точки зору взаємодії зовнішніх користувачів (акторів) із системою.

5. Що таке варіант використання?

Варіант використання (прецедент) — це специфікація послідовності дій, які система може виконувати у відповідь на дії актора для досягнення певної мети

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі використання можуть бути відображені відношення асоціації (між актором і прецедентом), включення (include), розширення (extend) та узагальнення.

7. Що таке сценарій?

Сценарій — це текстовий опис варіантів використання, який детально викладає основний потік подій та альтернативні шляхи реакції системи на дії користувача.

8. Що таке діаграма класів?

Діаграма класів — це графічне представлення статичної структури моделі, яке показує типи об'єктів системи (класи) та різноманітні відношення між ними.

9. Які зв’язки між класами ви знаєте?

Між класами існують такі основні типи зв'язків: асоціація, агрегація, композиція, узагальнення (спадкування), реалізація та залежність.

10. Чим відрізняється композиція від агрегації?

Композиція відрізняється від агрегації жорсткістю зв'язку: при композиції "частина" не може існувати без "цілого" (знищується разом з ним), тоді як при агрегації частини є незалежними.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на

діаграмах класів?

На діаграмах класів агрегація позначається лінією з порожнім ромбом на кінці, а композиція — лінією із зафарбованим чорним ромбом

12. Що являють собою нормальні форми баз даних?

Нормальні форми баз даних — це набір правил структурування даних, дотримання яких дозволяє усунути надлишковість інформації та запобігти аномаліям при вставці, оновленні чи видаленні даних.

13. Що таке фізична модель бази даних? Логічна?

Логічна модель описує структуру даних у термінах сутностей і зв'язків незалежно від СУБД, а фізична модель визначає конкретну реалізацію цієї структури (таблиці, типи даних) для обраної системи.

14. Який взаємозв'язок між таблицями БД та програмними класами?

Взаємозв'язок полягає в тому, що програмні класи зазвичай відображаються у таблиці бази даних, де екземпляри класу стають рядками, а атрибути — колонками таблиці.

Висновок: У результаті виконання лабораторної роботи було засвоєно принципи побудови UML-діаграм, зокрема діаграм варіантів використання та діаграм класів. Розроблені сценарії та побудовані моделі дали змогу краще зрозуміти предметну область і структуру проєктованої системи.