Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №2
Технології розроблення програмного забезпечення
«Основи проектування»

«System activity monitor»

Виконав
студент групи ІА–32:
Феклістов Д.С.

Київ 2025

Зміст

Теоретичні відомості

UML є універсальною мовою графічного моделювання, яка призначена для опису, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів і різних типів систем. UML забезпечує строгий і потужний інструментарій для створення концептуальних, логічних і графічних моделей складних систем різного призначення. Вона об'єднала в собі напрацювання й переваги методів програмної інженерії, які успішно застосовувалися для моделювання масштабних і складних систем упродовж останніх років.

Діаграма – це графічне подання елементів моделі у вигляді пов'язаного графа, вершини та ребра якого мають визначене смислове навантаження. Основним засобом побудови моделей на основі UML є нотація стандартних діаграм.

Діаграма варіантів використання – це UML-діаграма, яка дозволяє у графічній формі подати вимоги до майбутньої системи. Вона виступає початковою концептуальною моделлю проєктованої системи та не деталізує її внутрішню структуру. Такі діаграми є вихідним пунктом у процесі збору вимог до програмного забезпечення та його реалізації.

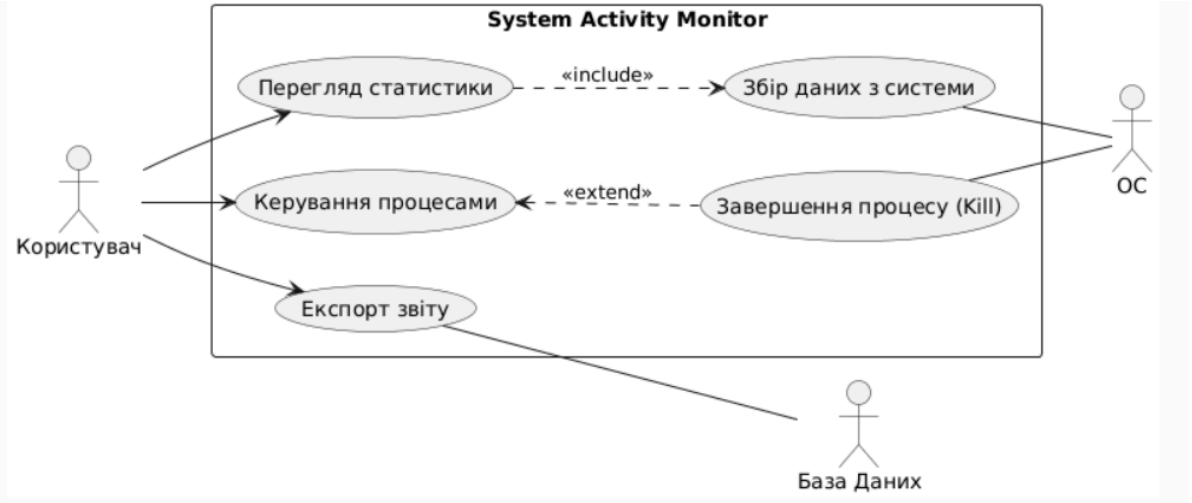
Сценарії використання – це текстові описи процесів взаємодії користувачів із системою. Вони подаються як формалізовані покрокові інструкції, які описують послідовність дій для досягнення певної мети. Такі сценарії однозначно визначають очікуваний результат.

Діаграми класів застосовуються в моделюванні програмних систем найчастіше. Вони належать до статичних моделей і відображають структуру системи з точки зору проєктування, показуючи класи, інтерфейси та взаємозв'язки між ними. При цьому діаграма класів не описує динамічної поведінки об'єктів.

Логічна модель бази даних – це структура, що містить таблиці, уявлення, індекси та інші логічні об'єкти бази даних, які забезпечують програмування й подальше використання БД.

Хід роботи

Діаграма варіантів використання



Опис: Діаграма відображає основні функції системи **System Activity Monitor**, включаючи **Моніторинг**, **Керування процесами** та **Генерацію звітів**. Головним актором є **Користувач** (Адміністратор), який взаємодіє із зовнішніми системами: **ОС** (для отримання даних і команд kill) та **Базою Даних** (для збереження та експорту).

Сценарій 1 - Перегляд статистики системи

Розділ	**Опис**
Актори	Користувач, ОС (Операційна система).
Основний потік	1. Користувач відкриває вкладку "Моніторинг". 2. Система через OsAdapter надсилає запит до ОС для отримання показників CPU, RAM та Disk. 3. ОС повертає поточні метрики навантаження. 4. Система відображає графіки та числові показники в інтерфейсі.
Винятки	Якщо Система не може отримати дані від ОС (відсутність прав) → Вивести повідомлення "Немає доступу до системних метрик".

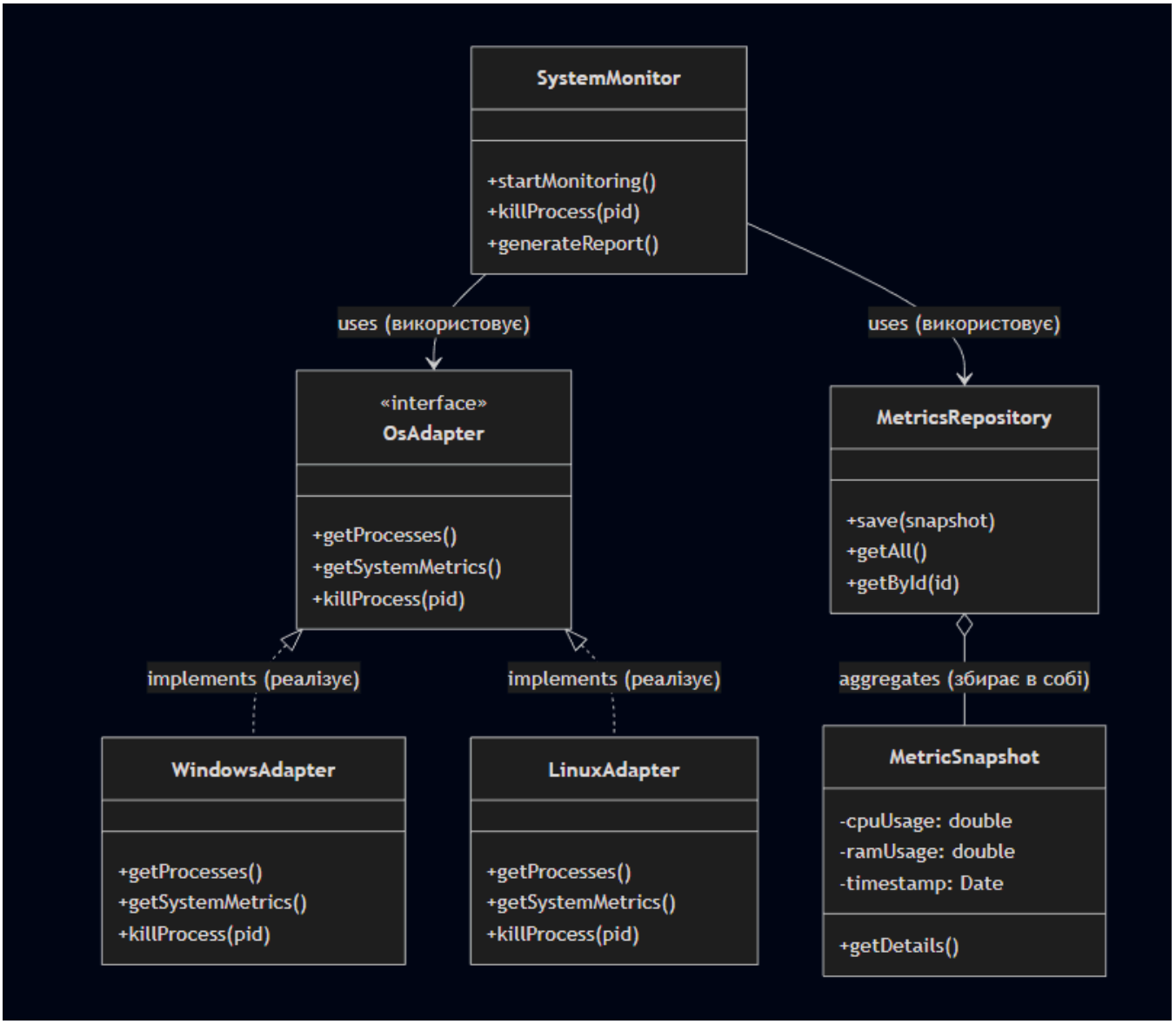
Сценарій 2 - Завершення процесу (Kill)

Розділ	Опис
Актори	Користувач, ОС (Операційна система).
Основний потік	1. Користувач обирає процес (за PID) зі списку та натискає "Завершити". 2. Система запитує підтвердження. 3. Користувач підтверджує. 4. Система через OsAdapter надсилає команду killProcess(pid) до ОС. 5. Система оновлює список процесів.
Винятки	ОС відхиляє команду, бо процес системний → Система показує повідомлення: "Відмовлено у доступі. Процес захищений.".

Сценарій 3 - Експорт звіту про активність

Розділ	Опис
Актори	Користувач, База даних (через MetricsRepository).
Основний потік	1. Користувач обирає опцію "Експорт/Звіт". 2. Користувач вказує період (наприклад, "Остання година"). 3. Система викликає метод MetricsRepository.getAll() для отримання історичних даних. 4. Система формує звітний документ (PDF/CSV). 5. Система зберігає файл і повідомляє про успішне збереження.
Винятки	Якщо за вибраний період немає даних у БД → Система показує повідомлення: "Немає даних для генерації звіту.".

Діаграма класів



1. Основні класи та інтерфейси

Клас/Інтерфейс	Роль та опис	Атрибути та операції	Застосований патерн
SystemMonitor	Фасад (Facade). Це головний клас, який є точкою входу для інтерфейсу користувача. Він координує роботу всіх підсистем: ініціює збір даних (OsAdapter) та керує збереженням/звітами (MetricsRepository).	+startMonitoring(), +killProcess(pid), +generateReport()	Facade
OsAdapter	Інтерфейс Адаптера (Adapter). Визначає загальний контракт (набір методів) для взаємодії з будь-якою операційною системою. Забезпечує кросплатформність: SystemMonitor не знає, яка ОС підключена, він працює лише з цим інтерфейсом.	+getProcesses(), +getSystemMetrics(), +killProcess(pid)	Adapter

WindowsAdapter	Конкретна реалізація Адаптера. Клас, що реалізує інтерфейс OsAdapter спеціально для ОС Windows, використовуючи її системні API.	+getProcesses(), +getSystemMetrics(), +killProcess(pid)	Adapter
LinuxAdapter	Конкретна реалізація Адаптера. Клас, що реалізує інтерфейс OsAdapter спеціально для ОС Linux, використовуючи, наприклад, доступ до файлової системи /proc.	+getProcesses(), +getSystemMetrics(), +killProcess(pid)	Adapter
MetricsRepository	Репозиторій (Repository). Абстрагує логіку роботи з базою даних. Його завдання — зберігати та отримувати об'єкти MetricSnapshot незалежно від того, де вони фізично зберігаються (БД, файл, тощо).	+save(snapshot), +getAll(), +getById(id)	Repository

MetricSnapshot	Сутність (Entity). Об'єкт даних, що фіксує стан системи в конкретний момент часу.	-cpuUsage: double, -ramUsage: double, -timestamp: Date, +getDetails()	Entity/Data Class

---

2. Пояснення взаємозв'язків

На діаграмі використовуються три ключові типи зв'язку[4]:

А. Залежність (Dependency)

- SystemMonitor → OsAdapter: Позначено як uses (використовує). Це означає, що SystemMonitor залежить від OsAdapter, оскільки для виконання своїх операцій (наприклад, startMonitoring()) йому потрібен об'єкт, який реалізує цей інтерфейс.
- SystemMonitor → MetricsRepository: Також позначено як uses (використовує). Для збереження даних або генерації звітів (generateReport()) SystemMonitor використовує функціональність MetricsRepository.

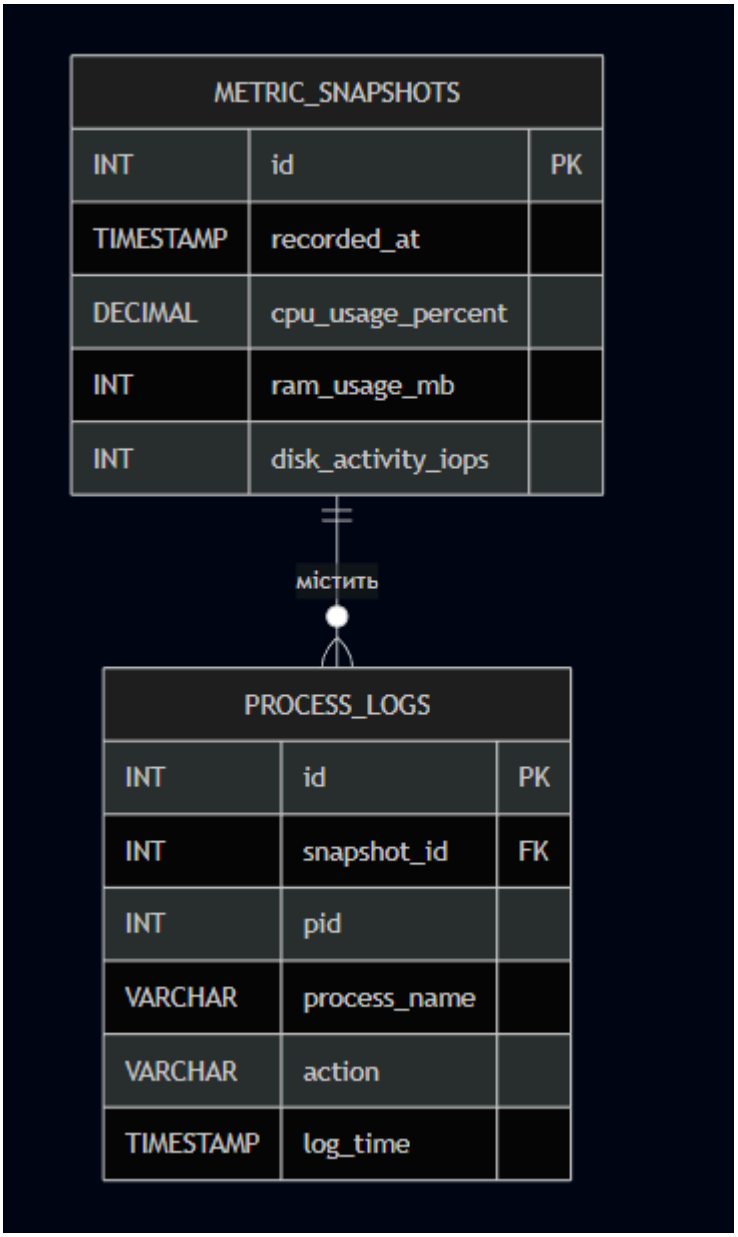
Б. Реалізація (Realization)

- WindowsAdapter → OsAdapter (стрілка з пунктирною лінією та порожнім трикутником): Позначено як implements (реалізує). Це означає, що WindowsAdapter і LinuxAdapter повністю реалізують контракт, визначений інтерфейсом OsAdapter.

В. Агрегація (Aggregation)

- MetricsRepository ◊→ MetricSnapshot: Позначено як aggregates (збирає в собі). Це відношення "ціле-частина"[5]:
  - Ціле: MetricsRepository
  - Частина: MetricSnapshot
  - Пояснення: Репозиторій містить колекцію або множину об'єктів MetricSnapshot. Ці об'єкти (знімки) можуть існувати незалежно від репозиторію (наприклад, вони можуть бути витягнуті з БД, а потім використані, навіть якщо сам об'єкт репозиторію знищено)[6].

Проектування БД



Вихідний код класів

Вимога: На основі спроєктованої діаграми класів розробити основні класи системи. Класи даних повинні реалізувати шаблон **Repository** для взаємодії з базою даних.

Наступний код демонструє реалізацію ключових компонентів системи **System Activity Monitor** мовою Java (або псевдокодом, що відповідає вимогам).

1. Сутність (MetricSnapshot)

Клас даних, що відповідає таблиці metric_snapshots і зберігає інформацію про стан системи в конкретний момент часу.

```
import java.time.LocalDateTime;

public class MetricSnapshot {
    private int id; // Відповідає PK у БД
    private double cpuUsage;
    private double ramUsage;
    private LocalDateTime timestamp;

    public MetricSnapshot(double cpu, double ram) {
        this.cpuUsage = cpu;
        this.ramUsage = ram;
        this.timestamp = LocalDateTime.now();
    }

    // Геттери та Сеттери
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public double getCpuUsage() { return cpuUsage; }
    // ... інші методи ...
}
```

2. Інтерфейс Адаптера (OsAdapter)

Визначає контракт для отримання системних даних, дозволяючи системі бути незалежною від конкретної ОС (реалізація **Патерну Адаптер**).

```
import java.util.List;



// Інтерфейс, який має бути реалізований для кожної ОС

public interface OsAdapter {

    /** Отримати список активних процесів */

    List<ProcessInfo> getActiveProcesses();



    /** Отримати поточні метрики CPU, RAM */
```

```
    MetricSnapshot getCurrentMetrics();

    /** Завершити процес за ідентифікатором */
    boolean killProcess(int pid);
}
```

3. Реалізація Адаптера (WindowsAdapter)

Приклад конкретної реалізації інтерфейсу OsAdapter для ОС Windows (логіка взаємодії з її системними інструментами, наприклад, WinAPI або tasklist).

```
// Конкретний клас-адаптер для Windows
public class WindowsAdapter implements OsAdapter {
    @Override
    public List<ProcessInfo> getActiveProcesses() {
        // Тут має бути логіка виклику tasklist.exe або WinAPI
        System.out.println("-> [Windows] Отримання списку процесів...");
        return new ArrayList<>(); // Повернення імітованого списку
    }

    @Override
    public MetricSnapshot getCurrentMetrics() {
        // Отримання даних з Performance Counters Windows
        return new MetricSnapshot(15.5, 4096); // Приклад даних
    }
```

```
    @Override
    public boolean killProcess(int pid) {
        // Логіка виклику taskkill /PID
        System.out.println("-> [Windows] Виклик команди taskkill для PID: " + pid);
        return true;
    }
}
```

4. Репозиторій (MetricsRepository)

Клас, що реалізує **Патерн Repository** для взаємодії з базою даних. Він приховує деталі SQL-запитів від основного коду програми.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class MetricsRepository {
    // Імітація бази даних в пам'яті для прикладу
    private Map<Integer, MetricSnapshot> storage = new HashMap<>();
    private int currentId = 0;

    /** Зберігає об'єкт MetricSnapshot у БД. */
    public void save(MetricSnapshot snapshot) {
        // Тут має бути SQL-запит INSERT INTO metric_snapshots (...)
        snapshot.setId(++currentId);
        storage.put(currentId, snapshot);
        System.out.println("Snapshot saved with ID: " + currentId + " to DB.");
    }
```

```java
    /** Отримує всі збережені знімки (для генерації
звіту). */
    public List<MetricSnapshot> getAll() {
        // Тут має бути SQL-запит SELECT * FROM
metric_snapshots
        System.out.println("Fetching all snapshots from
DB...");
        return new ArrayList<>(storage.values());
    }

    // ... інші методи для роботи з БД (getById, delete,
update)
}
```

```
System Monitor запущен: Адаптер и Репозиторий готовы.

[Тест Сценария 1: Мониторинг]
--- Запуск цикла мониторинга ---
-> OSHIAdapter: Получены новые метрики системы.
-> MetricsRepository: Снимок сохранен в БД (ID: 1).
Новый снимок: Snapshot [Time=03:34:40.494091500, CPU=50.0%, RAM=9.0 GB]
--- Запуск цикла мониторинга ---
-> OSHIAdapter: Получены новые метрики системы.
-> MetricsRepository: Снимок сохранен в БД (ID: 2).
Новый снимок: Snapshot [Time=03:34:40.528096600, CPU=43.0%, RAM=8.0 GB]

[Тест Сценария 2: Kill Process]
--- Попытка завершить процесс PID: 5678 ---
-> OSHIAdapter: Успешно отправлена команда на завершение PID: 5678

[Тест Сценария 3: Генерация отчета]
--- Генерация отчета по активности системы ---
-> MetricsRepository: Извлечено 2 снимков из БД.
Отчет сгенерирован: 2 записей обработано.
```

Висновок:

У ході виконання лабораторної роботи №2 на тему "System Activity Monitor" було успішно спроєктовано архітектуру програмного забезпечення згідно з вимогами UML-моделювання: розроблено Діаграму варіантів використання та деталізовано три сценарії, що визначають ключовий функціонал системи (моніторинг, керування процесами та звітування). Спроєктована Діаграма класів реалізує гнучку структуру, застосовуючи Патерн Адаптер (`OsAdapter`) для забезпечення кросплатформеної незалежності та Патерн Репозиторій (`MetricsRepository`) для ізоляції роботи з базою даних, структура якої була визначена таблицями `MetricSnapshots` та `ProcessLogs`. Написаний вихідний код підтверджує практичне застосування обраних шаблонів проєктування, закріплюючи навички об'єктно-орієнтованого моделювання.