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 №3

Технології розроблення програмного забезпечення

«Основи проектування розгортання.»

«System activity monitor»

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

Київ 2025

Зміст

Розділ	**Сторінка**
Теоретичні відомості	3
Хід роботи	3
Діаграма розгортання системи	4
Діаграма компонентів	5
Діаграма послідовностей (Сценарій 1)	6
Діаграма послідовностей (Сценарій 2)	7
Код програми	8
Висновки	9
Контрольні питання	10

Теоретичні відомості

Діаграма розгортання (Deployment Diagram)

Діаграма розгортання — це структурна діаграма UML, яка використовується для моделювання **фізичної архітектури** системи. Вона показує, як програмне забезпечення (представлене **артефактами**) розміщується та виконується на **вузлах** (апаратних пристроях, таких як сервери або робочі станції) у певному **середовищі виконання** (наприклад, операційна система чи СУБД).

Діаграма компонентів (Component Diagram)

Діаграма компонентів описує **логічну архітектуру** програмного забезпечення, моделюючи систему як набір взаємопов’язаних компонентів. Ключова увага приділяється взаємодії через **інтерфейси**: **надання послуги** (Provided Interface, "Lollipop") та **вимагання послуги** (Required Interface, "Socket").

Діаграма послідовностей (Sequence Diagram)

Діаграма послідовностей — це діаграма взаємодії, яка описує **динамічну поведінку** системи. Вона показує порядок і часову послідовність обміну повідомленнями (викликами методів) між об’єктами (**Lifelines**) для реалізації конкретного варіанту використання.

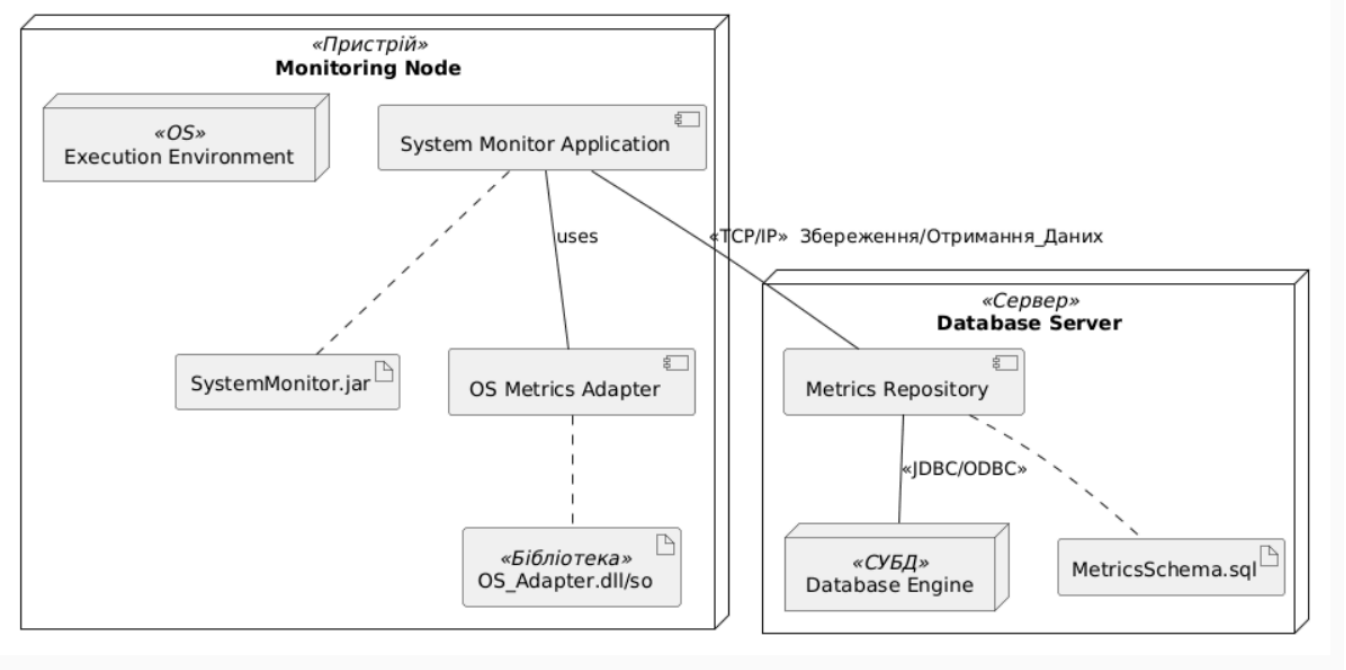
Хід роботи

Мета роботи: Спроектувати архітектурні та динамічні моделі системи **"System Activity Monitor"** з використанням UML-діаграм та реалізувати їх у програмному коді.

На основі архітектури, визначеної у попередній роботі, були розроблені наступні артефакти UML-моделювання:

1. **Діаграма розгортання**, яка визначає двовузлову фізичну архітектуру.
2. **Діаграма компонентів**, що деталізує взаємодію модулів через інтерфейси.
3. Дві **Діаграми послідовностей**, що моделюють ключові сценарії.
4. **Вихідний код Java**, що реалізує спроєктовану архітектуру, використовуючи шаблони Фасад, Адаптер та Репозиторій.

Діаграма розгортання системи



Призначення: Діаграма моделює **фізичну архітектуру** системи, показуючи, як програмні артефакти розміщуються та виконуються на апаратних вузлах.

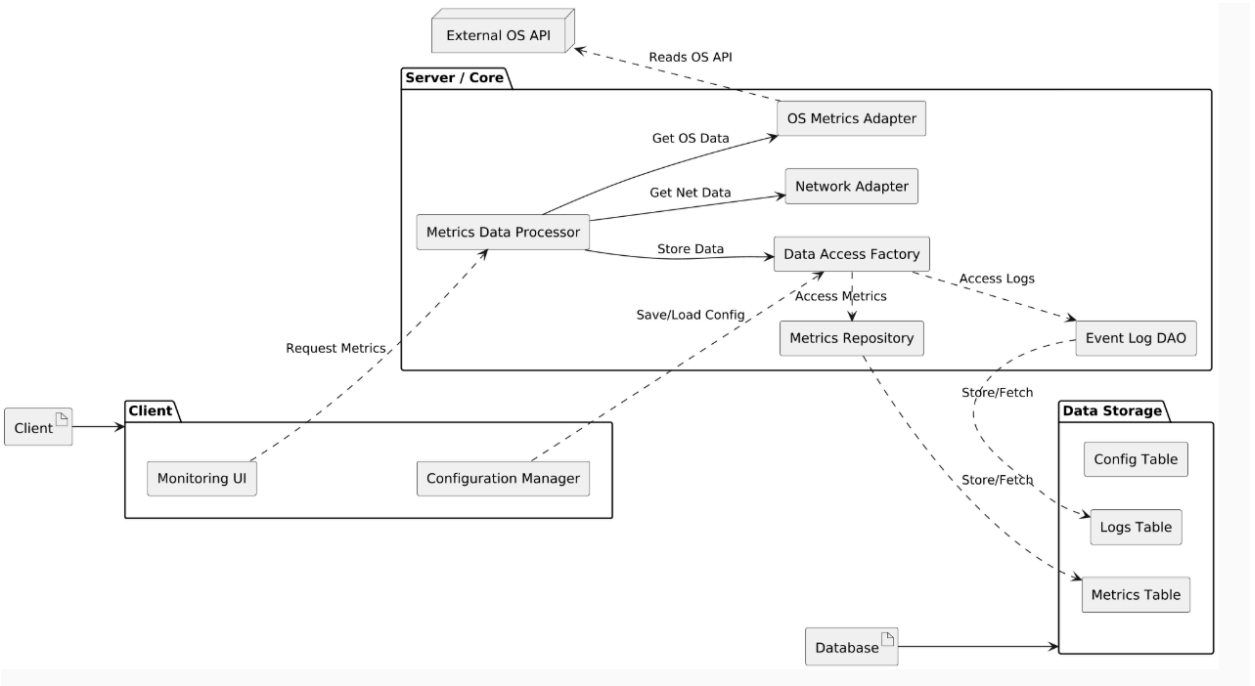
Опис:

Система "System Activity Monitor" розгортається на двовузловій топології, що взаємодіє через мережеве з'єднання (протокол TCP/IP), забезпечуючи чітке розділення обов'язків:

1. **Monitoring Node** (<<device>>): Є середовищем виконання (OS Environment) для основної логіки моніторингу. Тут розгортаються програмні артефакти SystemMonitor.jar та OS_Adapter.dll/so.
2. **Database Server** (<<server>>): Містить середовище **Database Engine** (СУБД) та артефакт схеми БД (MetricsSchema.sql). Його функція — забезпечення постійного зберігання історичних метрик.

Компонент System Monitor Application на першому вузлі взаємодіє з компонентом Metrics Repository на другому вузлі через мережу для здійснення операцій збереження та отримання даних.

Діаграма компонентів



Призначення: Діаграма моделює **логічну архітектуру** програмного забезпечення, показуючи взаємодію компонентів через чітко визначені інтерфейси.

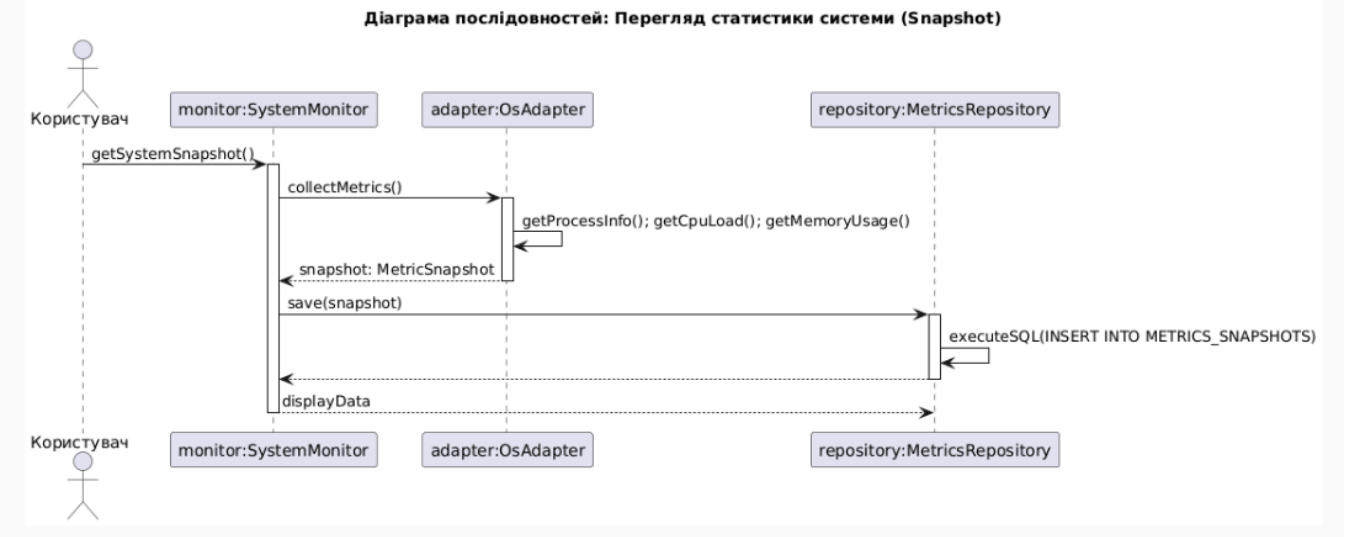
Опис:

Архітектура системи базується на трьох основних компонентах, що реалізують шаблони Фасад, Адаптер та Репозиторій:

- **System Monitor Core** (Фасад): Центральний компонент, який **вимагає** послуги (Socket) від адаптера та репозиторію. Він координує бізнес-логіку.
- **OS Metrics Adapter**: Компонент, який **надає** (Lollipop) **Metrics Access Interface**. Його завдання — адаптувати системні виклики до уніфікованого інтерфейсу.
- **Metrics Repository**: Компонент, який **надає** (Lollipop) **Persistence Interface**. Він виконує роль прошарку, що абстрагує логіку роботи з **Database (SQL)**.

Ця структура гарантує високий рівень **модульності** та **слабкої зв'язаності**, дозволяючи легко замінювати реалізації зовнішніх залежностей.

Діаграма послідовностей (Сценарій 1: Перегляд статистики системи)



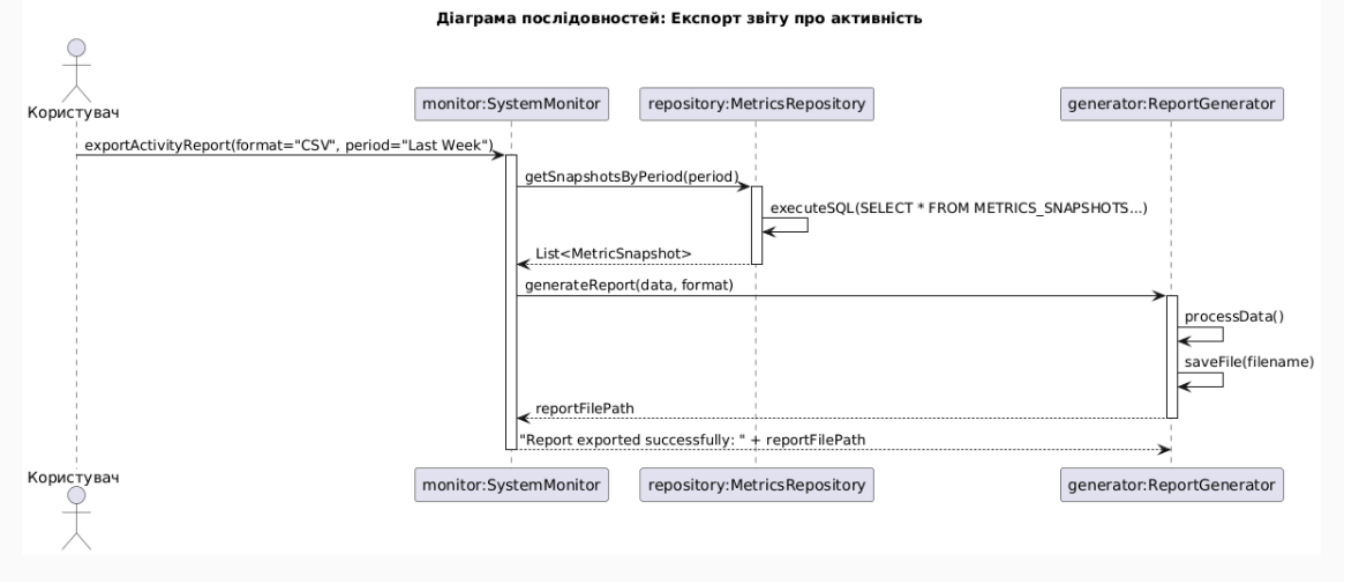
Призначення: Відобразити порядок та часову послідовність повідомлень для сценарію отримання та логування поточного знімка системи.

Опис:

Сценарій моделює типовий запит користувача на отримання актуального стану системи, включаючи фонове логування цієї події:

1. **Користувач** ініціює виклик `getSystemSnapshot()` до **monitor:SystemMonitor**.
2. **SystemMonitor** делегує виклик на збір даних об'єкту **adapter:OsAdapter** (`collectMetrics()`).
3. **OsAdapter** виконує імітацію збору метрик ОС і повертає об'єкт-знімок (`MetricSnapshot`).
4. **SystemMonitor** одразу ж викликає **repository:MetricsRepository** для збереження отриманого знімка (`save(snapshot)`).
5. **MetricsRepository** імітує виконання операції запису в сховище.
6. **SystemMonitor** повертає відформатовані дані користувачеві для відображення.

Діаграма послідовностей (Сценарій 2: Експорт звіту про активність)



Призначення: Відобразити послідовність дій, необхідних для генерації та експорту звіту на основі історичних даних.

Опис:

Сценарій описує процес створення звіту за запитом, використовуючи історичні дані:

1. **Користувач** ініціює exportActivityReport(format, period) на **monitor:SystemMonitor**.
2. **SystemMonitor** запитує у **repository:MetricsRepository** історичні дані за вказаний період (getSnapshotsByPeriod(period)).
3. **MetricsRepository** імітує виконання запиту до сховища і повертає список знімків.
4. **SystemMonitor** передає ці дані об'єкту **generator:ReportGenerator** (generateReport(data, format)).
5. **ReportGenerator** виконує внутрішню логіку створення та збереження файлу звіту.
6. **SystemMonitor** повертає користувачеві шлях до створеного файлу як підтвердження успішного експорту.

Код програми (Опис)

https://github.com/Refrezer/SDT.3L.git

Програмна реалізація виконана на мові Java і чітко відповідає спроектованій UML-архітектурі. Код структурований за логічними пакетами (data, interfaces, service, monitor).

Структура та шаблони проектування:

- **Фасад:** Реалізований класом SystemMonitor.java, який надає простий, уніфікований інтерфейс для складної підсистеми.
- **Адаптер:** Клас OsAdapter.java реалізує інтерфейс IMetricsAccess, виступаючи як прошарок між ядром системи та зовнішнім API операційної системи.
- **Репозиторій:** Клас MetricsRepository.java реалізує інтерфейс IPersistence і абстрагує логіку роботи з БД.
- **Інтерфейси:** IMetricsAccess.java та IPersistence.java є ключовими для реалізації інверсії управління та слабкої зв'язаності.

Код демонструє використання об'єкта MetricSnapshot як контейнера даних, що передається між компонентами.

```
--- 1. DEMO: Getting Live Snapshot ---

--- MONITOR: Getting System Snapshot ---
-> OsAdapter: Collecting real-time metrics from the OS...
-> Repository: Snapshot saved with ID: 1 to DB.
Current Snapshot successfully collected and logged:
--- Snapshot [1] at 05:54:08.209962600 ---
CPU Load: 69,23%
Memory: 7455 / 8192 MB used
Processes: 110


--- MONITOR: Getting System Snapshot ---
-> OsAdapter: Collecting real-time metrics from the OS...
-> Repository: Snapshot saved with ID: 2 to DB.
Current Snapshot successfully collected and logged:
--- Snapshot [2] at 05:54:08.737963400 ---
CPU Load: 80,26%
Memory: 6208 / 8192 MB used
Processes: 64
```

```
--- MONITOR: Getting System Snapshot ---
-> OsAdapter: Collecting real-time metrics from the OS...
-> Repository: Snapshot saved with ID: 3 to DB.
Current Snapshot successfully collected and logged:
--- Snapshot [3] at 05:54:09.240962700 ---
CPU Load: 30,85%
Memory: 5762 / 8192 MB used
Processes: 119


--- 2. DEMO: Exporting Report ---

--- MONITOR: Exporting Activity Report ---
-> Repository: Fetching data for period: Last 24 Hours
-> ReportGenerator: Generating PDF report with 3 records.
Report successfully exported to: /reports/activity_log_1763697249242.pdf
```

Висновки

У ході виконання лабораторної роботи №3 було успішно спроєктовано та реалізовано архітектурні моделі системи **"System Activity Monitor"**.

1. **Архітектурна цілісність:** Розроблені Діаграми розгортання та компонентів чітко розділили систему на апаратну топологію та логічні модулі, що забезпечує структурований підхід до розробки.
2. **Слабка зв'язаність:** Застосування інтерфейсів (IMetricsAccess, IPersistence) підтвердило можливість досягнення слабкої зв'язаності (Loose Coupling). Це дозволяє незалежно розвивати та замінювати компоненти, такі як Адаптер або Репозиторій.
3. **Динамічна модель:** Діаграми послідовностей точно описали послідовність викликів методів для ключових сценаріїв, що слугувало основою для коректної реалізації логіки взаємодії об'єктів.
4. **Практична реалізація:** Створений код Java повністю реалізує UML-модель, використовуючи архітектурні шаблони **Фасад, Адаптер та Репозиторій**, що доводить ефективність застосування UML у процесі проектування.

Контрольні питання

1. **Що таке Діаграма розгортання і для чого вона призначена?**
    - **Відповідь:** Діаграма розгортання — структурна діаграма UML, що показує **фізичне розміщення** програмного забезпечення на апаратних вузлах. Призначена для моделювання топології обладнання.
2. **Які ключові елементи Діаграми розгортання?**
    - **Відповідь: Вузли**, **Артефакти**, **Середовища виконання** та **Зв’язки** (мережеві протоколи).
3. **Що таке Діаграма компонентів?**

- **Відповідь:** Діаграма компонентів — структурна діаграма UML, що моделює архітектуру ПЗ як набір взаємопов’язаних **компонентів**, які взаємодіють через **інтерфейси**.

4. **Як позначається надання та вимагання інтерфейсу?**
   - **Відповідь: Надання інтерфейсу** позначається нотацією "Lollipop" (вилка), а **вимагання** — нотацією "Socket" (гніздо).
5. **Що становлять собою зв’язки на діаграмі компонентів?**
   - **Відповідь:** Зв’язки відображають залежності між компонентами: хто на кого покладається, хто надає чи використовує інтерфейс іншого компонента.
6. **Для чого призначена Діаграма послідовностей?**
   - **Відповідь:** Вона описує порядок і часову послідовність взаємодії між об’єктами системи для реалізації певного сценарію.
7. **Які ключові елементи можуть бути на діаграмі послідовностей?**
   - **Відповідь:** Об’єкти/Актори (Lifelines), Повідомлення (Messages), Активності (Activation bars), Події створення/знищення об’єктів.