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 № 3

Технології розроблення програмного забезпечення

«Основи проектування розгортання.»

Тема: IRC-Клієнт

Виконала:
студентка групи ІА-32
Красоха В.О.

Перевірив:
Мягкий М.Ю.

Київ 2025

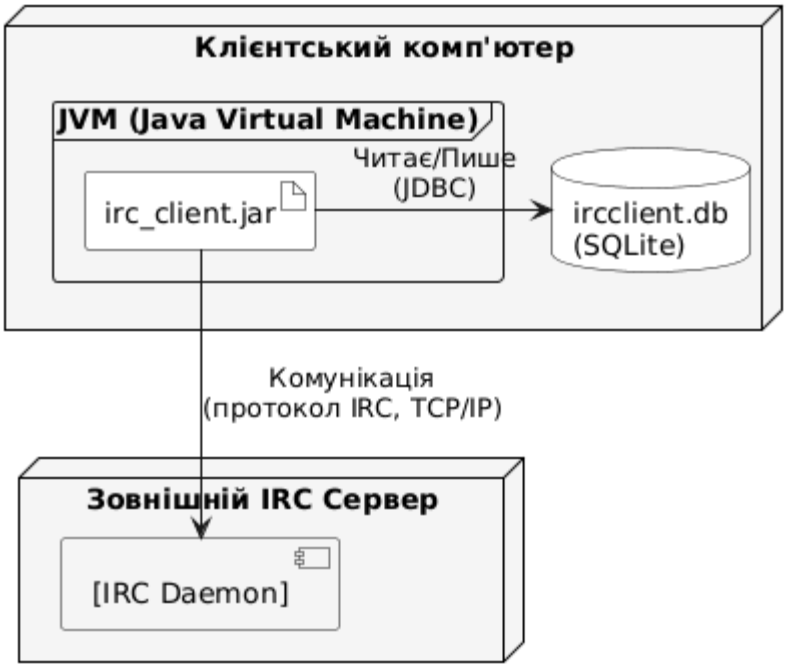
Тема: Основи проектування розгортання.

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Хід роботи

Діаграму розгортання та компонентів з описом:

Діаграма розгортання візуалізує фізичну архітектуру системи. Вона описує два вузли: "Клієнтський комп'ютер" та "Зовнішній IRC Сервер". На клієнтському вузлі розміщено виконавче середовище (JVM), в якому виконується артефакт програми, та локальний файл бази даних. На серверному вузлі знаходиться компонент "IRC Daemon". Діаграма ілюструє два типи комунікації: локальний зв'язок між програмою та її базою даних, та мережевий зв'язок за протоколом TCP/IP між програмою та сервером.



1. Вузол «Клієнтський комп'ютер»

Це локальна машина користувача, на якій розгорнуто основну частину системи:

- JVM (Java Virtual Machine): Оскільки застосунок розроблено на мові Java, він виконується всередині віртуальної машини, що забезпечує кросплатформеність.
- irc_client.jar: Виконавчий артефакт (скомпільована програма), який містить усі класи контролерів, сервісів та моделей.
- Локальна база даних (ircclient.db): На діаграмі чітко видно використання SQLite. Програма взаємодіє з нею локально через протокол JDBC (Java Database Connectivity). Це дозволяє зберігати історію чатів та налаштування серверів безпосередньо на пристрої користувача.

2. Вузол «Зовнішній IRC Сервер»

Представляє віддалений сервер у глобальній або локальній мережі:

- IRC Daemon: Спеціалізоване програмне забезпечення на стороні сервера, яке обробляє IRC-протокол, керує каналами та ретранслює повідомлення між клієнтами.

3. Комунікаційні шляхи та протоколи

На діаграмі виділено два ключові вектори взаємодії:

- Локальний (Читає/Пише JDBC): Забезпечує персистентність даних. Програма звертається до файлу ircclient.db для збереження кожного вхідного або вихідного повідомлення.
- Мережевий (TCP/IP, IRC): Використовується для реального зв'язку. Клієнт встановлює надійне TCP-з'єднання з сервером. Комунікація відбувається шляхом надсилання текстових команд (наприклад, PRIVMSG для повідомлень або LIST для списку каналів), що відповідає стандартам протоколу IRC.

Основні компоненти системи:

- Інтерфейс користувача (JavaFX View): Десктопна частина застосунку, побудована на основі FXML-файлів та CSS-стилів. Відповідає за візуалізацію чату, списку каналів та форм підключення.
- Контролери (MainController, ВікноПідключення): Вхідна точка взаємодії. Приймають події від користувача (натискання кнопок, введення тексту), оновлюють UI та передають команди до сервісного шару.
- Сервісний шар (IRCService): Центральний вузол бізнес-логіки. Координує процеси підключення, відправки повідомлень та синхронізації даних між мережею та базою даних.
- Мережевий модуль (IRCConnection, MessageParser): Відповідає за низькорівневу TCP/IP комунікацію. Виконує обмін даними з IRC-сервером через сокети та перетворює сирі текстові рядки у структуровані об’єкти повідомлень.
- Модель даних (Entities): Класи, що відображають сутності предметної області (Server, Channel, Message, User). Використовуються для передачі даних між усіма шарами системи.
- Рівень доступу до даних (DAO/Repositories): Набір класів (MessageDao, ServerDao), що реалізують шаблон Repository. Вони інкапсулюють у собі SQL-запити для взаємодії з SQLite.
- База Даних (SQLite): Локальний файл ircclient.db, що зберігає історію листування, налаштування серверів та список каналів для забезпечення персистентності даних.

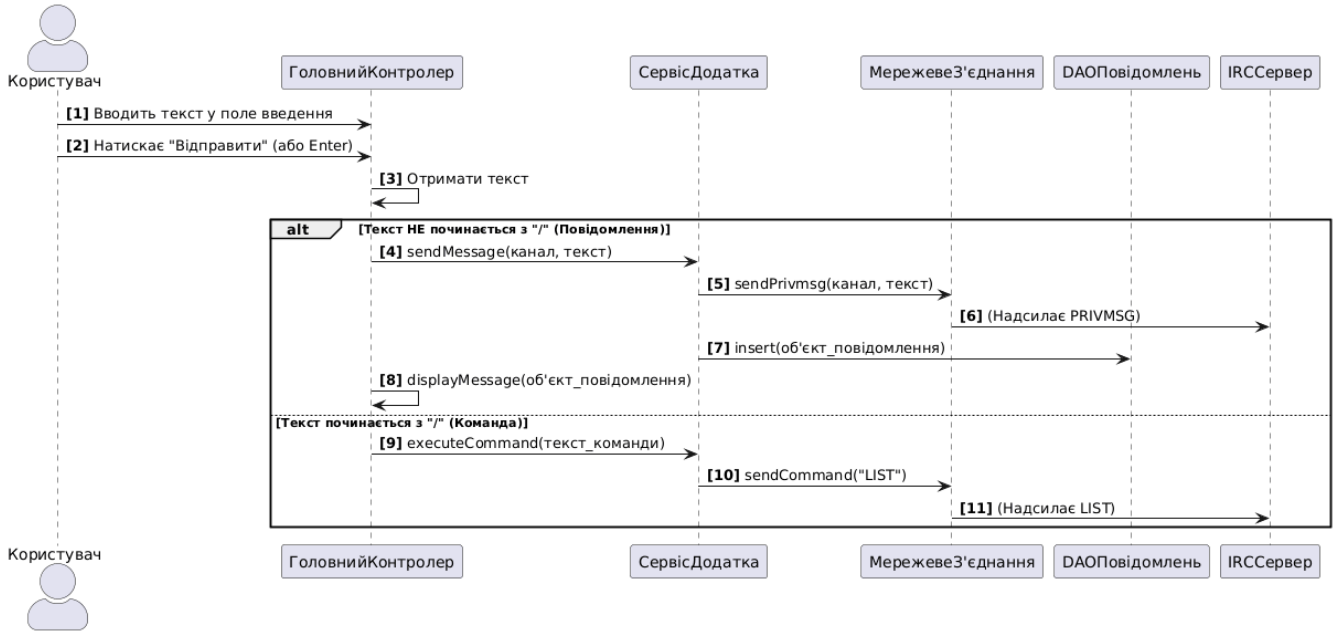
Взаємодії компонентів:

1. Користувач вводить текст у поле введення та натискає "Відправити" (через Контролер). Контролер викликає метод відправки у Сервісі.
2. Сервіс одночасно виконує дві дії:

- Передає команду до Мережевого модуля для відправки повідомлення на віддалений IRC Сервер.
- Створює об'єкт повідомлення та передає його до DAO, який виконує SQL-запит INSERT до Бази Даних.

3. При отриманні відповіді від сервера Мережевий модуль парсить дані, Сервіс оновлює Модель, а Контролер миттєво відображає нове повідомлення в Інтерфейсі.

Діаграма послідовностей:



Для ілюстрації динамічної поведінки системи розроблено діаграми послідовностей. Діаграма "Підключення до сервера" деталізує сценарій, що починається з ініціації користувачем, проходить через діалогове вікно, контролер, сервісний шар, і доходить до мережевого з'єднання та DAO для збереження профілю. Вона також моделює асинхронні відповіді від сервера, такі як успішне підключення або помилка. Інша діаграма, "Надіслати повідомлення", демонструє логіку обробки вводу користувача, показуючи альтернативні потоки: один для звичайного повідомлення (що включає відправку, збереження в БД та відображення в UI) та інший для обробки команди.

1. Учасники процесу:

На діаграмі представлені основні компоненти архітектури, які ви розробили:

- Користувач: Актор, що ініціює дію через графічний інтерфейс.
- ГоловнийКонтролер: Клас MainController, що обробляє події UI.
- СервісДодатка: Клас IRCService, де зосереджена бізнес-логіка.
- МережевеЗ'єднання: Клас IRCConnection, що працює з сокетами.
- DAOПовідомлень: Репозиторій MessageDao для роботи з БД SQLite.
- IRCСервер: Зовнішній вузол, з яким спілкується програма.

2. Основний сценарій: Відправка повідомлення

Коли користувач вводить текст і натискає "Відправити", система проходить такі кроки:

1. Вхідна подія (1-3): Контролер отримує текст із поля введення.
2. Розгалуження (блок alt): Система перевіряє, чи є текст звичайним повідомленням, чи командою (що починається з /).
3. Обробка повідомлення (4-8):
    - Контролер викликає метод sendMessage у Сервісу.
    - Сервіс звертається до Мережевого з'єднання (sendPrivmsg).
    - Мережева дія (6): Відправляється сира команда PRIVMSG на реальний IRC-сервер.
    - Збереження (7): Паралельно Сервіс викликає insert у DAO, щоб записати повідомлення в локальну базу даних ircclient.db.
    - Зворотний зв'язок (8): Контролер відображає відправлене повідомлення у вікні чату користувача.

3. Альтернативний сценарій: Виконання команди

Якщо текст починається з символу / (наприклад, /LIST або /JOIN):

1. Команда (9-11): Контролер ініціює executeCommand.
2. Мережевий протокол (11): Сервіс формує стандартну IRC-команду (наприклад, LIST) і через сокет відправляє її серверу. На відміну від повідомлень, команди зазвичай не зберігаються в локальній базі даних історії.

Вихідний код:

DatabaseService.java:

```
package org.ircclient.service;

import org.ircclient.dao.ChannelDao;
import org.ircclient.dao.MessageDao;
import org.ircclient.dao.ServerDao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class DatabaseService {
    private static final String DB_URL = "jdbc:sqlite:ircclient.db";

    private Connection connection;
    private ServerDao serverDao;
    private ChannelDao channelDao;
    private MessageDao messageDao;

    public DatabaseService() {
        try {
            Class.forName("org.sqlite.JDBC");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException("Помилка завантаження драйвера SQLite", e);
        }
        initializeDatabase();
    }

    private void initializeDatabase() {
        try {
            connection = DriverManager.getConnection(DB_URL);

            try (Statement stmt = connection.createStatement()) {
                stmt.execute("PRAGMA foreign_keys = ON;");
            }

            createTables();
            serverDao = new ServerDao(connection);
            channelDao = new ChannelDao(connection);
            messageDao = new MessageDao(connection);
```

```
    } catch (SQLException e) {
        throw new RuntimeException("Помилка ініціалізації бази даних SQLite", e);
    }
}

/**
 * Створення таблиць БД
 */
private void createTables() throws SQLException {
    try (Statement stmt = connection.createStatement()) {
        // Таблиця серверів
        // - Змінено: INT PRIMARY KEY AUTO_INCREMENT -> INTEGER PRIMARY KEY AUTOINCREMENT
        stmt.execute("""
            CREATE TABLE IF NOT EXISTS servers (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name VARCHAR(100),
                host VARCHAR(255) NOT NULL,
                port INT NOT NULL,
                nickname VARCHAR(50) NOT NULL
            )
        """);

        // Таблиця каналів
        // - Змінено: INT PRIMARY KEY AUTO_INCREMENT -> INTEGER PRIMARY KEY AUTOINCREMENT
        // - Змінено: TIMESTAMP -> DATETIME (в SQLite це буде TEXT або REAL)
        stmt.execute("""
            CREATE TABLE IF NOT EXISTS channels (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                server_id INT NOT NULL,
                name VARCHAR(100) NOT NULL,
                joined_at DATETIME,
                FOREIGN KEY (server_id) REFERENCES servers(id) ON DELETE CASCADE
            )
        """);

        // Таблиця повідомлень
        // - Змінено: INT PRIMARY KEY AUTO_INCREMENT -> INTEGER PRIMARY KEY AUTOINCREMENT
        // - Змінено: TIMESTAMP -> DATETIME
        // - Змінено: BOOLEAN DEFAULT FALSE -> INTEGER DEFAULT 0
        stmt.execute("""
            CREATE TABLE IF NOT EXISTS messages (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                channel_id INT,
                sender VARCHAR(50),
                content TEXT NOT NULL,
                timestamp DATETIME NOT NULL,
                is_private INTEGER DEFAULT 0,
                FOREIGN KEY (channel_id) REFERENCES channels(id) ON DELETE CASCADE
            )
        """);
    }
}

public Connection getConnection() {
    return connection;
}

public ServerDao getServerDao() {
    return serverDao;
}

public ChannelDao getChannelDao() {
    return channelDao;
```

```
    }

    public MessageDao getMessageDao() {
        return messageDao;
    }

    public void close() {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

IRCService.java:

```
package org.ircclient.service;

import org.ircclient.model.Channel;
import org.ircclient.model.Message;
import org.ircclient.model.Server;
import org.ircclient.model.User;
import org.ircclient.network.IRCConnection;
import org.ircclient.network.IRCMessage;
import org.ircclient.network.IRCMessageParser;

import java.io.IOException;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.function.Consumer;

public class IRCService {
    private IRCConnection connection;
    private DatabaseService databaseService;
    private Server currentServer;
    private String currentNickname;
    private Map<String, Channel> activeChannels = new ConcurrentHashMap<>();
    private Map<String, List<User>> channelUsers = new ConcurrentHashMap<>();
    private Map<String, List<String>> channelList = new ConcurrentHashMap<>();


    private Consumer<Message> onMessageReceived;
    private Consumer<Channel> onChannelJoined;
    private Consumer<String> onChannelLeft;
    private Consumer<String> onUserListUpdated;
    private Consumer<List<String>> onChannelListReceived;
    private Consumer<String> onError;
    private Consumer<String> onSystemMessage;

    public IRCService(DatabaseService databaseService) {
        this.databaseService = databaseService;
        this.connection = new IRCConnection();
        setupMessageHandler();
    }

    /**
     * Налаштування обробника повідомлень від IRC сервера
     */
```

```
private void setupMessageHandler() {
    connection.setMessageHandler(this::handleIRCMessage);
}

/**
 * Підключення до IRC сервера
 */
public void connect(Server server) throws IOException {
    this.currentServer = server;
    this.currentNickname = server.getNickname();

    connection.connect(server.getHost(), server.getPort());
    connection.sendNick(server.getNickname());
    connection.sendUser(server.getNickname(), server.getNickname());

    // Збереження сервера в БД
    try {
        databaseService.getServerDao().insert(server);
    } catch (Exception e) {
        // Сервер може вже існувати
    }
}

/**
 * Відключення від сервера
 */
public void disconnect() {
    connection.disconnect();
    activeChannels.clear();
    channelUsers.clear();
    currentServer = null;
}

/**
 * Приєднання до каналу
 */
public void joinChannel(String channelName) throws IOException {
    if (!channelName.startsWith("#")) {
        channelName = "#" + channelName;
    }
    connection.sendJoin(channelName);
}

/**
 * Вихід з каналу
 */
public void leaveChannel(String channelName) throws IOException {
    connection.sendPart(channelName);
}

/**
 * Відправка повідомлення
 */
public void sendMessage(String target, String content) throws IOException {
    connection.sendPrivmsg(target, content);

    // Збереження повідомлення в БД
    Channel channel = activeChannels.get(target);
    if (channel != null) {
        Message message = new Message(currentNickname, content);
        message.setChannelId(channel.getId());
        message.setIsPrivate(target.equals(currentNickname));
        try {
```

```
            databaseService.getMessageDao().insert(message);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

/**
 * Виконання IRC команди
 */
public void executeCommand(String commandLine) throws IOException {
    String[] parts = commandLine.substring(1).split(" ", 2);
    String command = parts[0].toUpperCase();
    String args = parts.length > 1 ? parts[1] : "";

    switch (command) {
        case "WHOIS":
            if (!args.isEmpty()) {
                connection.sendWhois(args);
            }
            break;
        case "LIST":
            connection.sendList();
            break;
        case "TOPIC":
            String[] topicParts = args.split(" ", 2);
            if (topicParts.length == 2) {
                connection.sendTopic(topicParts[0], topicParts[1]);
            } else if (topicParts.length == 1) {
                connection.sendTopic(topicParts[0], null);
            }
            break;
        case "JOIN":
            if (!args.isEmpty()) {
                joinChannel(args);
            }
            break;
        case "PART":
            if (!args.isEmpty()) {
                leaveChannel(args);
            }
            break;
        default:
            if (onError != null) {
                onError.accept("Невідома команда: " + command);
            }
    }
}

/**
 * Обробка IRC повідомлень від сервера
 */
private void handleIRCMessage(IRCMessage msg) {
    String command = msg.getCommand();

    switch (command) {
        case "001": // RPL_WELCOME - успішне підключення
            if (onSystemMessage != null) {
                onSystemMessage.accept("Підключено до сервера");
            }
            break;

        case "PING":
```

```
            // Відповідь на PING
            try {
                connection.sendPong(msg.getTrailing());
            } catch (IOException e) {
                e.printStackTrace();
            }
            break;

        case "PRIVMSG":
            handlePrivmsg(msg);
            break;

        case "JOIN":
            handleJoin(msg);
            break;

        case "PART":
            handlePart(msg);
            break;

        case "QUIT":
            handleQuit(msg);
            break;

        case "TOPIC":
            handleTopic(msg);
            break;

        case "332": // RPL_TOPIC - тема каналу
            handleTopicResponse(msg);
            break;

        case "353": // RPL_NAMREPLY - список користувачів каналу
            handleNamesReply(msg);
            break;

        case "366": // RPL_ENDOFNAMES - кінець списку користувачів
            handleEndOfNames(msg);
            break;

        case "322": // RPL_LIST - список каналів
            handleListReply(msg);
            break;

        case "323": // RPL_LISTEND - кінець списку каналів
            handleListEnd();
            break;

        case "311": // RPL_WHOISUSER
        case "312": // RPL_WHOISSERVER
        case "313": // RPL_WHOISOPERATOR
        case "317": // RPL_WHOISIDLE
        case "318": // RPL_ENDOFWHOIS
            handleWhoisResponse(msg);
            break;

        case "433": // ERR_NICKNAMEINUSE
            if (onError != null) {
                onError.accept("Нікнейм вже використовується");
            }
            break;

        default:
```

```
            // Інші числові відповіді
            if (command.matches("\\d{3}")) {
                handleNumericResponse(msg);
            }
        }
    }

    private void handlePrivmsg(IRCMessage msg) {
        String sender = IRCMessageParser.extractNickname(msg.getPrefix());
        String target = msg.getParam(0);
        String content = msg.getTrailing();

        Message message = new Message(sender, content);
        message.setType(Message.MessageType.NORMAL);

        // Визначення чи це приватне повідомлення
        boolean isPrivate = !target.startsWith("#");
        message.setIsPrivate(isPrivate);

        String channelKey = isPrivate ? sender : target;
        message.setChannelName(channelKey);  // Встановлюємо назву каналу для відображення

        Channel channel = activeChannels.get(channelKey);
        if (channel == null && isPrivate) {
            // Створення віртуального каналу для приватних повідомлень
            channel = new Channel(currentServer.getId(), sender);
            activeChannels.put(sender, channel);
        }

        if (channel != null) {
            message.setChannelId(channel.getId());
            try {
                databaseService.getMessageDao().insert(message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        if (onMessageReceived != null) {
            onMessageReceived.accept(message);
        }
    }

    private void handleJoin(IRCMessage msg) {
        String nickname = IRCMessageParser.extractNickname(msg.getPrefix());
        String channelName = msg.getParam(0);  // Канал завжди в першому параметрі

        if (nickname.equals(currentNickname)) {
            // Ми приєдналися до каналу
            Channel channel = new Channel(currentServer.getId(), channelName);
            try {
                databaseService.getChannelDao().insert(channel);
                activeChannels.put(channelName, channel);
                if (onChannelJoined != null) {
                    onChannelJoined.accept(channel);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            // Хтось інший приєднався
            Message message = new Message(nickname, "приєднався до каналу");
            message.setType(Message.MessageType.JOIN);
```

```
            message.setChannelName(channelName);
            if (onMessageReceived != null) {
                onMessageReceived.accept(message);
            }
        }
    }

    private void handlePart(IRCMessage msg) {
        String nickname = IRCMessageParser.extractNickname(msg.getPrefix());
        String channelName = msg.getParam(0);

        if (nickname.equals(currentNickname)) {
            // Ми вийшли з каналу
            activeChannels.remove(channelName);
            channelUsers.remove(channelName);
            if (onChannelLeft != null) {
                onChannelLeft.accept(channelName);
            }
        } else {
            // Хтось інший вийшов
            Message message = new Message(nickname, "вийшов з каналу");
            message.setType(Message.MessageType.PART);
            message.setChannelName(channelName);
            if (onMessageReceived != null) {
                onMessageReceived.accept(message);
            }
        }
    }

    private void handleQuit(IRCMessage msg) {
        String nickname = IRCMessageParser.extractNickname(msg.getPrefix());
        Message message = new Message(nickname, "відключився");
        message.setType(Message.MessageType.QUIT);
        if (onMessageReceived != null) {
            onMessageReceived.accept(message);
        }
    }

    private void handleTopic(IRCMessage msg) {
        String channelName = msg.getParam(0);
        String topic = msg.getTrailing();
        Channel channel = activeChannels.get(channelName);
        if (channel != null) {
            channel.setTopic(topic);
        }
    }

    private void handleTopicResponse(IRCMessage msg) {
        String channelName = msg.getParam(1);
        String topic = msg.getTrailing();
        Channel channel = activeChannels.get(channelName);
        if (channel != null) {
            channel.setTopic(topic);
        }
    }

    private void handleNamesReply(IRCMessage msg) {
        String channelName = msg.getParam(2);
        String names = msg.getTrailing();
        String[] userArray = names.split(" ");

        channelUsers.putIfAbsent(channelName, new ArrayList<>());
        List<User> users = channelUsers.get(channelName);
```

```
        for (String userStr : userArray) {
            User user = new User();
            // Витягування режимів (@, +)
            String modes = "";
            String nickname = userStr;
            if (userStr.startsWith("@")) {
                modes = "@";
                nickname = userStr.substring(1);
            } else if (userStr.startsWith("+")) {
                modes = "+";
                nickname = userStr.substring(1);
            }
            user.setNickname(nickname);
            user.setModes(modes);
            users.add(user);
        }
    }

    private void handleEndOfNames(IRCMessage msg) {
        String channelName = msg.getParam(1);
        if (onUserListUpdated != null) {
            onUserListUpdated.accept(channelName);
        }
    }

    private void handleListReply(IRCMessage msg) {
        String channelName = msg.getParam(1);
        String userCount = msg.getParam(2);
        String topic = msg.getTrailing();

        channelList.putIfAbsent("list", new ArrayList<>());
        channelList.get("list").add(channelName + " (" + userCount + " користувачів) - " + topic);
    }

    private void handleListEnd() {
        if (onChannelListReceived != null) {
            onChannelListReceived.accept(channelList.getOrDefault("list", new ArrayList<>()));
        }
        channelList.clear();
    }

    private void handleWhoisResponse(IRCMessage msg) {
        String info = msg.getTrailing();
        if (onSystemMessage != null) {
            onSystemMessage.accept(info);
        }
    }

    private void handleNumericResponse(IRCMessage msg) {
        // Обробка інших числових відповідей
        String content = msg.getTrailing();
        if (content != null && !content.isEmpty()) {
            if (onSystemMessage != null) {
                onSystemMessage.accept(content);
            }
        }
    }

    // Getters та Setters для callbacks
    public void setOnMessageReceived(Consumer<Message> handler) {
        this.onMessageReceived = handler;
    }
```

```
    public void setOnChannelJoined(Consumer<Channel> handler) {
        this.onChannelJoined = handler;
    }

    public void setOnChannelLeft(Consumer<String> handler) {
        this.onChannelLeft = handler;
    }

    public void setOnUserListUpdated(Consumer<String> handler) {
        this.onUserListUpdated = handler;
    }

    public void setOnChannelListReceived(Consumer<List<String>> handler) {
        this.onChannelListReceived = handler;
    }

    public void setOnError(Consumer<String> handler) {
        this.onError = handler;
    }

    public void setOnSystemMessage(Consumer<String> handler) {
        this.onSystemMessage = handler;
    }

    public Map<String, Channel> getActiveChannels() {
        return activeChannels;
    }

    public List<User> getChannelUsers(String channelName) {
        return channelUsers.getOrDefault(channelName, new ArrayList<>());
    }

    public boolean isConnected() {
        return connection.isConnected();
    }

    public Server getCurrentServer() {
        return currentServer;
    }
}
```

Контрольні запитання:

1. Що собою становить діаграма розгортання?
Діаграма розгортання — це діаграма UML, яка показує фізичне розміщення програмних компонентів на апаратних вузлах та взаємодію між цими вузлами.

2. Які бувають види вузлів на діаграмі розгортання?
Основні види вузлів:

- апаратні вузли (сервери, комп'ютери, пристрої)
- виконавчі середовища (операційні системи, JVM, контейнер застосунків)

3. Які бувають зв'язки на діаграмі розгортання?

- комунікаційні зв'язки (мережеві з'єднання)
- залежності між вузлами
- розміщення артефактів на вузлах (deployment)

4. Які елементи присутні на діаграмі компонентів?

- компоненти
- інтерфейси (надані та необхідні)
- порти
- залежності
- пакети (за потреби)

5. Що становлять собою зв'язки на діаграмі компонентів?
Зв'язки відображають залежності та взаємодію між компонентами, зазвичай через інтерфейси (використання або надання сервісів).

6. Які бувають види діаграм взаємодії?
До діаграм взаємодії UML належать:

- діаграма послідовностей
- діаграма комунікації

- діаграма огляду взаємодії
- діаграма часових діаграм

7. Для чого призначена діаграма послідовностей?
Вона призначена для моделювання обміну повідомленнями між об’єктами у часі під час виконання конкретного сценарію.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- учасники (актори, об’єкти)
- лінії життя
- повідомлення
- активації
- умовні конструкції (alt, opt, loop)

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?
Діаграми послідовностей деталізують сценарії варіантів використання, показуючи внутрішню взаємодію об’єктів при їх виконанні.

10. Як діаграми послідовностей пов’язані з діаграмами класів?

- учасники діаграми послідовностей є екземплярами класів
- повідомлення відповідають методам класів
- допомагають перевірити коректність структури класів і їх інтерфейсів