Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ:
CLIENT-SERVER, PEER-TO-PEER,
SERVICE-ORIENTED ARCHITECTURE»
Варіант №26

Виконав:
студент групи ІА-23
Мозоль В.О

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема.
Різні види взаємодії додатків: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Мета.
Метою лабораторної роботи є , вивчення та практичне впровадження різних моделей взаємодії додатків, зокрема, клієнт-серверної архітектури, Peer-to-Peer (P2P) архітектури, сервіс-орієнтованої архітектури (SOA)Набуття практичних навичок реалізації розподілених систем з використанням технологій.

Завдання.
1 . Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Обрана тема.
26 Download manager (iterator, command, observer, template method, composite, p2p)
Інструмент для скачування файлів з інтернету по протоколах http або https з можливістю продовження завантаження в зупиненому місці, розподілу швидкостей активним завантаженням, ведення статистики завантажень, інтеграції в основні браузери (firefox, opera, internet explorer, chrome).

Короткі теоретичні відомості.

На основі наданого документа я підготую короткі теоретичні відомості про клієнт-серверні додатки, peer-to-peer додатки, сервіс-орієнтовану архітектуру та мікро-сервісну архітектуру.

Клієнт-серверні додатки

Клієнт-серверні додатки - це розподілені додатки, які складаються з двох основних компонентів:

- **Клієнти**: відповідають за представлення додатку користувачеві
- **Сервери**: використовуються для зберігання і обробки даних

Типи клієнтів

1. **Тонкий клієнт**:
    - Передає майже всі операції на сервер для обробки
    - Зберігає лише візуальне представлення відповідей
    - Доцільний у захищених сценаріях та при необхідності централізації обчислень
2. **Товстий клієнт**:
    - Містить більшість логіки обробки даних на стороні клієнта
    - Розвантажує сервер
    - Сервер виступає точкою доступу до ресурсів

Структура клієнт-серверної взаємодії

Зазвичай організована у 3 рівні:

- **Клієнтська частина**: відображення інтерфейсу, логіка дій користувача
- **Загальна частина** (middleware): спільні класи та компоненти
- **Серверна частина**: основна бізнес-логіка, зберігання та обмін даними

Peer-to-Peer (P2P) Додатки

Характеристики:

- Рівноправність клієнтських програм
- Відсутність центрального серверу
- Взаємодія між клієнтами для досягнення спільних цілей

Основні виклики

- Синхронізація даних
- Пошук клієнтських застосувань

Механізми взаємодії

- Структуровані однорангові мережі
- Спеціальні протоколи обміну повідомленнями
- Алгоритми синхронізації (hash-алгоритми, узгодження)

Сервіс-орієнтована архітектура (SOA)

Модульний підхід до розробки ПЗ з наступними характеристиками:

- Розподілені, слабо пов'язані компоненти
- Стандартизовані інтерфейси
- Взаємодія за стандартизованими протоколами (SOAP, REST)

Переваги

- Інкапсуляція деталей реалізації
- Незалежність від платформ
- Повторне використання компонентів
- Масштабованість систем

Software as a Service (SaaS)

Бізнес-модель продажу ПЗ з такими особливостями:

- Веб-додаток, що керується постачальником
- Доступ через Інтернет
- Щомісячна абонентська плата або оплата за обсягом операцій
- Технічна підтримка включена в оплату
- Швидкі оновлення та модернізація

Мікро-сервісна архітектура

Підхід до створення серверного додатку як набору малих незалежних служб:

- Кожна служба в окремому процесі
- Взаємодія через HTTP/HTTPS, WebSockets, AMQP
- Автономний розвиток та розгортання
- Реалізація специфічних можливостей в обмеженому контексті

Переваги

- Гнучкість у довгостроковій перспективі
- Висока масштабованість
- Полегшене супроводження складних систем

Хід роботи.

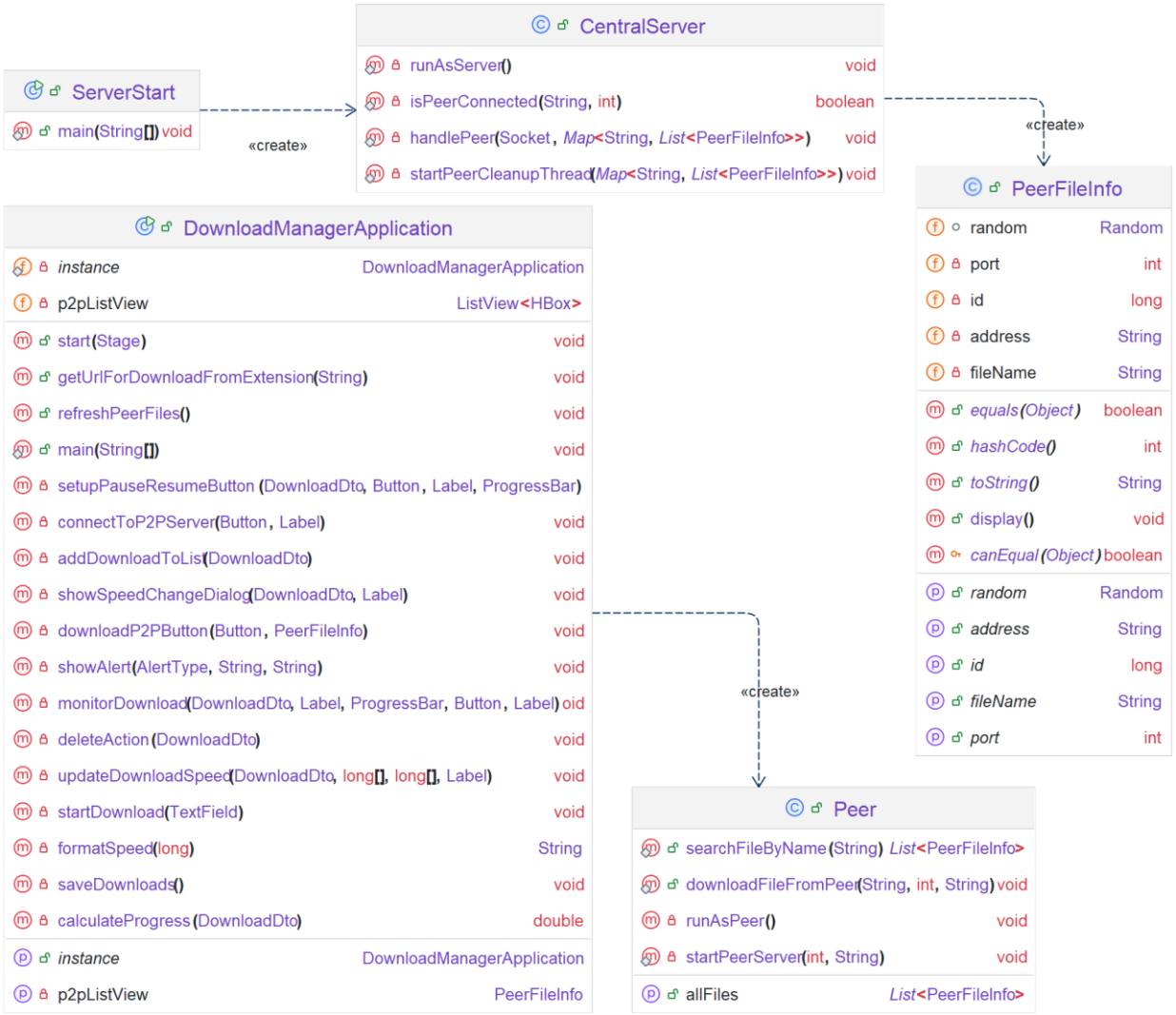

Рисунок №1 – Діаграма класів , згенерована IDE, реалізації шаблону Observer

Основні компоненти:

1. Peer (Клієнт-сервер):
   - Обмін файлами:
     Peer може завантажувати файли з інших вузлів та відправляти запити на пошук файлів. Також він приймає запити на передачу файлів від інших вузлів.
   - Реєстрація:
     Peer реєструє список доступних файлів на центральному сервері при запуску.
   - Завантаження файлів:
     Функціонал downloadFileFromPeer відповідає за завантаження файлу з іншого вузла.
2. Центральний сервер:

- Функція трекера:
  Сервер зберігає інформацію про файли та їх місцезнаходження, зареєстровані вузлами. Він допомагає вузлам знаходити один одного.
- Обробка запитів:
  Сервер виконує три основні дії:
  - REGISTER — реєстрація файлів, які доступні для завантаження.
  - SEARCH — пошук вузлів, які мають запитуваний файл.
  - FILES — отримання всього списку файлів у системі.
- Очищення неактивних вузлів:
  Сервер видаляє записи про вузли, які більше недоступні.

3. Обмін даними:
   - Вузли спілкуються через сокети (Socket), передаючи команди та дані через ObjectInputStream і ObjectOutputStream.
   - Завантаження файлів здійснюється через потоки (InputStream/OutputStream), що дозволяє передавати файли частинами

Код реалізації паттерну можна переглянути у GitHub репозиторії у папці DownloadManager або у Додатку А.

Робота паттерну.

Для демонстрації роботи паттерну, запустимо центральний сервер та приєднаємо до нього два піри з демонстраційними файлами.Також завантажимо тестовий файл. (Код для тестування є у GitHub репозиторії або у Додатку А) Результат виконання коду:

Рисунок №2 – Консоль піра на якому буде тестувати завантаження файлу

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Intelej\IntelliJ
Choose mode: (1) Server, (2) Peer
2
Enter central server IP address:
127.0.0.1
Enter central server port:
5000
Enter directory to share files from:
/test_peer2
Enter the port for this peer to listen on:
3001
Peer server is listening on port 3001
Files registered: [big_buck_bunny_720p_30mb.mp4, example.txt, Що нада.txt]
Enter the name of the file to search:
Uploading file: big_buck_bunny_720p_30mb.mp4
File uploaded successfully.
```

Рисунок №3 – Консоль піра який має потрібний файл для завантаження

Рисунок №4 – Консоль серверу для знаходження пірів

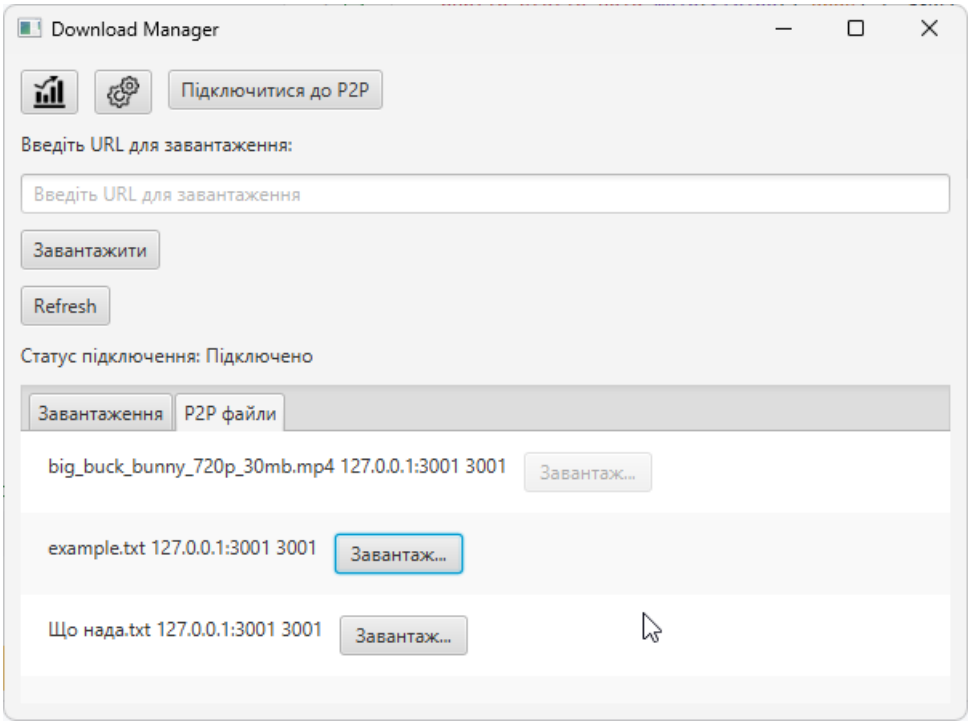

Рисунок №5 – Графічний інтерфейс для завантаження файлу (Кнопка неактивна тому що цей файл уже завантажено)

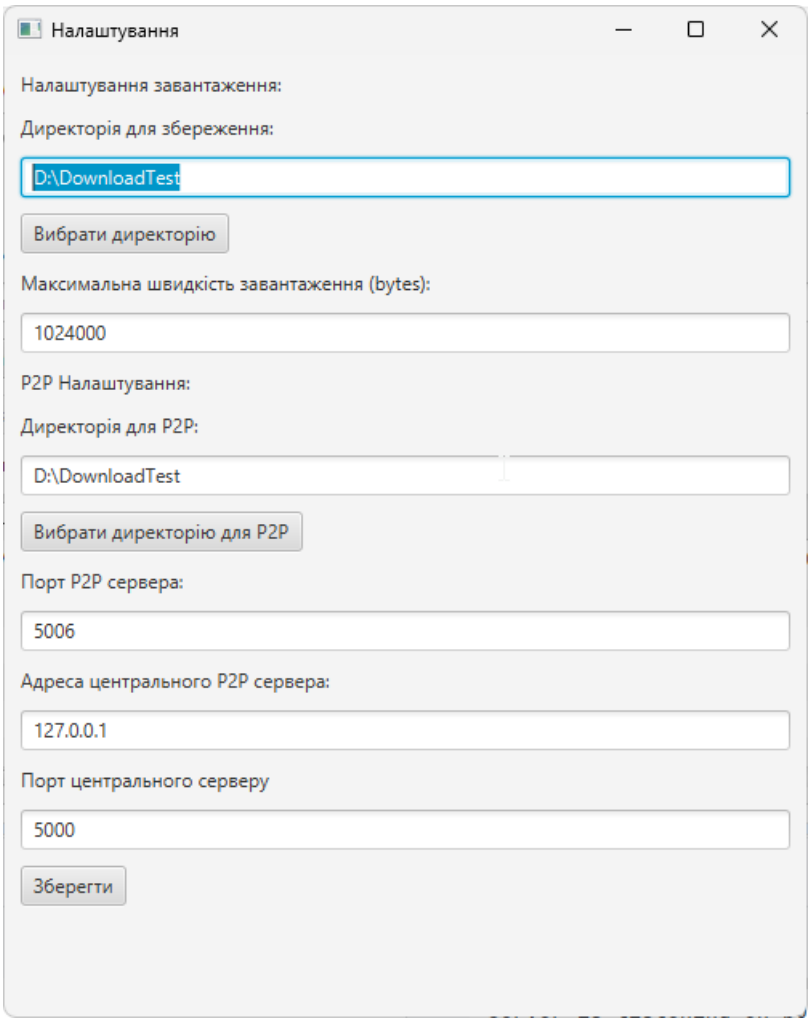

Рисунок №6 – Графічний інтерфейс налаштування для завантаження

Висновки.

За результатами виконання лабораторної роботи можна зробити наступні висновки. Успішно реалізовано peer-to-peer додаток для обміну файлами з центральним сервером координації. Додаток демонструє практичне впровадження розподілених архітектурних патернів та принципів проектування розподілених систем.Отримано практичний досвід роботи з мережевим програмуванням, багатопоточністю та розподіленими системами.

Додаток А.
CentralServer.java

```
package com.project.downloadmanager.util.p2p;

import com.project.downloadmanager.model.entity.PeerFileInfo;

import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

public class CentralServer {

    private static final Map<String, Set<PeerFileInfo>> connectedPeers = new
ConcurrentHashMap<>();
    private static final int port = 5000;

    public CentralServer() {
        try {
            runAsServer();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void runAsServer() throws IOException {
        Map<String, List<PeerFileInfo>> fileRegistry = new ConcurrentHashMap<>();

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);
            startPeerCleanupThread(fileRegistry);

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Peer connected: " + socket.getInetAddress() + ":" +
socket.getPort());

                new Thread(() -> handlePeer(socket, fileRegistry)).start();
            }
        }
    }

    private static void startPeerCleanupThread(Map<String, List<PeerFileInfo>>
fileRegistry) {
        new Thread(() -> {
            while (true) {
                try {
                    Thread.sleep(5 * 60 * 1000);

                    synchronized (fileRegistry) {
                        for (String file : fileRegistry.keySet()) {
                            fileRegistry.get(file).removeIf(peerInfo ->
                                    !isPeerConnected(peerInfo.getAddress(),
peerInfo.getPort()));
                        }

                        fileRegistry.entrySet().removeIf(entry ->
entry.getValue().isEmpty());
                    }
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                }
            }
        }).start();
    }
```

```
    private static boolean isPeerConnected(String address, int port) {
        try (Socket socket = new Socket(address.split(":")[0], port)) {
            return true;
        } catch (IOException e) {
            return false;
        }
    }

    private static void handlePeer(Socket socket, Map<String, List<PeerFileInfo>>
fileRegistry) {
        String peerAddress = null;
        try (InputStream inputStream = socket.getInputStream();
             ObjectOutputStream objectOutputStream = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream objectInputStream = new ObjectInputStream(inputStream)) {

            String command = (String) objectInputStream.readObject();

            if ("REGISTER".equals(command)) {
                @SuppressWarnings("unchecked")
                List<String> files = (List<String>) objectInputStream.readObject();
                int peerPort = objectInputStream.readInt();
                peerAddress = socket.getInetAddress().getHostAddress() + ":" + peerPort;

                for (String file : files) {
                    PeerFileInfo peerInfo = new PeerFileInfo(peerAddress, peerPort);
                    fileRegistry.computeIfAbsent(file, k -> new
ArrayList<>()).add(peerInfo);
                }
                System.out.println("Registered files from peer " + peerAddress + ": " +
files);

                String finalPeerAddress = peerAddress;
                connectedPeers.computeIfAbsent(peerAddress, k -> new HashSet<>()).addAll(
                        files.stream().map(file -> new PeerFileInfo(finalPeerAddress,
peerPort)).toList()
                );
            } else if ("SEARCH".equals(command)) {
                String fileName = (String) objectInputStream.readObject();
                String requesterAddress = socket.getInetAddress().getHostAddress() + ":"
+
                        objectInputStream.readInt();

                List<PeerFileInfo> peers = fileRegistry.getOrDefault(fileName,
Collections.emptyList())
                        .stream()
                        .filter(p -> !p.getAddress().equals(requesterAddress))
                        .toList();

                objectOutputStream.writeObject(peers);
                objectOutputStream.flush();
                System.out.println("Search request for file '" + fileName + "' returned
peers: " + peers);
            } else if ("FILES".equals(command)) {
                String requesterAddress = socket.getInetAddress().getHostAddress() + ":"
+
                        objectInputStream.readInt();
                List<PeerFileInfo> allFiles = new ArrayList<>();
                fileRegistry.values().forEach(allFiles::addAll);

                List<PeerFileInfo> returnedFiles = new ArrayList<>();

                for (PeerFileInfo peerInfo : allFiles.stream().filter(p ->
!p.getAddress().equals(requesterAddress)).toList()) {
                    for (Map.Entry<String, List<PeerFileInfo>> entry :
fileRegistry.entrySet()) {
                        if (entry.getValue().contains(peerInfo)) {
                            if (!returnedFiles.contains(new PeerFileInfo(entry.getKey(),
peerInfo.getAddress(), peerInfo.getPort()))) {
                                returnedFiles.add(new PeerFileInfo(entry.getKey(),
peerInfo.getAddress(), peerInfo.getPort()));
```

```
                            }
                        }
                    }
                }

                objectOutputStream.writeObject(returnedFiles);
                objectOutputStream.flush();
            }
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error handling peer: " + e.getMessage());
        } finally {
            if (peerAddress != null) {
                connectedPeers.remove(peerAddress);
            }
        }
    }
}
```

Peer.java

```
package com.project.downloadmanager.util.p2p;

import com.project.downloadmanager.config.ConfigLoader;
import com.project.downloadmanager.model.entity.PeerFileInfo;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

public class Peer {

    static String shareDirectory = ConfigLoader.getP2PDirectory();
    static int peerPort = ConfigLoader.getP2PPort();
    static String serverAddress = ConfigLoader.getP2PServerAddress();
    static int port = ConfigLoader.getCentralPort();

    public Peer () {
        try {
            runAsPeer();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private void runAsPeer() throws IOException {
        File directory = new File(shareDirectory);
        if (!directory.exists() || !directory.isDirectory()) {
            System.err.println("Invalid directory. Exiting.");
            return;
        }
        startPeerServer(peerPort, shareDirectory);

        List<String> filesToShare =
Arrays.asList(Objects.requireNonNull(directory.list()));
        try (Socket socket = new Socket(serverAddress, port);
             ObjectOutputStream objectOutputStream = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream objectInputStream = new
ObjectInputStream(socket.getInputStream())) {

            objectOutputStream.writeObject("REGISTER");
            objectOutputStream.writeObject(filesToShare);
            objectOutputStream.writeInt(peerPort);
            objectOutputStream.flush();

            System.out.println("Files registered: " + filesToShare);
        } catch (Exception e) {
            throw new IOException();
        }
```

```
    }

    public static List<PeerFileInfo> getAllFiles() throws IOException{
        try (Socket socket = new Socket(serverAddress, port);
             ObjectOutputStream objectOutputStream = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream objectInputStream = new
ObjectInputStream(socket.getInputStream())) {

            objectOutputStream.writeObject("FILES");
            objectOutputStream.writeInt(peerPort);
            objectOutputStream.flush();

            @SuppressWarnings("unchecked")
            List<PeerFileInfo> peers = (List<PeerFileInfo>)
objectInputStream.readObject();

            if (peers.isEmpty()) {
                System.out.println("No files found.");
            }
            return peers;
        } catch (Exception e) {
            System.err.println("Error during search files: " + e.getMessage());
            throw new IOException(e);
        }
    }

    public static List<PeerFileInfo> searchFileByName(String fileName) throws IOException
{
        try (Socket socket = new Socket(serverAddress, port);
             ObjectOutputStream objectOutputStream = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream objectInputStream = new
ObjectInputStream(socket.getInputStream())) {

            objectOutputStream.writeObject("SEARCH");
            objectOutputStream.writeObject(fileName);
            objectOutputStream.writeInt(peerPort);
            objectOutputStream.flush();

            @SuppressWarnings("unchecked")
            List<PeerFileInfo> peers = (List<PeerFileInfo>)
objectInputStream.readObject();

            if (peers.isEmpty()) {
                System.out.println("No peers have the requested file.");
                throw new IOException("No peers have the requested file.");
            }
            return peers;
        } catch (Exception e) {
            System.err.println("Error during search or download: " + e.getMessage());
            throw new IOException(e);
        }
    }

    public static void downloadFileFromPeer(String peerAddress, int peerPort, String
fileName) {
        try (Socket socket = new Socket(peerAddress.split(":")[0], peerPort);
             DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());
             InputStream inputStream = socket.getInputStream()) {

            dataOutputStream.writeUTF(fileName);
            dataOutputStream.flush();

            File file = new File(shareDirectory, fileName);
            try (FileOutputStream fileOutputStream = new FileOutputStream(file)) {
                System.out.println("Downloading file...");
                byte[] buffer = new byte[4096];
                int bytesRead;
                while ((bytesRead = inputStream.read(buffer)) != -1) {
                    fileOutputStream.write(buffer, 0, bytesRead);
```

```
                }
                System.out.println("File downloaded successfully and saved to: " +
file.getAbsolutePath());
            }
        } catch (IOException e) {
            System.err.println("Error during file download: " + e.getMessage());
        }
    }

    private static void startPeerServer(int port, String shareDirectory) {
        new Thread(() -> {
            try (ServerSocket serverSocket = new ServerSocket(port)) {
                System.out.println("Peer server is listening on port " + port);

                while (true) {
                    try (Socket socket = serverSocket.accept();
                         DataInputStream dataInputStream = new
DataInputStream(socket.getInputStream());
                         OutputStream outputStream = socket.getOutputStream()) {

                        String requestedFile = dataInputStream.readUTF();
                        File file = new File(shareDirectory, requestedFile);

                        if (file.exists()) {
                            try (FileInputStream fileInputStream = new
FileInputStream(file)) {
                                System.out.println("Uploading file: " + requestedFile);
                                byte[] buffer = new byte[4096];
                                int bytesRead;
                                while ((bytesRead = fileInputStream.read(buffer)) != -1)
{
                                    outputStream.write(buffer, 0, bytesRead);
                                }
                            }
                            System.out.println("File uploaded successfully.");
                        } else {
                            System.err.println("Requested file not found: " +
requestedFile);
                        }
                    } catch (IOException e) {
                        System.err.println("Error handling file request: " +
e.getMessage());
                    }
                }
            } catch (IOException e) {
                System.err.println("Peer server error: " + e.getMessage());
            }
        }).start();
    }

}
```

ServerStart.java

```
package com.project.downloadmanager;

import com.project.downloadmanager.util.p2p.CentralServer;

public class ServerStart {
    public static void main(String[] args) {
        new CentralServer();
    }
}
```

PeerFileInfo.java

```
package com.project.downloadmanager.model.entity;

import com.project.downloadmanager.util.composite.DownloadGroup;
import lombok.*;

import java.io.Serializable;
import java.util.Random;
```

```
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Data
public class PeerFileInfo implements Serializable, DownloadGroup {
    Random random = new Random();

    private long id = Math.abs(random.nextLong());
    private String fileName;
    private String address;
    private int port;

    public PeerFileInfo(String address, int port) {
        this.address = address;
        this.port = port;
    }

    public PeerFileInfo(String fileName, String address, int port) {
        this.fileName = fileName;
        this.address = address;
        this.port = port;
    }

    public PeerFileInfo(long id,String fileName, String address, int port) {
        this.id = id;
        this.fileName = fileName;
        this.address = address;
        this.port = port;
    }

    @Override
    public void display() {

    }
}
```

DownloadManagerApplication.java (Основний код для запуску)

```
private void connectToP2PServer(Button button, Label label) {
    try {
        new Peer();
        label.setText("Статус підключення: Підключено");
        button.setDisable(false);
        showAlert(Alert.AlertType.INFORMATION, "Підключено", "Натисність кнопку refresh
для перевірки доступних файлів");
    } catch (RuntimeException e) {
        button.setDisable(true);
        label.setText("Статус підключення: Не підключено");
        showAlert(Alert.AlertType.WARNING, "Увага", "Сталася помилка при підключенні до
сервера: " + e.getMessage() + " Перевірте правильність адреси та порта серверу");
    }
}
public void refreshPeerFiles() {
    try {
        p2pListView.getItems().clear();
        List<PeerFileInfo> peerFileInfos = Peer.getAllFiles();
        peerFileInfos.forEach(this::setP2pListView);
        if (peerFileInfos.isEmpty()) {
            showAlert(Alert.AlertType.INFORMATION, "На сервері не найдено жодний файлів",
"Файлів не знайдено");
        }
    } catch (Exception e) {
        showAlert(Alert.AlertType.ERROR, "Помилка", "Неможливо найти доступні файли");
    }
}
private void setP2pListView(PeerFileInfo peerFileInfo) {
    Label fileName = new Label(peerFileInfo.getFileName() + " " +
peerFileInfo.getAddress() + " " + peerFileInfo.getPort());
    HBox p2pItem = new HBox(10);

    Button downloadButton = new Button("Завантажити");
```

```
    downloadButton.setPrefWidth(80);

    p2pItem.setPadding(new Insets(10));
    p2pItem.getChildren().addAll(fileName, downloadButton);
    p2pListView.getItems().add(p2pItem);

    downloadP2PButton(downloadButton,peerFileInfo);
}
private void downloadP2PButton(Button button, PeerFileInfo peerFileInfo) {
    button.setOnAction(event -> {
        try {
            Peer.downloadFileFromPeer(peerFileInfo.getAddress(), peerFileInfo.getPort(),
peerFileInfo.getFileName());
            button.setDisable(true);
            peerFileService.save(peerFileInfo);
        } catch (Exception e) {
            e.printStackTrace();
        }});
}
```