**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 4

**З дисципліни:** *“Кросплатформне програмування”*

**На тему:** *“Багатопоточність в Java, організація різних типів взаємодій між потоками”*

**Лектор:**

доц. каф. ПЗ

Дяконюк Л. М.

**Виконав:**

ст. гр. ПЗ-35

Хруставчук М.Л.

**Прийняв:**

ст. викл. каф. ПЗ

Шкраб Р. Р.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2024 р.

∑= \_\_\_\_\_ .

Львів – 2024

**Тема роботи:** Багатопоточність в Java, організація різних типів взаємодій між потоками.

**Мета роботи:** Вивчити механізми синхронізації та організації взаємодії між потоками.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Багатопоточність є важливою складовою комп’ютерних програм, що дозволяє одночасно виконувати кілька завдань або процесів у межах одного програмного додатку. У Java багатопоточність реалізується через використання потоків (threads), які можуть виконуватися паралельно або в рамках одночасного виконання на різних процесорних ядрах.

*Пакет java.util.concurrent:*

* Java надає потужні інструменти для роботи з багатопоточністю через пакет java.util.concurrent. Цей пакет включає в себе інтерфейси та класи для керування потоками і синхронізації, що дозволяють створювати більш ефективні та надійні багатопоточні програми.
* Потік (thread) - це незалежний шлях виконання в програмі, що має свою власну послідовність команд, змінні, стани й контекст. Кожен потік виконується незалежно від інших, і завдяки багатопоточності програма може одночасно виконувати кілька задач.

У Java для роботи з потоками є два основні підходи:

1. Наслідування від класу Thread.
2. Реалізація інтерфейсу Runnable.

*Створення потоку через наслідування від класу Thread*

Для створення нового потоку можна створити підклас класу Thread і перевизначити його метод run(). Потік запускається за допомогою виклику методу start().

Основні методи класу Thread:

* start() - ініціює виконання потоку, викликає метод run().
* run() - метод, що містить код, який буде виконаний в окремому потоці.
* sleep(long millis) - ставить потік на паузу на вказаний час.
* join() - потік зупиняється і чекає завершення іншого потоку.
* interrupt() - перериває виконання потоку.

*Інтерфейс Executor і класи ExecutorService*

Executor - це інтерфейс для управління потоками, який дозволяє делегувати виконання завдань без необхідності створювати потоки вручну. ExecutorService є підтипом цього інтерфейсу і надає додаткові методи для керування життєвим циклом потоків.

ExecutorService дозволяє керувати пулом потоків, обробляти завдання та контролювати їх виконання, що значно спрощує роботу з багатопоточністю, оскільки не потрібно вручну створювати і запускати потоки.

За допомогою методу submit() можна запускати завдання асинхронно і отримувати об’єкт Future, через який можна отримати результат виконання завдання. Це дозволяє продовжувати виконання програми, не чекаючи завершення потоку.

Основні методи:

* submit() - дозволяє передати завдання для виконання і отримати Future об’єкт, що можна використовувати для відстеження виконання.
* invokeAll() - виконує кілька завдань і повертає список Future для кожного завдання.
* invokeAny() - виконує набір завдань і повертає результат першого завдання, яке завершиться без помилок.

**ЗАВДАННЯ**

*Варіант 5. Динамічна обробка великих даних у системі прогнозування*

Умова**:** У вас є система прогнозування погоди, яка отримує великі масиви даних від різних сенсорних станцій. Кожна станція передає дані з різною частотою, і ваша задача - обробляти ці дані в режимі реального часу.

Завдання:

1. Використовуйте ScheduledThreadPoolExecutor для регулярного збору даних з кожної станції, налаштувавши різний інтервал для кожної станції (наприклад, одна станція передає дані кожну секунду, інша — кожні 5 секунд).
2. Обробляйте отримані дані в окремих потоках, використовуючи FixedThreadPool, де кожен потік аналізує та передає результати в головну систему для побудови прогнозу.
3. Реалізуйте можливість динамічно додавати нові станції в процесі виконання програми, а також видаляти станції зі списку без зупинки програми.

Додаткові умови: встановіть обмеження на кількість одночасно активних потоків для збирання даних, щоб уникнути перевантаження.

Реалізуйте механізм агрегації результатів у реальному часі, де система повинна оновлювати загальний прогноз на основі нових даних кожні 5 секунд.

**ХІД ВИКОНАННЯ**

**1. Код розробленої програми**

WeatherApplication.java

package org.example.weatherapplication;  
  
import javafx.application.Application;  
import javafx.application.Platform;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.Alert;  
import javafx.scene.layout.Pane;  
import javafx.scene.layout.TilePane;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.\*;  
import java.util.concurrent.\*;  
  
public class WeatherApplication extends Application {  
 private static final int *INITIAL\_STATION\_COUNT* = 1;  
 private final ScheduledThreadPoolExecutor scheduler = new ScheduledThreadPoolExecutor(6);  
 private final ExecutorService fixedThreadPool = Executors.*newFixedThreadPool*(6);  
 private final Map<Integer, StationController> stationControllers = new HashMap<>();  
 private final Map<Integer, List<int[]>> stationDataHistory = new HashMap<>();  
 private final Map<Integer, Integer> updateCounts = new HashMap<>();  
 private final Map<Integer, ScheduledFuture<?>> stationScheduledTasks = new HashMap<>();  
 private Map<Integer, Pane> stationPanes = new HashMap<>();  
 private TilePane stationContainer;  
 private Pane PrognosisPane;  
 private StationController PrognosisController;  
 private int currentStationId = 0;  
  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("Main.fxml"));  
 Pane root = fxmlLoader.load();  
  
 MainController mainController = fxmlLoader.getController();  
 mainController.setApp(this);  
  
 stationContainer = mainController.getStationContainer();  
 PrognosisPane = mainController.getOverallAveragePane();  
  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("Station.fxml"));  
 Pane overallPane = loader.load();  
 PrognosisController = loader.getController();  
  
 PrognosisController.setStationName("Weather Prognosis");  
 PrognosisPane.getChildren().add(overallPane);  
 PrognosisPane.setUserData(PrognosisController);  
  
 for (int i = 0; i < *INITIAL\_STATION\_COUNT*; i++) {  
 addStation();  
 }  
  
 scheduler.scheduleAtFixedRate(() -> Platform.*runLater*(this::updateOverallAveragePane), 0, 10, TimeUnit.*SECONDS*);  
  
 stage.setTitle("Weather Application");  
 stage.setMaximized(true);  
 Scene scene = new Scene(root);  
 scene.getStylesheets().add(getClass().getResource("/styles/style.css").toExternalForm()); // Підключення CSS  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 public void addStation() {  
 if (stationControllers.size() >= 6) {  
 Alert alert = new Alert(Alert.AlertType.*WARNING*);  
 alert.setTitle("Station Limit Reached");  
 alert.setHeaderText(null);  
 alert.setContentText("Cannot add more than 6 stations. The maximum number of stations has been reached.");  
 alert.showAndWait();  
 return;  
 }  
  
 try {  
 int stationId = currentStationId++;  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("Station.fxml"));  
 Pane stationPane = loader.load();  
 StationController controller = loader.getController();  
 controller.setStationId(stationId);  
 controller.setApp(this);  
 stationContainer.getChildren().add(stationPane);  
  
 // Додаємо до мапи панелей  
 stationPanes.put(stationId, stationPane);  
  
 stationControllers.put(stationId, controller);  
 stationDataHistory.put(stationId, new ArrayList<>());  
 updateCounts.put(stationId, 0);  
 WeatherStation station = new WeatherStation();  
 controller.setStationName("Station " + (stationId + 1));  
  
 scheduleStationUpdates(stationId, station, controller);  
  
 System.*out*.println("Station " + stationId + " added.");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 public void removeStation(int stationId) {  
 if (!stationControllers.containsKey(stationId)) {  
 System.*out*.println("Station with ID " + stationId + " not found.");  
 return;  
 }  
  
 // Зупиняємо оновлення станції  
 stopStationUpdates(stationId);  
  
 // Отримуємо панель станції  
 Pane stationPane = stationPanes.get(stationId);  
 if (stationPane != null) {  
 Platform.*runLater*(() -> {  
 stationPane.getStyleClass().add("removed-station");  
 System.*out*.println("Station " + stationId + " marked as removed.");  
 });  
 }  
  
 // Видаляємо станцію з колекцій  
 stationControllers.remove(stationId);  
 stationDataHistory.remove(stationId);  
 updateCounts.remove(stationId);  
 stationPanes.remove(stationId);  
  
 System.*out*.println("Station " + stationId + " removed.");  
 }  
  
  
 private void stopStationUpdates(int stationId) {  
 // Перевіряємо, чи є заплановане завдання для станції  
 ScheduledFuture<?> future = stationScheduledTasks.get(stationId);  
 if (future != null) {  
 future.cancel(true);  
 stationScheduledTasks.remove(stationId);  
 }  
 }  
  
 private void scheduleStationUpdates(int stationId, WeatherStation station, StationController controller) {  
 // Генеруємо інтервал у діапазоні від 1 до 3 секунд  
 int intervalSeconds = new Random().nextInt(3) + 1;  
  
 // Плануємо завдання  
 ScheduledFuture<?> future = scheduler.scheduleAtFixedRate(() -> {  
 station.generateData();  
 Platform.*runLater*(() -> controller.setData(  
 String.*valueOf*(station.getTemperature()),  
 String.*valueOf*(station.getHumidity()),  
 String.*valueOf*(station.getWindSpeed()),  
 String.*valueOf*(station.getVisibility()),  
 String.*valueOf*(station.getPressure()),  
 String.*valueOf*(station.getCloudCover())  
 ));  
  
 storeDataAndCalculateAverage(stationId, station);  
 }, 0, intervalSeconds, TimeUnit.*SECONDS*);  
  
 // Зберігаємо завдання в картці  
 stationScheduledTasks.put(stationId, future);  
 }  
  
  
 private void storeDataAndCalculateAverage(int stationId, WeatherStation station) {  
 int[] currentData = {  
 station.getTemperature(),  
 station.getHumidity(),  
 station.getWindSpeed(),  
 station.getPressure(),  
 station.getVisibility(),  
 station.getCloudCover()  
 };  
  
 synchronized (stationDataHistory) {  
 stationDataHistory.get(stationId).add(currentData);  
 int updates = updateCounts.get(stationId) + 1;  
 updateCounts.put(stationId, updates);  
  
 if (updates == 6) {  
 updateCounts.put(stationId, 0);  
 List<int[]> dataHistory = new ArrayList<>(stationDataHistory.get(stationId));  
 stationDataHistory.get(stationId).clear();  
  
 // Використовуємо Future для обчислення середнього  
 Future<?> future = fixedThreadPool.submit(() -> {  
 long startTime = System.*nanoTime*(); // Початок вимірювання часу  
 calculateAndLogAverage(stationId, dataHistory);  
 long endTime = System.*nanoTime*(); // Кінець вимірювання часу  
  
 long executionTime = endTime - startTime;  
 System.*out*.println("Execution time for station " + (stationId + 1) + ": " + (executionTime / 1\_000\_000) + "ms");  
 });  
  
 try {  
 // Чекаємо результат обчислення  
 future.get();  
 System.*out*.println("Average calculation completed for station " + (stationId + 1 ));  
 } catch (InterruptedException | ExecutionException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 private final Map<Integer, int[]> stationAverages = new ConcurrentHashMap<>();  
  
 private void calculateAndLogAverage(int stationId, List<int[]> dataHistory) {  
 int[] totals = new int[6];  
 for (int[] data : dataHistory) {  
 for (int i = 0; i < data.length; i++) {  
 totals[i] += data[i];  
 }  
 }  
  
 int dataCount = dataHistory.size();  
 int[] averages = Arrays.*stream*(totals).map(total -> total / dataCount).toArray();  
  
 stationAverages.put(stationId, averages);  
  
 System.*out*.println("Station " + (stationId+1) + " Averages: " +  
 "Temperature=" + averages[0] + "°C, " +  
 "Humidity=" + averages[1] + "%, " +  
 "Wind Speed=" + averages[2] + " m/s, " +  
 "Pressure=" + averages[3] + " hPa, " +  
 "Visibility=" + averages[4] + " meters, " +  
 "Cloud Cover=" + averages[5] + "%");  
 }  
  
 private void updateOverallAveragePane() {  
 long startTime = System.*nanoTime*();  
 int stationCount = stationAverages.size();  
 if (stationCount == 0) return;  
  
 double[] totalAverages = new double[6];  
 for (int[] averages : stationAverages.values()) {  
 for (int i = 0; i < averages.length; i++) {  
 totalAverages[i] += averages[i];  
 }  
 }  
  
 for (int i = 0; i < totalAverages.length; i++) {  
 totalAverages[i] /= stationCount;  
 }  
  
 PrognosisController.setData(  
 String.*format*("%.0f", totalAverages[0]),  
 String.*format*("%.0f", totalAverages[1]),  
 String.*format*("%.0f", totalAverages[2]),  
 String.*format*("%.0f", totalAverages[4]),  
 String.*format*("%.0f", totalAverages[3]),  
 String.*format*("%.0f", totalAverages[5])  
 );  
  
 long endTime = System.*nanoTime*(); // Кінець вимірювання часу  
 long executionTime = endTime - startTime;  
  
 System.*out*.println("Overall average calculation time: " + (executionTime / 1\_000\_000) + " ms");  
 }  
  
 @Override  
 public void stop() throws Exception {  
 super.stop();  
 scheduler.shutdown();  
 fixedThreadPool.shutdown();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

MainController.java

package org.example.weatherapplication;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.Pane;  
import javafx.scene.layout.TilePane;  
import java.util.HashMap;  
import java.util.Map;  
  
public class MainController {  
 private WeatherApplication app;  
  
 @FXML  
 private TextField stationIdField;  
 @FXML  
 private TilePane stationContainer;  
  
 @FXML  
 private Pane overallAveragePane;  
  
 public void setApp(WeatherApplication app) {  
 this.app = app;  
 }  
  
 public TilePane getStationContainer() {  
 return stationContainer;  
 }  
  
 public Pane getOverallAveragePane() {  
 return overallAveragePane;  
 }  
  
 // Мапа для зберігання панелей станцій, де ключ - це ID станції  
 private final Map<Integer, Pane> stationPanels = new HashMap<>();  
  
  
 @FXML  
 private void addStation() {  
 if (app != null) {  
 app.addStation();  
  
 }  
 }  
  
 @FXML  
 public void handleRemoveStation() {  
 // Отримуємо ID станції з текстового поля  
 String idText = stationIdField.getText();  
 if (idText.isEmpty()) {  
 System.out.println("Please enter a station ID.");  
 return;  
 }  
  
 try {  
 int stationId = Integer.parseInt(idText);  
 app.removeStation(stationId); // Видалення станції  
 } catch (NumberFormatException e) {  
 System.out.println("Invalid ID format. Please enter a valid integer.");  
 }  
 }  
}

StationController.java

package org.example.weatherapplication;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import javafx.scene.layout.Pane;  
  
public class StationController {  
 private int stationId;  
 private WeatherApplication app;  
 private Pane stationPane;  
  
 public void handleRemoveStation() {  
 app.removeStation(stationId); // Викликаємо метод для видалення станції  
 }  
  
 @FXML  
 private Label stationNameLabel;  
  
 @FXML  
 private Label temperatureLabel;  
  
 @FXML  
 private Label humidityLabel;  
  
 @FXML  
 private Label windSpeedLabel;  
  
 @FXML  
 private Label visibilityLabel;  
  
 @FXML  
 private Label pressureLabel;  
  
 @FXML  
 private Label cloudCoverLabel;  
  
 public void setStationName(String name) {  
 stationNameLabel.setText(name);  
 }  
  
 public void setData(String temperature, String humidity, String windSpeed, String visibility, String pressure, String cloudCover) {  
 temperatureLabel.setText(temperature + " °C");  
 humidityLabel.setText(humidity + "%");  
 windSpeedLabel.setText(windSpeed + " m/s");  
 visibilityLabel.setText(visibility + " meters");  
 pressureLabel.setText(pressure + " hPa");  
 cloudCoverLabel.setText(cloudCover + "%");  
 }  
  
 public void setStationId(int stationId) {  
 this.stationId = stationId;  
 }  
  
 public int getStationId() {  
 return stationId;  
 }  
  
 public void setApp(WeatherApplication app) {  
 this.app = app;  
 }  
  
 public Pane getPane() {  
 return stationPane;  
 }  
}

WeatherStation.java

package org.example.weatherapplication;  
  
import java.util.Random;  
  
public class WeatherStation {  
 private int temperature;  
 private int humidity;  
 private int pressure;  
 private int windSpeed;  
 private int visibility;  
 private int cloudCover;  
  
 private final Random random = new Random();  
  
 public WeatherStation() {  
 generateData();  
 }  
  
 // Метод для оновлення даних станції  
 public void generateData() {  
 temperature = random.nextInt(36) - 10;  
 humidity = random.nextInt(50) + 30;  
 pressure = random.nextInt(100) + 950;  
 windSpeed = random.nextInt(20) + 1;  
 visibility = random.nextInt(9000) + 1000;  
 cloudCover = random.nextInt(100);  
 }  
  
 public int getTemperature() {  
 return temperature;  
 }  
  
 public int getHumidity() {  
 return humidity;  
 }  
  
 public int getPressure() {  
 return pressure;  
 }  
  
 public int getWindSpeed() {  
 return windSpeed;  
 }  
  
 public int getVisibility() {  
 return visibility;  
 }  
  
 public int getCloudCover() {  
 return cloudCover;  
 }  
  
 @Override  
 public String toString() {  
 return String.format("Temperature: %d°C, Humidity: %d%%, Pressure: %dhPa, Wind: %d m/s, Visibility: %d meters, Cloud Cover: %d%%",  
 temperature, humidity, pressure, windSpeed, visibility, cloudCover);  
 }  
}

Main.fxml

<?xml version="1.0" encoding="UTF-8"?>  
<?import javafx.scene.layout.BorderPane?>  
<?import javafx.scene.layout.TilePane?>  
<?import javafx.scene.layout.HBox?>  
<?import javafx.scene.layout.VBox?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.TextField?>  
<?import javafx.scene.control.Label?>  
<?import javafx.geometry.Insets?>  
  
<BorderPane xmlns="http://javafx.com/javafx/17"  
 xmlns:fx="http://javafx.com/fxml/1"  
 fx:controller="org.example.weatherapplication.MainController"  
 style="-fx-background-color: #F4F4F4;">  
  
 <top>  
 <HBox spacing="10" alignment="CENTER\_LEFT" style="-fx-padding: 10;">  
 <Label text="Weather Station Management" style="-fx-font-size: 20px; -fx-font-weight: bold;" />  
 <Button text="Add Station" onAction="#addStation" styleClass="button" style="-fx-background-color: #4CAF50; -fx-text-fill: white; -fx-font-weight: bold;"/>  
 </HBox>  
 </top>  
  
 <center>  
 <VBox spacing="20" alignment="CENTER" style="-fx-padding: 20;">  
 <TilePane fx:id="stationContainer" hgap="15" vgap="15"  
 alignment="TOP\_CENTER" prefTileWidth="200" prefTileHeight="200"  
 style="-fx-background-color: #FFFFFF; -fx-border-color: #BDBDBD; -fx-border-width: 1px; -fx-padding: 10;">  
 </TilePane>  
  
 <VBox fx:id="overallAveragePane" alignment="CENTER" spacing="10"  
 styleClass= "overall-average-pane">  
 <Label text="Overall Averages" style="-fx-font-size: 18px; -fx-font-weight: bold;"/>  
 </VBox>  
  
 <VBox spacing="10" alignment="CENTER" style="-fx-background-color: #FFFFFF; -fx-padding: 15; -fx-border-color: #BDBDBD; -fx-border-width: 1px;">  
 <Label text="Remove Station" style="-fx-font-size: 16px; -fx-font-weight: bold;"/>  
 <TextField fx:id="stationIdField" promptText="Enter Station ID to Remove"  
 style="-fx-background-color: #F8F9FA; -fx-border-color: #BDBDBD; -fx-border-radius: 5px; -fx-padding: 5;"/>  
 <Button text="Remove" onAction="#handleRemoveStation" styleClass="button"  
 style="-fx-background-color: #F44336; -fx-text-fill: white; -fx-font-weight: bold;"/>  
 </VBox>  
 </VBox>  
 </center>  
</BorderPane>  
Station.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
  
<VBox xmlns="http://javafx.com/javafx/17"  
 xmlns:fx="http://javafx.com/fxml/1"  
 fx:controller="org.example.weatherapplication.StationController"  
 styleClass="station-card"  
 alignment="TOP\_LEFT" spacing="20" prefWidth="250" prefHeight="250">  
 <Label fx:id="stationNameLabel" text="Weather Station" style="-fx-font-size: 18px; -fx-font-weight: bold;"/>  
 <GridPane hgap="10" vgap="10">  
  
 <Label text="Temperature:" styleClass="field-label" GridPane.rowIndex="0" GridPane.columnIndex="0"/>  
 <Label fx:id="temperatureLabel" text="" styleClass="field-value" GridPane.rowIndex="0" GridPane.columnIndex="1"/>  
  
 <Label text="Humidity:" styleClass="field-label" GridPane.rowIndex="1" GridPane.columnIndex="0"/>  
 <Label fx:id="humidityLabel" text="" styleClass="field-value" GridPane.rowIndex="1" GridPane.columnIndex="1"/>  
  
 <Label text="Wind Speed:" styleClass="field-label" GridPane.rowIndex="2" GridPane.columnIndex="0"/>  
 <Label fx:id="windSpeedLabel" text="" styleClass="field-value" GridPane.rowIndex="2" GridPane.columnIndex="1"/>  
  
 <Label text="Visibility:" styleClass="field-label" GridPane.rowIndex="3" GridPane.columnIndex="0"/>  
 <Label fx:id="visibilityLabel" text="" styleClass="field-value" GridPane.rowIndex="3" GridPane.columnIndex="1"/>  
  
 <Label text="Pressure:" styleClass="field-label" GridPane.rowIndex="4" GridPane.columnIndex="0"/>  
 <Label fx:id="pressureLabel" text="" styleClass="field-value" GridPane.rowIndex="4" GridPane.columnIndex="1"/>  
  
 <Label text="Cloud Cover:" styleClass="field-label" GridPane.rowIndex="5" GridPane.columnIndex="0"/>  
 <Label fx:id="cloudCoverLabel" text="" styleClass="field-value" GridPane.rowIndex="5" GridPane.columnIndex="1"/>  
  
 </GridPane>  
</VBox>

**2. Результати виконання розробленої програми**

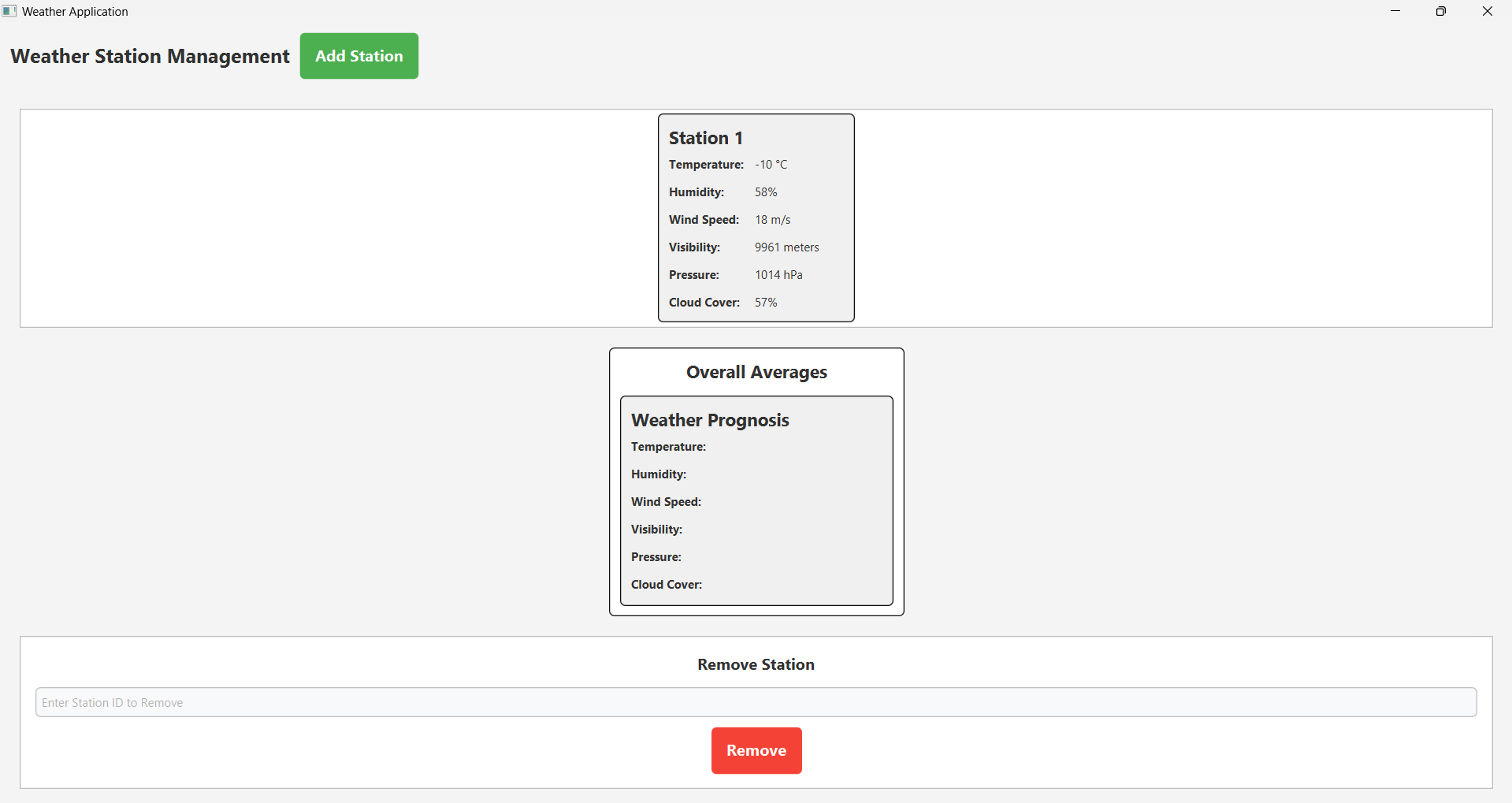


Рис. 1. Вікно програми при запуску програми

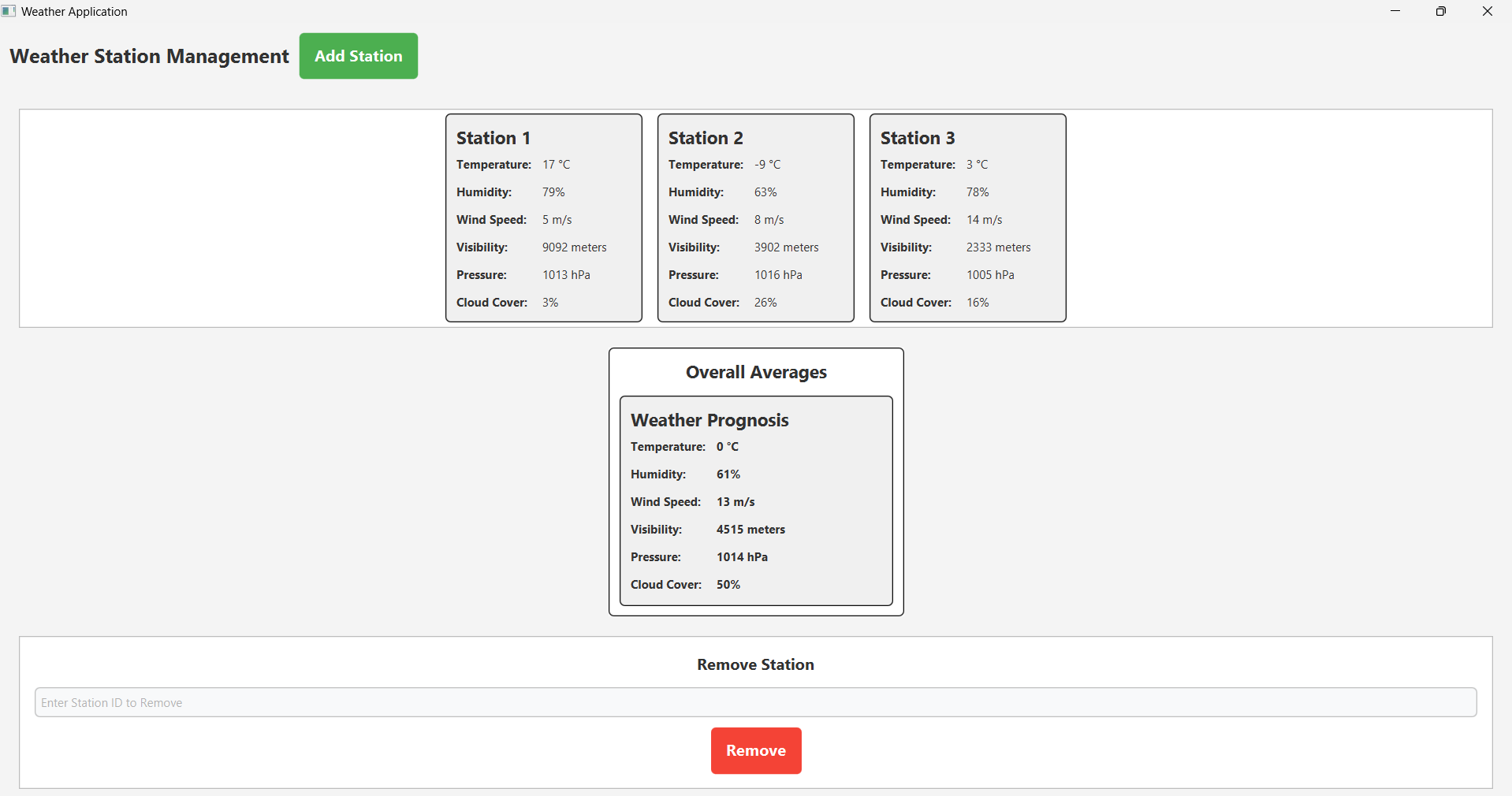


Рис. 2. Можливість створення нових потоків

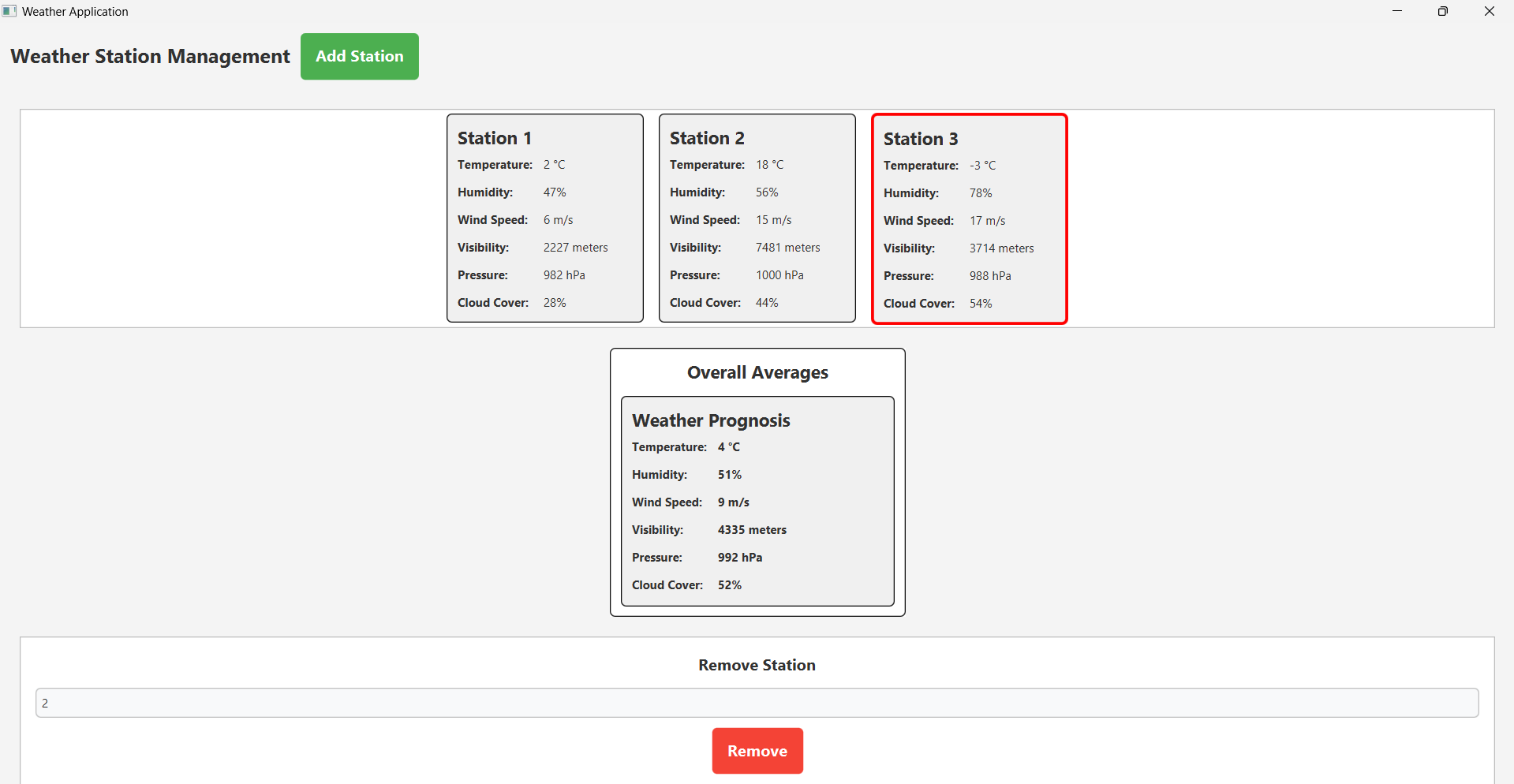


Рис. 3. Можливість зупинки потоку

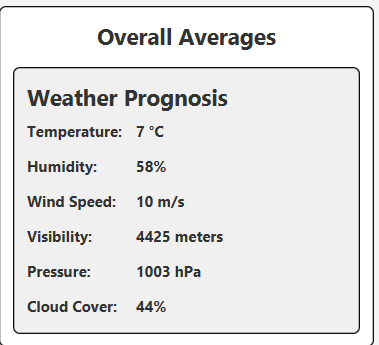


Рис. 4. Середнє значення із всіх станцій

**ВИСНОВКИ**

У ході виконання лабораторної роботи я вивчив механізми синхронізації та організації взаємодії між потоками. Написав програму, яка симулює прогноз погоди із використанням багатопоточності. Також реалізував інтерфейс за допомогою JavaFX.