**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №4

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

**з дисципліни** *“Кросплатформне програмування”*

**Лектор:**

доц. каф. ПЗ

Дяконюк Л. М.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

доц. каф. ПЗ

Баштовий А. В.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

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

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

# Теоретичні 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.

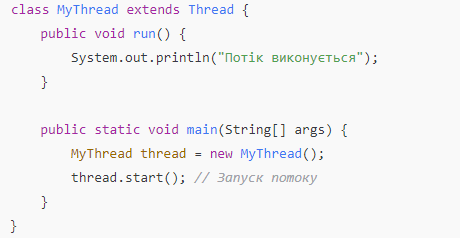
**2. Створення та запуск потоків**

У Java для роботи з потоками є два основні підходи:

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

**2.1. 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.

**2.2. Створення потоку через реалізацію інтерфейсу Runnable**

Інший спосіб створення потоку - реалізувати інтерфейс Runnable, що має єдиний метод run(). Потім потік можна передати в об'єкт класу Thread:

****

# Завдання

Потрібно реалізувати багатопотоковий застосунок з використанням пулу потоків, зазначеного в завданні.

1. Проаналізуйте, чи запропонований стандартний пул підходить для реалізації задачі.
2. Використовуйте механізм Future для отримання результатів від кожного потоку.
3. Оцінюйте, як зміна кількості потоків впливає на загальну продуктивність.
4. Продемонструйте задачу на прикладі з використанням невеликої кількості потоків.
5. Реалізуйте графічний інтерфейс користувача (на вибір студента Java FX або Swing).
   1. У додатку користувач має мати змогу задавати кількість потоків. Для роботи кожного потоку повинна бути створена панель. На панелі відображати ім’я потоку та його стан. Після завершення роботи кожного потоку відобразіть результат виконання завдання потоку.
   2. Стан виконання в головному потоці теж повинен контролюватися.
   3. Покажіть час виконання для кожного потоку та сумарний час виконання завдання.

**1. Завдання: Реалізація паралельної системи машинного навчання для розподіленого навчання моделі**

**Умова:** Уявіть, що ви розробляєте систему для паралельного навчання моделі машинного навчання. Ваше завдання – розподілити процес навчання на кілька частин:

* Дані розділяються на батчі, кожен з яких передається в окремий потік для обробки.
* Після обробки кожен потік передає результати в головний потік для агрегації та оновлення параметрів моделі.

**Завдання:**

1. Використайте FixedThreadPool для обробки батчів паралельно.
2. В кожному потоці симулюйте навчання частини даних і поверніть оновлені параметри в головний потік.
3. Організуйте механізм агрегації, щоб результати з кожного потоку об’єднувалися, а навчальна модель оновлювалася глобально

* Використовуйте механізм Future для отримання результатів від кожного потоку.

Оцінюйте, як зміна кількості потоків впливає на загальну продуктивність.

# Хід роботи

pom.xml:

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>main.javafx</groupId>  
 <artifactId>ThreadPools</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <name>ThreadPools</name>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <junit.version>5.10.2</junit.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-controls</artifactId>  
 <version>17.0.6</version>  
 </dependency>  
 <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-fxml</artifactId>  
 <version>17.0.6</version>  
 </dependency>  
 <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-media</artifactId>  
 <version>17.0.6</version>  
 </dependency>  
 <dependency>  
 <groupId>org.controlsfx</groupId>  
 <artifactId>controlsfx</artifactId>  
 <version>11.2.1</version>  
 </dependency>  
 <dependency>  
 <groupId>com.dlsc.formsfx</groupId>  
 <artifactId>formsfx-core</artifactId>  
 <version>11.6.0</version>  
 <exclusions>  
 <exclusion>  
 <groupId>org.openjfx</groupId>  
 <artifactId>\*</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <dependency>  
 <groupId>com.github.almasb</groupId>  
 <artifactId>fxgl</artifactId>  
 <version>17.3</version>  
 <exclusions>  
 <exclusion>  
 <groupId>org.openjfx</groupId>  
 <artifactId>\*</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>${junit.version}</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-engine</artifactId>  
 <version>${junit.version}</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.13.0</version>  
 <configuration>  
 <source>23</source>  
 <target>23</target>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-maven-plugin</artifactId>  
 <version>0.0.8</version>  
 <executions>  
 <execution>  
 <!-- Default configuration for running with: mvn clean javafx:run -->  
 <id>default-cli</id>  
 <configuration>  
 <mainClass>main.javafx.threadpools/main.javafx.threadpools.HelloApplication</mainClass>  
 <launcher>app</launcher>  
 <jlinkZipName>app</jlinkZipName>  
 <jlinkImageName>app</jlinkImageName>  
 <noManPages>true</noManPages>  
 <stripDebug>true</stripDebug>  
 <noHeaderFiles>true</noHeaderFiles>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
</project>

main.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.layout.VBox?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.layout.HBox?>  
<?import javafx.scene.control.TextField?>  
<?import javafx.scene.control.ScrollPane?>  
<?import javafx.scene.control.TextArea?>  
  
<VBox spacing="10" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="main.javafx.threadpools.MainController">  
 <HBox spacing="10">  
 <Label text="Кількість потоків:" />  
 <TextField fx:id="threadCountField" promptText="Введіть кількість потоків" />  
 <Button text="Запустити" onAction="#startProcessing" />  
 </HBox>  
 <Label text="Стан Завдань:" />  
 <ScrollPane fx:id="threadsPane" prefHeight="300" prefWidth="600">  
 <VBox fx:id="threadsBox" spacing="10" />  
 </ScrollPane>  
 <Label text="Результати:" />  
 <TextArea fx:id="resultArea" prefHeight="200" editable="false" />  
</VBox>

MainApplication.java:

// src/main/java/main/javafx/Application.java  
package main.javafx.threadpools;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class MainApplication extends Application {  
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 primaryStage.setTitle("Багатопоточна система навчання");  
 primaryStage.setScene(new Scene(FXMLLoader.*load*(getClass().getResource("main.fxml"))));  
 primaryStage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

MainController.java:

// src/main/java/main/javafx/Controller.java  
package main.javafx.threadpools;  
  
import javafx.application.Platform;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.layout.VBox;  
import main.util.ParallelTrainer;  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class MainController {  
 @FXML private TextField threadCountField;  
 @FXML private VBox threadsBox;  
 @FXML private TextArea resultArea;  
  
 private ExecutorService executor;  
  
 @FXML  
 public void startProcessing() {  
 // Отримання кількості потоків  
 int threadCount;  
 try {  
 threadCount = Integer.*parseInt*(threadCountField.getText());  
 } catch (NumberFormatException e) {  
 resultArea.setText("Помилка: Введіть правильне число потоків.");  
 return;  
 }  
  
 // Очищення перед запуском  
 threadsBox.getChildren().clear();  
 resultArea.clear();  
  
 // Ініціалізація обробки  
 executor = Executors.*newFixedThreadPool*(threadCount);  
 double[][] dataBatches = generateDataBatches(5, 10); // 5 батчів по 10 елементів  
 int weightSize = 10; // Розмір глобальних вагів  
  
 // Запуск тренування у фоновому потоці  
 new Thread(() -> {  
 try {  
 ParallelTrainer trainer = new ParallelTrainer(threadCount, threadsBox, resultArea);  
 trainer.trainModel(dataBatches, weightSize);  
 } catch (InterruptedException e) {  
 Platform.*runLater*(() -> resultArea.setText("Тренування було перервано."));  
 } finally {  
 executor.shutdown();  
 }  
 }).start();  
 }  
  
 // Генерація даних для навчання  
 private double[][] generateDataBatches(int batchCount, int batchSize) {  
 double[][] data = new double[batchCount][batchSize];  
 for (int i = 0; i < batchCount; i++) {  
 for (int j = 0; j < batchSize; j++) {  
 data[i][j] = Math.*random*();  
 }  
 }  
 return data;  
 }  
}

ParallelTrainer.java:

// src/main/java/main/util/ParallelTrainer.java  
package main.util;  
  
import javafx.application.Platform;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextArea;  
import javafx.scene.layout.VBox;  
import main.model.BatchProcessor;  
import main.model.ModelAggregator;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.\*;  
  
public class ParallelTrainer {  
 private final int threadCount;  
 private final VBox threadsBox;  
 private final TextArea resultArea;  
 private final ExecutorService executor;  
  
 public ParallelTrainer(int threadCount, VBox threadsBox, TextArea resultArea) {  
 this.threadCount = threadCount;  
 this.threadsBox = threadsBox;  
 this.resultArea = resultArea;  
 this.executor = Executors.*newFixedThreadPool*(threadCount);  
 }  
  
 public void trainModel(double[][] dataBatches, int weightSize) throws InterruptedException {  
 long startTime = System.*currentTimeMillis*();  
  
 List<Future<double[]>> futures = new ArrayList<>();  
 ModelAggregator aggregator = new ModelAggregator(weightSize);  
  
 // Створення віджетів для потоків  
 List<Label> threadLabels = new ArrayList<>();  
 for (int i = 0; i < dataBatches.length; i++) {  
 Label label = new Label("Завдання " + i + ": Очікування...");  
 Platform.*runLater*(() -> threadsBox.getChildren().add(label));  
 threadLabels.add(label);  
 }  
  
 // Відправка завдань на виконання  
 for (int i = 0; i < dataBatches.length; i++) {  
 int finalI = i;  
 futures.add(executor.submit(() -> {  
 Platform.*runLater*(() -> threadLabels.get(finalI).setText("Завдання " + finalI + ": Виконується..."));  
 BatchProcessor processor = new BatchProcessor(finalI, dataBatches[finalI]);  
 double[] result = processor.call();  
 Platform.*runLater*(() -> threadLabels.get(finalI).setText("Завдання " + finalI + ": Завершено."));  
 return result;  
 }));  
 }  
  
 List<double[]> batchResults = new ArrayList<>();  
  
 // Збір результатів  
 for (Future<double[]> future : futures) {  
 try {  
 batchResults.add(future.get());  
 } catch (ExecutionException e) {  
 Platform.*runLater*(() -> resultArea.appendText("Помилка під час обробки: " + e.getMessage() + "\n"));  
 }  
 }  
  
 // Агрегація результатів  
 aggregator.aggregateResults(batchResults);  
  
 long endTime = System.*currentTimeMillis*();  
 Platform.*runLater*(() -> {  
 resultArea.appendText("Загальний час виконання: " + (endTime - startTime) + " мс\n");  
 resultArea.appendText("Глобальні ваги: " + aggregator.getGlobalWeightsAsString() + "\n");  
 });  
  
 // Завершення виконання  
 executor.shutdown();  
 }  
}

ModelAggregator.java:

// src/main/java/main/model/ModelAggregator.java  
package main.model;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.concurrent.ExecutionException;  
  
public class ModelAggregator {  
 private double[] globalWeights;  
  
 public ModelAggregator(int weightSize) {  
 this.globalWeights = new double[weightSize];  
 }  
  
 // Агрегація результатів  
 public void aggregateResults(List<double[]> batchResults) {  
 for (double[] batchWeights : batchResults) {  
 for (int i = 0; i < globalWeights.length; i++) {  
 globalWeights[i] += batchWeights[i];  
 }  
 }  
 normalizeWeights();  
 }  
  
 // Нормалізація глобальних вагів (усереднення)  
 private void normalizeWeights() {  
 for (int i = 0; i < globalWeights.length; i++) {  
 globalWeights[i] /= globalWeights.length;  
 }  
 }  
  
 // Вивід глобальних вагів  
 public void printGlobalWeights() {  
 System.*out*.println("Global weights: " + Arrays.*toString*(globalWeights));  
 }  
  
 public String getGlobalWeightsAsString() {  
 return Arrays.*toString*(globalWeights);  
 }  
}

BatchProcessor.java:

// src/main/java/main/model/BatchProcessor.java  
package main.model;  
  
import java.util.Random;  
import java.util.concurrent.Callable;  
  
// Симуляція обробки одного батчу  
public class BatchProcessor implements Callable<double[]> {  
 private final int batchId;  
 private final double[] batchData;  
  
 public BatchProcessor(int batchId, double[] batchData) {  
 this.batchId = batchId;  
 this.batchData = batchData;  
 }  
  
 @Override  
 public double[] call() throws Exception {  
 System.*out*.println("Thread " + Thread.*currentThread*().getName() + " processing batch " + batchId);  
  
 // Симуляція "навчання" — обчислення  
 double[] updatedWeights = new double[batchData.length];  
 Random random = new Random();  
 for (int i = 0; i < batchData.length; i++) {  
 updatedWeights[i] = batchData[i] + random.nextDouble(); // Оновлення вагів  
 }  
  
 // Симуляція часу навчання  
 Thread.*sleep*(1000 + random.nextInt(500));  
  
 System.*out*.println("Thread " + Thread.*currentThread*().getName() + " completed batch " + batchId);  
 return updatedWeights;  
 }  
}

module-info.java:

module main.javafx.threadpools {  
 requires javafx.controls;  
 requires javafx.fxml;  
  
 requires org.controlsfx.controls;  
 requires com.dlsc.formsfx;  
 requires com.almasb.fxgl.all;  
  
 opens main.javafx.threadpools to javafx.fxml;  
 exports main.javafx.threadpools;  
}

Додаю [посилання](https://github.com/Akmitliviy/CPP) на GitHub репозиторій.

**Висновки:** Протягом виконання лабораторної роботи я вивчив механізми синхронізації та організації взаємодії між потоками.