**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

**ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №2

**на тему:** “ *Робота з текстовою інформацією”*

# з дисципліни: *“*Кросплатформне програмування*”*

**Лектор:**

доц. кафедри ПЗ

Дяконюк Л. М.

**Виконав:**

студент групи ПЗ-33

Шлом’як Д. В.

**Прийняв:**

доц. кафедри ПЗ

Баштовий А. В.

Львів – 2024

**Тема.** Робота з текстовою інформацією.

**Мета.** Навчитися використовувати основний інструментарій мови для пошуку, зміни, форматування текстової інформації, зокрема здобути навички роботи з датами та часовими змінними в різних часових зонах.

**Теоретичні відомості**

Важливе місце в обробці інформації займає робота з текстами. Як і багато чого, текстові рядки в мові Java являються об’єктами. Вони представляються екземплярами класу *String* або класу *StringBuffer*.

Класи *String, StringBuffer, StringBuilder* визначені в пакеті java.lang і доступні автоматично без оголошення імпорту. Всі три класи реалізують інтерфейс CharSequence.

На відміну від *String*, об'єкти типу *StringBuffer і StringBuilder* можуть бути змінені, вставляючи і додаючи рядки і символи. В багатьох випадках, коли треба змінити довжину рядка типу *String,* компілятор Java неявно перетворює його до типу *StringBuffer*, змінює довжину, потім перетворює назад в тип String.

**Конструктори класу String**

1. String() – створюється об’єкт з пустим рядком;
2. String (String str) – із одного об’єкта створюється інший, тому цей конструктор використовується рідко;
3. String (StringBuffer str) – перетворена копія об’єкту класу BufferString;
4. String(byte[] byteArray) – об’єкт створюється із масиву байтів byteArray;
5. String (char [] charArray) – об’єкт створюється із масиву charArray символів Unicode;
6. String (byte [] byteArray, int offset, int count) – об’єкт створюється із частини масиву байтів byteArray, починається з індексу offset і містить count байтів;
7. String (char [] charArray, int offset, int count) – те ж саме, але масив складається із символів Unicode;
8. String(byte[] byteArray, String encoding) – символи, записані в масиві байтів, задаються в Unicode-рядку з врахування кодування encoding;
9. String(byte[] byteArray, int offset, int count, String encoding) – те ж саме, але тільки для частини масиву.

**Завдання**

1. **Варіант 5:** Розробіть програму для аналізу та прогнозування курсів валют на основі історичних даних. Програма повинна завантажувати дані про курси валют з відкритих джерел, а потім аналізувати ці дані для виявлення тенденцій та можливих прогнозів майбутніх змін курсів валют.

**Додаткові умови:**

1. Завантажте історичні дані про курси валют для декількох валютних пар (наприклад, EUR/USD, GBP/USD) зі зручного джерела, такого як API валютних курсів.
2. Реалізуйте аналіз даних для виявлення патернів та тенденцій у зміні курсів валют.
3. Використовуйте математичні методи для створення моделей прогнозування майбутніх змін курсів валют.
4. **Варіант 5.** Знайти в тексті правильні записи цілих, дійсних з плаваючою крапкою та в експоненціальній формі чисел

**Хід роботи**

***Task.java***

public interface Task {  
 void executeTask();  
}

***CurrencyRatePrediction.java***

import java.io.IOException;  
import java.net.URI;  
import java.net.http.HttpClient;  
import java.net.http.HttpRequest;  
import java.net.http.HttpResponse;  
import java.util.\*;  
import org.json.JSONObject;  
  
import static java.lang.Thread.*sleep*;  
  
public class CurrencyRatePrediction implements Task {  
  
 // API key for the exchange rate provider  
 private static final String *API\_URL* = "http://api.currencylayer.com/historical";  
 private static final String *API\_KEY* = "bcf1434f6aeeba07d69db577e75437b6";  
 private static final int[] *API\_OLDEST\_DATE\_ENTRY* = {2019, 1, 1};  
 private static final int[] *API\_LATEST\_DATE\_ENTRY* = {2024, 1, 1};  
  
 private static final int *interval* = 1;  
  
 enum DateStructure{  
 *year*,  
 *month*,  
 *day* }  
  
 @Override  
 public void executeTask() {  
 String[] currencyPairs = {"UAH/USD"};  
  
 for (String pair : currencyPairs) {  
 try {  
 List<Double> historicalRates = *getSpecificDateRate*(pair);  
 *analyzeTrends*(historicalRates);  
 *predictFutureRates*(historicalRates);  
 } catch (IOException | InterruptedException e) {  
 System.*err*.println("Error fetching data for " + pair);  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private static List<Double> getSpecificDateRate(String currencyPair) throws IOException, InterruptedException {  
 String[] currencies = currencyPair.split("/");  
 String targetCurrency = currencies[0];  
 String baseCurrency = currencies[1];  
  
 List<Double> rates = new ArrayList<>();  
  
 int ageGap = *API\_LATEST\_DATE\_ENTRY*[DateStructure.*year*.ordinal()] -  
 *API\_OLDEST\_DATE\_ENTRY*[DateStructure.*year*.ordinal()];  
  
 int[] shiftingDate = *API\_OLDEST\_DATE\_ENTRY*;  
 for (int i = 0; i < ageGap; ++i) {  
 String url = *API\_URL* + "?access\_key=" + *API\_KEY* + "&source=" + baseCurrency + "&date=" +  
 shiftingDate[DateStructure.*year*.ordinal()] + "-" +  
 String.*format*("%02d", shiftingDate[DateStructure.*month*.ordinal()]) + "-" +  
 String.*format*("%02d", shiftingDate[DateStructure.*day*.ordinal()]);  
  
 HttpClient client = HttpClient.*newHttpClient*();  
 HttpRequest request = HttpRequest.*newBuilder*()  
 .uri(URI.*create*(url))  
 .build();  
  
 HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.*ofString*());  
  
 JSONObject jsonResponse = new JSONObject(response.body());  
  
 Double rate = jsonResponse.getJSONObject("quotes").getDouble(baseCurrency+targetCurrency);  
 rates.add(rate);  
  
 shiftingDate[DateStructure.*year*.ordinal()]++;  
 *sleep*(5000, 0);  
 }  
  
 return rates;  
 }  
  
 private static void analyzeTrends(List<Double> rates) {  
 if (rates == null || rates.isEmpty()) {  
 System.*out*.println("No data available for analysis.");  
 return;  
 }  
  
 double sum = 0;  
 for (double rate : rates) {  
 sum += rate;  
 }  
 double average = sum / rates.size();  
 System.*out*.println("Average Rate: " + average);  
  
 // Simple moving average (SMA)  
 int period = 1;  
  
 List<Double> sma = *calculateSMA*(rates, period);  
 System.*out*.println("Simple Moving Average: " + sma);  
 }  
  
 private static List<Double> calculateSMA(List<Double> rates, int period) {  
 List<Double> sma = new ArrayList<>();  
 for (int i = 0; i <= rates.size() - period; i++) {  
 double sum = 0;  
 for (int j = i; j < i + period; j++) {  
 sum += rates.get(j);  
 }  
 sma.add(sum / period);  
 }  
 return sma;  
 }  
  
 // Predicts using linear regression  
 private static void predictFutureRates(List<Double> rates) {  
 if (rates == null || rates.size() < 2) {  
 System.*out*.println("Not enough data to perform linear regression.");  
 return;  
 }  
  
 int n = rates.size();  
 double sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0;  
  
 for (int i = 0; i < n; i++) {  
 sumX += i;  
 sumY += rates.get(i);  
 sumXY += i \* rates.get(i);  
 sumX2 += i \* i;  
 }  
  
 double denominator = (n \* sumX2 - sumX \* sumX);  
 if (denominator == 0) {  
 System.*out*.println("Unable to perform linear regression: division by zero.");  
 return;  
 }  
  
 double m = (n \* sumXY - sumX \* sumY) / denominator;  
 double b = (sumY - m \* sumX) / n;  
  
 double nextRate = m \* n + b;  
 System.*out*.println("Predicted next rate: " + nextRate);  
 }  
}

***NumberFinder.java***

import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Scanner;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
public class NumberFinder implements Task {  
  
 @Override  
 public void executeTask() {  
 String filePath = "text.txt";  
 String text;  
  
 try {  
 text = *readFromFile*(filePath);  
 } catch (FileNotFoundException e) {  
 System.*err*.println("Error: The file " + filePath + " was not found.");  
 e.printStackTrace();  
 return;  
 }  
  
 // Regular expression for finding valid numbers only  
 String numberRegex = "(?<![\\d.])(-?\\d+(\\.\\d+)?([eE]-?\\d+)?)(?![\\d.])";  
  
 // Create pattern and matcher  
 Pattern = Pattern.*compile*(numberRegex);  
 Matcher = pattern.matcher(text);  
  
 *OutFoundNumbers*(matcher);  
 }  
  
 private static String readFromFile(String filePath) throws FileNotFoundException {  
 StringBuilder text = new StringBuilder();  
  
 try (Scanner = new Scanner(new File(filePath))) {  
 while (scanner.hasNextLine()) {  
 text.append(scanner.nextLine()).append("\n");  
 }  
 }  
  
 return text.toString();  
 }  
  
 private static void OutFoundNumbers(Matcher matcher) {  
 System.*out*.println("Found numbers:");  
  
 while (matcher.find()) {  
 System.*out*.println(matcher.group());  
 }  
 }  
}

***Main.java***

import java.util.HashMap;  
import java.util.Map;  
  
  
public class Main {  
  
  
 public static void main(String[] args){  
 Map<Integer, Task> tasks = new HashMap<>();  
 tasks.put(1, new CurrencyRatePrediction());  
 tasks.put(2, new NumberFinder());  
 }  
}

Посилання на проєкт на GitLab - <https://gitlab.com/crossplatformprog/cpp_lab2.git>

**Висновки:** протягом виконання цієї лабораторної роботи я навчився використовувати основний інструментарій мови для пошуку, зміни, форматування текстової інформації.