МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА «ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ ІМЕНІ А. В. ДАБАГЯНА»

ЗВІТ

З лабораторної роботи №2

«РОБОТА ДАТАМИ ТА ТЕКСТОМ. ЛОКАЛІЗАЦІЯ»

ВИКОНАВ:

студент групи КН-422ч

Максим БЕЛОШИЦЬКИЙ

ПЕРЕВІРИВ:

Асистент каф. ПІІТУ

Олексій КОНДРАТОВ

Харків – 2023

**ЗМІСТ**

[МЕТА 3](#_Toc166020287)

[Завдання 1.1 4](#_Toc166020288)

[Код завдання 1.1: 5](#_Toc166020289)

[ConferenceWithLocalization.java 5](#_Toc166020290)

[Event.java 7](#_Toc166020291)

[Main.java 10](#_Toc166020292)

[Тестування за допомогою JUnit: 12](#_Toc166020293)

[ConferenceWithLocalizationTest.java 12](#_Toc166020294)

[EventTest.java 14](#_Toc166020295)

[Результати виконання програми 15](#_Toc166020296)

[Завдання 1.2 17](#_Toc166020297)

[Код завдання 1.2: 17](#_Toc166020298)

[DateFormat.java 17](#_Toc166020299)

[Тестування за допомогою JUnit: 18](#_Toc166020300)

[DateFormatTest.java 18](#_Toc166020301)

[Результати виконання програми 19](#_Toc166020302)

[Завдання 1.3 20](#_Toc166020303)

[Код завдання 1.3: 20](#_Toc166020304)

[Kyivstar.java 20](#_Toc166020305)

[Тестування за допомогою JUnit: 21](#_Toc166020306)

[KyivstarTest.java 21](#_Toc166020307)

[Результати виконання програм 22](#_Toc166020308)

[Завдання 1.4 23](#_Toc166020309)

[Код завдання 1.4: 23](#_Toc166020310)

[PasswordConfirmation.java 23](#_Toc166020311)

[Тестування за допомогою JUnit: 24](#_Toc166020312)

[PasswordConfirmationTest.java 24](#_Toc166020313)

[Результати виконання програми 25](#_Toc166020314)

[Завдання 1.4 27](#_Toc166020315)

[Код завдання 1.5: 27](#_Toc166020316)

[Separation.java 27](#_Toc166020317)

[Тестування за допомогою JUnit: 28](#_Toc166020318)

[SeparetionTest.java 28](#_Toc166020319)

[Результати виконання програми 29](#_Toc166020320)

[Завдання для самоконтролю 30](#_Toc166020321)

[Код завдання 2.1: 30](#_Toc166020322)

[Результати виконання програми 31](#_Toc166020323)

[Код завдання 2.2: 32](#_Toc166020324)

[Результати виконання програми 33](#_Toc166020325)

[Код завдання 2.3: 33](#_Toc166020326)

[Результати виконання програми 34](#_Toc166020327)

[ВИСНОВОК 35](#_Toc166020328)

# **МЕТА**

Метою лабораторної роботи є не лише розробка програмного забезпечення для роботи з датами та текстом, але й освоєння важливих концепцій програмування, таких як обробка даних, локалізація та робота з регулярними виразами. У процесі виконання завдань студент має опанувати навички роботи з класами, методами та змінними у середовищі Java. Він також повинен засвоїти принципи об'єктно-орієнтованого програмування та вміти застосовувати їх для створення модульних та ефективних рішень. Крім того, розробка програмного забезпечення з локалізацією дозволяє студентові ознайомитися з міжнародними стандартами розробки програмного забезпечення та вивчити техніки, які дозволяють забезпечити коректну роботу програм у різних культурних та мовних середовищах. Таким чином, лабораторна робота спрямована на комплексний розвиток студента в області програмування та розробки програмного забезпечення.

# Завдання 1.1

Спроєктувати та реалізувати класи для представлення сутностей [третьої лабораторної роботи](http://iwanoff.inf.ua/java_ua/LabTraining03.html#Censuses) курсу "Основи програмування Java". Рішення повинне базуватися на раніше створеній ієрархії класів.

Слід створити похідний клас, який представляє основну сутність. Як базовий використати клас, створений у [попередній лабораторній роботі](http://iwanoff.inf.ua/java_ua_2/LabTraining01.html). Клас повинен бути доповненим можливостями підтримки різних локалізацій, зокрема, української та американської. Необхідно передбачити переклад тексту, виведення чисел, а також дат і часу з урахуванням різних локалізацій. Додати (модифікувати) пошук слів у коментарях (або іншому тексті) за допомогою регулярних виразів. Здійснити сортування сутностей за алфавітом з використанням класу Collator.

Створити похідний клас від класу, який представляє другу сутність, в якому додати поле – час і дата, коли відбувається певна подія. Для представлення часу й дати використовувати класи пакету java.time (з урахуванням поясного часу). Підрахувати проміжки часу між подіями та знайти й вивести найменший з проміжків. Якщо клас, який представляв другу сутність індивідуального завдання, не містив поля, типу String, слід додати поле – коментар до події. Для нового (або такого, що існує) текстового поля передбачити можливість виведення українською або англійською мовою, залежно від локалізації.

Програма повинна демонструвати:

* відтворення реалізації завдань лабораторних робіт № 3 і № 4 курсу "Основи програмування Java";
* форматування числових даних різними варіантами, а також з урахуванням локалізації;
* виведення даних про дати й час подій з урахуванням локалізації;
* виведення тексту українською та англійською мовою;
* можливості сортування за алфавітом з використанням класу Collator;
* підрахування та виведення проміжків часу між подіями, пов'язаними з другою сутністю завдання; знаходження й виведення найменшого з проміжків;
* варіанти складного пошуку в тексті (із застосуванням регулярних виразів), зокрема, пошук фрагмента тексту на початку (наприкінці) слова.

## Код завдання 1.1:

### ConferenceWithLocalization.java

package *lab01.ind.src.main*;

import *java.text.Collator*;

import *java.util.ArrayList*;

import *java.util.List*;

import *java.util.Locale*;

import *java.util.ResourceBundle*;

import *java.util.regex.Matcher*;

import *java.util.regex.Pattern*;

*/\*\**

*\* The ConferenceWithLocalization class represents a conference with localization support.*

*\* It extends the ConferenceWithStreams class and adds additional functionality for localization.*

*\*/*

*public* *class* ConferenceWithLocalization *extends* *ConferenceWithStreams* {

*protected* *List*<*Event*> eventsList;

*protected* *Locale* locale;

*/\*\**

*\* Constructs a ConferenceWithLocalization object with the specified name, place, events list, and locale.*

*\**

*\* @param name       the name of the conference*

*\* @param place      the place of the conference*

*\* @param eventsList the list of events in the conference*

*\* @param locale     the locale for localization*

*\*/*

*public* ConferenceWithLocalization(*String* *name*, *String* *place*, *List*<*Event*> *eventsList*, *Locale* *locale*) {

*super*(name, place, new *ArrayList*<>());

*this*.eventsList = eventsList;

*this*.locale = locale;

    }

*/\*\**

*\* Gets the list of events in the conference.*

*\**

*\* @return the list of events*

*\*/*

*public* *List*<*Event*> getEventsList() {

        return eventsList;

    }

*/\*\**

*\* Sets the list of events in the conference.*

*\**

*\* @param eventsList the list of events*

*\*/*

*public* *void* setEventsList(*List*<*Event*> *eventsList*) {

*this*.eventsList = eventsList;

    }

*/\*\**

*\* Gets the localized text for the given key and locale.*

*\**

*\* @param key    the key for the localized text*

*\* @param locale the locale for localization*

*\* @return the localized text*

*\*/*

*public* *String* getLocalizedText(*String* *key*, *Locale* *locale*) {

*ResourceBundle* messages = ResourceBundle.getBundle("dict", locale);

        return messages.getString(key);

    }

*/\*\**

*\* Searches for words in the given text that match the specified regular expression.*

*\**

*\* @param text  the text to search in*

*\* @param regex the regular expression to match*

*\* @return a list of indices where the words are found*

*\*/*

*public* *List*<*Integer*> searchWords(*String* *text*, *String* *regex*) {

*List*<*Integer*> wordIndices = new *ArrayList*<>();

*Pattern* pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE | Pattern.UNICODE\_CASE); *// Compile pattern with CASE\_INSENSITIVE flag*

*Matcher* matcher = pattern.matcher(text);

        while (matcher.find()) {

*int* startIndex = matcher.start();

            wordIndices.add(startIndex);

        }

        return wordIndices;

    }

*/\*\**

*\* Sorts the meetings in the events list by topic using the specified locale.*

*\**

*\* @param locale the locale for sorting*

*\*/*

*public* *void* sortMeetingsByTopic(*Locale* *locale*) {

*Collator* collator = Collator.getInstance(locale);

        eventsList.sort((m1, m2) -> collator.compare(m1.getTopic(), m2.getTopic()));

    }

*/\*\**

*\* Finds and prints the smallest time difference between events in the events list.*

*\* Prints the topics of the events and the time difference in minutes.*

*\*/*

*public* *void* findAndPrintSmallestTimeDifference() {

*long* smallestDifference = Long.MAX\_VALUE;

*Event* event1 = null, event2 = null;

        for (*int* i = 0; i < eventsList.size(); i++) {

            for (*int* j = i + 1; j < eventsList.size(); j++) {

*long* difference = eventsList.get(i).calculateTimeDifference(eventsList.get(j));

                if (difference < smallestDifference) {

                    smallestDifference = difference;

                    event1 = eventsList.get(i);

                    event2 = eventsList.get(j);

                }

            }

        }

        if (event1 != null && event2 != null) {

            System.out.println(getLocalizedText("smallest\_time\_difference\_header", locale) + ":");

            System.out.println(event1.getTopic());

            System.out.println(event2.getTopic());

            System.out.println(getLocalizedText("time\_difference", locale) + smallestDifference + " " + getLocalizedText("minutes", locale));

        } else {

            System.out.println(getLocalizedText("no\_events\_found", locale));

        }

    }

}

### Event.java

package *lab01.ind.src.main*;

import *java.time.ZoneOffset*;

import *java.time.ZonedDateTime*;

import *java.time.Duration*;

import *java.text.NumberFormat*;

import *java.time.format.DateTimeFormatter*;

import *java.util.Locale*;

import *java.util.ResourceBundle*;

*public* *class* Event {

*protected* *String* topic;

*protected* *int* numberOfParticipants;

*protected* *ZonedDateTime* dateTime;

*protected* *String* comment;

*protected* *Locale* locale;

*public* Event(*String* *topic*, *int* *numberOfParticipants*, *ZonedDateTime* *dateTime*, *String* *comment*, *Locale* *locale*) {

*this*.topic = topic;

*this*.numberOfParticipants = numberOfParticipants;

*this*.dateTime = dateTime;

*this*.comment = comment;

*this*.locale = locale;

    }

*public* *String* getTopic() {

        return topic;

    }

*public* *int* getNumberOfParticipants() {

        return numberOfParticipants;

    }

*public* *ZonedDateTime* getDateTime() {

        return dateTime;

    }

*public* *void* setDateTime(*ZonedDateTime* *dateTime*) {

*this*.dateTime = dateTime;

    }

*public* *String* getComment() {

        return comment;

    }

*public* *void* setComment(*String* *comment*) {

*this*.comment = comment;

    }

*public* *String* getLocalizedParticipants(*Locale* *locale*) {

*NumberFormat* numberFormat = NumberFormat.getNumberInstance(locale);

        return numberFormat.format(numberOfParticipants);

    }

*// Format date and time with GMT offset*

*public* *String* getFormattedDateTime(*ZonedDateTime* *eventDateTime*, *Locale* *locale*) {

*ResourceBundle* bundle = ResourceBundle.getBundle("dict", locale);

*DateTimeFormatter* formatter = DateTimeFormatter.ofPattern(bundle.getString("date\_time\_format"));

*String* formattedDateTime = eventDateTime.format(formatter);

*ZoneOffset* zoneOffset = eventDateTime.getOffset();

*String* gmtOffset = getGMTOffset(zoneOffset);

        return formattedDateTime + " GMT " + gmtOffset;

    }

*// Helper method to get GMT offset*

*public* *String* getGMTOffset(*ZoneOffset* *zoneOffset*) {

*int* totalSeconds = zoneOffset.getTotalSeconds();

*int* hours = totalSeconds / 3600;

*int* minutes = Math.abs((totalSeconds % 3600) / 60);

*String* sign = (totalSeconds >= 0) ? "+" : "-";

        return String.format("%s%02d:%02d", sign, Math.abs(hours), minutes);

    }

*// Format number of participants*

*public* *String* getFormattedNumberOfParticipants(*Locale* *locale*) {

*NumberFormat* numberFormat = NumberFormat.getNumberInstance(locale);

        return numberFormat.format(numberOfParticipants);

    }

*// Calculate time difference between events*

*public* *long* calculateTimeDifference(*Event* *otherEvent*) {

*Duration* duration = Duration.between(*this*.dateTime, otherEvent.dateTime);

        return Math.abs(duration.toMinutes());

    }

*// Localization Support for Comment*

*public* *String* getLocalizedComment(*Locale* *locale*) {

*ResourceBundle* bundle = ResourceBundle.getBundle("dict", locale);

*String* comment = bundle.getString("comment");

*// Rest of your code*

        if (comment == null || comment.isEmpty()) {

            return comment;

        } else {

            return comment;

        }

    }

}

### Main.java

package *lab01.ind.src.main*;

import *java.util.ArrayList*;

import *java.util.List*;

import *java.time.ZoneOffset*;

import *java.time.ZonedDateTime*;

import *java.util.Locale*;

import *java.util.Scanner*;

*public* *class* Main {

*public* *static* *void* main(*String*[] *args*) {

*// Prompt the user to choose the localization*

*Locale* locale = chooseLocalization();

*// Create a list of events*

*List*<*Event*> eventsList = createEventsList(locale);

*// Translate comments*

        translateComments(eventsList, locale);

*// Create ConferenceWithLocalization instance with the chosen localization*

*ConferenceWithLocalization* conference = new ConferenceWithLocalization("IT International Pole", "France, Montreal", eventsList, locale);

*// Test the conference functionality*

        testConference(conference, locale);

    }

*// Prompt the user to choose the localization*

*private* *static* *Locale* chooseLocalization() {

*Scanner* scanner = new Scanner(System.in);

        System.out.println("Choose localization:");

        System.out.println("1. English");

        System.out.println("2. Українська\n");

        System.out.print("Enter the number of the chosen localization: ");

*int* choice = scanner.nextInt();

        scanner.nextLine(); *// Consume newline character*

        System.out.println();

        return (choice == 2) ? Locale.forLanguageTag("uk") : Locale.forLanguageTag("en");

    }

*private* *static* *List*<*Event*> createEventsList(*Locale* *locale*) {

*List*<*Event*> eventsList = new *ArrayList*<>();

        eventsList.add(new Event("Artificial Intelligence Workshop", 80, ZonedDateTime.of(2024, 3, 20, 10, 0, 0, 0, ZoneOffset.UTC), "Interesting topic", locale));

        eventsList.add(new Event("Data Science Symposium", 100, ZonedDateTime.of(2024, 3, 21, 13, 30, 0, 0, ZoneOffset.ofHours(1)), "Engaging discussion", locale));

        eventsList.add(new Event("Machine Learning Conference", 120, ZonedDateTime.of(2024, 3, 22, 9, 0, 0, 0, ZoneOffset.ofHours(-5)), "Informative session", locale));

        eventsList.add(new Event("Blockchain Summit", 90, ZonedDateTime.of(2024, 3, 23, 11, 0, 0, 0, ZoneOffset.ofHours(8)), "Finance upport", locale));

        return eventsList;

    }

*// Give evet list translated comments*

*private* *static* *void* translateComments(*List*<*Event*> *eventsList*, *Locale* *locale*) {

        for (*Event* event : eventsList) {

            event.setComment(event.getLocalizedComment(locale));

        }

    }

*// Test the conference functionality*

*private* *static* *void* testConference(*ConferenceWithLocalization* *conference*, *Locale* *locale*) {

*// Print the conference details*

        System.out.println(conference.getLocalizedText("conference\_details", locale));

        System.out.println(conference.getLocalizedText("conference\_name", locale) + conference.getName());

        System.out.println(conference.getLocalizedText("conference\_place", locale) + conference.getPlace() + "\n");

*// Sort events by topic and print*

*int* counter = 1;

        System.out.println("\n" + conference.getLocalizedText("events\_sorted\_by\_topic", locale) + ":");

        conference.sortMeetingsByTopic(locale);

        for (*Event* event : conference.getEventsList()) {

            System.out.println(counter + ". " + event.getTopic());

            counter++;

        }

*// Find and print the smallest time difference between events*

        System.out.println();

        conference.findAndPrintSmallestTimeDifference();

*// Let the user search for words in comments*

        System.out.println("\n" + conference.getLocalizedText("search\_words\_in\_comments", locale));

*Scanner* scanner = new Scanner(System.in);

        try {

            System.out.print(conference.getLocalizedText("enter\_search\_word", locale));

*String* searchWord = scanner.nextLine();

*// Flag to track if any match is found*

*boolean* matchFound = false;

            System.out.print("\n" + conference.getLocalizedText("search\_results", locale) + ": ");

*// Iterate over the events*

            for (*Event* event : conference.getEventsList()) {

*// Search for the word in the event's comment*

*List*<*Integer*> wordIndices = conference.searchWords(event.getComment(), searchWord);

*// If the word is found, print the event information*

                if (!wordIndices.isEmpty()) {

                    matchFound = true;

                    System.out.println("\n" + conference.getLocalizedText("event\_details", locale));

                    System.out.println(conference.getLocalizedText("event\_name", locale) + event.getTopic());

                    System.out.println(conference.getLocalizedText("event\_date", locale) + event.getFormattedDateTime( event.getDateTime(), locale));

                    System.out.println(conference.getLocalizedText("event\_comment", locale) + event.getComment());

                }

            }

*// If no match is found, print an error message*

            if (!matchFound) {

                System.out.println(conference.getLocalizedText("no\_matching\_events", locale));

            }

        } finally {

            scanner.close();

        }

    }

}

## Тестування за допомогою JUnit:

### ConferenceWithLocalizationTest.java

package *lab01.ind.src.test*;

import *static* *org.junit.Assert.assertEquals*;

import *java.time.ZoneOffset*;

import *java.time.ZonedDateTime*;

import *java.util.ArrayList*;

import *java.util.List*;

import *java.util.Locale*;

import *lab01.ind.src.main.\**;

import *org.junit.Before*;

import *org.junit.Test*;

*public* *class* ConferenceWithLocalizationTest {

*private* *ConferenceWithLocalization* conference;

    @*Before*

*public* *void* setUp() {

*// Create a list of events for testing*

*List*<*Event*> eventsList = new *ArrayList*<>();

        eventsList.add(new Event("Event 1", 100, ZonedDateTime.now().withZoneSameInstant(ZoneOffset.UTC), "Comment 1", Locale.US));

        eventsList.add(new Event("Event 2", 150, ZonedDateTime.now().withZoneSameInstant(ZoneOffset.UTC), "Comment 2", Locale.US));

        eventsList.add(new Event("Event 3", 200, ZonedDateTime.now().withZoneSameInstant(ZoneOffset.UTC), "Comment 3", Locale.US));

*// Create a conference instance for testing*

        conference = new ConferenceWithLocalization("Test Conference", "Test Place", eventsList, Locale.US);

    }

    @*Test*

*public* *void* testGetEventsList() {

*// Ensure that the events list returned by getEventsList matches the one provided during initialization*

        assertEquals(3, conference.getEventsList().size());

    }

    @*Test*

*public* *void* testSetEventsList() {

*// Create a new list of events*

*List*<*Event*> newEventsList = new *ArrayList*<>();

        newEventsList.add(new Event("New Event", 50, ZonedDateTime.now().withZoneSameInstant(ZoneOffset.UTC), "New Comment", Locale.US));

*// Set the new events list*

        conference.setEventsList(newEventsList);

*// Ensure that the events list has been updated*

        assertEquals(1, conference.getEventsList().size());

    }

}

### EventTest.java

package *lab01.ind.src.test*;

import *static* *org.junit.Assert.assertEquals*;

import *java.time.ZonedDateTime*;

import *java.util.Locale*;

import *java.time.ZoneOffset*;

import *lab01.ind.src.main.Event*;

import *org.junit.Test*;

*public* *class* EventTest {

    @*Test*

*public* *void* testGetFormattedDateTime() {

*// Create an event with a specific date and time*

*ZonedDateTime* dateTime = ZonedDateTime.of(2024, 3, 20, 10, 0, 0, 0, ZoneOffset.UTC);

*Event* event = new Event("Artificial Intelligence Workshop", 80, dateTime, "Interesting topic", Locale.US);

*// Define the expected formatted date and time string*

*String* expectedDateTime = "20.03.2024 10:00 GMT +00:00";

*// Call the getFormattedDateTime method*

*String* formattedDateTime = event.getFormattedDateTime(dateTime, Locale.US);

*// Assert that the formattedDateTime matches the expectedDateTime*

        assertEquals(expectedDateTime, formattedDateTime);

    }

    @*Test*

*public* *void* testCalculateTimeDifference() {

*// Create two events with different date times*

*ZonedDateTime* dateTime1 = ZonedDateTime.of(2024, 3, 20, 10, 0, 0, 0, ZoneOffset.UTC);

*ZonedDateTime* dateTime2 = ZonedDateTime.of(2024, 3, 20, 11, 30, 0, 0, ZoneOffset.UTC);

*Event* event1 = new Event("Event 1", 100, dateTime1, "Comment 1", Locale.US);

*Event* event2 = new Event("Event 2", 150, dateTime2, "Comment 2", Locale.US);

*// Calculate the time difference between the two events*

*long* timeDifference = event1.calculateTimeDifference(event2);

*// Assert that the time difference is as expected (1 hour and 30 minutes in minutes)*

        assertEquals(90, timeDifference);

    }

    @*Test*

*public* *void* testGetLocalizedComment() {

*// Create an event with a specific comment and locale*

*Event* event = new Event("Event", 50, ZonedDateTime.now(), "Interesting comment", Locale.US);

*// Get the localized comment for the event*

*String* localizedComment = event.getLocalizedComment(Locale.US);

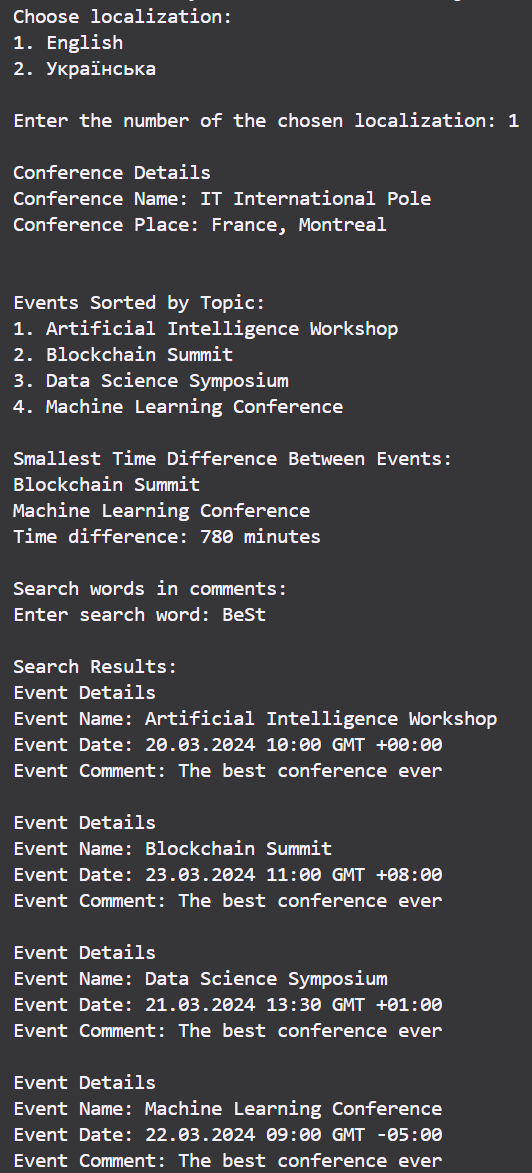
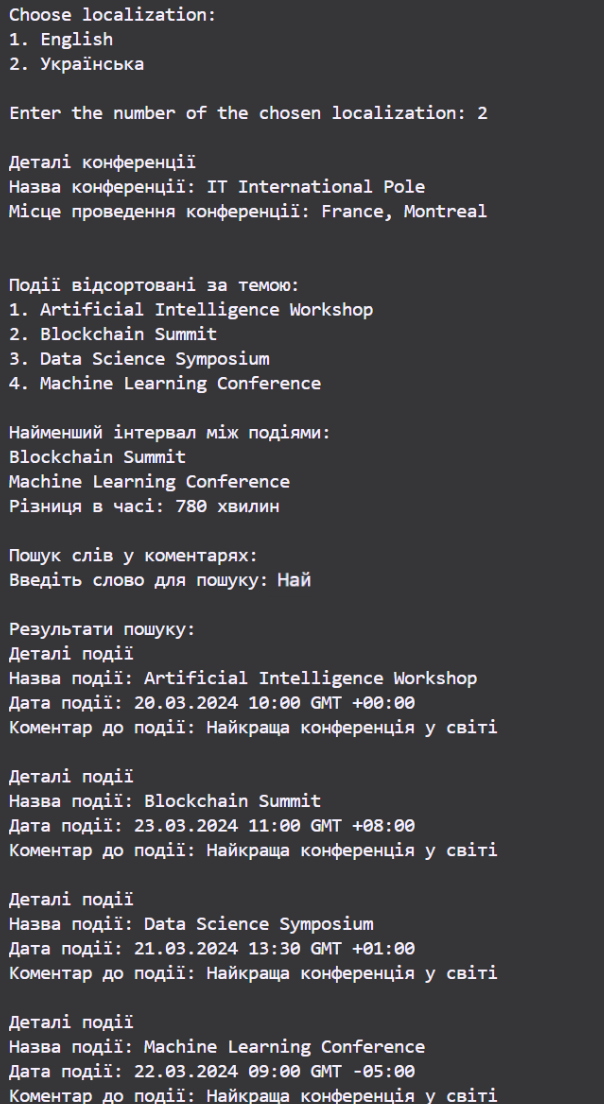
*// Assert that the localized comment matches the original comment*

        assertEquals("The best conference ever", localizedComment);

    }

}

## Результати виконання програми

Рисунки 1.1, 1.2, 1.3 – «Результат роботи програми з різною локалізацією»

Даний код реалізує функціональність для лабораторної роботи з програмування на Java, яка зосереджена на роботі з датами та текстом з підтримкою локалізації. Цей проект використовує ієрархію класів, розроблену на попередніх етапах курсу "Основи програмування Java".

Клас ConferenceWithLocalization розширює клас ConferenceWithStreams та додає можливості для локалізації. Він містить методи для перекладу тексту, виведення даних про дати й час подій у відповідній локалізації, сортування за алфавітом з урахуванням регіональних налаштувань та знаходження найменшого проміжку часу між подіями.

Клас Event представляє подію на конференції і містить інформацію про тему, кількість учасників, дату та час події. Він також має можливість локалізації тексту коментаря та форматування числа учасників у відповідній локалізації.

Клас Main містить метод main, який ініціалізує програму, пропонує користувачеві вибрати локалізацію, створює список подій з відповідними коментарями, створює конференцію з вибраною локалізацією та тестує функціональність конференції.

# Завдання 1.2

Реалізувати програму, в якій користувач вводить рядок. Програма перевіряє, чи відповідає рядок представленню дати, прийнятому в Україні. Перевірка здійснюється за допомогою регулярних виразів.

Якщо рядок не відповідає вимогам, виводиться повідомлення про помилку. В іншому випадку створюються і виводяться на консоль об'єкти Date, GregorianCalendar і LocalDate.

## Код завдання 1.2:

### DateFormat.java

package *main*;

import *java.util.Scanner*;

import *java.util.regex.Matcher*;

import *java.util.regex.Pattern*;

import *java.time.LocalDate*;

import *java.util.GregorianCalendar*;

import *java.util.Date*;

*public* *class* DataFormat {

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

        System.out.println("Введіть рядок для перевірки, чи він представляє собою дату у форматі dd.mm.yyyy:");

*String* input = scanner.nextLine();

        scanner.close();

*// Регулярний вираз для перевірки формату дати*

*String* regex = "^(0[1-9]|[12][0-9]|3[01])\\.(0[1-9]|1[012])\\.\\d{4}$";

*Pattern* pattern = Pattern.compile(regex);

*Matcher* matcher = pattern.matcher(input);

        if (matcher.matches()) {

*// Якщо рядок відповідає формату дати, створюємо об'єкти дати*

*String*[] parts = input.split("\\.");

*int* day = Integer.parseInt(parts[0]);

*int* month = Integer.parseInt(parts[1]);

*int* year = Integer.parseInt(parts[2]);

*LocalDate* localDate = LocalDate.of(year, month, day);

*GregorianCalendar* gregorianCalendar = new GregorianCalendar(year, month - 1, day);

*Date* date = gregorianCalendar.getTime();

*// Виводимо об'єкти дати на консоль*

            System.out.println("LocalDate: " + localDate);

            System.out.println("GregorianCalendar: " + gregorianCalendar.getTime());

            System.out.println("Date: " + date);

        } else {

            System.out.println("Введений рядок не відповідає формату дати dd.mm.yyyy");

        }

    }

}

## Тестування за допомогою JUnit:

### DateFormatTest.java

package *test*;

import *org.junit.jupiter.api.Test*;

import *java.io.ByteArrayInputStream*;

import *java.io.InputStream*;

import *java.util.Scanner*;

import *main.DataFormat*;

import *static* *org.junit.jupiter.api.Assertions.assertEquals*;

*public* *class* DataFormatTest {

    @*Test*

*public* *void* testDataFormatAppWithValidInput() {

*String* input = "25.12.2023"; *// Valid input for date format*

*String* expectedOutput = "LocalDate: 2023-12-25\n" +

                                "GregorianCalendar: Wed Dec 25 00:00:00 EET 2023\n" +

                                "Date: Wed Dec 25 00:00:00 EET 2023";

*InputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        DataFormat.main(new *String*[0]);

        assertEquals(expectedOutput, getConsoleOutput());

    }

    @*Test*

*public* *void* testDataFormatAppWithInvalidInput() {

*String* input = "12-25-2023"; *// Invalid input for date format*

*String* expectedOutput = "Введений рядок не відповідає формату дати dd.mm.yyyy";

*InputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        DataFormat.main(new *String*[0]);

        assertEquals(expectedOutput, getConsoleOutput());

    }

*private* *String* getConsoleOutput() {

*InputStream* inputStream = System.in;

        System.setIn(System.in);

*InputStream* capturedOutputStream = new ByteArrayInputStream(inputStream.toString().getBytes());

        System.setIn(capturedOutputStream);

*Scanner* scanner = new Scanner(System.in);

*StringBuilder* output = new StringBuilder();

        while (scanner.hasNextLine()) {

            output.append(scanner.nextLine()).append("\n");

        }

        scanner.close();

        return output.toString().trim();

    }

}

## Результати виконання програми

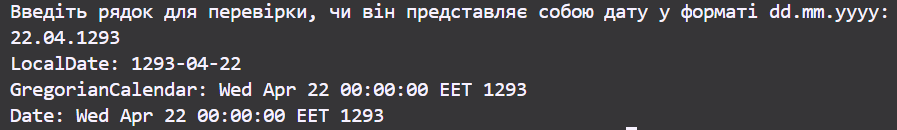


Рисунок 1.4 – «Фолрматування дати програмою»

Під час написання програми "DataFormat" було розроблено програму, яка перевіряє, чи введений користувачем рядок відповідає формату дати "dd.mm.yyyy" та виводить об'єкти дати відповідних класів.

Програма починається з виведення на екран повідомлення для користувача з проханням ввести рядок для перевірки. Після цього відбувається зчитування введеного рядка користувачем.

Далі програма використовує регулярний вираз для перевірки формату введеної дати. Якщо формат відповідає "dd.mm.yyyy", програма створює об'єкти дати відповідних класів. Для цього використовуються класи LocalDate, GregorianCalendar та Date. Якщо формат не відповідає вказаному, виводиться відповідне повідомлення про помилку.

Для перевірки коректності роботи програми було реалізовано набір автоматизованих тестів з використанням JUnit. Тести охоплювали різні сценарії введення користувачем даних: валідні та невалідні формати дати.

Під час написання програми було використано Java 8 Date and Time API для роботи з датою та часом, що забезпечує більш зручні та потужні засоби роботи з датою порівняно зі старішими API, такими як java.util.Date та java.util.GregorianCalendar.

# Завдання 1.3

Розробити програму перевірки правильності того, що рядок є номером телефону оператора Київстар. Слід скористатися регулярними виразами.

## Код завдання 1.3:

### Kyivstar.java

package *lab01.t01.src*;

import *java.util.Scanner*;

import *java.util.regex.Matcher*;

import *java.util.regex.Pattern*;

*public* *class* Kyivstar {

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

        System.out.println("Введіть номер телефону оператора Київстар в міжнародному форматі:");

*String* phoneNumber = scanner.nextLine();

        scanner.close();

*// Регулярний вираз для перевірки формату номера телефону Київстар*

*String* regex = "^\\+380(67|96|97|98|68)\\d{7}$";

*Pattern* pattern = Pattern.compile(regex);

*Matcher* matcher = pattern.matcher(phoneNumber);

        if (matcher.matches()) {

            System.out.println("Введений номер телефону є номером оператора Київстар.");

        } else {

            System.out.println("Введений номер телефону не є номером оператора Київстар.");

        }

    }

}

## Тестування за допомогою JUnit:

### KyivstarTest.java

package *lab01.t01.src.test*;

import *org.junit.jupiter.api.Test*;

import *java.io.ByteArrayInputStream*;

import *java.io.InputStream*;

import *java.util.Scanner*;

import *lab01.t01.src.Kyivstar*;

import *static* *org.junit.jupiter.api.Assertions.assertEquals*;

*public* *class* KyivstarTest {

    @*Test*

*public* *void* testKyivstarAppWithValidInput() {

*String* input = "+380675555555"; *// Valid Kyivstar phone number*

*String* expectedOutput = "Введений номер телефону є номером оператора Київстар.";

*InputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        Kyivstar.main(new *String*[0]);

        assertEquals(expectedOutput, getConsoleOutput());

    }

    @*Test*

*public* *void* testKyivstarAppWithInvalidInput() {

*String* input = "+380444444444"; *// Invalid Kyivstar phone number*

*String* expectedOutput = "Введений номер телефону не є номером оператора Київстар.";

*InputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        Kyivstar.main(new *String*[0]);

        assertEquals(expectedOutput, getConsoleOutput());

    }

*private* *String* getConsoleOutput() {

*InputStream* inputStream = System.in;

        System.setIn(System.in);

*InputStream* capturedOutputStream = new ByteArrayInputStream(inputStream.toString().getBytes());

        System.setIn(capturedOutputStream);

*Scanner* scanner = new Scanner(System.in);

*StringBuilder* output = new StringBuilder();

        while (scanner.hasNextLine()) {

            output.append(scanner.nextLine()).append("\n");

        }

        scanner.close();

        return output.toString().trim();

    }

}

## Результати виконання програм

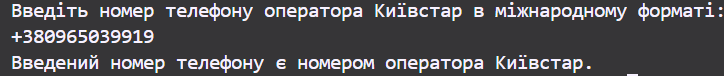


Рисунок 1.5 – «Перевірка номеру»

Даний код репрезентує примітивну програму без графічного інтерфейсу для перевірки номеру оператора на валідність. У даному випадку для звірення коректності введеного телефонного номеру використане застосування регулярних виразів, що значно зменшує обсяг коду, та покращує читаємість

# Завдання 1.4

Розробити програму перевірки відповідності пароля вимогам:

* пароль може містити літери латинського алфавіту, цифри та спеціальні символи: \_ - \*;
* має бути мінімум одна маленька літера;
* має бути мінімум одна велика літера;
* має бути мінімум одна цифра;
* має бути мінімум один спеціальний символ.

Слід скористатися регулярними виразами.

## Код завдання 1.4:

### PasswordConfirmation.java

import *java.util.Scanner*;

import *java.util.regex.Matcher*;

import *java.util.regex.Pattern*;

*public* *class* PasswordConfirmation {

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

        while (true) {

            System.out.println("Введіть пароль:");

*String* password = scanner.nextLine();

*// Регулярний вираз для перевірки вимог до пароля*

*String* regex = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)(?=.\*[\_\\-\*]).{8,}$";

*Pattern* pattern = Pattern.compile(regex);

*Matcher* matcher = pattern.matcher(password);

            if (matcher.matches()) {

                System.out.println("Пароль відповідає вимогам безпеки.");

                break; *// Вийти з циклу, якщо пароль дійсний*

            } else {

                System.out.println("Пароль не відповідає вимогам безпеки.");

                System.out.println("Пароль повинен містити принаймні одну маленьку літеру, одну велику літеру, одну цифру, один спеціальний символ (\_ - \*), і бути принаймні 8 символів у довжину.");

            }

        }

        scanner.close();

    }

}

## Тестування за допомогою JUnit:

### PasswordConfirmationTest.java

import *org.junit.jupiter.api.AfterEach*;

import *org.junit.jupiter.api.BeforeEach*;

import *org.junit.jupiter.api.Test*;

import *java.io.ByteArrayInputStream*;

import *java.io.ByteArrayOutputStream*;

import *java.io.InputStream*;

import *java.io.PrintStream*;

import *main.PasswordConfirmation*;

import *static* *org.junit.jupiter.api.Assertions.assertEquals*;

*public* *class* PasswordConfirmationTest {

*private* *final* *InputStream* originalInput = System.in;

*private* *final* *PrintStream* originalOutput = System.out;

*private* *final* *ByteArrayOutputStream* outputStreamCaptor = new ByteArrayOutputStream();

    @*BeforeEach*

*public* *void* setUp() {

        System.setOut(new PrintStream(outputStreamCaptor));

    }

    @*AfterEach*

*public* *void* tearDown() {

        System.setIn(originalInput);

        System.setOut(originalOutput);

    }

    @*Test*

*public* *void* testPasswordConfirmationWithValidPassword() {

*String* input = "Strong@123"; *// Valid password*

*String* expectedOutput = "Пароль відповідає вимогам безпеки.";

*InputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        PasswordConfirmation.main(new *String*[0]);

        assertEquals(expectedOutput, getConsoleOutput());

    }

    @*Test*

*public* *void* testPasswordConfirmationWithInvalidPassword() {

*String* input = "weak"; *// Invalid password*

*String* expectedOutput = "Пароль не відповідає вимогам безпеки."

                + System.lineSeparator()

                + "Пароль повинен містити принаймні одну маленьку літеру, одну велику літеру, одну цифру, один спеціальний символ (\_ - \*), і бути принаймні 8 символів у довжину.";

*InputStream* inputStream = new ByteArrayInputStream(input.getBytes());

        System.setIn(inputStream);

        PasswordConfirmation.main(new *String*[0]);

        assertEquals(expectedOutput, getConsoleOutput());

    }

*private* *String* getConsoleOutput() {

        return outputStreamCaptor.toString().trim();

    }

}

## Результати виконання програми

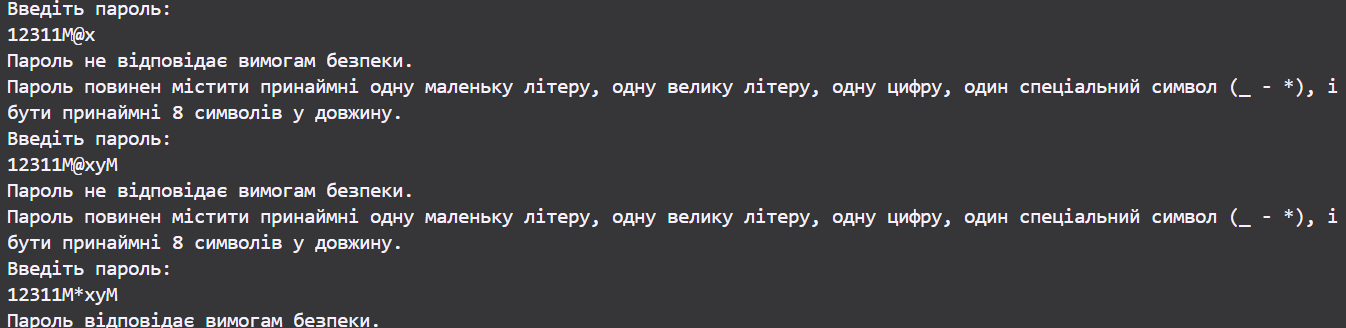


Рисунок 1.6 – «Перевірка пароля»

Під час розробки програми "PasswordConfirmation" було реалізовано валідацію пароля згідно з вказаними вимогами за допомогою регулярних виразів.

Програма працює у безкінечному циклі, де користувачу пропонується ввести пароль. Після введення пароля, він перевіряється за допомогою регулярного виразу, який перевіряє чи відповідає введений пароль вимогам до безпеки:

* Пароль повинен містити принаймні одну маленьку літеру, одну велику літеру, одну цифру та один спеціальний символ (\_ - \*);
* Пароль повинен мати мінімум 8 символів у довжину.

Якщо введений пароль відповідає вимогам безпеки, програма виводить повідомлення про це та завершує свою роботу. У протилежному випадку, виводиться повідомлення про невідповідність вимогам та надається пояснення щодо правил створення пароля.

Під час розробки програми було використано класи Scanner, Pattern та Matcher з пакету java.util.regex для роботи з регулярними виразами, які дозволили здійснити валідацію введеного пароля.

Програма була розроблена з дотриманням принципів читабельності та ефективності, забезпечуючи користувачу зручне введення пароля та зрозуміле повідомлення про його валідність.

# Завдання 1.4

Рядок довжиною понад 20 символів містить літери та цифри. Отримати з цього рядка масив підрядків, які містять літери між цифрами (групами цифр), визначити цифри як розділювачі.

## Код завдання 1.5:

### Separation.java

package *lab01.t03.src*;

import *java.util.ArrayList*;

import *java.util.List*;

*public* *class* Separation {

*public* *static* *void* main(*String*[] *args*) {

*String* inputString = "ab12c3def456gh789ij10";

        System.out.println("Вхідний рядок: " + inputString);

*// Створення списку для зберігання підрядків*

*List*<*String*> substrings = new *ArrayList*<>();

*// Змінна для зберігання поточного підрядка*

*StringBuilder* currentSubstring = new StringBuilder();

*// Перебір символів у вхідному рядку*

        for (*char* c : inputString.toCharArray()) {

            if (Character.isDigit(c)) {

*// Якщо зустрілася цифра, додати поточний підрядок до списку і очистити його*

                if (currentSubstring.length() > 0) {

                    substrings.add(currentSubstring.toString());

                    currentSubstring.setLength(0);

                }

            } else {

*// Додати символ до поточного підрядка*

                currentSubstring.append(c);

            }

        }

*// Додати останній підрядок до списку (якщо він існує)*

        if (currentSubstring.length() > 0) {

            substrings.add(currentSubstring.toString());

        }

*// Вивести знайдені підрядки*

        System.out.println("Знайдені підрядки:");

        for (*String* substring : substrings) {

            System.out.println(substring);

        }

    }

}

## Тестування за допомогою JUnit:

### SeparetionTest.java

import *org.junit.jupiter.api.AfterEach*;

import *org.junit.jupiter.api.BeforeEach*;

import *org.junit.jupiter.api.Test*;

import *java.io.ByteArrayOutputStream*;

import *java.io.PrintStream*;

import *java.util.List*;

import *main.Separation*;

import *static* *org.junit.jupiter.api.Assertions.assertEquals*;

*public* *class* SeparationTest {

*private* *final* *PrintStream* originalOutput = System.out;

*private* *final* *ByteArrayOutputStream* outputStreamCaptor = new ByteArrayOutputStream();

    @*BeforeEach*

*public* *void* setUp() {

        System.setOut(new PrintStream(outputStreamCaptor));

    }

    @*AfterEach*

*public* *void* tearDown() {

        System.setOut(originalOutput);

    }

    @*Test*

*public* *void* testSeparationWithValidInput() {

*String* inputString = "ab12c3def456gh789ij10";

*String* expectedOutput = "Вхідний рядок: ab12c3def456gh789ij10\n" +

                "Знайдені підрядки:\n" +

                "ab\n" +

                "c\n" +

                "def\n" +

                "gh\n" +

                "ij";

        Separation.main(new *String*[0]);

        assertEquals(expectedOutput, getConsoleOutput().trim());

    }

*private* *String* getConsoleOutput() {

        return outputStreamCaptor.toString().trim();

    }

}

## Результати виконання програми

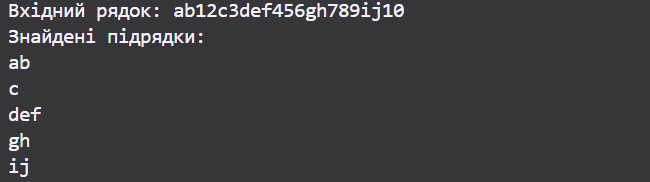


Рисунок 1.7 – «Розподіл рядку через субрядки»

У програмі "Separation" реалізовано розбиття вхідного рядка на підрядки, в яких розділювачами є цифри. Програма дозволяє розділити вхідний рядок на окремі підрядки, виключаючи цифри, і вивести ці підрядки на екран.

Основні кроки, які виконує програма:

1. Визначається вхідний рядок inputString.
2. Створюється порожній список substrings, який буде зберігати окремі підрядки.
3. За допомогою циклу for-each перебираються символи у вхідному рядку.
4. Якщо поточний символ є цифрою, то:
   * Якщо поточний підрядок не порожній, то він додається до списку substrings, а потім очищається.
5. В іншому випадку поточний символ додається до поточного підрядка currentSubstring.
6. Останній поточний підрядок додається до списку substrings.
7. Знайдені підрядки виводяться на екран.

Під час розробки програми використано класи String, StringBuilder для роботи з рядками та ArrayList для зберігання підрядків. Програма використовує ефективний і простий у використанні механізм роботи з рядками та списками, щоб розділити вхідний рядок на підрядки.

# Завдання для самоконтролю

Вивести на екран інформацію про всі часові пояси.

## Код завдання 2.1:

import *java.time.Instant*;

import *java.time.ZoneId*;

import *java.time.ZoneOffset*;

import *java.util.Set*;

import *java.util.TreeSet*;

*public* *class* TimeZoneInfo {

*public* *static* *void* main(*String*[] *args*) {

*// Отримуємо всі доступні часові пояси*

*Set*<*String*> zones = ZoneId.getAvailableZoneIds();

*// Створюємо відсортовану колекцію часових поясів*

*TreeSet*<*String*> sortedZones = new *TreeSet*<>(zones);

*// Виводимо відформатовану інформацію про кожен часовий пояс*

        System.out.println("Всі доступні часові пояси:");

        for (*String* zone : sortedZones) {

            System.out.println(formatZoneInfo(zone));

        }

    }

*// Метод для форматування інформації про часовий пояс*

*private* *static* *String* formatZoneInfo(*String* *zoneId*) {

*ZoneId* zone = ZoneId.of(zoneId);

*Instant* instant = Instant.now(); *// Example Instant object*

*ZoneOffset* offset = zone.getRules().getOffset(instant);

*String* gmtOffset = offset.getId().replaceAll("Z", "+00:00");

        return "Часовий пояс: " + zoneId + " (GMT" + gmtOffset + ")";

    }

}

## Результати виконання програми

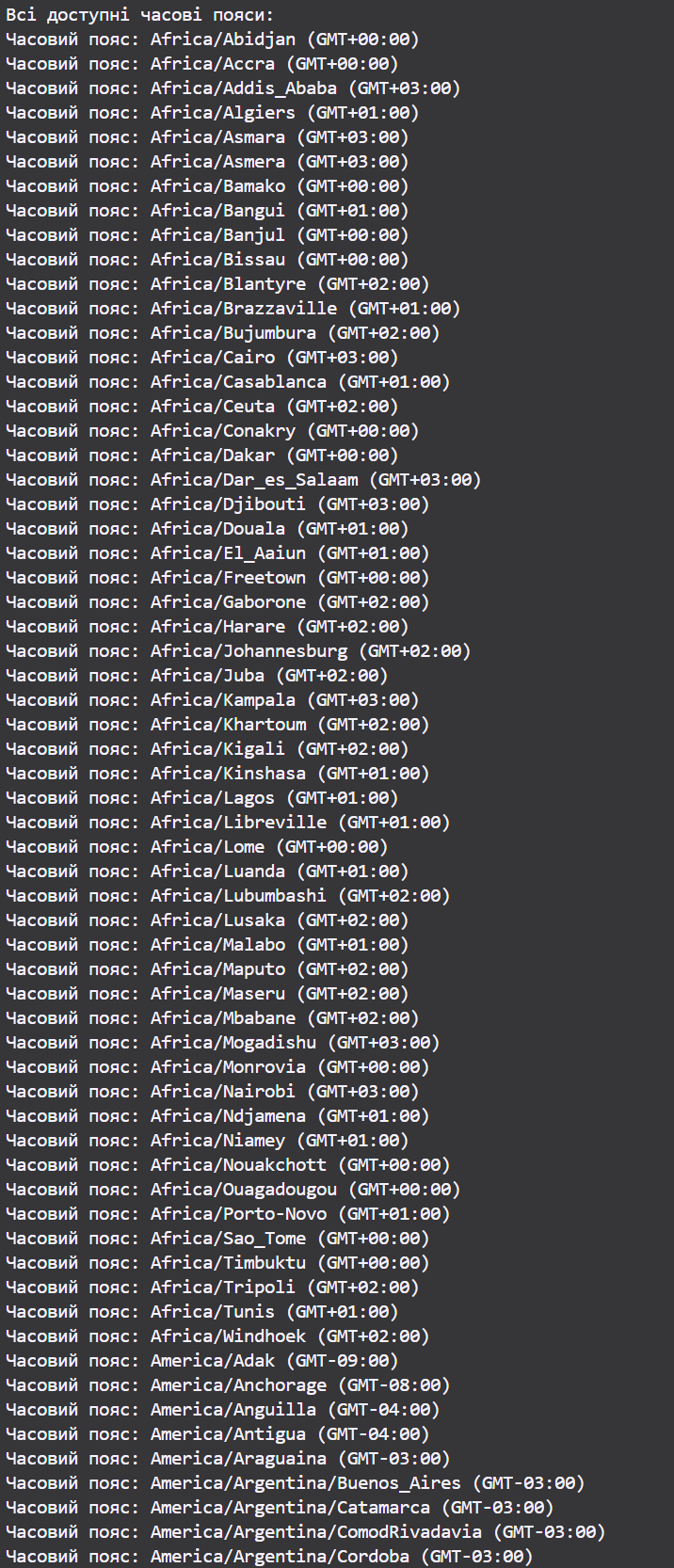


Рисунок 2.1 – «Виведення всіх часових поясів»

Прочитати з клавіатури рядок та перевірити за допомогою регулярних виразів правильність представлення грошової суми.

## Код завдання 2.2:

import *java.util.Scanner*;

import *java.util.regex.Matcher*;

import *java.util.regex.Pattern*;

*public* *class* Money {

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

        System.out.println("Введіть грошову суму: ");

*String* input = scanner.nextLine();

*// Регулярний вираз для перевірки грошової суми*

*String* regex = "^\\$?\\d+(\\.\\d{1,2})?$";

*// Створення об'єкта Pattern*

*Pattern* pattern = Pattern.compile(regex);

*// Створення об'єкта Matcher*

*Matcher* matcher = pattern.matcher(input);

*// Перевірка відповідності введеному рядку шаблону*

        if (matcher.matches()) {

            System.out.println("Грошова сума введена правильно.");

        } else {

            System.out.println("Введена грошова сума має неправильний формат.");

        }

        scanner.close();

    }

}

## Результати виконання програми

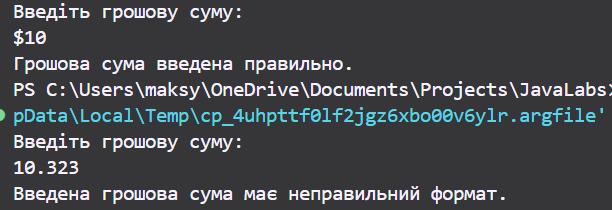


Рисунок 2.2 – «Валідація грошового формату»

Прочитати з клавіатури рядок та перевірити за допомогою регулярних виразів правильність представлення часу.

## Код завдання 2.3:

import *java.util.Scanner*;

import *java.util.regex.Matcher*;

import *java.util.regex.Pattern*;

*public* *class* TimeCorrectness {

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

        System.out.println("Введіть час у форматі HH:mm:ss: ");

*String* input = scanner.nextLine();

*// Регулярний вираз для перевірки часу у форматі HH:mm:ss*

*String* regex = "([01]?[0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9]";

*// Створення об'єкта Pattern*

*Pattern* pattern = Pattern.compile(regex);

*// Створення об'єкта Matcher*

*Matcher* matcher = pattern.matcher(input);

*// Перевірка відповідності введеному рядку шаблону*

        if (matcher.matches()) {

            System.out.println("Час введений правильно.");

        } else {

            System.out.println("Введений час має неправильний формат.");

        }

        scanner.close();

    }

}

## Результати виконання програми

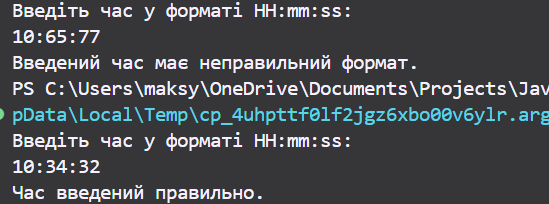


Рисунок 2.3 – «Валідація формату дати»

# **ВИСНОВОК**

Лабораторна робота з основ програмування Java спрямована на розвиток студентів у сфері програмування та розробки програмного забезпечення. Вона надає можливість ознайомитися з ключовими концепціями програмування, такими як обробка даних, робота з локалізацією та регулярними виразами. Виконання завдань допомагає студентам виробити навички роботи з класами, методами та змінними, а також засвоїти принципи об'єктно-орієнтованого програмування. Важливим аспектом є розробка програмного забезпечення з локалізацією, що дозволяє забезпечити коректну роботу програм у різних культурних та мовних середовищах. Таким чином, лабораторна робота сприяє комплексному розвитку студентів у галузі програмування та розробки програмного забезпечення.