

Лабораторна робота №11

Тема: Застосування класів Java для роботи з датою та часом.

Мета: Навчитися створювати та реалізовувати додатки з використанням методів класів Java для встановлення та отримання компонентів дати та часу.

Обладнання: ОС Windows, інтегроване середовище JetBrains IntelliJ IDEA.

1. Теоретичні відомості

Оскільки комп'ютерні програми у переважній більшості мають прикладне призначення й обслуговують різні технологічні та бізнесові процеси, в них часто виникає необхідність обробляти дані про час та дату. Цей різновид даних у порівнянні з іншими типами даних, має певні особливості представлення та обробки, обумовлені історичними причинами.

Територія України знаходитьться у трьох часових поясах, але переважна її частина лежить у часовому поясі GMT+02, оскільки відповідний меридіан проходить через Київ.

Через складний процес розвитку систем відліку часу та дат, в різних частинах світу існують різні системи запису дати та часу. Навіть стандартизація часу UTC, яка дозволила уникати помилок при обчисленні відхилень місцевого часу від часу за Гринвіцьким меридіаном (GMT), не дозволила повністю прибрати розбіжності у записі дат. Наприклад, дати в Україні у вигляді чисел часто записують так: 01.03.2021. Першим позначають день місяця, другим номер місяця у році, й нарешті номер року. А скажімо, в Америці розповсюджений інакший запис: 03/01/2020. Тобто спочатку місяць, потім число, а потім рік. А бувають ще системи, засновані на номерах тижнів у році, та номерах днів на тижні тощо. Таким чином, при роботі з датою та часом в комп'ютерних системах виникає проблема форматування дат та часу, або, навпаки, розпізнавання формату дати та часу.

Крім того, у різного роду задачах існує проблема арифметики часу. Виникає необхідність обраховувати тривалість проміжків часу з урахуванням потрібних одиниць вимірювання (секунд, хвилин тощо), або виникає необхідність з'ясування, яка була дата певну кількість часу тому т. і.

Задля розв'язання проблеми в комп'ютерній техніці з'явився спеціальний вигляд дати та часу, який дозволяє легко представляти дату та час у вигляді великого цілого числа. Як правило, це кількість секунд, яка минула від опорної дати та часу. Для форматів UNIX time опорною датою було обрано 1 січня 1970 року 00:00:00 (GMT). Проте треба бути уважним щодо цього, оскільки іноді буває, що опорна дата може бути іншою, наприклад 1 січня 1900 року або 1 січня 2000 року. Крім того, часто буває так, що замість кількості секунд, це число відбиває кількість мілісекунд, або наносекунд, як, наприклад, це зроблено в мові Java.

Таким чином, станом на зараз існує два основні різновиди представлення дат та часу в комп'ютерних системах - у вигляді рядків форматованого тексту та у вигляді великих цілих чисел. Відповідно виникають наступні проблеми роботи з датою та часом:

- перетворення дати та часу з текстового у ціле представлення з урахуванням формату;
- перетворення дати та часу з цілого представлення у текстове з урахуванням формату;
- зміна одного текстового формату дати та часу на інший текстовий;
- обчислення над датою та часом без або з урахуванням часових поясів;

Мова Java має два окремі набори класів, що виконують зазначені операції з датою та часом. Перший набір класів входить до складу пакету `java.util`, а другий утворив самостійний пакет `java.time`.

Перед тим, як братися до вивчення відповідного мовного інструментарію, треба ще раз обговорити деякі питання термінології. Часто в описах функцій, крім звичних понять, пов'язаних з календарем, датами та часом, можна зустріти наступні поняття:

- Часовий пояс — частина поверхні земної кулі, на якій ухвалено один стандартний час, який ще часто називають місцевим часом. Для зручності відліку поточного часу вся територія земної кулі поділена на 24 часові пояси шириною в середньому 15° . В межах кожного такого поясу встановлюється свій єдиний поясний час;
- GMT - Середній сонячний час меридіана, що проходить через колишнє місце розташування Гринвіцької обсерваторії;
- UTC - універсальний корегований час. Для того, щоб не прив'язувати базову величину для виміру часу до певної географічної назви, був утворений нащадок GMT, який прийнято вважати еквівалентом GMT, хоча основою обчислення UTC є атомний годинник. UTC не є аж надто точним, відрізняючись на кілька секунд від суверо рівного вимірювання часу, та в межах секунди від квазірівномірного;
- Локальний стандартний час – сонячний час кожного меридіану, який утворює певну часову зону. Наприклад, через Київ проходить меридіан, який утворює часовий пояс GMT+02;
- Локальний годинниковий час - попри те, що певна місцина належить до якогось часового поясу, зовсім не обов'язково, щоб вона лежала на меридіані, яким утворений цей часовий пояс. Тож іноді, аби не просто знати час у певному часовому поясі, а знати точний час у певній місцині, треба врахувати її зсув відносно меридіану того часового поясу, де знаходиться місцина. Тож точний годинниковий час може трохи відрізнятися від часу меридіана часового пояса. Проте часто цим нехтують, спрощуючи вимірювання часу й ототожнюючи ці поняття;
- DayLight Saving Time - ознака переходу на літній/зимовий час.

1.1 Класи зі складу java.util.

Класи для роботи з датою та часом, включені до складу `java.util`, умовно можна поділити на основні, що безпосередньо маніпулюють з відповідними абстракціями, та допоміжні. До допоміжних класів належать клас опису часового поясу та клас опису мовного стандарту. Оскільки вони використовуються як типи даних методів та їхніх аргументів, що входять до складу основних класів, то допоміжні класи слід розглянути першими.

1.1.1 Клас `java.util.Timezone`

Цей клас призначений для того, аби позначити певний часовий пояс з його зсувом відносно UTC та тим, чи підтримує він перехід на літній/зимовий час. Кожен часовий пояс можна позначити саме використовуючи зсув у годинах та хвилинах відносно Гринвіцького часу. Але переважна частина часових поясів має власні позначки (ID). Здебільшого ID містить назву міста чи місця, де проходить головний меридіан часового поясу.

Переглянемо, які часові пояси відомі класу `TimeZone`:

```
import java.util.*;
class TimeZoneDemo {
    public static void main(String[] argv) {
        System.out.println("Наявні часові пояси: ");
        for (String timezoneID : TimeZone.getAvailableIDs())
            System.out.println(timezoneID);
        System.out.println("Загальна кількість часових поясів:
"+TimeZone.getAvailableIDs().length);
    }
}
```

Виконавши цей код, можна побачити довгий список ID часових поясів та їхню загальну кількість. Тобто за необхідності можна створити об'єкт `TimeZone` для будь-якого з цих часових поясів, або за допомогою методу `getDefault()` можна отримати об'єкт `TimeZone` поточного часового поясу, використованого Вашою операційною системою.

Уявімо, що необхідно дізнатися різницю у часі між поточним часовим поясом та часовим поясом, де лежить місто Лондон:

```
import java.util.*;
class TimeZoneDemo {
    public static void main(String[] argv) {
        TimeZone currentTZ = TimeZone.getDefault();
        TimeZone LondonTZ =
TimeZone.getTimeZone("Europe/London");
        // Зсув часового поясу зберігається у мілісекундах
        long TimeDiff = (currentTZ.getRawOffset() -
LondonTZ.getRawOffset())/1000;
```

```

        System.out.println("Різниця складає "+TimeDiff/3600+" годин та "+(TimeDiff%3600)/60+" хвилин");
    }
}

```

Але попри чималу кількість часових поясів, параметри яких заздалегідь визначені, Java дозволяє створити й власні часові пояси. При цьому варто врахувати правила іменування користувачьких часових поясів. Офіційна документація класу TimeZone рекомендує використовувати один з таких форматів:

- GMT<знак><кількість годин>;
- GMT<знак><кількість годин><кількість хвилин>;
- GMT<знак><кількість годин>:<кількість хвилин>,

де:

- <знак> - "+" для часових поясів на схід від Гринвіча та "-" для часових поясів на захід від Гринвіча;
- <кількість годин> - однозначне чи двозначне число у межах від 0 до 23;
- <кількість хвилин> - двозначне число у межах від 00 до 59.

Наприклад, можливими назвами користувачьких часових поясів є такі: GMT+02, GMT-0100, GMT+1:00 тощо.

Від того, яку кількість знаків було використано під час позначення кількості годин, та який саме формат з наведених вище було використано, при зворотному запиті об'єкт завжди поверне відповідь наступного вигляду:

- GMT<знак><кількість годин>:<кількість хвилин>,

де:

- <знак> - "+" для часових поясів на схід від Гринвіча та "-" для часових поясів на захід від Гринвіча;
- <кількість годин> - двозначне число у межах від 00 до 23;
- <кількість хвилин> - двозначне число у межах від 00 до 59.

Зауважте, що рядок вхідного та вихідного формату користувачького часового поясу не містить жодної інформації про перехід на літній/зимовий час. На жаль, користувачькі часові пояси позбавлені можливості позначати такий перехід. Тож у випадку, якщо необхідно враховувати такий перехід, доведеться все-таки скористатися заздалегідь визначеними у класі TimeZone часовими поясами, які підтримують таку можливість. Дізнаєтися про те, чи підтримує певний часовий пояс можливість переходу, можна, викликавши метод useDaylightTime(), як буде показано у наступному прикладі.

Порівнямо часовий пояс для стандартного локального київського часу та користувачький часовий пояс, який на 15 хвилин відстас від часу за Києвом:

```

import java.util.*;
class TimeZoneDemo {
    public static void printInfo(TimeZone timezone) {

```

```

        System.out.println("Часовий пояс:"
+timezone.getDisplayName(timezone.useDaylightTime(),
TimeZone.SHORT));
        System.out.println("ID часового поясу:
"+timezone.getID());
        System.out.println("Довга назва часового поясу:
"+timezone.getDisplayName());
        String suffix = "";
        switch (timezone.getRawOffset()/3600000) {
            case 1: suffix = "a";
            break;
            case 2: case 3: case 4: suffix = "и";
            break;
            default: suffix = "";
        }
        System.out.println("Зсув відносно UTC: "
+timezone.getRawOffset()/3600000+" годин"+suffix+
+(timezone.getRawOffset()%3600000)/60000+" хвилин");
        String status = (timezone.useDaylightTime()) ? "" :
"не ";
        System.out.println("Часовий пояс "+status+"підтримує
перехід на літній/зимовий час");
    }
    public static void main(String[] argv) {
        TimeZoneDemo.printInfo(TimeZone.getTimeZone("Europe/Kiev"));
        TimeZoneDemo.printInfo(TimeZone.getTimeZone("GMT+145"));

    }
}

```

1.1.2 Клас java.util.Locale

Для чималої кількості застосунків є важливим питання якою природною мовою або мовами вони здатні вводити і виводити інформацію. Крім того у різних країнах місцинах та навіть іноді в межах однієї країни мови можуть відрізнятися. Тому час від часу постає питання яким чином позначити місцевість та мову яка використовується в цій місцевості задля розв'язання цього питання було утворено купу міжнародних стандартів якими позначається місцевість (країна, або її частина) та позначка мови яку треба використовувати стосовно тої чи іншої країни або частини країни.

Процес інтеграції певної національної мови у якості основної мови застосунку найчастіше називають локалізацією, а процес інтеграції кількох мов інтернаціоналізацією. Задля збереження мовної інформації використовується клас Locale бібліотеки java.util - він здатен зберігати стандартизовані дво чи три символльні позначки мови, двозначні позначки країни, а також деяку додаткову інформацію щодо питання мов. Зокрема одним зі стандартів визначені стандартні назви локалізацій для певних країн, які в класі Locale вказані як

константи. В поточній роботі задля отримання об'єкту класа Locale використовуватиметься статичний метод *forLanguageTag* який дозволяє створити об'єкт для поданої мови. Зауважте що українська мова у міжнародних стандартах позначена тегом UK, а не UA як ми з вами самі звикли себе позначати. У разі якщо операція створення об'єкту Locale не може бути правильно виконана через помилкове позначення мови або країни то автоматично буде створено об'єкт який позначатиме стандартну англійську мову.

1.1.3. Клас java.util.Calendar

Починаючи з JDK цей клас став основним інтерфейсом для маніпуляції з датами та часом. Клас Calendar є абстрактним, тому аби створити об'єкт календаря необхідно викликати статичний метод *get.Instance()*. Цей клас містить поля, які зберігають інформацію про рік, місяць, день, годину, хвилину, секунду та частку секунди певного моменту часу з урахуванням інформації про часовий пояс, тому для виконання операцій встановлення значень тих чи інших полів треба звернути увагу або на операцію *set* або на операцію *set_time* яка здатна встановити значення відповідних полів, використовуючи поданий об'єкт класу Date у якості джерела. Також вкрай важливою властивістю класу Calendar є можливість застосування до інформації про час так званої арифметики часу. Арифметика часу - це пошук моменту у майбутньому чи минулому шляхом додавання певного зсуву, у випадку класу Calendar виклик методу *add* дозволяє додати або відняти ціле значення від заданого поля об'єкту Calendar. Задання поля календаря виконується через використання відповідної статичної змінної.

1.1.4 Клас java.util.Date

Клас Date містить інформацію про рік, місяць, день, годину, хвилину, секунду та наносекунду певного моменту часу. Внутрішній формат зберігання цієї інформації виглядає як велике ціле число, яке містить кількість наносекунд, що минули від початку епохи, але у випадку читання цього цілого числа, відповідні методи повертають кількість мілісекунд від початку епохи. Тому задля отримання наносекунд, треба використовувати методи, що повертають кількість саме наносекунди. Варто зазначити, що оскільки починаючи з версії JDK 1.1, переважна більшість операцій з датою та часом покладена саме на клас Calendar та його нащадків, об'єкти класу Date найчастіше виступають у допоміжній ролі. Основні операції щодо трансформації відомостей про дату та час варто виконувати за допомогою класу календар, його нащадків, а також за допомогою класів *DateFormat* та *SimpleDateFormat* бібліотеки *java.text*.

1.1.5 Клас java.util.TimeStamp

Типи даних що містять інформацію про час часто використовуються в базах даних при цьому форматів збереження часової інформації так само розрізняють два: у вигляді цілого числа та у вигляді рядка. Найкомпактнішим та

найзручнішим для маніпуляцій є ціле число. Тому стандартний інтерфейс підключення баз даних реалізований у бібліотеці JDBS (Java Database Connectivity), містить тип даних TimeStamp який фактично показує кількість секунд що минули від початку епохи. Але ж відомо що типи date та calendar бібліотеки java.util здатні працювати навіть з наносекундами. Тому до складу бібліотеки java.util був доданий спеціальний клас, який є містком між класом Date та типом TIMESTAMP структурованої мови запитів SQL. Цей клас крім іншого містить методи, що дозволяють розбирати інформацію про час, подані у базі даних у вигляді рядка, а також форматувати відповідну часову інформацію у вигляді рядка перед записом до бази даних.

1.1.6 Клас java.text.SimpleDateFormat

Зважаючи на те, що найзручнішим представленням дати та часу є саме текстовий рядок, в програмах, які обробляють дані про дату та час, часто виникає потреба вивести відформатовану часову інформацію, або навпаки, взяти інформацію про дату та час у певному вигляді з тексту. Класи з пакета java.util не містять методів, які дозволили впоратись з цими завданнями. Тож пакет java.text містить класи для форматування або обробки форматованого тексту, зокрема, для дат та часу. В цьому пакеті є два класи: SimpleDateFormat та DateFormat. В поточній роботі буде розглянуто тільки SimpleDateFormat, оскільки він дозволяє під час створення об'єкта задати формат для розбору рядка або форматування, використовуючи спеціальні позначки.

Після створення об'єкта SimpleDateFormat, найчастіше вживаються методи цього об'єкту parse та format. Зауважте, що у разі використання локалізованої інформації, наприклад, українських позначок місяців та днів тижня, інформацію про локалізацію можна передати, як у момент створення об'єкта SimpleDateFormat, так безпосередньо в момент аналізу або створення рядка з датою. Це виконується шляхом створення відповідного об'єкту Locale.

1.1.7 Приклади використання класів

Найчастіше дані про дату та час використовуються в інформаційних системах, де накопичуються інформація, аналіз якої у часі важить. Часто такі системи передбачаються у якості сховищ бази даних або форматованих текстових файлів, для яких під час обробки використовуються різні колекції об'єктів включно з масивами об'єктів. Щодо баз даних, то використання там дати та часу залежить від машини баз даних та підтримуваних нею стандартних типів даних. Сучасні реляційні бази даних мають власний інструментар щодо обробки інформації щодо дати та час, який доступний програмістам шляхом використання структурованої мови запитів SQL. В цьому випадку програміст стикається з необхідністю обробляти інформацію такого штибу тільки задля виведення на екран чи друк або у випадку введення користувачем подібних даних. Розглянемо два найпростіші приклади в одному з яких, поточну дату та час буде

відформатовано у певний спосіб (приклад 1), а в другому, буде розібрано рядок, що містить інформацію про дату у певному форматі (приклад 2).

Приклад 1. Demo.java

```
import java.util.*;
import java.text.*;
public class Demo {
    public static void main(String[] argv) {
        Date currentDate = new Date();
        SimpleDateFormat f = new SimpleDateFormat("EEE, d MMM
yyyy", Locale.forLanguageTag("uk"));
        System.out.println(f.format(currentDate));
    }
}
```

Приклад 2. Demo1.java

```
import java.util.*;
import java.text.*;
public class Demol {
    public static void main(String[] argv) {
        try {
            SimpleDateFormat f = new SimpleDateFormat("dd MMM
yyyy", Locale.forLanguageTag("uk"));
            Date d = f.parse("01 березня 2023");
            System.out.println(d);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

В тих системах, в яких сховищем виступають форматовані файли, окрім питань щодо введення та виведення дати та часу, постає також куча питань, пов'язаних з арифметикою дат та усілякими порівняннями про дату та час. Це потрібно задля сортування та фільтрації даних в інформаційній системі, а також можливості їх аналітичної обробки.

Арифметика дат передбачає обчислення дати та часу спираючись на наявну дату та час шляхом додавання або віднімання певного проміжку. Для виконання цих дій щодо класів з `java.util`, існує два способи. Оскільки внутрішній формат збереження дати та часу для класів цього пакету виглядає як велике ціле число, що містить кількість мілісекунд, які минули від початку епохи, то можна або власноруч додати чи відняти певну кількість мілісекунд, аби отримати бажаний зсув у часі, або скористатися методом `add` класу `Calendar`. При цьому додатній зсув, що передається в `add`, знаходить точку часу у майбутньому, а від'ємний зсув

- у минулому. Наступний приклад демонструє відповідні обрахунки обома способами.

Приклад 3. Demo2.java

```
import java.util.*;
import java.text.*;
public class Demol {
    public static void main(String[] argv) {
        Calendar c = Calendar.getInstance();
        // минуле
        // 1 хвилина тому
        c.setTime(new Date());
        Date d = c.getTime();
        d.setTime(d.getTime() - 60000);
        c.setTime(d);
        System.out.println(c.getTime());
        // 1 година тому
        c.setTime(new Date());
        c.add(Calendar.HOUR, -1);
        System.out.println(c.getTime());
        // 1 доба тому
        c.setTime(new Date());
        Date d1 = new Date();
        d1.setTime(d1.getTime() - 24 * 3600 * 1000);
        c.setTime(d1);
        System.out.println(c.getTime());
        // 1 місяць тому
        c.setTime(new Date());
        c.add(Calendar.MONTH, -1);
        System.out.println(c.getTime());
        // 1 рік тому
        c.setTime(new Date());
        Date d2 = c.getTime();
        d2.setTime(d2.getTime() - 365 * 24 * 2600 * 1000);
        c.setTime(d2);
        System.out.println(c.getTime());
        // майбутнє
        // 1 хвилина під тому
        c.setTime(new Date());
        c.add(Calendar.MINUTE, 1);
        System.out.println(c.getTime());
        // 1 година по тому
        c.setTime(new Date());
        Date d3 = new Date();
        d3.setTime(d3.getTime() + 3600 * 1000);
        c.setTime(d3);
        System.out.println(c.getTime());
        // 1 доба по тому
```

```

        c.setTime(new Date());
        c.add(Calendar.DAY_OF_MONTH, 1);
        System.out.println(c.getTime());
        // 1 місяць по тому
        c.setTime(new Date());
        Date d4 = c.getTime();
        d4.setTime(d2.getTime() + 30 * 24 * 3600 * 1000);
        c.setTime(d4);
        System.out.println(c.getTime());
        // 1 рік по тому
        c.setTime(new Date());
        c.add(Calendar.YEAR, 1);
        System.out.println(c.getTime());
    }
}

```

У випадку якщо пошук, сортування та фільтрація даних, що містять часову компоненту, має виконуватись в коді програми, варто створити окремі класи, що моделюють такі дані і надалі колекціонувати їх, використовуючи весь наявний інструментар мови Java. У наступному прикладі розглянуто найпростіший випадок. Для обліку книг у бібліотеці, бібліотекар робить записи про видані читачам книжки. Запис містить назву книжки, ПІБ читача, дату видачу книжки та дату повернення. Клас журналу з відповідними записами передбачає внутрішнє представлення колекціонованих записів у вигляді масиву. Демонстраційний код містить заповнення масиву випадковими даними. Також клас журналу має методи, які дозволяють перевірити чи передує один запис у журналі іншому та методи сортування та друку записів. Прошу зауважити, що цей код написано задля демонстрації принципу обробки колекцій, що містять елементи з інформацією про час. Реальна інформаційна система виглядатиме суттєво складніше і відповідний журнал у такій системі теж виглядатиме інакше. Проте, приклад наочно демонструє як можна виконувати обробку та, відповідно, будувати аналітику для таких колекцій.

Приклад 4. Journal.java

```

import java.util.*;
import java.text.*;
class Record {
    // назва книжки
    private String BookTitle;
    // ім'я читача
    private String ReaderName;
    // дата отримання книги читачем
    private Date IssueDate;
    // дата повернення книги читачем
    private Date TurnBackDate;
}

```

```
    public Record(String booktitle, String readername, Date issuedate) {
        BookTitle = booktitle;
        ReaderName = readername;
        IssueDate = issuedate;
        TurnBackDate = null;
    }
    String getBookTitle() {
        return BookTitle;
    }
    void setBookTitle(String title) {
        BookTitle = title;
    }
    String getReaderName() {
        return ReaderName;
    }
    void setReaderName(String name) {
        ReaderName = name;
    }

    Date getIssueDate() {
        return IssueDate;
    }

    void setIssueDate(Date issue) {
        IssueDate = issue;
    }

    Date getTurnBackDate() {
        return TurnBackDate;
    }

    void setTurnBackDate(Date turnback) {
        TurnBackDate = turnback;
    }
}

public class Journal {
    private Record[] records;

    public Journal(int size) {
        records = new Record[size];
    }

    Record[] getRecords() {
        return records;
    }

    void setRecord(int index, Record rec) {
```

```

        records[index] = rec;
    }

    static boolean after(Record rec1, Record rec2) {
        return rec1.getIssueDate().after(rec2.getIssueDate());
    }

    void print() {
        DateFormat df = new SimpleDateFormat("dd.MM.yyyy");
        for (int i=0; i<records.length; i++) {
            System.out.print(records[i].getBookTitle() + "\t"
                            + records[i].getReaderName() +
"\t"
                +
df.format(records[i].getIssueDate()));
            if (records[i].getTurnBackDate() != null)
                System.out.println(df.format(records[i].getTur
nBackDate()));
            else
                System.out.println();
        }
    }

    void sort() {
        for (int i=0; i<records.length-1; i++)
            for (int j=0; j<records.length-1-i; j++)
                if (after(records[j], records[j+1])) {
                    Record r = records[j+1];
                    records[j+1] = records[j];
                    records[j] = r;
                }
    }

    public static void main(String[] argv) {

        SimpleDateFormat df = new
SimpleDateFormat("dd.MM.yyyy");
        Journal j = new Journal(5);
        try {
            j.setRecord(0, new Record("Кобзар", "Сокольська
Ю.К.", df.parse("16.02.2023")));
            j.setRecord(1, new Record("Лісова пісня", "Юхимович
О.П.", df.parse("09.01.2023")));
            j.setRecord(2, new Record("Інтернаціоналізм або
русифікація?", "Сінчук Є.М.", df.parse("25.01.2023")));
            j.setRecord(3, new Record("Сагайдачний", "Доленко
К.І.", df.parse("19.01.2023")));
            j.setRecord(4, new Record("Чорний ворон.
Залишеньець.", "Коваль М.", df.parse("17.12.2022")));
        }
    }
}

```

```

        j.sort();
        j.print();
    } catch (Exception e) {
        System.out.println("Виникла помилка");
    }
}
}
}

```

2. Порядок виконання роботи

2.1 Для здобуття мінімально необхідного похідного балу за виконання лабораторної роботи з таблиці 2.1 обрати завдання до виконання. Номер варіанту за списком для кожної підгрупи.

2.3 Для здобуття більш високого балу за виконання лабораторної роботи виконати додатково одне завдання (рівень 1) або два завдання (рівень 2) з таблиці 2.2.

2.3 Створити та надати викладачеві звіт з виконання лабораторної роботи. Звіт повинен містити тему, мету та обладнання для виконання лабораторної роботи, короткий опис постанови задачи для кожного з завдань, лістинг програми, скріншоти результатів тестування програми.

3. Варіанти індивідуальних завдань

Таблиця 2.1

Варіант	Завдання
1	Роздрукувати дати останніх двох тижнів, коли проводилися заняття з дисципліни «Технологія програмування мовою Java», задавши на вході дні тижня.
2	Роздрукувати дні тижня для своїх днів народження за останні 5 років.
3	Визначити місцевий час прильоту літака, що вилетів з Києва о 8:00 ранку до Парижа, якщо час польоту дорівнює 3:50.
4	Задати деяку дату і роздрукувати список 10 високосних років, що передували цій даті.
5	Задати деяку дату та визначити кількість днів, годин та хвилин між заданою датою та поточною.
6	Запустити одну із створених вами програм та визначити час її виконання.
7	Роздрукувати графік свят на поточний рік, використовуючи форматування.
8	Роздрукувати дні тижня для видачі зарплат співробітникам деякої фірми за останні 6 місяців, враховуючи дати 14 та 28 число кожного місяця.

Варіант	Завдання
9	Розрахувати кількість років, місяців та днів між поточною датою та днями народження членів вашої родини.
10	Вести у вигляді рядків дати народження членів вашої сім'ї та роздрукувати їх у форматі "уууу-ММ-дд".

Таблиця 2.2

Варіант	Завдання
1	Інтернет-крамниця закуповує три види товарів, та продає їх з націнкою. Закупівельна ціна кожної партії товару може відрізнятися від попередньої. Ціна продажу товару також може відрізнятися від попередньої, оскільки можуть бути застосовані персональні або акційні знижки. Створіть класи, щоб дозволити вести облік придбання та продажів. Розроблене програмне забезпечення має надати можливість переглядати хронологію торговельних операцій, та обраховувати зиск крамниці за певний період часу. Задля збереження даних про закупівлі та продажі використовуйте масив чи будь-яку відому Вам колекцію. При підрахунку зиску період має бути довільним. Межі періоду повинні вводитись з клавіатури. Зиском вважати різницю між витратами на закупівлі та виторгом від продажу.
2	Телевізійний канал ретранслює свої програми у Західній, Центральній та Східній Європі (GMT+0, GMT+1 та GMT+2 відповідно). Програми можуть транслюватися одночасно у всіх трьох регіонах, або можуть бути призначені для кожного регіону окремо. Треба створити редактор розкладу програм, який би не дозволяв додавати програму для певного часового поясу, якщо у цей час вже заплановано якісь інші програми. Розроблене програмне забезпечення має на запит друкувати розклад програм для вказаної користувачем частини Європи.
3	Бібліотекар веде облік виданих та повернутих читачами бібліотеки книжок. За правилами бібліотеки читач може ознайомлюватись з вмістом книжки протягом одного календарного місяця та одночасно тримати на руках не більше 5 видань. Розробити програмне забезпечення, що дозволяє вести такий облік. Розроблене програмне забезпечення може використовувати масив чи будь-яку відому Вам колекцію для збереження інформації про видачу/повернення книжок. Також повинен бути функціонал, що не

	дозволяв би видавати книжок більше, ніж дозволено, та дозволяв би переглянути перелік користувачів, хто не повернув книжки вчасно.
4	Мережа захищеного зв'язку складається з трьох каналів. Вимір завантаження мережі відбувається що 20 секунд. Використовуючи генератор випадкових чисел імітувати навантаження мережевим трафіком. Знайти найдовший проміжок протягом доби, коли навантаження було найменшим, та найкоротший проміжок, коли навантаження було найбільшим.
5	Рецептура виготовлення суміші для будівництва передбачає, що вона складатиметься з трьох компонентів з частками 45%, 33% та 22% відповідно. Обладнання для змішування складається з 4 ємностей: по одній для кожного компонента та ємність для змішування (по 3 м ³ та 1 м ³ відповідно). Змішування відбувається протягом 3 хвилин за умови, що наявні всі три компоненти. Якщо якогось з компонентів не вистачає, процес зупиняється на дозаповнення, що триває від 1 хвилини 15 секунд до 1 хвилини 30 секунд (визначити за допомогою генератора випадкових чисел). За цей час відповідна ємність заповнюється по вінця. Створити програмне забезпечення, яке імітує журнал роботи обладнання. Програмне забезпечення повинно забезпечити можливість розрахунку кількості суміші, виробленої за визначений користувачем проміжок часу, а також загальний час простою за одну добу роботи.

4. Питання для самоконтролю

1. Які основні різновиди представлення дат та часу існують в комп'ютерних системах.
2. Опишіть різницю між поняттями локальний стандартний час та локальний годинниковий час.
3. Яким чином можна визначити, коли відбувається перехід на літній/зимовий час для окремого часового поясу.
4. Поясніть різницю між процесами локалізації та інтернаціоналізації національної мови.
5. Як виглядає внутрішній формат зберігання інформації у класі Date.
6. Який спеціальний клас є інтерфейсом між класом Date та типом TIMESTAMP структурованої мови запитів SQL.