

**Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»**

**Навчально-науковий інститут атомної і теплової енергетики  
Кафедра цифрових технологій в енергетиці**

## **ЗВІТ**

**з лабораторної роботи №3**

**з дисципліни «Програмування на мові Java»**

**Тема: «Розробка програм в середовищі INTELLIJ IDEA для  
дослідження та особливостей забезпечення безпеки даних»**

**Варіант №22**

**Виконав:  
Студент групи ТР-12  
Ковальов О. О.**

**Дата здачі: 29.10.2023**

**Мета роботи:** набуття практичних навичок під час створення програмних проєктів на мові Java, які виявляють, аналізують або виправляють потенційні проблеми безпеки даних.

### **Загальне завдання на лабораторну роботу:**

1. Написати програму мовою Java. Варіант обрати за списком групи. Представити виконання в IntelliJ IDEA. Продемонструвати детальні скріншоти виконання коду програми з поясненням. Обов'язково – наявність висновків.

2. Написати програму мовою Java. Варіант обрати за списком групи. Представити виконання в IntelliJ IDEA. Продемонструвати детальні скріншоти виконання коду програми з поясненням. Обов'язково – наявність висновків.

3. Зробити звіт з лабораторної роботи та вчасно надіслати викладачу на перевірку (дедлайн для надсилання звітів по **Лаб\_3 – до 29.10.2023 року до 23:59**).

**Завдання 1:** Написати програму мовою Java для обчислення часу, що необхідно для перебору всіх можливих варіантів паролів у системі із наступними параметрами: A=26 (що включає цифри та літери верхнього регістру), що складається з 12 елементів (L=12). Обрати інтерактивну швидкість підбору паролів (V=5 паролів/сек). Вивести результат у вигляді текстового повідомлення, що покаже скільки годин, хвилин, секунд потрібно для виконання завдання (приклад виведеного повідомлення: *Час перебору всіх паролів: ... годин, ... хвилин, ... секунд*). Якщо час більше 24 годин – додати поле «дні». Обов'язково додати перевірку на валідність введеної інформації користувачем.

Для того, щоб розв'язати задачу було написано декілька класів.

Клас BruteForceCalculator визначає обчислення часу, необхідного для зламу паролю методом "грубої сили". Цей клас має три приватні поля: cardinality, passwordLength та speed, і включає методи для їх отримання та встановлення.

Конструктор класу BruteForceCalculator використовується для ініціалізації об'єкта і приймає три аргументи. В конструкторі викликаються методи «сеттери» для встановлення відповідних полів класу.

Метод seconds() обчислює час, необхідний для зламу паролю на основі значень відповідних полів. Результат повертається у формі числа секунд.

Клас також містить методи доступу (getters та setters) для полів класу, вони використовують валідатор BruteForceValidator для перевірки правильності вхідних даних перед встановленням значень полів.

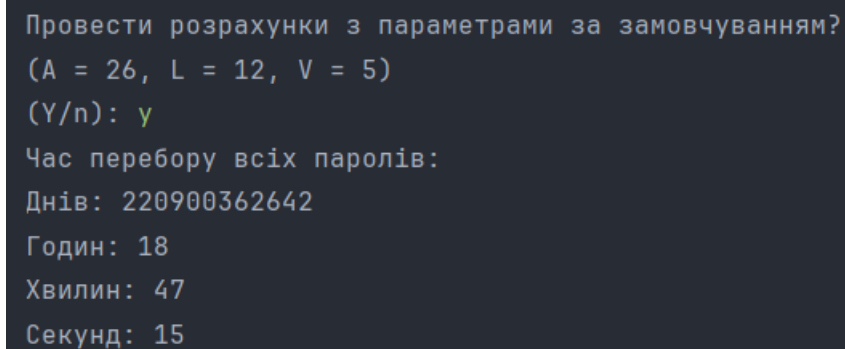
В цілому, клас допомагає розраховувати час для зламу паролю, враховуючи кількість можливих символів (cardinality), довжину паролю (passwordLength) і швидкість перебору паролів (speed).

Клас BruteForceValidator містить три статичні методи для валідації числових параметрів: cardinality, passwordLength, і speed. Вони перевіряють, чи передані значення більше нуля і генерують IllegalArgumentException, якщо перевірка не пройдена.

Клас `TimeFormatter` має статичний метод `seconds`, який форматує кількість секунд у текстовий рядок, що представляє час у більших одиницях, таких як дні, години, хвилини та секунди. Метод перевіряє, чи передане значення `value` є невід'ємним. У випадку від'ємного значення, генерується виключення `IllegalArgumentException` з повідомленням `"Seconds' value must be non-negative."` Потім він розраховує кількість днів, годин, хвилин і секунд, поділяючи вхідне значення `value` на відповідні кількості секунд в цих одиницях. Якщо значення більше нуля для певної одиниці часу, воно додається до результуючого рядка. В результаті отримуємо текстове представлення часу, що включає в себе дні, години, хвилини і секунди, і повертає його як рядок.

В класі `Main` відбувається введення даних користувачем.

**Висновок:** В задачі був застосований основний принцип ООП – інкапсуляція, були застосовані методи «гетери» та «сетери», також класи були розбиті по підзадачам. Все працює.



```
Провести розрахунки з параметрами за замовчуванням?  
(A = 26, L = 12, V = 5)  
(Y/n): y  
Час перебору всіх паролів:  
Днів: 220900362642  
Годин: 18  
Хвилин: 47  
Секунд: 15
```

Рис.1. Демонстрація роботи програми

**Завдання 2:** Написати програму мовою Java, яка приймає пароль від користувача та перевіряє його за вказаними критеріями безпеки.

Програма повинна видати попередження, якщо пароль користувача не відповідає наступним вимогам:

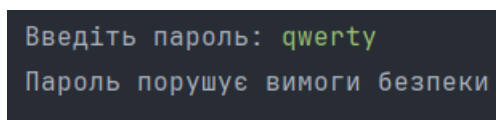
- довжина не менше 12 символів;
- не повинен містити наступну інформацію (тобто – не повинен бути простим) – `password`, `qwerty` або `123456`, і т. д.;

Обов'язково додати перевірку на валідність введеної інформації користувачем. При умові виконання всіх вимог, на екран виводиться повідомлення «Пароль валідний». При умові порушення хоча б однієї вимоги щодо безпеки – «Пароль порушує вимоги безпеки».

Клас `Validator` містить статичні поля `validLength` і `passwords`. У ньому є статичний метод `check`, який перевіряє переданий пароль і виводить відповідне повідомлення щодо його валідності. Цей клас також містить два приватних методи: `isPasswordValid`, який перевіряє відсутність пробілів у паролі, і `isPasswordStrong`, який визначає, чи пароль відповідає вимогам безпеки.

Клас `FileReader` містить метод `readPasswordDictionary`, який відповідає за читання списку паролів з файлу `"PasswordDictionary.txt"`, де містяться 100 найбільш вживаних паролів. Метод створює порожній список рядків з назвою `passwords`. Потім відкриває файл за вказаним шляхом `PATHNAME` за допомогою об'єкта `Scanner`. Починає послідовно зчитувати рядки з файлу, додаючи їх до списку `passwords`. Після завершення читання файлу, він закриває об'єкт `Scanner`. Нарешті, метод повертає список `passwords`, який містить усі паролі, зчитані з файлу. Цей метод допомагає отримати список паролів з файлу для подальшої перевірки їх на валідність та вимоги безпеки у класі `Validator`.

**Висновок:** У класі `Main` використовується клас `Scanner` для отримання введеного користувачем пароля з консолі. Після того, як користувач вводить пароль, він зчитується, видаляються зайві пробіли з початку і кінця рядка за допомогою методу `trim()`, і конвертується в нижній регістр за допомогою методу `toLowerCase()`. Потім викликається метод `Validator.check(password)`, який відповідає за перевірку валідності та відповідності вимогам безпеки введеного пароля. В залежності від результату перевірки, в консоль виводиться відповідне повідомлення щодо пароля - чи він валідний, чи порушує вимоги безпеки. Таким чином, цей клас представляє спосіб введення користувачем пароля та використання класу `Validator` для його перевірки.



```
Введіть пароль: qwerty
Пароль порушує вимоги безпеки
```

Рис.2. Демонстрація роботи програми

**Завдання 3:** Написати програму мовою Java, яка приховує конфіденційні дані в текстовому повідомленні. Користувач вводить будь-яке текстове повідомлення (студент особисто обирає приклади текстових повідомлень: в першому повідомленні повинна бути вказана інформація з умовними даними платіжної картки (16 цифр, послідовно або через пробіл), в другому повідомленні – повинен з'явитися номер телефону (у форматі 0970000000 або 097-000-00-00), в третьому – комбінування двох попередніх варіантів повідомлень.

При появі інформації про номер телефону чи реквізити платіжних карток, програма повинна приховати конфіденційні дані, замінивши їх на символ `"*"` на кожен прихований символ. Наприклад, дані платіжної картки програма замінює на `«*****»`, а номер на `«0*****»` або `«0**-***-**-**»`.

Клас `SecureText` містить метод `mask`, який виконує обробку тексту з метою маскуванню номерів телефонів та кредитних карт. Цей метод викликає два приватні методи: `maskPhone` для маскуванню номерів телефонів і `maskCreditCard` для маскуванню номерів кредитних карт.

Метод `maskPhone` використовує регулярні вирази для пошуку та маскуванню номерів телефонів, які можуть мати різні формати, такі як `"097-123-45-67"` або `"097`

123 45 67". Він замінює цифри в номерах на символи "\*", залишаючи перший символ незмінним.

Метод `maskCreditCard` також використовує регулярні вирази для пошуку та маскувння номерів кредитних карт, які можуть бути 16 цифр у рядок або розділені пробілами або дефісами. Він також замінює цифри на символи "\*".

У класі `Main` користувач вводить текстове повідомлення через консоль, яке зчитується за допомогою класу `Scanner`. Потім це текстове повідомлення передається до методу `SecureText.mask(message)`, який відповідає за обробку тексту та маскувння номерів телефонів і кредитних карт, якщо вони присутні в тексті. Результат обробки виводиться в консоль за допомогою `System.out.printf()`. Основна функція цього класу - захистити конфіденційну інформацію в тексті, маскуючи її так, щоб вона не була видима в виводі.

**Висновок:** Ці класи вдало виконують свою задачу, а саме допомагають захистити конфіденційну інформацію у тексті, маскуючи чутливі дані, щоб їх не було видно у виводі або логах.

```
Введіть текстове повідомлення:
Lorem ipsum 097-380-12-23 Lorem 0972314153 Some text4149414911111111somet s4149 1234 1423 1233s
Результат:
Lorem ipsum 0**-***-**-** Lorem 0***** Some text*****somet s**** **** **** ****
Process finished with exit code 0
```

Рис.3. Демонстрація роботи програми

## Додаток 1. Лістинги

### Завдання 1.

```
package Lab3.Task1;
```

```
public class BruteForceCalculator {
    private int cardinality;
    private int passwordLength;
    private int speed;

    public BruteForceCalculator(int cardinality, int passwordLength, int
speed) {
        setCardinality(cardinality);
        setPasswordLength(passwordLength);
        setSpeed(speed);
    }

    public long seconds() {
        long systemCardinality = (long) Math.pow(cardinality,
passwordLength);

        return systemCardinality / speed;
    }

    public int getCardinality() {
        return cardinality;
    }

    public void setCardinality(int cardinality) {
        BruteForceValidator.cardinality(cardinality);
    }
}
```

```

        this.cardinality = cardinality;
    }

    public int getPasswordLength() {
        return passwordLength;
    }

    public void setPasswordLength(int passwordLength) {
        BruteForceValidator.passwordLength(passwordLength);

        this.passwordLength = passwordLength;
    }

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        BruteForceValidator.speed(speed);

        this.speed = speed;
    }
}

package Lab3.Task1;

public class BruteForceValidator {
    public static void cardinality(int c) {
        if (c <= 0) throw new IllegalArgumentException("Cardinal must be
greater than 0");
    }

    public static void passwordLength(int l) {
        if (l <= 0) throw new IllegalArgumentException("Password length must
be greater than 0");
    }

    public static void speed(int s) {
        if (s <= 0) throw new IllegalArgumentException("Password picking
speed must be greater than 0");
    }
}

package Lab3.Task1;

public class TimeFormatter {
    public static String seconds (long value) {
        if (value < 0) throw new IllegalArgumentException("'Seconds' value
must be non-negative.");

        var sb = new StringBuilder();
        sb.append("Час перебору всіх паролів:\n");

        int seconds = (int) (value % 60);
        int minutes = (int) (value % 3600 / 60);
        int hours = (int) (value % 86400 / 3600);
        long days = value / 86400;

        if (days != 0) sb.append(String.format("Днів: %d\n", days));
        if (hours != 0) sb.append(String.format("Годин: %d\n", hours));
        if (minutes != 0) sb.append(String.format("Хвилин: %d\n", minutes));
        sb.append(String.format("Секунд: %d", seconds));
    }
}

```

```

        return sb.toString();
    }
}

package Lab3.Task1;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int A = 26;
        int L = 12;
        int V = 5;

        var scanner = new Scanner(System.in);

        System.out.println("Провести розрахунки з параметрами за замовчуванням?");
        System.out.printf("(A = %d, L = %d, V = %d)\n", A, L, V);
        System.out.print("(Y/n): ");
        if (!scanner.next().toLowerCase().trim().equals("y")) {
            System.out.print("1. Кардинал алфавіту: ");
            A = scanner.nextInt();

            System.out.print("2. Довжина паролю: ");
            L = scanner.nextInt();

            System.out.print("3. Швидкість підбору (в сек.): ");
            V = scanner.nextInt();
        }

        var calculator = new BruteForceCalculator(A, L, V);
        var seconds = calculator.seconds();
        System.out.println(TimeFormatter.seconds(seconds));
    }
}

```

## Завдання 2.

```

package Lab3.Task2;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class FileReader {
    final static String PATHNAME = "D:\\Programming\\University\\Java-Programming\\src\\Lab3\\Task2\\PasswordDictionary.txt";

    public static List<String> readPasswordDictionary() throws
    FileNotFoundException {
        var passwords = new ArrayList<String>();

        var scanner = new Scanner(new File(PATHNAME));

        while (scanner.hasNextLine()) {
            passwords.add(scanner.nextLine());
        }

        scanner.close();

        return passwords;
    }
}

```

```

package Lab3.Task2;

import java.io.FileNotFoundException;
import java.util.List;

public class validator {
    private final static int validLength = 12;
    private final static List<String> passwords;

    static {
        try {
            passwords = FileReader.readPasswordDictionary();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    public static void check(String password) {
        if (!isPasswordValid(password))
            System.out.println("Пароль не валідний");
        else if (!isPasswordStrong(password))
            System.out.println("Пароль порушує вимоги безпеки");
        else
            System.out.println("Пароль валідний");
    }

    private static boolean isPasswordValid(String password) {
        if (password.contains(" ")) return false;

        return true;
    }

    private static boolean isPasswordStrong(String password) {
        if (password.length() < validLength) return false;

        for (var p : passwords) {
            if (password.equalsIgnoreCase(p)) return false;
        }

        return true;
    }
}

```

```

package Lab3.Task2;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        var scanner = new Scanner(System.in);

        System.out.print("Введіть пароль: ");
        var password = scanner.nextLine().trim().toLowerCase();
        validator.check(password);
    }
}

```

### **Завдання 3.**

```

package Lab3.Task3;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class SecureText {
    public static String mask(String message) {

```



```

        return maskCreditCard(maskPhone(message));
    }

    private static String maskPhone(String message) {
        var phonePattern = Pattern.compile("\\b097[-\\s]?\\d{3}[-\\s]?\\d{2}[-\\s]?\\d{2}\\b");
        Matcher matcher = phonePattern.matcher(message);

        String result = message;
        while (matcher.find()) {
            var phone = matcher.group();
            var masked = phone.charAt(0)
                + phone.replaceAll("\\d", "*").substring(1);

            result = result.replace(phone, masked);
        }

        return result;
    }

    private static String maskCreditCard(String message) {
        var cardPattern = Pattern.compile("\\d{16}|\\d{4}(?:[-\\s]\\d{4}){3}");
        var matcher = cardPattern.matcher(message);

        String result = message;
        while (matcher.find()) {
            var card = matcher.group();
            var masked = card.replaceAll("\\d", "*");

            result = result.replace(card, masked);
        }

        return result;
    }
}

package Lab3.Task3;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        var scanner = new Scanner(System.in);

        System.out.println("Введіть текстове повідомлення:");
        var message = scanner.nextLine();

        var result = SecureText.mask(message);

        System.out.printf("Результат: \n%s", result);
    }
}

```