Тема 12. Регулярні вирази. Обробка тексту

Мета:

* Ознайомлення з принципами використання регулярних виразів для обробки тексту.

**1 ВИМОГИ**

**1.1 Розробник**

Інформація про розробника:

- Гряник Георгій Володимирович

- КІТ-119Д;

- 6 варіант.

**1.2 Загальне завдання**

1. Використовуючи програми рішень попередніх задач, продемонструвати ефективне (оптимальне) використання регулярних виразів при вирішенні [прикладної задачі](https://oop-khpi.gitlab.io/%D0%B7%D0%B0%D0%B2%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F/task12/#_3).
2. Передбачити можливість незначної зміни умов пошуку.
3. Продемонструвати розроблену функціональність в діалоговому та автоматичному режимах.

1.3 Задача

### **Поліцейська картотека**

Поліцейська картотека. Знайти всіх ув'язнених не молодше 20 років з прізвищем, що починається з голосної та містить комбінацію "ко". Наприклад: Архипенко, Ішкова, Єрмаков, Янковський.

2 ОПИС ПРОГРАМИ

2.1 Засоби ООПРозробити Regex-регулярний вираз(-и) для перевірки та контролю даних при вводі даних чи зчитуванні файлів.

2.2 Ієрархія та структура класівКлас “PoliceFile ” – описує поліцейську картотеку з можливістю додавати та виводити дані класу. Клас описує дані про злочинця відповідно до завданн. Клас «Date” – опису формат часу :день, місяць, рік . Створений для ергономічного запису дат відомостей про злочинця. Клас « Console\_program» - клас керування програми , створений щоб надавати користувачеві можливість керувати програмою. Клас «СontainerList» - клас-контейнер створений для зберігання даних у список. Реалізовано додавання, видалення та інші можливості для керування даними. Клас «Serializator» - клас розроблений для збереження даних контейнеру у файл. При цьому зберігання прохоже у звичайний файл та файл типу .xml. При цьому в класі реалізовано методи для відновлення даних як із звичайного файлу так із .xml файлу. Клас Console\_File розроблений для роботи із файлами розміщені в директоріях. Цей клас забезпечує можливість користувачеві обирати файл та перемінятися між директоріями. Клас «Helper» - реалізація допоміжних методів які реалізують допоміжні дії в основній програмі. Клас «Console\_program\_auto» - клас розроблений для автоматизації програмою.

Клас «Regex» - клас реалізація методів перевірок текстів за допомогою регулярних виразів.

2.3 Важливі фрагменти програми **/\*\***

**\* @author <Георгiй>**

**\***

**\*/**

**import java.util.Date;**

**import java.util.regex.\*;**

**public class Regex {**

**public static boolean Name(String name){**

**Pattern p = Pattern.compile("^[A-ZА-ЯЁЇІ][a-zа-яёїі]{1,15}$");**

**Matcher m = p.matcher(name);**

**return m.matches();**

**}**

**public static boolean Data(String data){**

**Pattern p = Pattern.compile("^([0-2]\\d|3[0-1])[\\,\\.]((0\\d)|(1[0-2]))[\\,\\.](20\\d{2}|19\\d{2})$");**

**Matcher m = p.matcher(data);**

**return m.matches();**

**}**

**public static boolean SupperPF(String supper){**

**Pattern p = Pattern.compile("^Name:[A-ZА-ЯЁЇІ][a-zа-яёїі]{1,15}; Surname:[A-ZА-ЯЁЇІ][a-zа-яёїі]{1,15}; Lastname:[A-ZА-ЯЁЇІ][a-zа-яёїі]{1,15}"**

**+ "; ДН:([0-2]\\d|3[0-1])[\\,\\.](0?\\d|1[0-2])[\\,\\.](200[0-1]|19\\d{2})"**

**+ "; ОПВ:([0-2]\\d|3[0-1])[\\,\\.](0?\\d|1[0-2])[\\,\\.](20\\d{2}|19\\d{2})"**

**+ "; ОЗ:([0-2]\\d|3[0-1])[\\,\\.](0?\\d|1[0-2])[\\,\\.](20\\d{2}|19\\d{2})"**

**+ "; ДС:(([0-2]\\d|3[0-1])[\\,\\.]?(0\\d|1[0-2])[\\,\\.]?(20\\d{2}|19\\d{2})( )?){1,};$");**

**Matcher m = p.matcher(supper);**

**return m.matches();**

**}**

**public static boolean LitePF(String supper){**

**Pattern p = Pattern.compile("^Name:[A-ZА-ЯЁЇІa-zа-яёїі]{1,15}; Surname:[A-ZА-ЯЁЇІa-zа-яёїі]{1,15}; Lastname:[A-ZА-ЯЁЇІa-zа-яёїі]{1,15}"**

**+ "; ДН:([0-9\\,\\.]){8,10}"**

**+ "; ОПВ:([0-9\\,\\.]){8,10}"**

**+ "; ОЗ:([0-9\\,\\.]){8,10}"**

**+ "; ДС:(([0-9\\,\\.]){8,10}( )?){1,};.\*$");**

**Matcher m = p.matcher(supper);**

**return m.matches();**

**}**

**public static boolean NotYoung(String supper){**

**Date date = new Date();**

**Pattern p = Pattern.compile("^([0-2]\\d|3[0-1])[\\,\\.]?(0\\d|1[0-2])[\\,\\.]?(200[0-1]|19\\d{2})$");**

**Matcher m = p.matcher(supper);**

**return m.matches();**

**}**

**public static boolean KoName(String KoName){**

**Pattern p = Pattern.compile("^.\*([KkКк][OoОо]).\*$");**

**Matcher m = p.matcher(KoName);**

**return m.matches();**

**}**

**public static boolean KoNameNotYoung(String KoName){**

**Pattern p = Pattern.compile("^.\*Lastname:[AАОУЭИЫЕЁЯЮІЇЄ].\*([KkКк][OoОо]).\*;.\*ДН:(\\d|[1-2]\\d|3[0-1])[\\,\\.](0?\\d|1[0-2])[\\,\\.](200[0-1]|19\\d{2});.\*$");**

**Matcher m = p.matcher(KoName);**

**return m.matches();**

**}**

**public static boolean SupperData(String supper){**

**Pattern p = Pattern.compile("^(?:(?:31(\\/|-|\\.)(?:0?[13578]|1[02]))\\1|(?:(?:29|30)(\\/|-|\\.)"**

**+ "(?:0?[1,3-9]|1[0-2])\\2))(?:(?:1[6-9]|[2-9]\\d)?\\d{2})$|^(?:29(\\/|-|\\.)0?2"**

**+ "\\3(?:(?:(?:1[6-9]|[2-9]\\d)?(?:0[48]|[2468][048]|[13579][26])|(?:(?:16|[2468][048]|[3579][26]"**

**+ ")00))))$|^(?:0?[1-9]|1\\d|2[0-8])(\\/|-|\\.)(?:(?:0?[1-9])|(?:1[0-2]))\\4(?:(?:1[6-9]|[2-9]\\"**

**+ "d)?\\d{2})$");**

**Matcher m = p.matcher(supper);**

**return m.matches();**

**}**

**}**//////////////////////////////////////

**3 ВАРІАНТИ ВИКОРИСТАННЯ**

Регулярні вирази реалізовані таким чином аби можна було перевірити ім’я ,дату на правильність вводу. При вводі даних регулярний вираз контролює введені дані і у разі помилки вводу програма ігнорує введені дані та просто їх не записує до списку.

Оновлено пошук, Пошук виконує прикладне завдання шляхом перебору даних та перевірці цього об’єкта за допомогою регулярного виразу

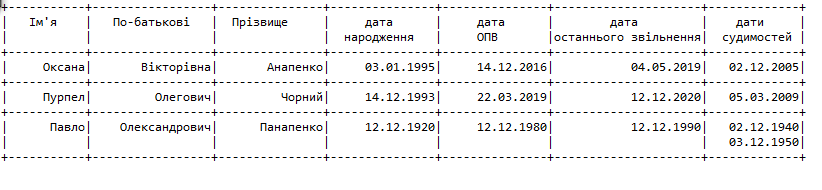


Рисунок 1. Вивід даних масиву

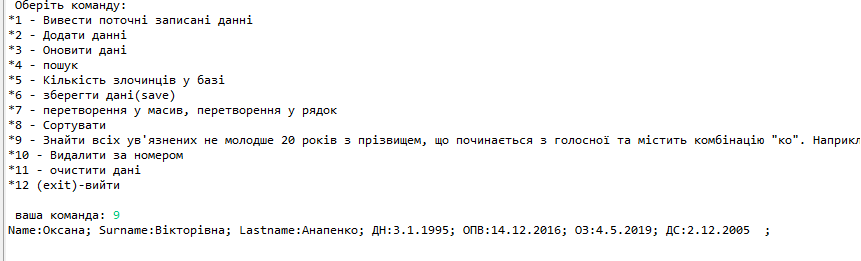


Рисунок 2. Результат пошуку даних за допомогою перевірці регулярним виразом

ВИСНОВКИ

Під час виконання лабораторної роботи було набуто навички роботи створення регулярних виразів та використання їх при зчитування даних з файлу та пошуку.