Звіт

Автор: Васильченко С., 1.КІТ118б

Дата: 10.02.2020

Лабораторна робота №11

РЕГУЛЯРНІ ВИРАЗИ. ПЕРЕВІРКА ДАНИХ

Мета:

- Ознайомлення з принципами використання регулярних виразів для перевірки рядка на відповідність шаблону.

Вимоги:

Продемонструвати ефективне (оптимальне) використання регулярних виразів для перевірки коректності (валідації) даних, що вводяться, перед записом в domain-об’єкти відповідно до призначення кожного поля для заповнення розробленого контейнера:

- при зчитуванні даних з текстового файла в автоматичному режимі;
- при введенні даних користувачем в діалоговому режимі.

ПРИКЛАДНА ЗАДАЧА:

Кадрове агентство. Сортування за назвою фірми, за назвою запропонованої спеціальності, за вказаною освітою.

ОПИС ПРОГРАМИ

2.1 Опис змінних:

LinkedContainer<Recruitment> stringLinked = new LinkedContainer<>();// об’єкт параметризованого контейнера

Recruitment rec1 = new Recruitment(); // об’єкт класа кадрового агенства

Scanner scan = new Scanner(System.in); // змінна для активування зчитування з консолі

2.2 Ієрархія та структура класів.

final class Lab11 – головний клас. Містить метод main(точку входу у програму) та методи по роботі з програмою для реалізації індивідуального завдання.

interface Linked - інтерфейс контенеру

class Recruitment - клас прикладної задачі кадрового агенства

class LinkedContainer - параметризований клас-контейнер, котрий зберігає інформацію агенства

class DescendingIterator – клас, який реалізує обратний ітератор для переміщення по списку

class Util - класс зберігаючий утиліти для обробки контейнера

class Demo – клас для демонстрації виконання програми в різних її варіаціях

class Exe – клас, який має два варіанта виконання, один з яких – меню з варіантами дії.

ТЕКСТ ПРОГРАМИ

File Lab11.java:

```
package ua.khpi.oop.vasilchenko10.Tests;



public class Lab10 {

    public static void main(String[] args) {

        if (args.length != 0) {

            if (args[0].equals("-auto")) {

                Exe.auto();

            } else {

                System.out.println("Repeat entered with params -auto");

            }

        } else {

            Exe.menu();

        }

    }

}
```

Recruitment.java :

```
package ua.khpi.oop.vasilchenko11.First;

import java.util.Comparator;
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

```
public class Recruitment {
    private String firm;
    private String specialty;
    private String workingConditions;
    private int payment;


    private String needsSpeciality;
    private int experience;
    private String education;

    private boolean confirm;

    public void setFirm(final String firm) throws InputMismatchException {
        if (checkFirm(firm)) {
            this.firm = firm;
        } else {
            throw new InputMismatchException();
        }
    }

    private boolean checkFirm(final String firm) {
        Pattern pattern = Pattern.compile("[\\sa-z0-9.%$+#@^()=!_\\\\-]*",
Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(firm);
        return matcher.matches();
    }

    public void setSpecialty(final String specialty) throws InputMismatchException {
        if (checkSpecialty(specialty)) {
            this.specialty = specialty;
        } else {
            throw new InputMismatchException();
        }
    }

    private boolean checkSpecialty(final String specialty) {
        Pattern pattern = Pattern.compile("[\\s\\w.#%@&+=]*", Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(specialty);
        return matcher.matches();
    }

    public void setWorkingConditions(final String workingConditions) throws
InputMismatchException {
        if (checkWorkingConditions(workingConditions)) {
            this.workingConditions = workingConditions;
        } else {
            throw new InputMismatchException();
        }
    }

    private boolean checkWorkingConditions(final String workingConditions) {
        Pattern pattern = Pattern.compile("[\\s\\w%$+#@^()=!_\\\\-]*",
Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(workingConditions);
        return matcher.matches();
    }

    public void setPayment(final int payment) {
        this.payment = payment;
    }

    public void setConfirm(final boolean confirm) {
        this.confirm = confirm;
    }

    public int getPayment() {
        return payment;
```

```
    }

    public String getWorkingConditions() {
        return workingConditions;
    }

    public String getSpecialty() {
        return specialty;
    }

    public String getFirm() {
        return firm;
    }

    public Recruitment() {
        firm = null;
        specialty = null;
        workingConditions = null;
        payment = 0;
        needsSpeciality = null;
        experience = 0;
        education = null;
        confirm = false;

    }

    public String show() {
        String show;
        show = "Фирма: " + firm + "\n" +
                "Специальность: " + specialty + "\n" +
                "Условия работы: " + workingConditions + "\n" +
                "Оплата: " + payment + "\n";
        if (confirm) {
            show += "Необходимая специальность: " + needsSpeciality + "\n";
            show += "Опыт: " + experience + "\n";
            show += "Образование: " + education + "\n";
        }
        return show;
    }

    public Recruitment(final Recruitment obj) {
        firm = obj.firm;
        specialty = obj.specialty;
        workingConditions = obj.workingConditions;
        payment = obj.payment;
        needsSpeciality = obj.needsSpeciality;
        experience = obj.experience;
        education = obj.education;
        confirm = obj.confirm;
    }

    public void generateVacancy() throws InputMismatchException {
        try {
            Scanner scan = new Scanner(System.in);
            Scanner scan2 = new Scanner(System.in);
            int choose;
            System.out.print("\nВведите фирму: ");
            setFirm(scan.nextLine());
            System.out.print("\nВведите специальность: ");
            setSpecialty(scan.nextLine());
            System.out.print("\nВведите условия работы: ");
            setWorkingConditions(scan.nextLine());
            System.out.print("\nВведите оплату: ");
            setPayment(scan.nextInt());
            System.out.println("Желаете добавить дополнительные условия работы? 1 - Да. 0 -
Нет: ");

            boolean loop = true;
            while (loop) {
```

```
                choose = scan.nextInt();
                switch (choose) {
                    case 1:
                        System.out.print("\nВведите необходимую специальность: ");
                        setNeedsSpeciality(scan2.nextLine());
                        System.out.print("\nНеобходимое образование: ");
                        setEducation(scan2.nextLine());
                        System.out.print("\nНеобходимый опыт работы: ");
                        setExperience(scan2.nextInt());
                        confirm = true;
                        loop = false;
                        break;
                    case 0:
                        needsSpeciality = null;
                        experience = 0;
                        education = null;
                        loop = false;
                        break;
                    default:
                        System.out.println("Ошибка! Повторите ввод: ");
                        break;
                }
            }
        } catch (InputMismatchException ex) {
            System.out.println("Wrong input!");
            firm = null;
            specialty = null;
            workingConditions = null;
            payment = 0;
            needsSpeciality = null;
            experience = 0;
            education = null;
            confirm = false;
        }
    }


    public void setExperience(final int experience) throws InputMismatchException {
        if (checkExperience(experience)) {
            this.experience = experience;
        } else {
            throw new InputMismatchException();
        }
    }

    /**
     * Проверка правильности ввода опыта.
     * до 49 лет включительно.
     *
     * @param experience - опыт.
     * @return подтверждение сравнения.
     */
    private boolean checkExperience(final int experience) {
        Pattern pattern = Pattern.compile("^[0-4][0-9]?", Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(String.valueOf(experience));
        return matcher.matches();
    }

    public int getExperience() {
        return experience;
    }

    public void setNeedsSpeciality(final String needsSpeciality) throws InputMismatchException
{
        if (checkNeedsSpeciality(needsSpeciality)) {
            this.needsSpeciality = needsSpeciality;
        } else {
            throw new InputMismatchException();
        }
```

```
    }

    private boolean checkNeedsSpeciality(final String needsSpeciality) {
        Pattern pattern = Pattern.compile("[\\s\\w%$+#@^()=!_\\\\-]*",
Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(needsSpeciality);
        return matcher.matches();
    }


    public String getNeedsSpeciality() {
        return needsSpeciality;
    }

    public void setEducation(final String education) throws InputMismatchException {
        if (checkEducation(education)) {
            this.education = education;
        } else {
            throw new InputMismatchException();
        }
    }

    private boolean checkEducation(final String education) {
        Pattern pattern = Pattern.compile("[a-z]*-?\\s?", Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(education);
        return matcher.matches();
    }

    public String getEducation() {
        return education;
    }

    public boolean getConfirms() {
        return confirm;
    }

    @Override
    public String toString() {
        return show();
    }

    public static final Comparator<Recruitment> compareByEducation = new
Comparator<Recruitment>() {
        @Override
        public int compare(Recruitment o1, Recruitment o2) {
            return o1.getEducation().compareTo(o2.getEducation());
        }
    };
    public static final Comparator<Recruitment> compareByFirm = new Comparator<Recruitment>()
{
        @Override
        public int compare(Recruitment o1, Recruitment o2) {
            return o1.getFirm().compareTo(o2.getFirm());
        }
    };
    public static final Comparator<Recruitment> compareBySpecialty = new
Comparator<Recruitment>() {
        @Override
        public int compare(Recruitment o1, Recruitment o2) {
            return o1.getSpecialty().compareTo(o2.getSpecialty());
        }
    };
}
```

Linked.java:

```
package ua.khpi.oop.vasilchenko09.MyList;
```

```
import java.io.Serializable;

public interface Linked<T> extends DescendingIterator<T>, Serializable, Iterable<T> {
    void addLast(T obj);
    void addFirst(T obj);
    int size();
    T getElementByIndex(int index);
    void saveAll();
    void saveRec();
    void add(T obj);
    void clear();
    boolean notEmpty();
    void readRec();
    void readAll();
}
```

DescendingIterator.java:

```
package ua.khpi.oop.vasilchenko09.MyList;

import java.util.Iterator;

public interface DescendingIterator<T> {
    Iterator<T> descendingIterator();
}
```

Util.java:

```
package ua.khpi.oop.vasilchenko10.MyList;

import ua.khpi.oop.vasilchenko10.First.Recruitment;

import java.util.Arrays;

public class Util {
    public static <T extends Recruitment> void sortFirm(LinkedContainer<T> obj) {
        Recruitment[] array = new Recruitment[obj.size()];
        for (int i = 0; i < obj.size(); i++) {
            array[i] = obj.getElementByIndex(i);
        }
        Arrays.sort(array, Recruitment.compareByFirm);
        obj.clear();
        for (int i = 0; i < array.length; i++) {
            obj.add((T) array[i]);
        }
    }

    public static <T extends Recruitment> void sortSpecialty(LinkedContainer<T> obj) {
        Recruitment[] array = new Recruitment[obj.size()];
        for (int i = 0; i < obj.size(); i++) {
            array[i] = obj.getElementByIndex(i);
        }
        Arrays.sort(array, Recruitment.compareBySpecialty);
        obj.clear();
        for (int i = 0; i < array.length; i++) {
            obj.add((T) array[i]);
        }
    }

    public static <T extends Recruitment> void sortEducation(LinkedContainer<T> obj) {
        Recruitment[] array = new Recruitment[obj.size()];
        int count = 0;
        for (int i = 0; i < obj.size(); i++) {
            array[i] = obj.getElementByIndex(i);
            if (array[i].getEducation() == null) {
                count++;
            }
```

```
        }
        Recruitment[] buff = new Recruitment[count];
        for (int i = 0, j = 0; i < array.length; i++) {
            if (array[i].getEducation() == null) {
                buff[j] = array[i];
                j++;
            }
        }
        Recruitment[] temp = new Recruitment[obj.size() - count];
        for (int i = 0, j = 0; i < array.length; i++) {
            if (array[i].getEducation() != null) {
                temp[j] = array[i];
                j++;
            }
        }
        Arrays.sort(temp, Recruitment.compareByEducation);
        obj.clear();
        for (int i = 0; i < temp.length; i++) {
            obj.add((T) temp[i]);
        }
        for (int i = 0; i < buff.length; i++) {
            obj.add((T) buff[i]);
        }
    }

    public static void chooseMenu() {
        System.out.println();
        System.out.println("1. Add vacancy.");
        System.out.println("2. Show all vacancies.");
        System.out.println("3. Clear container.");
        System.out.println("4. Check elements in container.");
        System.out.println("5. Size of container.");
        System.out.println("6. Get element by index.");
        System.out.println("7. Save data to file.");
        System.out.println("8. Read data from file.");
        System.out.println("9. Sorting data in container.");
        System.out.println("0. End of work.");
        System.out.print("Write your choose there: ");
    }
}
```

Demo.java:

```
package ua.khpi.oop.vasilchenko10.Tests;

public class Demo {
    public static void main(String[] args) {
        //Lab10.main(new String[] {"-auto"});
       Lab10.main(args);
    }
}
```

Exe.java

```
package ua.khpi.oop.vasilchenko10.Tests;


import ua.khpi.oop.vasilchenko10.First.Recruitment;
import ua.khpi.oop.vasilchenko10.MyList.LinkedContainer;
import ua.khpi.oop.vasilchenko10.MyList.Util;

import java.util.Scanner;

public class Exe {
    public static void auto() {
        System.out.println("It is auto mode");
        System.out.println("Creating object container:");
```

```
        LinkedContainer<Recruitment> linkedContainer = new LinkedContainer<>();
        System.out.println("Reading from file: ");
        linkedContainer.readRec();
        System.out.println("Print container:");
        for (Recruitment s : linkedContainer) {
            System.out.println(s);
        }
        System.out.println("Size: ");
        System.out.println(linkedContainer.size());
        System.out.println("Clear: ");
        linkedContainer.clear();
        System.out.println("Size: ");
        System.out.println(linkedContainer.size());
        System.out.println("Reading from file: ");
        linkedContainer.readRec();
        System.out.print("Not empty? : ");
        System.out.println(linkedContainer.notEmpty());
        System.out.println("Element with index 0");
        System.out.println(linkedContainer.getElementByIndex(0));
    }

    public static void menu() {
        System.out.println("It is menu mode");
        LinkedContainer<Recruitment> linkedContainer = new LinkedContainer<>();
        Recruitment rec1 = new Recruitment();
        Scanner scan = new Scanner(System.in);
        boolean loop = true;
        while (loop) {
            Util.chooseMenu();
            int choose = scan.nextInt();
            switch (choose) {
                case 1:
                    rec1.generateVacancy();
                    linkedContainer.add(rec1);
                    System.out.println("Done!");
                    break;
                case 2:
                    if (linkedContainer.notEmpty()) {
                        for (Recruitment s : linkedContainer) {
                            System.out.println(s);
                        }
                    } else {
                        System.out.println("Ошибка! Список пустой!");
                    }
                    break;
                case 3:
                    if (linkedContainer.notEmpty()) {
                        linkedContainer.clear();
                        System.out.println("Успешно!");
                    } else {
                        System.out.println("Ошибка! Массив пустой!");
                    }
                    break;
                case 4:
                    if (linkedContainer.notEmpty()) {
                        System.out.println("Your container have data.");
                    } else {
                        System.out.println("Your container doesn`t have data.");
                    }
                    break;
                case 5:
                        System.out.println("Size of container: " + linkedContainer.size());
                    break;
                case 6:
                    if (linkedContainer.notEmpty()) {
                        System.out.print("Entered index from 0 to " + (linkedContainer.size()
- 1) + ": ");
                        int choose1 = scan.nextInt();
                        if (choose1 > (linkedContainer.size() - 1) || choose1 < 0) {
```

```
                            System.out.println("Repeat enter");
                        } else {
                            System.out.println(linkedContainer.getElementByIndex(choose1));
                        }
                        break;
                    }
                case 7:
                    if (linkedContainer.notEmpty()) {
                        linkedContainer.saveRec();
                    } else {
                        System.out.println("Your container is empty");
                    }
                    break;
                case 8:
                    if (linkedContainer.notEmpty()) {
                        linkedContainer.clear();
                    }
                    linkedContainer.readRec();
                    break;
                case 9:
                    if (linkedContainer.notEmpty()) {
                        System.out.println("What field do you want to sort by?");
                        System.out.println("1. Sort by firm");
                        System.out.println("2. Sort by specialty");
                        System.out.println("3. Sort by education");
                        int choose2 = scan.nextInt();
                        boolean loop2 = true;
                        while (loop2) {
                            switch (choose2) {
                                case 1:
                                    Util.sortFirm(linkedContainer);
                                    loop2 = false;
                                    break;
                                case 2:
                                    Util.sortSpecialty(linkedContainer);
                                    loop2 = false;
                                    break;
                                case 3:
                                    Util.sortEducation(linkedContainer);
                                    loop2 = false;
                                    break;
                                default:
                                    System.out.println("Ошибка. Неверное число!");
                                    break;
                            }
                        }
                    } else {
                        System.out.println("Your container is empty");
                    }
                    break;
                case 0:
                    System.out.println("Thanks for working!");
                    loop = false;
                    break;
                default:
                    System.out.println("Ошибка. Неверное число!");
                    break;
            }
        }
    }
}
```

ВАРІАНТИ ВИКОРИСТАННЯ

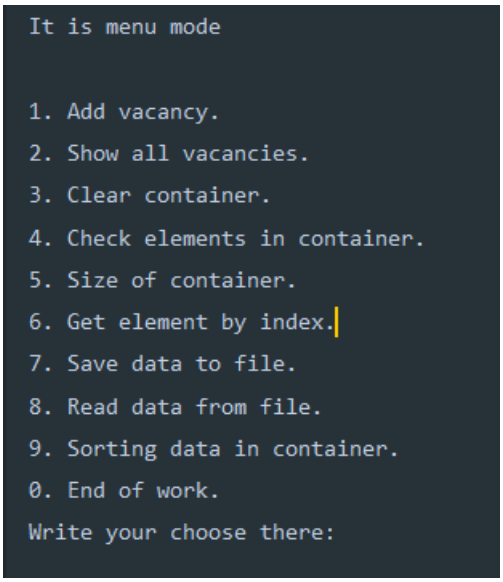

Рис. 11.1 – Результат роботи програми

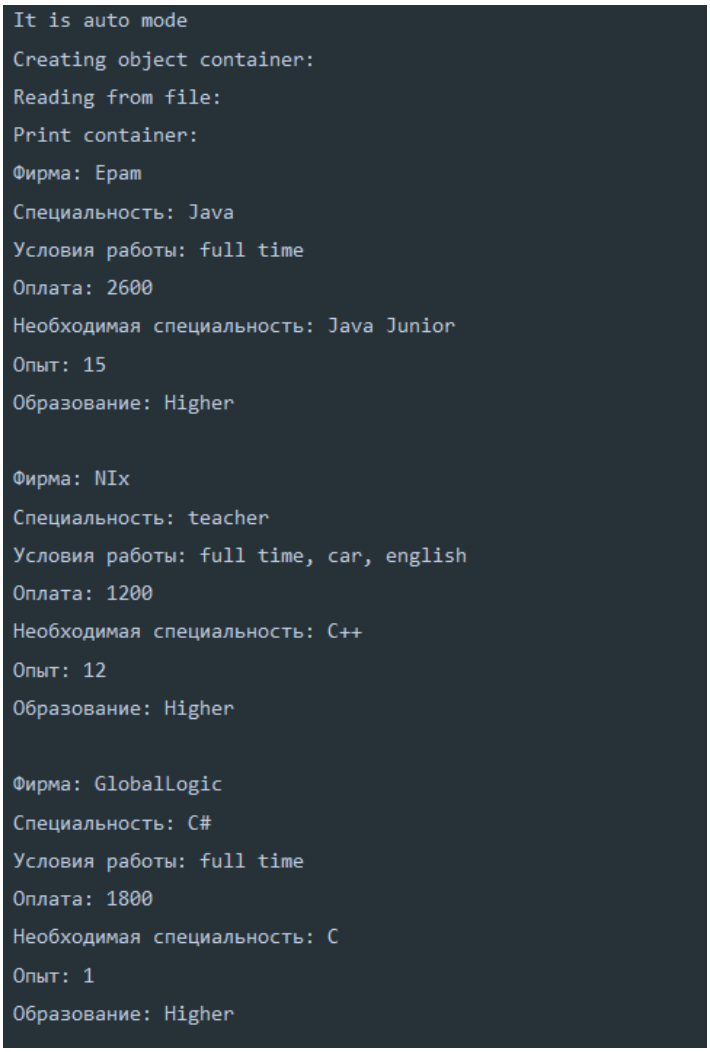

Рис. 11.2 – Результат роботи програми

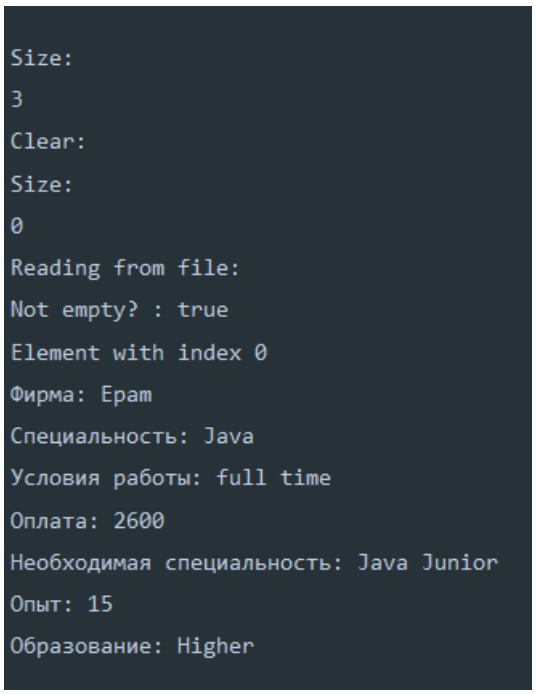

Рис. 11.3 – Результат роботи програми

Програму можна використовувати задля створення бази даних. Завдяки параметризації зв’язного списка, базу даних можна використати для будь-яких типів даних. Переважно у нашому варіанті - кадрове агенство, в якому представляються різноманітні вакансії. Для кожного поля було прописано регулярны вирази, для перевірки валідності введених даних. Також для вибору доступно багато інших можливостей. Реалізовано меню для поліпшення користування програмою.

ВИСНОВКИ

При виконанні лабораторної роботи набуто практичних навичок щодо розробки параметризованих класів. Завдяки цієї можливості в JAVA, можливо створювати колекції та інші класи на основі будь-яких типів. Також навчився обробляти параметризовані контейнери. Додано регулярні вирази для перевірки валідності введених даних. Завдання виконане! Програма працює успішно!