Звіт

Автор: Лунгол О., КІТ101.8б

ЛАБОРАТОРНА РОБОТА №12

# [Регулярні вирази. Обробка тексту](https://oop-khpi.gitlab.io/#tasks)

МЕТА:

Ознайомлення з принципами використання регулярних виразів для обробки тексту.

ВИМОГИ

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.

ОПИС ПРОГРАМИ

**Опис змінних**

**private** **long** NumberPassport;

**private** String specialty;

**private** String workingConditions;

**private** **int** marks;

**private** **int** payment;

**private** String data;

**private** String position;

**private** String education;

**private** String department;

Створили функцію яка вибирає картки людей з певними параметрами.

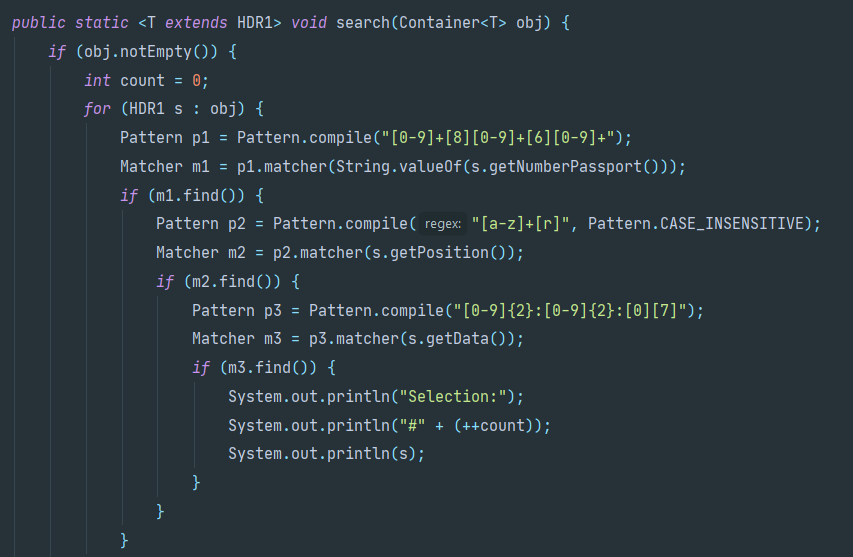


Рисунок 1 – Функція

В результаті буде відібрано картки у яких:

1. Цифра паспорта ..8..6..
2. Посада закінчується на букву r
3. Рік прийняття на посаду 2007

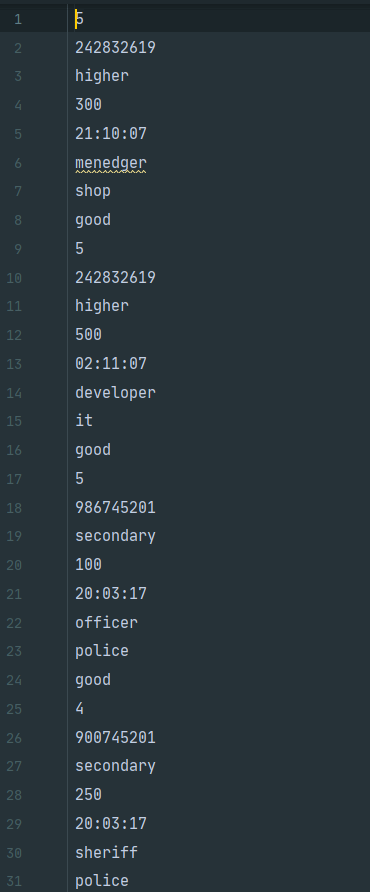


Рисунок 2 – Всі картки

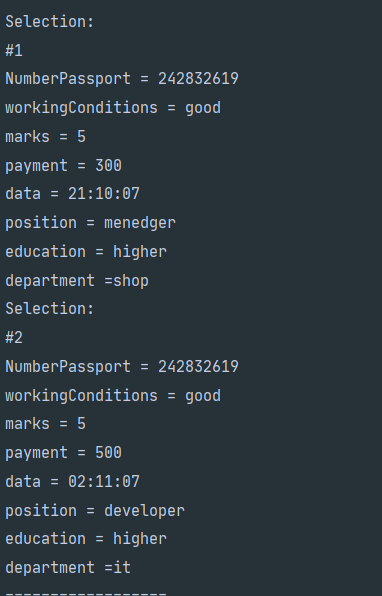


Рисунок 3 – Вибірка катрок за критерієм

**Ієрархія та структура класів**

**class** Container- клас в якому описуються та знаходяться контейнеры

**class** Lab09- головний клас

**class** HDR1 – клас в якому данні про відділ кадрів

**class** Util - клас в якому знаходяться параметрезовані методи

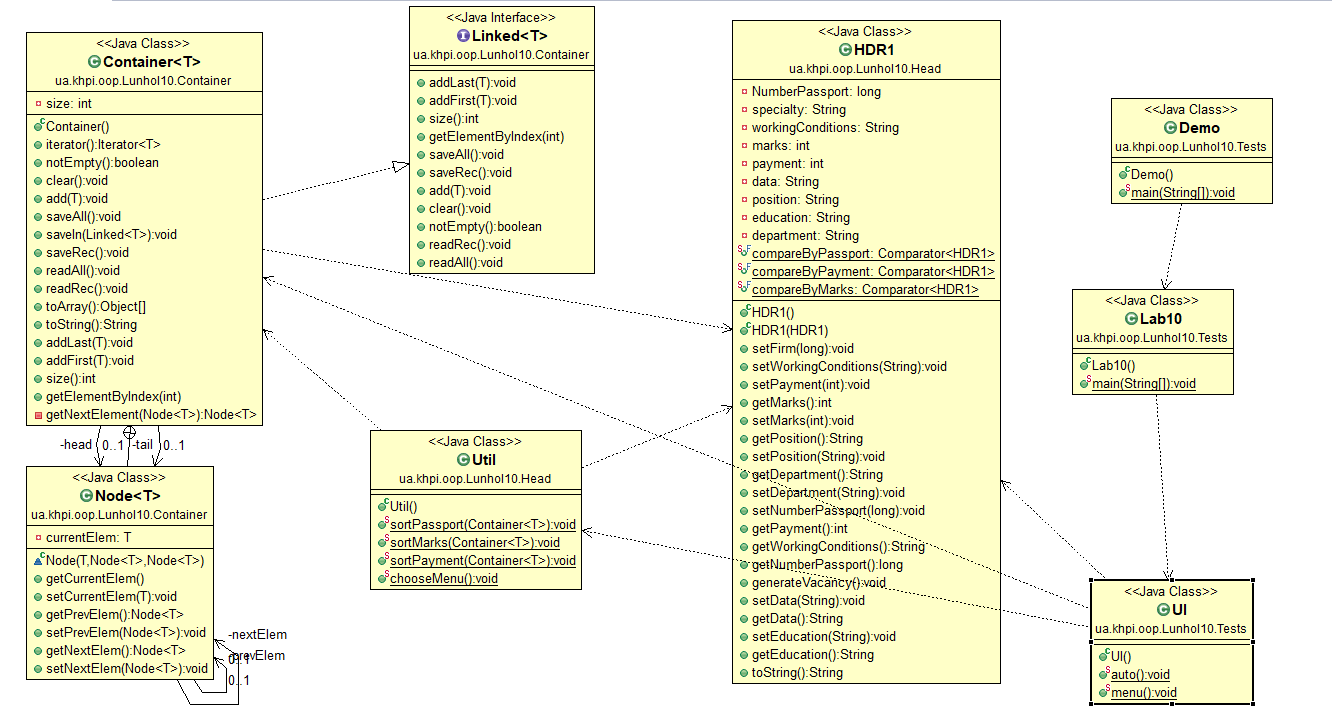


Рисунок 1 – Ієрархія класів

ТЕКСТ ПРОГРАМИ

РОБОТА ПРОГРАМИ

**package ua.khpi.oop.Lunhol10.Container;**

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.ObjectOutputStream;

**import** java.io.PrintWriter;

**import** java.io.Serializable;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**import** ua.khpi.oop.Lunhol10.Head.HDR1;

**public** **class** Container<T **extends** HDR1> **implements** Linked<T>, Serializable {

//конструктор инициализации

**public** Container() {

head = **new** Node<>(**null**, **null**, tail);

tail = **new** Node<>(**null**, head, **null**);

head = **new** Node<>(**null**, **null**, tail);

}

@Override

**public** Iterator<T> iterator() {

**return** **new** Iterator<>() {

**private** **int** position = 0;

@Override

**public** **boolean** hasNext() {

**return** position < size;

}

@Override

**public** T next() {

**if** (**this**.hasNext()) {

**return** getElementByIndex(position++);

} **else** {

**throw** **new** NoSuchElementException();

}

}

};

}

@Override

**public** **boolean** notEmpty() {

**return** size > 0;

}

@Override

**public** **void** clear() {

**for** (Node<T> x = head; x != **null**; ) {

Node<T> next = x.nextElem;

x.currentElem = **null**;

x.nextElem = **null**;

x.prevElem = **null**;

x = next;

}

head = **null**;

tail = **null**;

tail = **new** Node<>(**null**, head, **null**);

head = **new** Node<>(**null**, **null**, tail);

size = 0;

}

@Override

**public** **void** add(**final** T obj) {

addLast(obj);

}

@Override

**public** **void** saveAll() {

**try** {

File file = **new** File("save.txt");

**if** (!file.exists()) {

file.createNewFile();

}

PrintWriter pw = **new** PrintWriter(file);

System.***out***.println();

pw.println(size);

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

pw.println(getElementByIndex(i));

}

pw.close();

} **catch** (IOException e) {

System.***out***.println("Error" + e);

}

}

**public** **void** saveIn(Linked<T> obj) **throws** IOException {

FileOutputStream file = **new** FileOutputStream("save.sasha");

ObjectOutputStream object = **new** ObjectOutputStream(file);

object.writeObject(obj);

object.close();

}

@Override

**public** **void** saveRec() {

**try** {

File file = **new** File("save.txt");

**if** (!file.exists()) {

file.createNewFile();

}

PrintWriter pw = **new** PrintWriter(file);

HDR1 temp;

System.***out***.println();

pw.println(size);

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

temp = (HDR1) getElementByIndex(i);

pw.println(temp.getNumberPassport());

pw.println(temp.getEducation());

pw.println(temp.getPayment());

pw.println(temp.getData());

pw.println(temp.getPosition());

pw.println(temp.getDepartment());

pw.println(temp.getWorkingConditions());

pw.println(temp.getMarks());

}

pw.close();

} **catch** (IOException e) {

System.***out***.println("Error" + e);

}

}

@Override

**public** **void** readAll() {

**try** (BufferedReader br = **new** BufferedReader(**new** FileReader("save.txt"))) {

Object temp;

String line;

line = br.readLine();

**int** count = Integer.*parseInt*(line);

**for** (**int** i = 0; i < count; i++) {

line = br.readLine();

temp = line;

add((T) temp);

}

} **catch** (IOException ex) {

ex.printStackTrace();

}

}

@Override

**public** **void** readRec() **throws** IOException {

BufferedReader br = **null**;

HDR1 temp = **new** HDR1();

**try** {

br = **new** BufferedReader(**new** FileReader("save.txt"));

String line;

line = br.readLine();

**int** count = Integer.*parseInt*(line);

**for** (**int** i = 0; i < count; i++) {

line = br.readLine();

temp.setNumberPassport(Integer.*parseInt*(line));

line = br.readLine();

temp.setEducation(line);

line = br.readLine();

temp.setPayment(Integer.*parseInt*(line));

line = br.readLine();

temp.setData(line);

line = br.readLine();

temp.setPosition(line);

line = br.readLine();

temp.setDepartment(line);

line = br.readLine();

temp.setWorkingConditions(line);

line = br.readLine();

temp.setMarks(Integer.*parseInt*(line));

add((T) **new** HDR1(temp));

}

} **catch** (IOException ex) {

ex.printStackTrace();

} **finally** {

br.close();

}

}

**public** Object[] toArray() {

Object[] result = **new** Object[size];

**int** i = 0;

**for** (Node<T> temp = head; i < size; temp = temp.nextElem) {

result[i++] = temp.currentElem;

}

**return** result;

}

@Override

**public** String toString() {

**return** Arrays.*toString*(toArray());

}

**private** Node<T> head; //первый элемент

**private** Node<T> tail; //последний элемент

**private** **int** size = 0; //размер списка

@Override

**public** **void** addLast(**final** T obj) {

Node<T> prev = tail; //сохранение данных хвоста

prev.setCurrentElem(obj); //установка значения

tail = **new** Node<>(**null**, prev, **null**); //изменение указателя хвоста

prev.setNextElem(tail); //установка указателя на хвост

size++; //увелечение размера списка

}

@Override

**public** **void** addFirst(**final** T obj) {

Node<T> next = head;

next.setCurrentElem(obj);

head = **new** Node<>(**null**, **null**, next);

next.setPrevElem(head);

size++;

}

@Override

**public** **int** size() {

**return** size;

}

@Override

**public** T getElementByIndex(**final** **int** index) {

Node<T> target = head.getNextElem(); //след элемент первого узла

**for** (**int** i = 0; i < index; i++) {

target = getNextElement(target);

}

**return** target.getCurrentElem();

}

**private** Node<T> getNextElement(**final** Node<T> index) {

**return** index.getNextElem();

}

// head -> null & tail -> null

// null <- prevElem [head(t = null)] nextElem-> & <- prevElem [head(t = null)] nextElem -> null

**private** **class** Node<T> **implements** Serializable {

**private** T currentElem; //текущий

**private** Node<T> prevElem; //предыдущий

**private** Node<T> nextElem; //следущий

//конструктор копирования

Node(**final** T currentElem, **final** Node<T> prevElem, **final** Node<T> nextElem) {

**this**.currentElem = currentElem;

**this**.prevElem = prevElem;

**this**.nextElem = nextElem;

}

//геттеры и сеттеры

**public** T getCurrentElem() {

**return** currentElem;

}

**public** **void** setCurrentElem(**final** T currentElem) {

**this**.currentElem = currentElem;

}

**public** Node<T> getPrevElem() {

**return** prevElem;

}

**public** **void** setPrevElem(**final** Node<T> prevElem) {

**this**.prevElem = prevElem;

}

**public** Node<T> getNextElem() {

**return** nextElem;

}

**public** **void** setNextElem(**final** Node<T> nextElem) {

**this**.nextElem = nextElem;

}

}

}

**package ua.khpi.oop.Lunhol10.Container;**

**import** java.io.IOException;

**import** java.io.Serializable;

//создаем интерфейс линкд

//обьявление прототипов функций

// Т джунерик (обобщение)

**public** **interface** Linked<T> **extends** 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() **throws** IOException;

**void** readAll();

}

**package ua.khpi.oop.Lunhol10.Head;**

*package* ua.khpi.oop.Lunhol11.Head;  
  
*import* java.io.*Serializable*;  
*import* java.util.*Comparator*;  
*import* java.util.InputMismatchException;  
*import* java.util.Scanner;  
*import* java.util.regex.Matcher;  
*import* java.util.regex.Pattern;  
  
*public class* HDR1 *implements Serializable* {  
 *private long* numberPassport;  
 *private* String specialty;  
 *private* String workingConditions;  
 *private int* marks;  
 *private int* payment;  
  
 *private* String data;  
 *private* String position;  
 *private* String education;  
 *private* String department;  
   
  
 *public* HDR1() {  
 numberPassport = 0;  
 workingConditions = *null*;  
 marks = 0;  
 payment = 0;  
 data = *null*;  
 position = *null*;  
 education = *null*;  
 department = *null*;  
 }  
   
 *public* HDR1 (*final* HDR1 obj) {  
 numberPassport = obj.numberPassport;  
 specialty = obj.specialty;  
 workingConditions = obj.workingConditions;  
 marks = obj.marks;  
 payment = obj.payment;  
 data = obj.data;  
 position = obj.position;  
 education = obj.education;  
 department = obj.department;  
 }  
  
 *public void* setFirm(*long* NumberPassport) {  
 *this*.numberPassport = NumberPassport;  
 }  
  
 *public void* setWorkingConditions(String workingConditions) {  
 *if* (checkWorkingConditions(workingConditions)) {  
 *this*.workingConditions = workingConditions;  
 } *else* {  
 *throw new* InputMismatchException();  
 }  
 }  
  
 *//проверка на правильность ввода  
 private boolean* checkWorkingConditions(*final* String workingConditions) {  
 *//регулярное выражение соответсвующая набору символов латинского алфавита и знаков разделителя* Pattern pattern = Pattern.*compile*("[\\s\\w%$+#@^()=!\_\\\\-]\*", Pattern.*CASE\_INSENSITIVE*);  
 *//провверка переменной на совпадение с регулярным выражением* Matcher matcher = pattern.matcher(workingConditions);  
 *//возврат совпало/несовпало  
 return* matcher.matches();  
 }  
  
 *private boolean* checkNumberPassport(*final long* NumberPassport) {  
 Pattern pattern = Pattern.*compile*("^[0-9]{9}");  
 Matcher matcher = pattern.matcher(String.*valueOf*(NumberPassport));  
 *return* matcher.matches();  
 }  
  
 *public void* setPayment(*int* payment) {  
 *this*.payment = payment;  
 }  
  
 *public int* getMarks() {  
 *return* marks;  
 }  
  
 *public void* setMarks(*int* marks) {  
 *this*.marks = marks;  
 }  
  
 *public* String getPosition() {  
 *return* position;  
 }  
  
 *public void* setPosition(String position) {  
 *if* (checkPosition(position)) {  
 *this*.position = position;  
 } *else* {  
 *throw new* InputMismatchException();  
 }  
 }  
  
 *private boolean* checkPosition(*final* String position) {  
 Pattern pattern = Pattern.*compile*("[a-z]\*-?\\s?", Pattern.*CASE\_INSENSITIVE*);  
 Matcher matcher = pattern.matcher(position);  
 *return* matcher.matches();  
 }  
  
 *public* String getDepartment() {  
 *return* department;  
 }  
  
 *public void* setDepartment(String department) {  
 *if* (checkDepartment(department)) {  
 *this*.department = department;  
 } *else* {  
 *throw new* InputMismatchException();  
 }  
 }  
  
 *private boolean* checkDepartment(*final* String department) {  
 *//регулярное выражение соответсвующая набору символов латинского алфавита и знаков разделителя* Pattern pattern = Pattern.*compile*("[\\s\\w%$+#@^()=!\_\\\\-]\*", Pattern.*CASE\_INSENSITIVE*);  
 *//провверка переменной на совпадение с регулярным выражением* Matcher matcher = pattern.matcher(department);  
 *//возврат совпало/несовпало  
 return* matcher.matches();  
 }  
  
 *public void* setNumberPassport(*long* numberPassport) {  
 *if* (checkNumberPassport(numberPassport)) {  
 *this*.numberPassport = numberPassport;  
 } *else* {  
 *throw new* InputMismatchException();  
 }  
 }  
  
 *public int* getPayment() {  
 *return* payment;  
 }  
 *public* String getWorkingConditions() {  
 *return* workingConditions;  
 }  
  
 *public long* getNumberPassport() {  
 *return* numberPassport;  
 }  
  
 *public void* generateVacancy() {  
 Scanner scan = *new* Scanner(System.*in*);  
 Scanner scan2 = *new* Scanner(System.*in*);  
 *int* choose = 0;  
 System.*out*.print("\nВведите номер паспорта: ");  
 setNumberPassport(scan.nextLong());  
 System.*out*.print("\nВведите образование: ");  
 setEducation(scan2.nextLine());  
 System.*out*.print("\nОплата: ");  
 setPayment(scan.nextInt());  
 System.*out*.print("\nДата взятия на должность: ");  
 setData(scan2.nextLine());  
 System.*out*.print("\nВведите должность: ");  
 setPosition(scan2.nextLine());  
 System.*out*.print("\nВведите отдел: ");  
 setDepartment(scan2.nextLine());  
 System.*out*.print("\nНабор свойств: ");  
 setWorkingConditions(scan2.nextLine());  
 System.*out*.println("\nОценка кандидата: ");  
 setMarks(scan.nextInt());  
 }  
  
  
   
  
 *public void* setData(String data) {  
 *if* (checkData(data)) {  
 *this*.data = data;  
 } *else* {  
 *throw new* InputMismatchException();  
 }  
 }  
  
 *private boolean* checkData(*final* String data) {  
 Pattern pattern = Pattern.*compile*("[0-9]{2}:[0-9]{2}:[0-9]{2}");  
 Matcher matcher = pattern.matcher(data);  
 *return* matcher.matches();  
 }  
  
 *public* String getData() {  
 *return* data;  
 }  
  
 *public void* setEducation(String education) {  
 *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;  
 }  
  
 @Override  
 *public* String toString() {  
 *return* "NumberPassport = " + numberPassport + "\n" + "workingConditions = "  
 + workingConditions + "\n" +"marks = " + marks + "\n" +"payment = " + payment + "\n" + "data = " + data + "\n" + "position = "  
 + position + "\n" +"education = " + education + "\n" +"department =" + department;  
 }  
 *//компоратор - это функция для сравнения обьектов по определённому полю в моем контейнере и  
 //использованию в встроеном методе сорт  
  
 public static final Comparator*<HDR1> *compareByPassport* = *new* Comparator<HDR1>() {  
 @Override  
 *public int* compare(HDR1 o1, HDR1 o2) {  
 *return* (*int*) (o1.getNumberPassport() - o2.getNumberPassport());  
 }  
 };  
 *public static final Comparator*<HDR1> *compareByPayment* = *new* Comparator<HDR1>() {  
 @Override  
 *public int* compare(HDR1 o1, HDR1 o2) {  
 *return* o1.getPayment() - o2.getPayment();  
 }  
 };  
 *public static final Comparator*<HDR1> *compareByMarks* = *new* Comparator<HDR1>() {  
 @Override  
 *public int* compare(HDR1 o1, HDR1 o2) {  
 *return* o1.getMarks() - o2.getMarks();  
 }  
 };

**package ua.khpi.oop.Lunhol10.Head;**

**import** java.util.Arrays;

**import** ua.khpi.oop.Lunhol10.Container.Container;

//параметрез. методы

**public** **class** Util {

**public** **static** <T **extends** HDR1> **void** sortPassport(Container<T> obj) {

HDR1[] array = **new** HDR1[obj.size()];

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

array[i] = obj.getElementByIndex(i);

}

Arrays.*sort*(array, HDR1.***compareByPassport***);

obj.clear();

**for** (**int** i = 0; i < array.length; i++) {

obj.add((T) array[i]);

}

}

**public** **static** <T **extends** HDR1> **void** sortMarks(Container<T> obj) {

HDR1[] array = **new** HDR1[obj.size()];

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

array[i] = obj.getElementByIndex(i);

}

Arrays.*sort*(array, HDR1.***compareByMarks***);

obj.clear();

**for** (**int** i = 0; i < array.length; i++) {

obj.add((T) array[i]);

}

}

**public** **static** <T **extends** HDR1> **void** sortPayment(Container<T> obj) {

HDR1[] array = **new** HDR1[obj.size()];

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

array[i] = obj.getElementByIndex(i);

}

Arrays.*sort*(array, HDR1.***compareByPayment***);

obj.clear();

**for** (**int** i = 0; i < array.length; i++) {

obj.add((T) array[i]);

}

*public static* <T *extends* HDR1> *void* search(Container<T> obj) {  
 *if* (obj.notEmpty()) {  
 *int* count = 0;  
 *for* (HDR1 s : obj) {  
 Pattern p1 = Pattern.compile("[0-9]+[8][0-9]+[6][0-9]+");  
 Matcher m1 = p1.matcher(String.valueOf(s.getNumberPassport()));  
 *if* (m1.find()) {  
 Pattern p2 = Pattern.compile("[a-z]+[r]", Pattern.CASE\_INSENSITIVE);  
 Matcher m2 = p2.matcher(s.getPosition());  
 *if* (m2.find()) {  
 Pattern p3 = Pattern.compile("[0-9]{2}:[0-9]{2}:[0][7]");  
 Matcher m3 = p3.matcher(s.getData());  
 *if* (m3.find()) {  
 System.out.println("Selection:");  
 System.out.println("#" + (++count));  
 System.out.println(s);  
 }  
 }  
 }  
 }  
 }  
}

}

**public** **static** **void** chooseMenu() {

System.***out***.println();

System.***out***.println("1. Add candidat.");

System.***out***.println("2. Show all candidats.");

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: ");

}}

**package ua.khpi.oop.Lunhol10.Tests;**

**import** java.io.IOException;

**public** **class** Demo {

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

Lab10.*main*(**new** String[]{"-auto"});

//Lab10.main(args);

}

}

**package ua.khpi.oop.Lunhol10.Tests;**

**import** java.io.IOException;

**public** **class** Lab10 {

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

**if** (args.length != 0) {

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

UI.*auto*();

} **else** {

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

}

} **else** {

UI.*menu*();

}

}

}

**package ua.khpi.oop.Lunhol10.Tests;**

**import** java.io.IOException;

**import** java.util.Scanner;

**import** ua.khpi.oop.Lunhol10.Container.Container;

**import** ua.khpi.oop.Lunhol10.Head.HDR1;

**import** ua.khpi.oop.Lunhol10.Head.Util;

**public** **class** UI {

**public** **static** **void** auto() **throws** IOException {

HDR1 temp = **new** HDR1();

Container<HDR1> container = **new** Container<>();

System.***out***.println(container.size());

container.add(temp);

System.***out***.println(container.size());

container.readRec();

System.***out***.println("==================");

**int** counter = 0;

**for** (HDR1 s : container) {

System.***out***.println("#" + ++counter);

System.***out***.println(s);

}

System.***out***.println("==================");

System.***out***.println(container.size());

System.***out***.println("==================");

container.saveIn(container);

System.***out***.println(container.getElementByIndex(0));

container.clear();

System.***out***.println(container.size());

}

**public** **static** **void** menu() **throws** IOException {

System.***out***.println("It is menu mode");

Container<HDR1> linkedContainer = **new** Container<>();

HDR1 rec1 = **new** HDR1();

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()) {

**int** count = 0;

**for** (HDR1 s : linkedContainer) {

System.***out***.println();

System.***out***.println(++count);

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 passport");

System.***out***.println("2. Sort by marks");

System.***out***.println("3. Sort by payment");

**int** choose2 = scan.nextInt();

**boolean** loop2 = **true**;

**while** (loop2) {

**switch** (choose2) {

**case** 1:

Util.*sortPassport*(linkedContainer);

loop2 = **false**;

**break**;

**case** 2:

Util.*sortPayment*(linkedContainer);

loop2 = **false**;

**break**;

**case** 3:

Util.*sortMarks*(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**;

}

}

}

}

Висновок

Розробили параметризовані методи ([Generic Methods](https://docs.oracle.com/javase/tutorial/java/generics/methods.html)) для обробки колекцій об’єктів згідно [прикладної задачі](https://oop-khpi.github.io/" \l "task_10_app).

Продемонстрували розроблену функціональність (створення, управління та обробку власних контейнерів) в діалоговому та автоматичному режимах.

Автоматичний режим виконання програми задали параметром командного рядка **-auto**. Наприклад, java ClassName -auto.

В автоматичному режимі діалог з користувачем відсутній, необхідні данні генеруються, або зчитуються з файлу.