Третьяков І.С.

КІТ-118а

Лабораторна робота №12

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

***Мета:***

* Розширення функціональності параметризованих класів.

## **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.

ПРИКЛАДНА ЗАДАЧА:

Планувальник. Сортування за датою заходу, за тривалістю, за кількістю учасників..

ОПИС ПРОГРАМИ

***2.1 Опис змінних:***

LinkedContainer<Event > linkedContainer = new LinkedContainer<>();

// об’єкт параметризованого контейнера

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

***2.2 Ієрархія та структура класів.***

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

interface Linked - інтерфейс контенеру

class Event - клас прикладної задачі планувальника

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

**ТЕКСТ ПРОГРАМИ**

File Main.java:

package ua.khpi.oop.tretiakov12;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.TransformerException;

import java.io.\*;

import java.util.Arrays;

import java.util.Comparator;

import java.util.LinkedList;

import java.util.Scanner;

import java.io.\*;

import java.util.concurrent.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Main implements Serializable {

public static void fileRead() throws IOException, ParserConfigurationException, TransformerException, InterruptedException,

ExecutionException, TimeoutException {

LinkedContainer<Event> linkedContainer = new LinkedContainer<Event>();

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

Scanner scanner = new Scanner(file);

String line = scanner.nextLine();

String[] array = line.split(" ");

// Участники

Participant par1[] = new Participant[2];

par1[0] = new Participant("Роман","Жолин",25);

par1[1] = new Participant("Евгений","Розумовский",24);

Participant[] par2 = new Participant[2];

par2[0] = new Participant("Игнат", "Третьяков", 18);

par2[1] = new Participant("Ростислав", "Малахов", 19);

String date = null;

String startTime = null;

int duration = 0;

String venue = null;

String description = null;

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

date = array[0].toString();

startTime = array[1].toString();

duration = Integer.parseInt(array[2]);

venue = array[3].toString();

description = array[4].toString();

}

regCheck(date, startTime, duration, venue, description);

Event firstEvent = new Event(date, startTime, duration, venue, description, par1);

Event secondEvent = new Event("2019-12-22", "10:00", 1441, "Kharkov", "Corporate", par2);

Event thirdEvent = new Event("2017-12-22", "10:00", 1440, "Kharkov","Corporate", par1);

regCheck(secondEvent.getData(), secondEvent.getStartTime(), secondEvent.getDuration(),

secondEvent.getVenue(), secondEvent.getDescription());

Event[] events = {firstEvent, secondEvent};

/\*System.out.println("SORT BY DATE");

//Arrays.sort(events);

for (Event e : events) {

System.out.println(e);

}

System.out.println("SORT BY DURATION");

ComparatorDuration comparatorDuration = new ComparatorDuration();

Arrays.sort(events, comparatorDuration);

for(Event tmpss : events)

{

System.out.println(tmpss);

}

System.out.println("SORT BY PARTICIPANT");

ComparatorParticipant comparatorParticipant = new ComparatorParticipant();

Arrays.sort(events, comparatorParticipant);

for(Event tmpp : events)

{

System.out.println(tmpp);

}\*/

linkedContainer.addFirst(firstEvent);

linkedContainer.addLast(secondEvent);

linkedContainer.addLast(thirdEvent);

System.out.println("Container size");

System.out.println(linkedContainer.size());

choise(linkedContainer);

//serialize(linkedContainer);

//linkedContainer.addLast(secondEvent);

}

public static void choise(LinkedContainer<Event> linkedContainer) {

for(Event ev : linkedContainer)

{

if (ev.getData().compareTo("2017-01-01") > 0 && ev.getVenue() == "Kharkov" && ev.getDuration() >= 1440) {

System.out.println(ev);

}

}

}

public static void serialize(LinkedContainer<Event> linkedContainer) throws IOException, ParserConfigurationException,

TransformerException{

XmlRead xmlRead = new XmlRead();

XmlWrite xmlWrite = new XmlWrite();

xmlWrite.write(linkedContainer,"XML.xml");

LinkedContainer<Event> newXml = XmlRead.read("XML.xml");

for(Event e : newXml )

{

System.out.println(e);

}

}

public static void regCheck(String date, String startTime, int duration, String venue, String description)

{

// Дата в формате dd-mm-yyyy

if(date.matches("[0-9]{4}-(0[1-9]|1[012])-(0[1-9]|1[0-9]|2[0-9]|3[01])") == true)

{

System.out.println("OK");

}

else

{

System.out.println("Change date");

}

if(startTime.matches("^([0-1]\\d|2[0-3])(:[0-5]\\d)$"))

{

System.out.println("OK");

}else{System.out.println("Change time");}

}

public static void manual() throws IOException, ClassNotFoundException, FileNotFoundException, TransformerException,

ParserConfigurationException {

System.out.println("U have chosen manual mode");

int choose;

LinkedContainer<Event> linkedContainer = new LinkedContainer<>();

Event event1 = null;

do{

System.out.println("Choose action ");

Scanner in = new Scanner(System.in);

System.out.println("0. Exit");

System.out.println("1. Create new element");

System.out.println("2. Add elem ");

System.out.println("3. Del elem");

System.out.println("4. Clear container ");

System.out.println("5. Convert to Array ");

System.out.println("6. Convert to String");

System.out.println("7. Serialize ");

System.out.println("8. Deserialize ");

System.out.println("9. Xml serialize");

System.out.println("10. Xml deserialize");

System.out.println("11. Kharkov events");

choose = in.nextInt();

switch (choose) {

case 1:

Participant[] par2 = new Participant[2];

par2[0] = new Participant("Игнат", "Третьяков", 18);

par2[1] = new Participant("Ростислав", "Малахов", 19);

//par2[2] = new Participant("Данил", "Макаров", 17);

Scanner din = new Scanner(System.in);

Scanner cin = new Scanner(System.in);

System.out.println("Enter date(format yyyy-mm-dd)");

String date = din.nextLine();

if(date.matches("[0-9]{4}-(0[1-9]|1[012])-(0[1-9]|1[0-9]|2[0-9]|3[01])")==true) {

System.out.println("OK");

}else {System.out.println("NOT OK CHANGE DATE");date=din.nextLine();}

System.out.println("Enter start time(format HH:MM)");

String startTime=din.nextLine();

if(startTime.matches("^([0-1]\\d|2[0-3])(:[0-5]\\d)$")==true) {

System.out.println("OK");

}else {System.out.println("NOT OK CHANGE TIME");startTime=din.nextLine();}

System.out.println("Enter duration");

int duration=cin.nextInt();

System.out.println("Enter venue");

String venue=din.nextLine();

System.out.println("Enter description");

String description=din.nextLine();

event1 = new Event(date,startTime,duration,venue,description,par2);

break;

case 2:

System.out.println(linkedContainer.size());

linkedContainer.addLast(event1);

System.out.println(linkedContainer.size());

for(Event tmp : linkedContainer)

{

System.out.println(tmp);

}

break;

case 3:

System.out.println(linkedContainer.size());

System.out.println("Еnter index: ");

int index = in.nextInt();

linkedContainer.removeByIndex(index);

System.out.println(linkedContainer.size());

break;

case 4:

linkedContainer.clean();

System.out.println(linkedContainer.size());

break;

case 5:

Object []arr = linkedContainer.toArray().toArray();

for(int i=0; i<linkedContainer.size();i++)

{

System.out.println(arr[i]);

}

break;

case 6:

linkedContainer.toString();

break;

case 7:

ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream("event.txt"));

objectOutputStream.writeObject(linkedContainer);

objectOutputStream.close();

break;

case 8:

ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream("event.txt"));

LinkedContainer<Event> newTravels = (LinkedContainer<Event>) objectInputStream.readObject();

objectInputStream.close();

for (Event e : newTravels) {

System.out.println(e);

}

break;

case 9:

XmlWrite xmlWrite = new XmlWrite();

xmlWrite.write(linkedContainer,"XML.xml");

break;

case 10:

LinkedContainer<Event> newXml = XmlRead.read("XML.xml");

for(Event e : newXml )

{

System.out.println(e);

}

break;

case 11:

choise(linkedContainer);

break;

default:

System.out.println("Exit");

break;

}}while(choose!=0);

}

public static void main(String args[]) throws IOException, ParserConfigurationException, TransformerException,

ClassNotFoundException, InterruptedException, ExecutionException, TimeoutException {

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

{

System.out.println("U chose auto mode.");

System.out.println("1. Reading from file");

fileRead();

}

else

{

manual();

}

}

}

Event.java :

**package** ua.khpi.oop.tretiakov12;

**import** java.io.Serializable;

**public** **class** Event **implements** Comparable<Event>, Serializable {

**private** String data; // Дата

**private** String startTime; // Время начала

**private** **int** duration; // Продолжительность

**private** String venue; // Место проведения

**private** String description; // Описание

**private** Participant[] participants; // Участники

**public** Event(String data, String startTime, **int** duration,

String venue, String description, Participant[] participants) {

**this**.data = data;

**this**.startTime = startTime;

**this**.duration = duration;

**this**.venue = venue;

**this**.description = description;

**this**.participants = participants;

}

**public** Event() {

}

**public** String getData() {

**return** **this**.data;

}

**public** **void** setData(String data) {

**this**.data = data;

}

**public** String getStartTime() {

**return** **this**.startTime;

}

**public** **void** setStartTime(String startTime) {

**this**.startTime = startTime;

}

**public** **int** getDuration() {

**return** **this**.duration;

}

**public** **void** setDuration(**int** duration) {

**this**.duration = duration;

}

**public** String getVenue() {

**return** **this**.venue;

}

**public** **void** setVenue(String venue) {

**this**.venue = venue;

}

**public** String getDescription() {

**return** **this**.description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** Participant[] getParticipants() {

**return** **this**.participants;

}

**public** **void** setParticipants(Participant[] participants) {

**this**.participants = participants;

}

@Override

**public** String toString() {

StringBuffer buf = **new** StringBuffer();

buf.append("Дата: " + **this**.data + "\n");

buf.append("Время начала: " + **this**.startTime + "\n");

buf.append("Продолжительность: " + **this**.duration + "\n");

buf.append("Место проведения: " + **this**.venue + "\n");

buf.append("Описание: " + **this**.description + "\n");

buf.append("Участники: ");

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

buf.append(**this**.participants[i].toString());

}

**return** buf.toString();

}

@Override

**public** **int** compareTo(Event o) {

Event entry = (Event) o;

**int** tmp = o.data.compareTo(entry.data);

// this.payment - ((SecondCreate)o).payment;

**return** tmp;

}

}

Participant.java:

**package** ua.khpi.oop.tretiakov12;

**import** java.io.Serializable;

**public** **class** Participant **implements** Serializable {

**private** String name;

**private** String surname;

**private** **int** age;

**public** Participant(String name, String surname, **int** age){

**this**.name = name;

**this**.surname = surname;

**this**.age = age;

}

**public** Participant(){

}

**public** String getName() {

**return** **this**.name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getSurname() {

**return** **this**.surname;

}

**public** **void** setSurname(String surname) {

**this**.surname = surname;

}

**public** **int** getAge() {

**return** **this**.age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

StringBuffer buf = **new** StringBuffer();

buf.append("Имя: " + **this**.name + "\n");

buf.append("Фамилия: " + **this**.surname + "\n");

buf.append("Возраст: " + **this**.age + "\n");

**return** buf.toString();

}

}

Comparable.java

**package** ua.khpi.oop.tretiakov12;

**public** **interface** Comparable<T> {

**public** **int** compareTo(T o);

}

LinkedContainer.java

package ua.khpi.oop.tretiakov12;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.LinkedList;

public class LinkedContainer<E> implements Linked<E>, Iterable<E>, Serializable{

private int size = 0;

private Node<E> first;

private Node<E> last;

public LinkedContainer() {

last = new Node<E>(first,null, null);

first = new Node<E>(null, null, last);

}

@Override

public void addLast(E e) {

Node<E> prev = last;

prev.setItem(e);

last = new Node<E>(prev,null, null);

prev.setNext(last);

size++;

}

@Override

public void addFirst(E e) {

Node<E> next = first;

next.setItem(e);

first = new Node<E>(null, null, next);

next.setPrev(first);

size++;

}

@Override

public int size() {

return this.size;

}

@Override

public E getElementByIndex(int index) {

Node<E> target = first.getNext();

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

if (target == null) return null;

target = target.getNext();

}

return target.getItem();

}

@Override

public void removeByIndex(int index) {

Node<E> target = first.getNext();

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

if (target == null) return;

target = target.getNext();

}

Node<E> PrevRemoved = target.prev;

Node<E> NextRemoved = target.next;

PrevRemoved.next = NextRemoved;

NextRemoved.prev = PrevRemoved;

target.setItem(null);

target.setPrev(null);

target.setNext(null);

size--;

}

@Override

public void clean() {

Node<E> target = first.getNext();

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

target.setItem(null);

target = target.getNext();

}

last = new Node<E>(first,null, null);

first = new Node<E>(null, null, last);

size = 0;

}

ArrayList<E> toArray(){

ArrayList<E> result = new ArrayList<E>();

Node<E> target = first.getNext();

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

result.add(target.getItem());

target = target.getNext();

}

return result;

}

@Override

public String toString() {

StringBuilder builder = new StringBuilder();

Node<E> target = first.getNext();

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

builder.append(target.item.toString());

target = target.getNext();

}

return builder.toString();

}

boolean isEmpry(){

if(first.next == last){

return true;

}else {

return false;

}

}

@Override

public Iterator<E> iterator() {

Iterator<E> iterator = new Iterator<E>() {

int counter = 0;

@Override

public boolean hasNext() {

return counter < size;

}

@Override

public E next() {

return getElementByIndex(counter++);

}

};

return iterator;

}

private static class Node<E> implements Serializable {

E item;

Node<E> next;

Node<E> prev;

Node(Node<E> prev, E element, Node<E> next) {

this.item = element;

this.next = next;

this.prev = prev;

}

public E getItem() {

return item;

}

public void setItem(E item) {

this.item = item;

}

public Node<E> getNext() {

return next;

}

public void setNext(Node<E> next) {

this.next = next;

}

public Node<E> getPrev() {

return prev;

}

public void setPrev(Node<E> prev) {

this.prev = prev;

}

}

}

Linked.java:

**package** ua.khpi.oop.tretiakov12;

**public** **interface** Linked<E> {

**void** addLast(E e);

**void** addFirst(E e);

**int** size();

E getElementByIndex(**int** index);

**void** removeByIndex(**int** index);

**void** clean();

}

XmlRead.java

package ua.khpi.oop.tretiakov12;

import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;

import org.xml.sax.SAXException;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

public class XmlRead {

static public LinkedContainer<Event> read(String fileName) throws IOException, ParserConfigurationException {

LinkedContainer<Event> resEvent = new LinkedContainer<>();

try {

// Создается построитель документа

DocumentBuilder documentBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();

// Создается дерево DOM документа из файла

Document document = documentBuilder.parse(fileName);

// Получаем корневой элемент

Node root = document.getDocumentElement();

System.out.println(root.getNodeName());

// Просматриваем все подэлементы корневого - т.е. книги

NodeList event = root.getChildNodes();

String data = new String();

String startTime = new String();

int duration = 0;

String venue = new String();

String description = new String();

Participant[] participants = new Participant[2];

for (int i = 0; i < event.getLength(); i++){

if(event.item(i).getNodeType() != Node.TEXT\_NODE){

if(event.item(i).getNodeName() == "Data"){

data = event.item(i).getTextContent();

}

if(event.item(i).getNodeName() == "StartTime"){

startTime = event.item(i).getTextContent();

}

if(event.item(i).getNodeName() == "Duration"){

duration = Integer.valueOf(event.item(i).getTextContent());

}

if(event.item(i).getNodeName() == "Venue"){

venue = event.item(i).getTextContent();

}

if(event.item(i).getNodeName() == "Description"){

description = event.item(i).getTextContent();

}

if(event.item(i).getNodeName() == "Participant"){

String str[] = event.item(i).getTextContent().split(" ");

String name = new String();

String surname = new String();

int age = 0;

int j = 0;

for (int m = 0; m < str.length; m+=3){

name = str[m];

surname = str[m+1];

age = Integer.valueOf(str[m+2]);

participants[j] = new Participant(name,surname,age);

j++;

}

}

}

Event e1 = new Event(data, startTime, duration, venue,description,participants);

resEvent.addLast(e1);

}

//Event e1 = new Event(data, startTime, duration, venue,description,participants);

//resEvent.addLast(e1);

} catch (ParserConfigurationException ex) {

ex.printStackTrace(System.out);

} catch (SAXException ex) {

ex.printStackTrace(System.out);

} catch (IOException ex) {

ex.printStackTrace(System.out);

}

return resEvent;

}

}

XmlWrite.java

package ua.khpi.oop.tretiakov12;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.TransformerFactoryConfigurationError;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

public class XmlWrite {

static public void write(LinkedContainer<Event> linkedContainer, String fileName) throws ParserConfigurationException, TransformerFactoryConfigurationError, FileNotFoundException, TransformerException {

DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

Document doc = docBuilder.newDocument();

for(Event e : linkedContainer)

{

Element Event = doc.createElement("Event");

Element Data = doc.createElement("Data");

Element StartTime = doc.createElement("StartTime");

Element Duration = doc.createElement("Duration");

Element Venue = doc.createElement("Venue");

Element Description = doc.createElement("Description");

Element Participant = doc.createElement("Participant");

////////////////////////////////////

doc.appendChild(Event);

Event.appendChild(Data);

Event.appendChild(StartTime);

Event.appendChild(Duration);

Event.appendChild(Venue);

Event.appendChild(Description);

Event.appendChild(Participant);

/////////////////////////////////////

Data.appendChild(doc.createTextNode(e.getData()));

StartTime.appendChild(doc.createTextNode(e.getStartTime()));

Duration.appendChild(doc.createTextNode(Integer.toString(e.getDuration())));

Venue.appendChild(doc.createTextNode(e.getVenue()));

Description.appendChild(doc.createTextNode(e.getDescription()));

StringBuffer participant = new StringBuffer();

for (int i = 0; i < e.getParticipants().length; i++) {

participant.append(e.getParticipants()[i].getName() + " ");

participant.append(e.getParticipants()[i].getSurname() + " ");

participant.append(e.getParticipants()[i].getAge() + " ");

}

Participant.appendChild(doc.createTextNode(participant.toString()));

}

Transformer t = TransformerFactory.newInstance().newTransformer();

t.setOutputProperty(OutputKeys.INDENT, "yes");

t.transform(new DOMSource(doc), new StreamResult(new FileOutputStream(fileName)));

}

}

**ВАРІАНТИ ВИКОРИСТАННЯ**

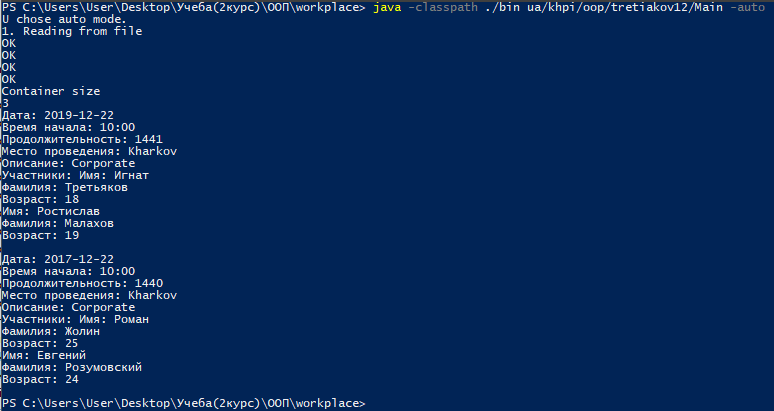


Рис. 12.1 – Результат роботи програми

Після строки Reading from file йдуть чотири строки з написом ОК це означає, що введені данні були перевірені і підпадають під дозволені.

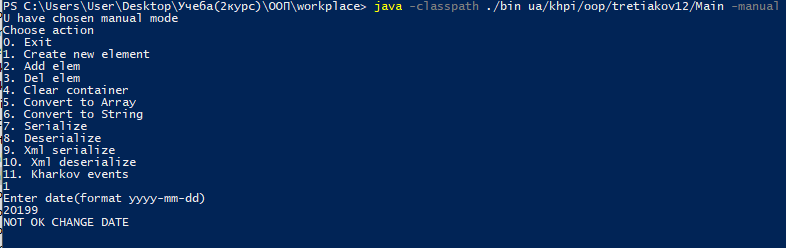


Рис . 12.2 – Результат роботи програми

Як видно зі скріншоту ім’я було введено не вірно і программа запросила ввести дату знову. Так само є перевірка введення часу початку заходу.

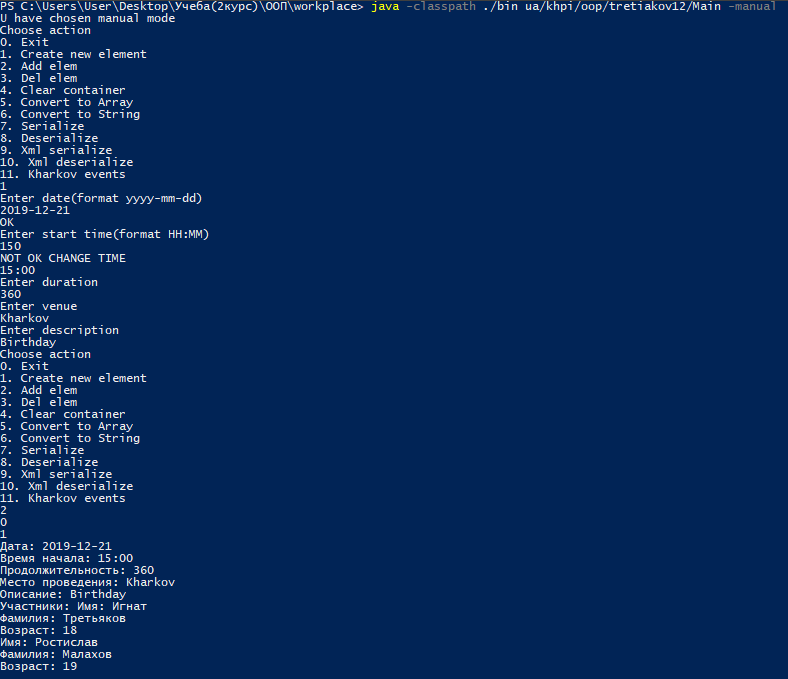


Рис. 12.3 – Результат роботи програми

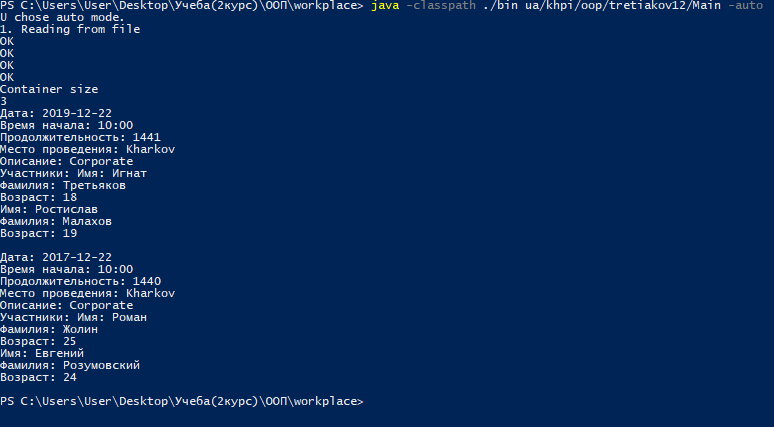


Рисунок 12.4 – Пошук елементів які задовольняють заданій умові (Дата > 2017, тривалість >= 1440, місто Харків).

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 - планувальник, в якому представляються різноманітні заходи. Також до програми додана функція перевірки введених користувачем данних, і при некоректно введених данних программа попросить користувача ввести данні знову.

**ВИСНОВКИ**

При виконанні лабораторної роботи набуто практичних навичок щодо розробки параметризованих класів, перевіряти введені данні за допомогою регулярних виразів та при невірно введених данних вони повинні бути введені знову згідно шаблону.