Третьяков І.С.

КІТ-118а

Лабораторна робота №14

# [Паралельне виконання. Ефективність використання](https://oop-khpi.gitlab.io/#tasks)

***Мета:***

* Розширення функціональності параметризованих класів.

## **Вимоги**

1. Використовуючи програми рішень попередніх задач, продемонструвати можливість паралельної обробки елементів контейнера: створити не менше трьох додаткових потоків, на яких викликати відповідні методи обробки контейнера.
2. Забезпечити можливість встановлення користувачем максимального часу виконання (таймаута) при закінченні якого обробка повинна припинятися незалежно від того знайдений кінцевий результат чи ні.
3. Для паралельної обробки використовувати алгоритми, що не змінюють початкову колекцію.
4. Кількість елементів контейнера повинна бути досить велика, складність алгоритмів обробки колекції повинна бути зіставна, а час виконання приблизно однаковий, наприклад:
   * пошук мінімуму або максимуму;
   * обчислення середнього значення або суми;
   * підрахунок елементів, що задовольняють деякій умові;
   * відбір за заданим критерієм;
   * власний варіант, що відповідає обраній прикладної області.

ПРИКЛАДНА ЗАДАЧА:

Планувальник. Знайти всі конференції, що пройшли за останні три роки в Харкові та області з тривалістю не менше доби.

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.tretiakov14;

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.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", 1000, "Kharkov", "Corporate", par2);

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

Event[] events = {firstEvent, secondEvent, thirdEvent};

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);

System.out.println("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

for(int i = 0; i<99999;i++) {

linkedContainer.addLast(secondEvent);

}

// Начало потока с огранечением по времени

long begin = System.currentTimeMillis();

Threads.MyTread1 myTread1 = new Threads.MyTread1();

myTread1.set(linkedContainer);

Threads.MyThread2 myThread2 = new Threads.MyThread2();

myThread2.set(linkedContainer);

Threads.MyThread3 myThread3 = new Threads.MyThread3();

myThread3.set(linkedContainer);

ExecutorService executorService = Executors.newFixedThreadPool(3);

Future<Boolean> future = executorService.submit(myTread1);

Future<Boolean> future1 = executorService.submit(myThread2);

Future<Boolean> future2 = executorService.submit(myThread3);

future.get();

future1.get();

future2.get();

future.cancel(true);

future1.cancel(true);

future2.cancel(true);

executorService.shutdown();

try {

// В следующей строке первым параметром идет кол-во секунд сколько будет выполняться поток

future.get(2000, TimeUnit.SECONDS);

} catch (TimeoutException ex) {

future.cancel(true);

}

executorService.shutdown();

long finnish = System.currentTimeMillis();

long result = finnish - begin;

System.out.println("Time parralell threads was working " + result + " Milliseconds");

// Конец потока с ограничением по времени

// Два потока без ограничения по времени

long start = System.currentTimeMillis();

/\*FirstThread threadFirst = new FirstThread(linkedContainer);

ExecutorService executorServiceFirst = Executors.newFixedThreadPool(1);

executorServiceFirst.submit(threadFirst);

executorServiceFirst.shutdown();

SecondThread threadSecond = new SecondThread(linkedContainer);

ExecutorService executorServiceSecond = Executors.newFixedThreadPool(1);

executorServiceSecond.submit(threadSecond);

executorServiceSecond.shutdown();

ThirdThread threadThird = new ThirdThread(linkedContainer);

ExecutorService executorServiceThird = Executors.newFixedThreadPool(1);

executorServiceThird.submit(threadThird);

executorServiceThird.shutdown();\*/

FirstThread ft = new FirstThread(linkedContainer);

ft.run();

SecondThread st = new SecondThread(linkedContainer);

st.run();

ThirdThread th = new ThirdThread(linkedContainer);

th.run();

long stop = System.currentTimeMillis();

long res = stop - start;

System.out.println("Time consecutive threads was working = " + res + " milliseconds");

linkedContainer.addLast(secondEvent);

linkedContainer.addLast(thirdEvent);

// конец потоков без ограничения по времени

// Поиск элемента соответствующего заданым критериям

choise(linkedContainer);

}

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();

}

}

}

ComparatorDuration.java

**package** ua.khpi.oop.tretiakov14;

**import** java.util.Comparator;

**public** **class** ComparatorDuration **implements** Comparator<Event> {

@Override

**public** **int** compare(Event o1, Event o2) {

**if** (o1.getDuration() > o2.getDuration()) {

**return** 1;

}

**else** **if** (o1.getDuration() == o2.getDuration()) {

**return** 0;

}

{

**return** -1;

}

}

}

ComparatorParticipant.java

**package** ua.khpi.oop.tretiakov14;

**import** java.util.Comparator;

**public** **class** ComparatorParticipant **implements** Comparator<Event>

{

@Override

**public** **int** compare(Event o1, Event o2)

{

**if** (o1.getParticipants().length > o2.getParticipants().length) {

**return** 1;

}

**else** **if** (o1.getParticipants().length == o2.getParticipants().length) {

**return** 0;

}

{

**return** -1;

}

}

}

Event.java

**package** ua.khpi.oop.tretiakov14;

**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.getData().compareTo(entry.getData());

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

**return** tmp;

}

}

FirstThread.java

**package** ua.khpi.oop.tretiakov14;

**public** **class** FirstThread **implements** Runnable {

LinkedContainer<Event> linkedContainer;

**public** FirstThread(LinkedContainer<Event> linkedContainer)

{

**this**.linkedContainer = linkedContainer;

}

@Override

**public** **void** run() {

**int** max = linkedContainer.getElementByIndex(0).getDuration();

**int** min = linkedContainer.getElementByIndex(0).getDuration();

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

**try** {

Thread.*sleep*(0);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**if** (linkedContainer.getElementByIndex(i).getDuration() < min) {

min = linkedContainer.getElementByIndex(i).getDuration();

}

**if** (linkedContainer.getElementByIndex(i).getDuration() > max) {

max = linkedContainer.getElementByIndex(i).getDuration();

}

}

System.***out***.println("Max duration = " + max+" minute");

System.***out***.println("Min duration = " + min+" minute");

}

}

Linked.java

**package** ua.khpi.oop.tretiakov14;

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

**void** addLast(E e);

**void** addFirst(E e);

**int** size();

E getElementByIndex(**int** index);

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

**void** clean();

}

LinkedContainer.java

package ua.khpi.oop.tretiakov14;

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;

}

}

}

Participant.java

**package** ua.khpi.oop.tretiakov14;

**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();

}

}

secondThread.java

**package** ua.khpi.oop.tretiakov14;

**public** **class** SecondThread **implements** Runnable {

LinkedContainer<Event> linkedContainer;

**public** SecondThread(LinkedContainer<Event> linkedContainer)

{

**this**.linkedContainer = linkedContainer;

}

@Override

**public** **void** run() {

**int** sum = 0;

**int** avg = 0;

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

**try** {

Thread.*sleep*(0);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

sum += linkedContainer.getElementByIndex(i).getDuration();

}

avg = sum / linkedContainer.size();

System.***out***.println("Sum duration = " + sum + " minute");

System.***out***.println("Avg duration = " + avg + " minute");

}

}

ThirdThread.java

package ua.khpi.oop.tretiakov14;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class ThirdThread implements Runnable {

LinkedContainer<Event> linkedContainer;

public ThirdThread(LinkedContainer<Event> linkedContainer)

{

this.linkedContainer = linkedContainer;

}

@Override

public void run() {

for(Event ev : linkedContainer)

{

try {

Thread.sleep(0);

} catch (InterruptedException e) {

e.printStackTrace();

}

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

System.out.println(ev);

}

}

}

}

Threads.java

**package** ua.khpi.oop.tretiakov14;

**import** java.util.concurrent.Callable;

**public** **class** Threads {

**public** **static** **class** MyThread3 **implements** Callable<Boolean> {

**public** **static** **final** **int** ***HIGHER\_DURATION*** = 1000;

LinkedContainer<Event> linkedContainer;

@Override

**public** Boolean call() **throws** Exception {

countHigherDuration();

**return** **true**;

}

**private** **void** countHigherDuration() **throws** InterruptedException {

**int** count = 0;

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

Thread.*sleep*(0);

**if** (linkedContainer.getElementByIndex(i).getDuration() > ***HIGHER\_DURATION***) {

count++;

}

}

//System.out.println("Number of events with higher duration: " + count);

}

**public** **void** set(LinkedContainer<Event> linkedContainer) {

**this**.linkedContainer = linkedContainer;

}

}

**public** **static** **class** MyThread2 **implements** Callable<Boolean> {

LinkedContainer<Event> linkedContainer;

**private** **void** sumAvgDuration() **throws** InterruptedException {

**long** sum = 0;

**long** avg = 0;

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

// Thread.sleep(2);

sum += linkedContainer.getElementByIndex(i).getDuration();

}

avg = sum / linkedContainer.size();

System.***out***.println("Sum duration = " + sum +" minute");

System.***out***.println("Avg duration = " + avg +" minute");

}

**public** **void** set(LinkedContainer<Event> linkedContainer) {

**this**.linkedContainer = linkedContainer;

}

@Override

**public** Boolean call() **throws** Exception {

sumAvgDuration();

**return** **true**;

}

}

**public** **static** **class** MyTread1 **implements** Callable<Boolean> {

LinkedContainer<Event> linkedContainer;

**public** **void** set(LinkedContainer<Event> linkedContainer) {

**this**.linkedContainer = linkedContainer;

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

}

@Override

**public** Boolean call() **throws** Exception {

run();

**return** **true**;

}

**public** **void** run() **throws** InterruptedException {

**int** max = linkedContainer.getElementByIndex(0).getDuration();

**int** min = linkedContainer.getElementByIndex(0).getDuration();

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

//Thread.sleep(10);

**if** (linkedContainer.getElementByIndex(i).getDuration() < min) {

min = linkedContainer.getElementByIndex(i).getDuration();

}

**if** (linkedContainer.getElementByIndex(i).getDuration() > max) {

max = linkedContainer.getElementByIndex(i).getDuration();

}

}

System.***out***.println("Max duration = " + max + " minute");

System.***out***.println("Min duration = " + min + " minute");

}

}

}

**ВАРІАНТИ ВИКОРИСТАННЯ**

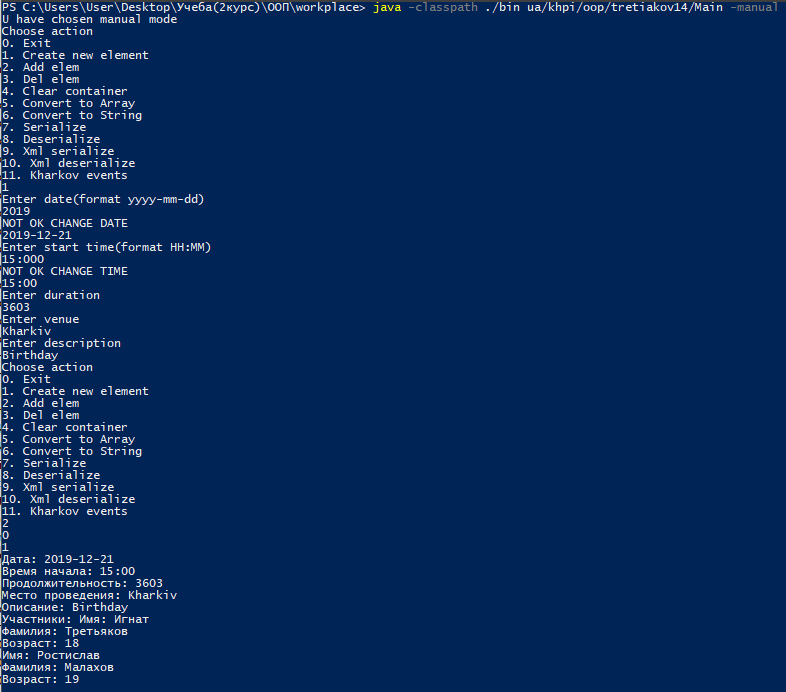


Рис. 14.1 – Результат роботи програми

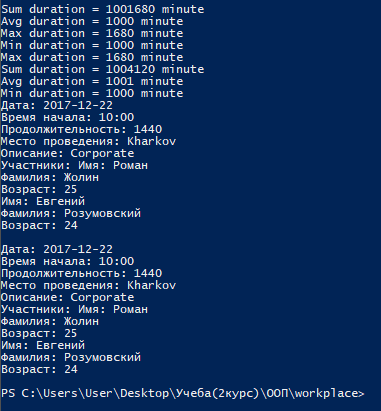


Рис 14.2 – Виконання роботи 3 додаткових потоків.



Рис 14.3 – Результат роботи программи змінився, адже швидкість роботи потоку була обмежена 2 секундами, а у циклі пошуку елементів задовольняючих умовам була встановлена штучна затримка 2 секунди, тому виконалися лише ті потоки, які були без обмеження по часу виконання.

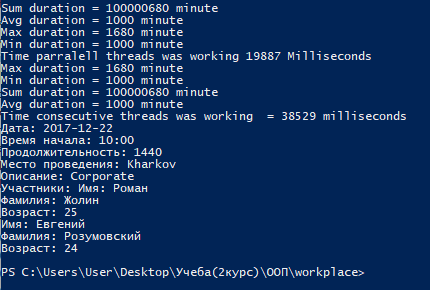


Рис 14.4 – Результат роботи паралельної і послідовної роботи потоків.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Кількість ел | Паралельне | Послідовне | Затримка | Різниця в часі | В середньому |
| 1 | 100 | 1100мс | 3209мс | 10мс | 2,917 | 5,27 рази паралельна обробка швидша за послідовну |
| 2 | 500 | 2852мс | 8712мс | 5мс | 3,054 |
| 3 | 1000 | 2758мс | 8382мс | 2мс | 3,039 |
| 4 | 10000 | 19703мс | 296965мс | 1мс | 15,07 |
| 5 | 100000 | 24806мс | 56475мс | 0мс | 2,27 |
|  |

Програму можна використовувати задля створення бази даних. Завдяки параметризації зв’язного списку, базу даних можна використати для будь-яких типів даних. В данній лабораторній роботі було проведено порівняння швидкості виконання послідовних та паралельних потоків.

**ВИСНОВКИ**

При виконанні лабораторної роботи набуто практичних навичок щодо розробки параметризованих класів. Порівняв швидкість роботи паралельних та послідовних потоків, та було виявлено у скільки разів виконання паралельних потоків буде швидше за послідовне виконання. На комп’ютері, на якому виконувалося порівняння швидкості було 2 фізичних ядра. Згідно з результатами ми можемо побачити, що в більшості тестів результат був швидше у 3 рази.