1

Автор:Горянський І.,1.КІТ101.8б

Лабораторна робота № 13

## ПАРАЛЕЛЬНЕ ВИКОНАННЯ. БАГАТОПОТОЧНІСТЬ

**1. ЗАВДАННЯ ДО РОБОТИ**

**Загальне завдання:**

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.

**1.1 Опис змінних:**

**private** T currElem; поточний елемент списку

**private** Node<T> prevElem; попередній елемент списку

**private** Node<T> nextElem; наступний елемент списку

**private** Node<T> fstNode;

**private** Node<T> lstNode;

**private** **int** size = 0; змінна розміру списку

**private** String Address; поле адресу прикладної галузі

**private** String Name; поле імені прикладної галузі

**private** String Special; поле спеціалізіції прикладної галузі

**private** String Phone; поле телефону прикладної галузі

**private** String StartTime; поле початку робочого дня прикладної галузі

**private** String EndTime; поле кінця робочого дня прикладної галузі

Текст програми

**package** ua.khpi.oop.gorianckiy09;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Threadik {

**public** **static** **class** ExampleThread **implements** Runnable {

LinkedContainer<Directory> dir;

**int** count = 0;

@Override

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

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

**if** (dir.getElementByIndex(i).getFreeTime().equals("Yes")) {

// try {

// Thread.sleep(40);

// } catch (InterruptedException e) {

//

// e.printStackTrace();

// }

count++;

}

}

System.***out***.println("FreeDay: ");

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

}

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

**this**.dir = linkedContainer;

}

}

**public** **static** **class** ExampleThread1 **implements** Runnable {

LinkedContainer<Directory> dir;

**int** count1 = 0;

**public** **void** set1(LinkedContainer<Directory> linkedContainer) {

**this**.dir = linkedContainer;

}

@Override

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

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

**if** (dir.getElementByIndex(i).getTime().equals("11:00")) {

// try {

// Thread.sleep(40);

// } catch (InterruptedException e) {

//

// e.printStackTrace();

// }

count1++;

}

}

System.***out***.println("StartTime: ");

System.***out***.println(count1);

}

}

**public** **static** **class** ExampleThread2 **implements** Runnable {

LinkedContainer<Directory> dir;

**int** count2 = 0;

**public** **void** set2(LinkedContainer<Directory> linkedContainer) {

**this**.dir = linkedContainer;

}

@Override

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

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

**if** (dir.getElementByIndex(i).getTime1().equals("23:00")) {

// try {

// Thread.sleep(40);

// } catch (InterruptedException e) {

//

// e.printStackTrace();

// }

count2++;

}

}

System.***out***.println("EndTime: ");

System.***out***.println(count2);

}

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.util.Iterator;

**public** **interface** DescendingIterator<T> {

Iterator<T> descendingIterator();

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.util.Scanner;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**import** java.text.ParseException;

**import** java.io.IOException;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.text.SimpleDateFormat;

**public** **class** Directory {

**private** String Address;

**private** String Name;

**private** String Special;

**private** String Phone;

**private** String StartTime;

**private** String EndTime;

**private** String FreeTime;

**private** String Consultation;

String str1;

String str2;

Directory(){

Address = **null**;

Name = **null**;

Special = **null**;

Phone = **null**;

StartTime = **null**;

EndTime = **null**;

FreeTime = **null**;

Consultation = **null**;

}

@Override

**public** String toString() {

**return** "\nAddress=" + Address + "\nName=" + Name + "\nSpecial=" + Special + "\nPhone=" + Phone

+ "\nStartTime=" + StartTime + "\nEndTime=" + EndTime + "\nFreeDay =" + FreeTime + "\nConsulation=" + Consultation;

}

Directory(**final** Directory obj){

Address = obj.Address;

Name = obj.Name;

Special = obj.Special;

Phone = obj.Phone;

StartTime = obj.StartTime;

EndTime = obj.EndTime;

FreeTime = obj.FreeTime;

Consultation = obj.Consultation;

}

**public** **void** setAddress(String Address) {

**this**.Address =Address;

}

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

**this**.Name = Name;

}

**public** **void** setSpecial(String Special) {

**this**.Special = Special;

}

**public** **void** setPhone(String Phone) {

**this**.Phone = Phone;

}

**public** **void** setTime(String Time) {

**this**.StartTime = Time;

}

**public** **void** setTime1(String Time1) {

**this**.EndTime = Time1;

}

**public** **void** setFreeTime(String FreeTime) {

**this**.FreeTime = FreeTime;

}

**public** **void** setConsultation(String Consult) {

**this**.Consultation = Consult;

}

String getFreeTime() {

**return** FreeTime;

}

String getConsultation() {

**return** Consultation;

}

String getAddress() {

**return** Address;

}

String getName() {

**return** Name;

}

String getSpecial() {

**return** Special;

}

String getPhone() {

**return** Phone;

}

**public** String getTime() {

// SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

// return sdf.format(this.StartTime);

**return** StartTime;

}

**public** String getTime1() {

// SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

// return sdf.format(this.EndTime);

**return** EndTime;

}

**public** **void** generateStore() **throws** ParseException,IOException {

Scanner s = **new** Scanner(System.***in***);

Pattern pat = Pattern.*compile*("^[a-zA-Z]{3,10}[0-9]{0,2}|[a-zA-Z]{3,10}\\s[a-zA-Z]{3,10}[0-9]{0,2}");

String str3 = **null**;

Matcher mat3 = **null**;

**do** {

System.***out***.println("Enter name of outlet");

str3 = s.nextLine();

mat3 = pat.matcher(str3);

}**while**(!mat3.matches());

Name = str3;

Pattern patt = Pattern.*compile*("^[a-zA-Z]{3,10}[0-9]{0,2}|[a-zA-Z]{3,10}\\s[0-9]{0,2}|[a-zA-Z]{3,10}\\s[a-zA-Z]{3,10}[0-9]{0,2}");

String str4 = **null**;

Matcher mat4 = **null**;

**do** {

System.***out***.println("Enter Address");

str4 =s.nextLine();

mat4 = patt.matcher(str4);

}**while**(!mat4.matches());

Address = str4;

Pattern patter = Pattern.*compile*("^[a-zA-Z]{3,10}");

String str5 =**null**;

Matcher mat5 = **null**;

**do** {

System.***out***.println("Enter Specialization");

str5 = s.nextLine();

mat5 = patter.matcher(str5);

}**while**(!mat5.matches());

Special = str5;

Pattern pattern = Pattern.*compile*("\\d{10,12}");

String str = **null**;

Matcher m = **null**;

**do** {

System.***out***.println("Enter Phone number");

str = s.nextLine();

m = pattern.matcher(str);

}**while**(!m.matches());

Phone = str;

Pattern pattern1 = Pattern.*compile*("[0-9]{2}:[0-9]{2}");

String str1 = **null**;

Matcher mat1 = **null**;

**do** {

System.***out***.println("Enter StartTime:");

str1 =s.nextLine();

mat1 = pattern1.matcher(str1);

}**while**(!mat1.matches());

StartTime = str1;

Pattern pattern2 = Pattern.*compile*("[0-9]{2}:[0-9]{2}");

String str2 = **null**;

Matcher mat2 = **null**;

**do** {

System.***out***.println("Enter EndTime:");

str2 =s.nextLine();

mat2 = pattern2.matcher(str2);

}**while**(!mat2.matches());

EndTime = str2;

System.***out***.println("Enter availability of FreeTime: ");

FreeTime =s.nextLine();

System.***out***.println("Enter availability of Consultation: ");

Consultation = s.nextLine();

}

**public** **static** Comparator<Directory> *comparatorName* = **new** Comparator<Directory>() {

@Override

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

**return** o1.getName().compareTo(o2.getName());

}

};

**public** **static** Comparator<Directory> *comparatorAddres* = **new** Comparator<Directory>() {

@Override

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

**return** o1.getAddress().compareTo(o2.getAddress());

}

};

**public** **static** Comparator<Directory> *comparatorSpecailization* = **new** Comparator<Directory>() {

@Override

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

**return** o1.getSpecial().compareTo(o2.getSpecial());

}

};

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.IOException;

**import** java.text.ParseException;

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

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

Main.*main*(args);

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.util.Arrays;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** FuncNine {

**public** **static** **void** ChooseMenu() {

System.***out***.println("\n\nChoose what you want to do:");

System.***out***.println("1. Delete all from Container");

System.***out***.println("2. Output to File ");

System.***out***.println("3. Output from File ");

System.***out***.println("4. Check for emptiness");

System.***out***.println("5. Output container to screen");

System.***out***.println("6. Add to our directory new data");

System.***out***.println("7. Sort directory by Name");

System.***out***.println("8. Sort directory by Addres");

System.***out***.println("9. Sort directory by Spicialization");

System.***out***.println("10. Word processing");

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

System.***out***.print("Write:");

}

**public** **static**<T **extends** Directory> **void** sortByName(LinkedContainer<T> t) {

Directory [] array = **new** Directory[t.size()];

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

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

}

Arrays.*sort*(array,Directory.*comparatorName*);

t.clear();

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

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

}

}

**public** **static**<T **extends** Directory> **void** sortByAddres(LinkedContainer<T>t) {

Directory []array = **new** Directory[t.size()];

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

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

}

Arrays.*sort*(array,Directory.*comparatorAddres*);

t.clear();

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

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

}

}

**public** **static**<T **extends** Directory> **void** sortBySpecial(LinkedContainer<T>t ) {

Directory [] array = **new** Directory[t.size()];

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

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

}

Arrays.*sort*(array,Directory.*comparatorSpecailization*);

t.clear();

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

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

}

}

**public** **static** <T **extends** Directory> **void** find(LinkedContainer<T>t ) {

**if**(t!=**null**) {

**for**(Directory s : t) {

Pattern p = Pattern.*compile*("No");

Matcher m = p.matcher(s.getFreeTime());

**if**(m.find()) {

Pattern pat = Pattern.*compile*("Yes");

Matcher mat = pat.matcher(s.getConsultation());

**if**(mat.find()) {

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

}

}

}

}**else** {

System.***out***.println("No matches found");

}

}

**public** **static** <T **extends** Directory> **void** find1(LinkedContainer<T>t,String filter,String filter1 ) {

**if**(t!=**null**) {

**for**(Directory s : t) {

Pattern p = Pattern.*compile*("No");

Matcher m = p.matcher(s.getFreeTime());

**if**(m.find()) {

Pattern pat = Pattern.*compile*("Yes");

Matcher mat = pat.matcher(s.getConsultation());

**if**(mat.find()) {

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

}

}

}

}**else** {

System.***out***.println("No matches found");

}

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.Serializable;

**import** java.text.ParseException;

**import** java.util.Iterator;

**public** **interface** Linked<T> {

**void** addLast(T obj);

**void** addFirst(T obj);

**int** size();

T getElementByIndex(**int** index);

**void** add(T obj);

**void** clear();

**void** toFile();

**void** FromFile() **throws** ParseException;

**void** readAll();

**void** DelAll();

**void** toFileAll();

**void** checkEmptiness();

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.ParseException;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.ParseException;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** LinkedContainer<T **extends** Directory> **implements** Linked<T>, Iterable<T>,DescendingIterator<T> {

//конструктор инициализации

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

**private** T currElem; //текущий

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

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

//конструктор копирования

Node( T currElem, Node<T> prevElem, Node<T> nextElem) {

**this**.currElem = currElem;

**this**.prevElem = prevElem;

**this**.nextElem = nextElem;

}

//геттеры и сеттеры

**public** T getCurrentElem() {

**return** currElem;

}

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

**this**.currElem = currentElem;

}

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

**return** prevElem;

}

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

**this**.prevElem = prevElem;

}

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

**return** nextElem;

}

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

**this**.nextElem = nextElem;

}

}

**private** Node<T> fstNode;

**private** Node<T> lstNode;

**private** **int** size = 0;

**public** LinkedContainer() {

fstNode = **new** Node<T>(**null**,**null**,lstNode);

lstNode = **new** Node<T>(**null**,fstNode,**null**);

fstNode = **new** Node<T>(**null**,**null**,lstNode);

}

@Override

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

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

**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** Iterator<T> descendingIterator() {

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

**int** position = size - 1;

@Override

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

**return** position >= 0;

}

@Override

**public** T next() {

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

position--;

**return** getElementByIndex(position--);

} **else** {

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

}

}

};

}

@Override

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

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

Node<T> next = x.nextElem;

x.currElem = **null**;

x.nextElem = **null**;

x.prevElem = **null**;

x = next;

}

fstNode = **null**;

lstNode = **null**;

lstNode = **new** Node<>(**null**, fstNode, **null**);

fstNode = **new** Node<>(**null**, **null**, lstNode);

size = 0;

}

@Override

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

addLast(t);

}

@Override

**public** **void** toFile() {

**try** {

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

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

file.createNewFile();

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

Directory temp ;

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

pw.println(size);

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

temp =(Directory)getElementByIndex(i);

pw.println(temp.getName());

pw.println(temp.getAddress());

pw.println(temp.getSpecial());

pw.println(temp.getPhone());

pw.println(temp.getTime() );

pw.println(temp.getTime1());

}

pw.close();

}

**catch**(IOException e){

System.***out***.print("Error: " + e);

}

}

@Override

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

**try** (BufferedReader br = **new** BufferedReader(**new** FileReader("File1.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** FromFile() **throws** ParseException {

BufferedReader br = **null**;

Directory temp = **new** Directory();

**try** {

br = **new** BufferedReader(**new** FileReader("C:\\Users\\User\\eclipse-workspace\\laba11\\File1.txt"));

String Line;

Line = br.readLine();

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

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

Pattern pat = Pattern.*compile*("^[a-zA-Z]{3,10}[0-9]{0,2}|[a-zA-Z]{3,10}\\s[a-zA-Z]{3,10}[0-9]{0,2}");

Matcher mat3 = **null**;

**do** {

System.***out***.println("Enter name of outlet");

Line = br.readLine();

mat3 = pat.matcher(Line);

}**while**(!mat3.matches());

temp.setName(Line);

Pattern patt = Pattern.*compile*("^[a-zA-Z]{3,10}[0-9]{0,2}|[a-zA-Z]{3,10}\\s[0-9]{0,2}|[a-zA-Z]{3,10}\\s[a-zA-Z]{3,10}[0-9]{0,2}");

Matcher mat4 = **null**;

**do** {

System.***out***.println("Enter Address");

Line = br.readLine();

mat4 = patt.matcher(Line);

}**while**(!mat4.matches());

temp.setAddress(Line);

Pattern patter = Pattern.*compile*("^[a-zA-Z]{3,10}|[a-zA-Z]{3,10}\\s[a-zA-Z]{3,10}");

Matcher mat5 = **null**;

**do** {

System.***out***.println("Enter Specialization");

Line = br.readLine();

mat5 = patter.matcher(Line);

}**while**(!mat5.matches());

temp.setSpecial(Line);

Pattern pattern = Pattern.*compile*("\\d{10,12}");

Matcher m = **null**;

**do** {

System.***out***.println("Enter Phone number");

Line = br.readLine();

m = pattern.matcher(Line);

}**while**(!m.matches());

temp.setPhone(Line);

Pattern pattern1 = Pattern.*compile*("[0-9]{2}:[0-9]{2}");

Matcher mat1 = **null**;

**do** {

System.***out***.println("Enter StartTime:");

Line = br.readLine();

mat1 = pattern1.matcher(Line);

}**while**(!mat1.matches());

temp.setTime(Line);

Pattern pattern2 = Pattern.*compile*("[0-9]{2}:[0-9]{2}");

Matcher mat2 = **null**;

**do** {

System.***out***.println("Enter EndTime:");

Line = br.readLine();

mat2 = pattern2.matcher(Line);

}**while**(!mat2.matches());

temp.setTime1(Line);

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

}

}**catch**(IOException e) {

System.***out***.print("Error: " + e);

}**finally** {

**try** {

br.close();

}**catch**(IOException e ) {

System.***out***.print("Error: " + e);

}

}

}

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

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

**int** i = 0;

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

result[i++] = temp.currElem;

}

**return** result;

}

@Override

**public** String toString() {

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

}

@Override

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

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

prev.setCurrentElem((T)**new** Directory(obj)); //установка значения

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

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

size++; //увелечение размера списка

}

@Override

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

Node<T> next = fstNode;

next.setCurrentElem(obj);

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

next.setPrevElem(fstNode);

size++;

}

@Override

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

**return** size;

}

@Override

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

Node<T> target = fstNode.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();

}

@Override

**public** **void** DelAll() {

fstNode = **null**;

lstNode = **null**;

lstNode = **new** Node<>(**null**, fstNode, **null**);

fstNode = **new** Node<>(**null**, **null**, lstNode);

size = 0;

}

@Override

**public** **void** toFileAll() {

**try** {

File file = **new** File("File.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***.print("Error: " + e);

}

}

@Override

**public** **void** checkEmptiness() {

**if**(size >0) {

System.***out***.println("Container is not empty");

}**else** {

System.***out***.println("Conteiner is empty");

}

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.IOException;

**import** java.text.ParseException;

**public** **class** Main {

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

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

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

TypeOfWork.*auto*();

}

} **else** {

System.***out***.println("Could not find value: -auto\n"

+ "Transition to manual mode");

TypeOfWork.*menu*();

}

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.IOException;

**import** java.text.ParseException;

**import** java.util.Scanner;

**import** java.util.Date;

**import** java.util.InputMismatchException;

**import** java.util.concurrent.\*;

**import** ua.khpi.oop.gorianckiy09.Threadik.ExampleThread;

**import** ua.khpi.oop.gorianckiy09.Threadik.ExampleThread1;

**import** ua.khpi.oop.gorianckiy09.Threadik.ExampleThread2;

**public** **class** TypeOfWork {

**public** **static** **void** menu() **throws** ParseException, IOException, ExecutionException, InterruptedException {

LinkedContainer<Directory> stringLinked = **new** LinkedContainer<>();

Directory dir = **new** Directory();

System.***out***.println("Welocme to Manual Mode");

**int** loop = 1;

**while** (loop == 1) {

FuncNine.*ChooseMenu*();

Scanner sc = **new** Scanner(System.***in***);

**int** choose = sc.nextInt();

**switch** (choose) {

**case** 1:

stringLinked.clear();

**break**;

**case** 2:

stringLinked.toFileAll();

**break**;

**case** 3:

stringLinked.readAll();

**break**;

**case** 4:

stringLinked.checkEmptiness();

**break**;

**case** 5:

**for** (Directory s : stringLinked) {

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

}

**break**;

**case** 6:

dir.generateStore();

stringLinked.add(dir);

**break**;

**case** 7:

FuncNine.*sortByName*(stringLinked);

**break**;

**case** 8:

FuncNine.*sortByAddres*(stringLinked);

**break**;

**case** 9:

FuncNine.*sortBySpecial*(stringLinked);

**break**;

**case** 10:

FuncNine.*find*(stringLinked);

**break**;

**case** 11:

ExampleThread exampleThread = **new** ExampleThread();

exampleThread.set(stringLinked);

//new Thread(exampleThread).start();

ExampleThread1 exampleThread1 = **new** ExampleThread1();

exampleThread1.set1(stringLinked);

//new Thread(exampleThread1).start();

ExampleThread2 exampleThread2 = **new** ExampleThread2();

exampleThread2.set2(stringLinked);

//new Thread(exampleThread2).start();

**int** time = sc.nextInt();

**long** start = System.*currentTimeMillis*();

ExecutorService executorService = Executors.*newFixedThreadPool*(3);

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

Future<Boolean> future1 = (Future<Boolean>) executorService.submit(exampleThread1);

Future<Boolean> future2 = (Future<Boolean>) executorService.submit(exampleThread2);

**try** {

//Thread.sleep(40000);

future.get(time, TimeUnit.***SECONDS***);

future1.get(time, TimeUnit.***SECONDS***);

future2.get(time,TimeUnit.***SECONDS***);

}**catch**(TimeoutException ex){

future.cancel(**true**);

future1.cancel(**true**);

future2.cancel(**true**);

}

**long** finish = System.*currentTimeMillis*();

**long** res = finish - start;

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

**break**;

**case** 0:

loop = 0;

}

}

}

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

LinkedContainer<Directory> dir = **new** LinkedContainer<>();

System.***out***.println("Welcome to auto-mode");

System.***out***.println("Add to container new elements");

dir.FromFile();

System.***out***.println("Check if there is something in container");

dir.checkEmptiness();

System.***out***.println("Let's see what we added");

**for** (Directory s : dir) {

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

}

System.***out***.println("Check size of container");

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

dir.toFile();

}

}

**2. Діаграми**

3.Результат роботи:

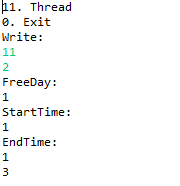


Рисунок 1 – Результат роботи

ВИСНОВОК

При виконанні цієї лабораторної роботи були набуті навички роботи з потоками.