Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №8

Виконав:

студент групи ІО-22

Коломієць О.

Перевірив:

Невдащенко М. В.

Номер залікової книжки:

2209

Київ 2013р.

**Тема:** Робота з потоками в мові програмування Java.

**Мета:** Здобуття навичок у використанні потоків в мові програмування Java.

1) Завдання

1. Створити клас, що складається з методів збереження до файлу та зчитування з файлу колекції (лабораторна робота №6), як єдиного об’єкту; як послідовності об’єктів узагальненого класу (лабораторна робота №5); як послідовності об’єктів у вигляді тексту.

2. Написати JUnit-тести для перевірки працездатності усіх методів та виключних ситуацій.

3. Код повинен відповідати стандартам JCC та бути детально задокументований.

3) Код програми

**import** java.util.Scanner;

**import** java.io.Serializable;

/\*\*

\* Clas Travel from Lab\_5

\* **@author** Kolomiets Oleh

\*

\*/

**public** **class** Travel **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1064004830324310885L;

/\*\*

\* **@param** d

\* time of travel

\* **@param** f

\* type of food

\*/

**public** Travel(**int** d, **int** f){

days = d;

food = f;

}

**public** Travel(Scanner in){

in.next(); setDays(in.nextInt());

in.next(); setFood(in.nextInt());

}

/\*\* gets days of travel

\* **@return** days

\*/

**public** **int** getDays(){

**return** days;

}

/\*\* gets type of food

\* **@return**

\*/

**public** **int** getFood(){

**return** food;

}

/\*\*Sets days of travel

\* **@param** days

\*/

**public** **void** setDays(**int** days) {

**this**.days = days;

}

/\*\*

\* sets type of food

\* **@param** food

\*/

**public** **void** setFood(**int** food) {

**this**.food = food;

}

/\*

\* @Override

\*/

**public** String toString(){

**return** "Travel " + "daysOfTravel: " + days + " food: " + food;

}

/\*\*

\* Days field

\*/

**private** **int** days;

/\*\*

\* Food field

\*/

**private** **int** food;

}

/\*\*

\* Node of list

\* **@author** Kolomiets Oleh

\*

\*/

**public** **class** Node **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 7623897119915359744L;

/\*\*

\* data field

\*/

**private** Travel data;

/\*\*

\* nextNode field

\*/

**private** Node nextNode;

/\*\*

\* Constructor of Node with data parameter

\* **@param** data object of Travel class

\*/

**public** Node(Travel data){

**this**.data = data;

}

/\*\*

\* Constructor of Node with data and link to the next node parameters

\* **@param** data object of Travel class

\* **@param** node object of Node class

\*/

**public** Node(Travel data, Node node){

**this**.data = data;

**this**.nextNode = node;

}

/\*\*

\* gets data

\* **@return** data

\*/

**public** Travel getData() {

**return** data;

}

/\*\*

\* sets data

\* **@param** data

\*/

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

**this**.data = data;

}

/\*\*

\* gets NextNode

\* **@return** NextNode

\*/

**public** Node getNextNode() {

**return** nextNode;

}

/\*\* sets NextNode

\* **@param** nextNode

\*/

**public** **void** setNextNode(Node nextNode) {

**this**.nextNode = nextNode;

}

}

**import** java.io.BufferedReader;

**import** java.io.EOFException;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.Serializable;

**import** java.util.Collection;

**import** java.util.Iterator;

**import** java.util.Scanner;

**import** java.util.Set;

/\*\*

\* Class which implements Set through list data structure

\* **@author** Kolomiets Oleh

\*

\*/

**public** **class** TravelSet **implements** Set<Travel> ,Serializable{

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = -5653508480273150671L;

/\*\*

\* size field

\*/

**private** **int** size;

/\*\*

\* link for the first node

\*/

**private** Node first = **new** Node(**null**);

/\*\*

\* link for the last node

\*/

**private** Node last = **new** Node(**null**);

/\*\*

\* empty constructor

\*/

**public** TravelSet(){

first = **new** Node(**null**);

last = **new** Node(**null**);

}

/\*\*

\* constructor which have one parameter of class Travel

\* **@param** t object of class Travel

\*/

**public** TravelSet(Travel t){

add(t);

}

/\*\*

\* constructor with collection as parameter

\* **@param** a collection

\*/

**public** TravelSet(Collection<Travel> a ){

addAll(a);

}

/\* adds element to set.

\* adds element to the end of set

\*/

@Override

**public** **boolean** add(Travel a) {

**if**(a == **null**)

**throw** **new** NullTravelPointerException();

**if**(!contains(a)){

**if**(size == 0) {

first.setData(a);

last = first;

}

**else**{

last.setNextNode(**new** Node(a));

last = last.getNextNode();

}

size++;

**return** **true**;

}

**else**

**return** **false**;

}

/\*

\* adds collection to set.

\* iterates over collection and adds elements to the end of set

\*/

@Override

**public** **boolean** addAll(Collection<? **extends** Travel> arg0) {

**boolean** result = **false**;

**for**(Travel t : arg0){

**if**(add(t))

result = **true**;

**else**

result = **false**;

}

**return** result;

}

/\*

\* clears the set

\*/

@Override

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

first = last = **new** Node(**null**);

size = 0;

}

/\*

\* Checks is element present in collection

\* return true if it is, else returns false

\*/

@Override

**public** **boolean** contains(Object o) {

**if**( o == **null**)

**throw** **new** NullTravelPointerException();

**for**(Node curr = first; curr != **null**; curr = curr.getNextNode())

**if**(o.equals(curr.getData()))

**return** **true**;

**return** **false**;

}

/\*

\* Checks is collection contains in collection

\* return true if it is, else returns false

\*/

@Override

**public** **boolean** containsAll(Collection<?> coll) {

**boolean** result = **false**;

**for**(Iterator<?> iterator = coll.iterator(); iterator.hasNext();){

**if**(contains(iterator.next()))

result = **true**;

}

**return** result;

}

/\*

\* Checks is collection empty

\* return true if it is, else returns false

\*/

@Override

**public** **boolean** isEmpty() {

**return** size == 0;

}

/\*

\* create Iterator to iterate over the collection

\*/

@Override

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

Iterator<Travel> it = **new** Iterator<Travel>(){

@SuppressWarnings("unused")

Node temp = **new** Node(**null**);

Node current = first;

@Override

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

**if** (current == **null**)

**return** **false**;

**else**

**return** **true**;

}

@Override

**public** Travel next() {

Travel result = **null**;

**if**(hasNext()){

temp = current;

result = current.getData();

current = current.getNextNode();

//result = temp.getData();

}

**return** result;

}

@Override

**public** **void** remove() {

}

};

**return** it;

}

/\*

\* Remove element from set if it there

\*/

@Override

**public** **boolean** remove(Object arg0) {

**if**(arg0 == **null**)

**throw** **new** NullTravelPointerException();

Node current = first;

**boolean** result = **false**;

**if**(current.getData().equals(arg0)){

first = current.getNextNode();

result = **true**;

size--;

}

**else**{

**while**(current.getNextNode() != **null**){

Node next = current.getNextNode();

**if**(next.getData().equals(arg0)){

current.setNextNode(next.getNextNode());

size--;

last = current;

result = **true**;

}

**else**

current = current.getNextNode();

}

}

**return** result;

}

/\*

\* Remove collection from set if it there

\*/

@Override

**public** **boolean** removeAll(Collection<?> coll) {

**boolean** result = **false**;

**for**(Iterator<?> iterator = coll.iterator(); iterator.hasNext();){

Object obj = (Object)iterator.next();

result = remove(obj);

}

**return** result;

}

@Override

**public** **boolean** retainAll(Collection<?> arg0) {

**return** **false**;

}

/\*

\* returne size of the set

\*/

@Override

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

**return** size;

}

/\*

\* convert set to array

\* return array of Object

\*/

@Override

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

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

**int** i = 0;

**for**(Iterator<?> iterator = iterator(); iterator.hasNext();){

Object obj = (Object)iterator.next();

a[i] = obj;

i++;

}

**return** a;

}

/\*

\* convert set to array of given type

\* return array

\*/

@SuppressWarnings("unchecked")

@Override

**public** <T> T[] toArray(T[] arg0) {

**return** (T[])toArray();

}

/\*\*

\* Perform collection to string.

\*/

@Override

**public** String toString() {

StringBuffer sb = **new** StringBuffer();

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

o = **this**.toArray();

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

sb.append(o[i].toString());

sb.append(" ");

}

**return** sb.toString();

}

/\*\*

\* Writes elements of collection to file

\* **@param** fileName

\* **@throws** IOException

\*/

**public** **void** writeToFile(String fileName) **throws** IOException{

ObjectOutputStream oos = **null**;

**try**{

FileOutputStream file = **new** FileOutputStream(fileName);

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

**for**(Iterator<Travel> iter = iterator(); iter.hasNext();){

oos.writeObject(iter.next());

}

}**catch**(FileNotFoundException ex){

ex.printStackTrace();

}

**finally**{

**if**(oos != **null**)

**try**{

oos.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

/\*\*

\* Reads elements from file and add them to collection

\* **@param** fileName

\* **@throws** IOException

\*/

**public** **void** readFromFile(String fileName) **throws** IOException{

ObjectInputStream ois = **null**;

**try**{

FileInputStream file = **new** FileInputStream(fileName);

ois = **new** ObjectInputStream(file);

Object o = **null**;

**while**(**true**){

**try**{

o = ois.readObject();

**if**(o **instanceof** Travel){

**this**.add((Travel)o);

}

}

**catch**(EOFException ex){

**break**;

}

**catch**(ClassNotFoundException ex){

ex.printStackTrace();

}

}

}**catch**(FileNotFoundException ex){

ex.printStackTrace();

}

**finally**{

**if**(ois != **null**)

**try**{

ois.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

/\*\*

\* Writes collection to file as object.

\* **@param** fileName

\* **@throws** IOException

\*/

**public** **void** writeToFileAsObject(String fileName) **throws** IOException {

ObjectOutputStream oos = **null**;

**try** {

oos = **new** ObjectOutputStream(**new** FileOutputStream(fileName));

oos.writeObject(**this**);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

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

**try**{

oos.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

}

/\*\*

\* Read collection as object from file.

\* **@param** fileName

\* **@throws** IOException

\*/

**public** **void** readFromFileAsObject(String fileName) **throws** IOException {

ObjectInputStream ois = **null**;

**try** {

ois = **new** ObjectInputStream(**new** FileInputStream(fileName));

**try** {

Object s = ois.readObject();

**this**.addAll((Collection<Travel>) s);

} **catch** (ClassNotFoundException ex) {

ex.printStackTrace();

} **catch** (IOException ex) {

ex.printStackTrace();

}

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

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

**try**{

ois.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

}

/\*\*

\* Write collection as text to file.

\* **@param** fileName

\* **@throws** IOException

\*/

**public** **void** writeToFileAsText(String fileName) **throws** IOException {

FileWriter fw = **null**;

**try** {

fw = **new** FileWriter(**new** File(fileName));

fw.write( **this**.toString());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

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

**try**{

fw.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

}

/\*\*

\* Read collection as text from file.

\* **@param** fileName

\* **@throws** IOException

\*/

**public** **void** readFromFileAsText(String fileName) **throws** IOException {

FileReader file = **new** FileReader(fileName);

BufferedReader br = **new** BufferedReader(file);

Scanner in = **new** Scanner(br);

**while** (in.hasNext()) {

String s = in.next();

**if** (s.equals("Travel")) {

**this**.add(**new** Travel(in));

} **else** **if** (s.equals("Assignment")) {

**this**.add(**new** Assignment(in));

} **else** **if** (s.equals("Sanatorium")) {

**this**.add(**new** Sanatorium(in));

} **else** **if** (s.equals("Cruise")) {

**this**.add(**new** Cruise(in));

}

}

in.close();

}

}

**import** **static** org.junit.Assert.\*;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

/\*\*

\* Class created for testing methods from TravelSet class

\* **@author** Kolomiets Oleh

\*

\*/

**public** **class** TravelSetTest {

**static** TravelSet *tr*;

**static** Travel *t1* = **new** Travel(7, 3);

**static** Travel *t2* = **new** Travel(5, 3);

**static** Travel *t3* = **new** Travel(14,3);

**static** ArrayList<Travel> *a*;

@BeforeClass

**public** **static** **void** BeforeTravelSetTest(){

*a* = **new** ArrayList<Travel>();

*a*.add(*t1*);

*a*.add(*t2*);

}

@Test

**public** **void** testTravelSet() {

*tr* = **new** TravelSet();

*assertTrue*("Constructor with out parameters" , *tr*.isEmpty());

}

@Test

**public** **void** testTravelSetTravel() {

*tr* = **new** TravelSet(*t1*);

*assertTrue*("Constructor with Travel type parameter", *tr*.contains(*t1*));

}

@Test

**public** **void** testTravelSetCollection() {

*tr* = **new** TravelSet(*a*);

*assertTrue*("Constructor with collectoin as parameter", *tr*.containsAll(*a*));

}

@Test

**public** **void** testAdd() {

*tr* = **new** TravelSet();

*tr*.add(*t1*);

*assertTrue*("add", *tr*.contains(*t1*));

**try**{

*tr*.add(**null**);

*fail*("Exeption");

} **catch** (NullTravelPointerException e) {

}

}

@Test

**public** **void** testAddAll() {

*tr* = **new** TravelSet();

*tr*.addAll(*a*);

*assertTrue*("addAll", *tr*.containsAll(*a*));

}

@Test

**public** **void** testClear() {

*tr* = **new** TravelSet();

*tr*.add(*t1*);

*tr*.add(*t2*);

*tr*.clear();

*assertTrue*(*tr*.isEmpty());

}

@Test

**public** **void** testContains() {

*tr* = **new** TravelSet();

*tr*.add(*t1*);

*assertTrue*("contains", *tr*.contains(*t1*));

**try**{

*tr*.contains(**null**);

*fail*("Exeption");

} **catch**(NullTravelPointerException a){

}

}

@Test

**public** **void** testContainsAll() {

*tr* = **new** TravelSet();

*tr*.addAll(*a*);

*assertTrue*("containsAll", *tr*.containsAll(*a*));

}

@Test

**public** **void** testIsEmpty() {

*tr* = **new** TravelSet();

*assertTrue*("isEmpty", *tr*.isEmpty());

}

@Test

**public** **void** testRemove() {

*tr* = **new** TravelSet();

*tr*.add(*t1*);

*tr*.remove(*t1*);

*assertFalse*("remove", *tr*.contains(*t1*));

**try**{

*tr*.remove(**null**);

*fail*("Exeption");

} **catch**(NullTravelPointerException a){

}

}

@Test

**public** **void** testRemoveAll() {

*tr* = **new** TravelSet();

*tr*.addAll(*a*);

*tr*.removeAll(*a*);

*assertTrue*("removeAll", *tr*.isEmpty());

}

@Test

**public** **void** testsize() {

*tr* = **new** TravelSet();

*assertEquals*("deflaute constructor", 0, *tr*.size());

*tr*.add(*t1*);

*assertEquals*("added one element", 1, *tr*.size());

}

@Test

**public** **void** writeReadToFile() {

**try** {

String fileName = "testFile1.dat";

*tr* = **new** TravelSet();

*tr*.add(*t1*);

*tr*.writeToFile(fileName);

TravelSet d = **new** TravelSet();

d.readFromFile(fileName);

*assertEquals*(d.toString(), *tr*.toString());

} **catch** (IOException e) {

e.printStackTrace();

*fail*();

}

}

@Test

**public** **void** testWriteReadToFileAsObject() {

**try** {

String fileName = "testFile2.dat";

*tr* = **new** TravelSet();

*tr*.add(*t1*);

*tr*.add(*t3*);

*tr*.writeToFileAsObject(fileName);

TravelSet tr1 = **new** TravelSet();;

tr1.readFromFileAsObject(fileName);

*assertEquals*(*tr*.toString(), tr1.toString());

} **catch** (IOException e) {

e.printStackTrace();

*fail*();

}

}

@Test

**public** **void** testWriteReadToFileAsText() {

**try** {

String fileName = "testFile3.txt";

*tr* = **new** TravelSet();

*tr*.add(*t1*);

*tr*.add(*t2*);

*tr*.add(*t3*);

*tr*.writeToFileAsText(fileName);

TravelSet tr1 = **new** TravelSet();

tr1.readFromFileAsText(fileName);

*assertEquals*(*tr*.toString(), tr1.toString());

} **catch** (IOException e) {

e.printStackTrace();

*fail*();

}

}

}

4) Висновок

Під час лабораторної роботи я закріпив вміння працювати із колекціями та в мові програмування Java, навчився працювати із потоками, зчитувати/записувати колекцій до файлу у вигляді окремого об’єкту, послідовності елементів та у вигляді тексту, написав JUnit-тести для перевірки працездатності усіх методів та виключних ситуацій.