Національний Технічний Університет України

«Київський Політехнічний Інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №8

Обробка виключних ситуацій та основи тестування в мові програмування Java

|  |  |
| --- | --- |
| Прийняв  Ст.в. Невдащенко М.В.  «\_\_»\_\_\_\_\_\_\_\_ 2014 р. | Виконала Студентка 1-ого курсу ФІОТ  групи ІО-32  Шапран К.О |

**Завдання:**

1. Модифікувати класи з попередніх лабораторних робіт (лабораторні роботи №6 та №7) таким чином, щоб обробка виключних ситуацій відбувалась за допомогою стандартних засобів мови програмування Java. Створити власний клас обробник виключних ситуацій.

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

3. Всі початкові дані задаються у виконавчому методі. Код повинен відповідати стандартам JCC та бути детально задокументований.

**Лістинг:**

Клас StoneCollection

/\*\*

\* @(#) StoneCollection.java 1.0 03/06/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

**import** java.lang.reflect.Array;

**import** java.util.Collection;

**import** java.util.Iterator;

**import** java.util.Set;

/\*\* Class StoneCollection, which has methods

\* and constructors for collection of the stones

\*

\* **@version** 1.0 03 June 2014

\* **@author** Karina Shapran

\* **@since** 1.0

\*/

**public** **class** StoneCollection <T> **implements** Set<Stones> {

/\*\*

\* **@param** s, new list

\* **@param** size, size of the list

\*/

**private** List s = **new** List();

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

/\*\*

\* Constructs an empty list

\*/

**public** StoneCollection (){

}

/\*\*

\* Passing an object to the list

\* **@param** stn, an element of the Stones

\*/

**public** StoneCollection (Stones stn){

add(stn);

}

/\*\*

\* Constructs a list containing

\* the elements of the specified collection

\* **@param** stn, the collection whose elements

\* are to be placed into this list

\*/

**public** StoneCollection(Collection<? **extends** Stones> stn){

addAll(stn);

}

/\*\*

\* Method, which adds an element to the beginning of a list

\* **@param** e, an element of the Stones

\* **@param** flag

\* **@return** flag

\*/

@Override

**public** **boolean** add(Stones stn) {

**boolean** flag = **true**;

ListElement t = s.getHead();

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

**if** (t.data==stn) {

flag = **false**;

**break**;

}

t = t.next;

}

**if** (flag) {

s.addBack(stn);

size++;

}

**return** flag;

}

/\*\*

\* Method, which adds all elements of the collection

\* **@param** c, the collection whose elements

\* are to be placed into this list

\* **@param** flag

\* **@return** flag

\*/

@Override

**public** **boolean** addAll(Collection<? **extends** Stones> c) {

Iterator<? **extends** Stones> iterator = c.iterator();

**boolean** flag = **false**;

**while** (iterator.hasNext()) {

flag = add((Stones) iterator.next());

}

**return** flag;

}

/\*\*

\* Method, which clears the list

\*/

@Override

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

s = **null**;

s = **new** List();

}

/\*\*

\* Method, which checks for the existence of the object

\* **@param** flag

\* **@return** flag

\*/

@Override

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

**try** {

**boolean** flag = add((Stones) o);

**if** (flag) {

s.delEl((Stones) o);;

}

**return** flag==**false**;

} **catch** (ClassCastException e) {

System.*out*.println("Неправильний тип переданого об'єкта");

**return** **false**;

}

}

/\*\*

\* Method, which checks for the existence of the collection

\* **@param** flag

\* **@return** flag

\*/

@Override

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

**boolean** flag = **true**;

Iterator<Stones> iterator = (Iterator<Stones>) c.iterator();

**while** (iterator.hasNext()) {

flag = contains((Stones) iterator.next());

**if** (flag==**false**){

**break**;

}

}

**return** flag;

}

/\*\*

\* Method, which checks the emptiness of the list

\*/

@Override

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

**return** s.getHead()==**null**;

}

/\*\*

\* Method, which return iterator

\*

\*/

@Override

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

**return** **new** StoneIterator();

}

/\*\*

\* Method, which removes an element of the list

\* **@param** o, object of the collection

\*/

@Override

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

**try** {

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

**throw** **new** MyException();

}

**if** (contains(o)) {

s.delEl((Stones) o);

size--;

**return** **true**;

} **else** {

**return** **false**;

}

} **catch** (ClassCastException e) {

System.*out*.println("Неправильний тип параметру");

**return** **false**;

}**catch** (MyException e) {

e.printStackTrace();

**return** **false**;

}

}

/\*\*

\* Method, which removes all elements of the collection

\* **@param** flag;

\* **@return** flag;

\*/

@Override

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

**boolean** flag = **false**;

**for** (Object object : c) {

flag = remove(object);

}

**return** flag;

}

/\*\*

\* Method, which implements the set intersection

\* **@param** c, the collection whose elements

\* are to be placed into this list

\* **@param** flag

\* **@return** flag

\*/

@Override

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

**boolean** flag = **true**;

Iterator<Stones> iterator = iterator();

**while** (iterator.hasNext()) {

Stones stones = (Stones) iterator.next();

flag = c.contains(stones);

**if** (flag == **false**)

remove(stones);

}

**return** flag == **false**;

}

/\*\*

\* Method, which have size of the collection

\* **@return** size;

\*/

@Override

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

**return** size;

}

/\*\*

\* Method, which turns objects of collection to array

\* **@param** i

\* **@return** mas

\*/

@Override

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

**int** i = 0;

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

Iterator<Stones> iterator = iterator();

**while** (iterator.hasNext()) {

mas[i] = iterator.next();

i++;

}

**return** mas;

}

/\*\*

\* Method, which records elements of the collection

\* in the array until it has a place

\* **@param** array;

\* **@param** size;

\* **@param** i;

\* **@return** array;

\*/

@Override

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

**int** size = size();

**if** (array.length < size) {

array = (T[]) Array.*newInstance*(array.getClass().getComponentType(), size);

} **else** **if** (array.length > size) {

array[size] = **null**;

}

**int** i = 0;

**for** (Stones e: **this**) {

array[i] = (T) e;

i++;

}

**return** array;

}

/\*\*

\* Interal class StoneIterator, which implements interface of Iterator

\***@version** 1.0 03 June 2014

\* **@author** Karina Shapran

\* **@since** 1.0

\*

\*/

**class** StoneIterator **implements** Iterator {

/\*\*

\* **@param** thisElement

\*/

**private** ListElement thisElement = s.getHead();

/\*\*

\* Method, which (checks if the next element exists)

\*/

@Override

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

**return** thisElement != **null**;

}

/\*\*

\* Method,which have next element of the collection

\* **@param** s, an element of the Stones

\* **@return** s;

\*/

@Override

**public** Stones next() {

Stones s = thisElement.data;

thisElement = thisElement.next;

**return** s;

}

/\*\*

\* Method, which remove elements of the collection

\*/

@Override

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

s.delEl(thisElement.data);

}

}

**public** **void** testCallNullPointerException() {

**throw** **new** NullPointerException();

}

}

Клас TestStoneCollection

/\*\*

\* @(#) TestStoneCollection.java 1.0 03/06/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

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

**import** java.util.Iterator;

**import** org.junit.Before;

**import** org.junit.Test;

/\*\*

\* Class TestStoneCollection, is testing

\* all method of StoneCollection

\* **@version** 1.0 03 June 2014

\* **@author** Karina Shapran

\* **@since** 1.0

\*/

**public** **class** TestStoneCollection {

/\*\*

\* **@param** col

\* **@param** col1

\* **@param** col2

\* **@param** col13

\* **@param** s1

\* **@param** s2

\* **@param** s3

\*/

**private** StoneCollection col;

**private** StoneCollection<Stones> col1;

**private** StoneCollection<Stones> col2;

**private** StoneCollection<Stones> col3;

**private** Stones s1;

**private** Stones s2;

**private** Stones s3;

@Before

**public** **void** setParam(){

col = **new** StoneCollection();

col1 = **new** StoneCollection<Stones>();

col2 = **new** StoneCollection<Stones>();

col3 = **new** StoneCollection<Stones>();

s1 = **new** Stones("1", 1, 1, 1);

s2 = **new** Stones("2", 2, 2, 2);

s3 = **new** Stones("3", 3, 3, 3);

}

@Test

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

*assertTrue*(col1.add(s1));

*assertFalse*(col1.add(s1));

}

@Test

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

col2.add(s1);

col2.add(s2);

*assertTrue*(col1.addAll(col2));

*assertFalse*(col1.addAll(col2));

}

@Test

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

col.clear();

*assertTrue*(col.size()==0);

}

@Test

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

col.add(s1);

*assertTrue*(col.contains(s1));

*assertFalse*(col.contains(s2));

}

@Test

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

col1.add(s1);

col1.add(s2);

col2.add(s3);

*assertFalse*(col1.containsAll(col2));

}

@Test(timeout = 1000)

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

col1.add(s1);

*assertFalse*(col1.size()== 0 );

col1.remove(s1);

*assertTrue*(col1.size()== 0 );

}

@Test

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

col1.add(s1);

*assertTrue*(col1.remove(s1));

*assertFalse*(col1.remove(s1));

}

@Test

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

col1.add(s1);

col1.add(s2);

col2.add(s1);

*assertTrue*(col1.removeAll(col2));

*assertFalse*(col1.removeAll(col2));

}

@Test

**public** **void** testSize(){

col1.add(s1);

*assertTrue*(col1.size()!= 0);

}

@Test

**public** **void** testToArray(){

col1.add(s1);

col1.add(s2);

col1.add(s3);

Object [] mas = col1.toArray();

*assertTrue*(((Stones)mas[0]).equals(s1));

*assertTrue*(((Stones)mas[1]).equals(s2));

*assertTrue*(((Stones)mas[2]).equals(s3));

}

@Test

**public** **void** testToArrayType(){

col1.add(s1);

col1.add(s2);

col1.add(s3);

Stones [] array = **new** Stones[0];

Stones [] mas = col1.toArray(array);

*assertTrue*(((Stones)mas[0]).equals(s1));

*assertTrue*(((Stones)mas[1]).equals(s2));

*assertTrue*(((Stones)mas[2]).equals(s3));

}

@Test(expected = NullPointerException.**class**)

**public** **void** testNullPointerException(){

col.testCallNullPointerException();

}

@Test

**public** **void** testEquals(){

*assertFalse*(col1.equals(col2));

col1 = col2;

*assertTrue*(col1.equals(col2));

}}

Клас MyException

/\*\*

\* @(#) MyException.java 1.0 03/06/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\*

\* Class MyException

\* **@version** 1.0 03 June 2014

\* **@author** Karina Shapran

\* **@since** 1.0

\*/

**public** **class** MyException **extends** Exception {

@Override

**public** **void** printStackTrace() {

System.*out*.println("Помилка Null");

**super**.printStackTrace();

}}

Клас Test

/\*\*

\* @(#) Test.java 1.0 03/06/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

**import** java.lang.reflect.Array;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Iterator;

**import** java.util.Set;

/\*\* Class Test

\* **@version** 1.0 03 June 2014

\* **@author** Karina Shapran

\* **@since** 1.0

\*/

**public** **class** Test {

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

StoneCollection<Stones> col1 = **new** StoneCollection<Stones>();

StoneCollection<Stones> col2 = **new** StoneCollection<Stones>();

Stones s1 = **new** Stones("Rubin", 3.95, 600, 2.1);

Stones s2 = **new** Stones("Onyx", 8.34, 720, 4.21);

col1.add(s1);

col1.add(s2);

Necklace [] n = **new** Necklace[4];

col1.remove(**null**);

col2.add(**new** Stones ("Diamond", 2.72, 11830, 15.1));

col2.add(s2);

System.*out*.println(col2.contains(s2));

}}

**Опис програми:**

TestStoneCollection - клас, який тестує всі методи класа StoneCollection на працездатність за допомогою J-Unit тестів. StoneCollection містить обробку деяких виключних ситуацій за допомогою стандартних засобів мови програмування Java.

**Висновок:**

У даній лабораторній роботі ми здобули навичок у використанні механізму обробки виключних ситуацій та написанні тестів для перевірки працездатності методів в мові програмування Java, створивши клас, що тестує методи іншого класу, а також обробивши виключні ситуації допомогою стандартних засобів мови програмування Java.