**Міністерство освіти і науки, молоді та спорту України**

**Національний технічний університет України**

**«Київський політехнічний інститут»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №8**

**Тема: «Обробка виключних ситуацій**

**та основи тестування в мові програмування Java.»**

**Виконав: Студент групи ІО-33**

**Заколенко Р.К.**

**Перевірив: Викладач**

**Невдащенко М.В.**

**Київ 2014**

1. **Завдання**
2. Модифікувати класи з попередніх лабораторних таким чином, щоб обробка виключних ситуацій відбувалась за допомогою стандартних засобів мови програмування Java.

1. Написати JUnit-тести для перевірки працездатності усіх методів та виключних ситуацій.
2. **Опис програми**

Програма містить 2 класи:

1. Клас DoubleList (implements List<Munitions>);
2. Клас TestDoubleList (клас, у якому відбувається тестування методів класу DoubleList);

У класі TestDoubleList за допомогою бібліотеки JUnit відбувається тестування методів класу DoubleList. Клас TestDoubleList має поле типу DoubleList list, яке ініціалізується кожного разу перед виконанням тесту(за допомогою методу @Before public void setList()) та використовується у даному тесті.

Виключні ситуації обробляються за допомогою конструкції try{} catch(){}.

1. **Код програми**

Клас TestDoubleList, у якому тестуються методи класу DoubleList:

**package** ua.kpi.io33.zakolenko.lab8;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.ListIterator;

**import** org.junit.\*;

**import** ua.kpi.io33.zakolenko.lab7.DoubleList;

**import** ua.kpi.io33.zakolenko.lab7.Munitions;

/\*\*

\* Class TestDoubleList is used for testing methods of class DoubleList.

\*

\* **@author** Roman Zakolenko.

\* **@version** 1,6 27 May 2014.

\*/

**public** **class** TestDoubleList {

**private** DoubleList list;

/\*\*

\* Performed before each test. Initializes the field "list".

\*/

@Before

**public** **void** setList() {

**this**.list = **new** DoubleList();

**for** (**int** i = 0; i < 15; i++) {

list.add(**new** Munitions(i, i + 1, "iron"));

}

}

/\*\*

\* Test the method size().

\*/

@Test

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

Assert.*assertEquals*(15, list.size());

}

/\*\*

\* Test the method equals(Object o).

\*/

@Test

**public** **void** testEqualsMunitions() {

Assert.*assertEquals*(**true**,

list.getFirst().getData().equals(**new** Munitions(0, 1, "iron")));

Assert.*assertEquals*(

**false**,

list.getFirst().getNext().getData()

.equals(**new** Munitions(4, 5, "iron")));

}

/\*\*

\* Test the empty constructor.

\*/

@Test

**public** **void** testEmptyConstructor() {

DoubleList sample = **new** DoubleList();

Assert.*assertNull*(sample.getFirst());

}

/\*\*

\* Test the constructor.

\*/

@Test

**public** **void** testConstructor() {

DoubleList sample = **new** DoubleList(**new** Munitions(20, 25, "platina"));

Assert.*assertTrue*(sample.getFirst().getData()

.equals(**new** Munitions(20, 25, "platina")));

}

/\*\*

\* Test the constructor.

\*/

@Test

**public** **void** testConstructorCollection() {

DoubleList sample = **new** DoubleList(**this**.list);

Assert.*assertEquals*(15, sample.size());

}

/\*\*

\* Test the method get(int).

\*/

@Test

**public** **void** testGet() {

Assert.*assertTrue*(list.get(2).equals(**new** Munitions(2, 3, "iron")));

}

/\*\*

\* Test the method add(int, Munitions).

\*/

@Test

**public** **void** testAddByIndex() {

list.add(3, **new** Munitions(3, 4, "ferrum"));

list.add(5, **new** Munitions(34, 4, "iron"));

Assert.*assertEquals*(17, list.size());

Assert.*assertTrue*(list.get(3).equals(**new** Munitions(3, 4, "ferrum")));

}

/\*\*

\* Test the method add(Munitions).

\*/

@Test

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

list.add(**new** Munitions(23, 24, "cuprum"));

Assert.*assertEquals*(16, list.size());

Assert.*assertTrue*(list.get(15).equals(**new** Munitions(23, 24, "cuprum")));

}

/\*\*

\* Test the method addAll(Collection<? extends Munitions>).

\*/

@Test

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

DoubleList sample = **new** DoubleList(**new** Munitions(2, 4, "ferrum"));

sample.addAll(list);

Assert.*assertEquals*(16, sample.size());

Assert.*assertTrue*(sample.get(0).equals(**new** Munitions(2, 4, "ferrum")));

Assert.*assertTrue*(sample.get(1).equals(list.get(0)));

Assert.*assertTrue*(sample.get(15).equals(list.get(14)));

}

/\*\*

\* Test the method addAll(int, Collection<? extends Munitions>).

\*/

@Test

**public** **void** testAddAllInt() {

DoubleList sample = **new** DoubleList(**new** Munitions(2, 4, "ferrum"));

sample.add(**new** Munitions(10, 5, "ferrum"));

sample.addAll(1, list);

Assert.*assertEquals*(17, sample.size());

Assert.*assertTrue*(sample.get(0).equals(**new** Munitions(2, 4, "ferrum")));

Assert.*assertTrue*(sample.get(1).equals(list.get(0)));

Assert.*assertTrue*(sample.get(16).equals(**new** Munitions(10, 5, "ferrum")));

}

/\*\*

\* Test the method clear().

\*/

@Test

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

list.clear();

Assert.*assertNull*(list.getFirst());

Assert.*assertEquals*(0, list.size());

}

/\*\*

\* Test the method contains(Object).

\*/

@Test

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

Assert.*assertTrue*(list.contains(**new** Munitions(2, 3, "iron")));

Assert.*assertFalse*(list.contains(**new** Munitions(2, 5, "ferrum")));

}

/\*\*

\* Test the method containsAll(Collection<?>).

\*/

@Test

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

DoubleList sample = **new** DoubleList(**new** Munitions(2, 3, "iron"));

Assert.*assertTrue*(list.containsAll(sample));

sample.add(**new** Munitions(2, 2, "iron"));

Assert.*assertFalse*(list.containsAll(sample));

}

/\*\*

\* Test the method indexOf(Object).

\*/

@Test

**public** **void** testIndexOf() {

Assert.*assertTrue*(list.indexOf(**null**) == -1);

Assert.*assertEquals*(2, list.indexOf(**new** Munitions(2, 3, "iron")));

}

/\*\*

\* Test the method isEmpty().

\*/

@Test

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

Assert.*assertFalse*(list.isEmpty());

list.clear();

Assert.*assertTrue*(list.isEmpty());

}

/\*\*

\* Test the method iterator().

\*/

@Test

**public** **void** testIterator() {

Iterator<Munitions> iter = list.iterator();

**while** (iter.hasNext()) {

Assert.*assertNotNull*(iter.next());

}

}

/\*\*

\* Test the method lastIndexOf(Object).

\*/

@Test

**public** **void** testLastIndexOf() {

Assert.*assertTrue*(list.lastIndexOf(**null**) == -1);

Assert.*assertEquals*(2, list.lastIndexOf(**new** Munitions(2, 3, "iron")));

list.add(**new** Munitions(2, 3, "iron"));

Assert.*assertEquals*(15, list.lastIndexOf(**new** Munitions(2, 3, "iron")));

}

/\*\*

\* Test the method listIterator().

\*/

@Test

**public** **void** testListIterator() {

ListIterator<Munitions> iter = list.listIterator();

**while** (iter.hasNext()) {

Assert.*assertNotNull*(iter.next());

}

**while** (iter.hasPrevious()) {

Assert.*assertNotNull*(iter.previous());

}

}

/\*\*

\* Test the method listIterator(int).

\*/

@Test

**public** **void** testListIteratorIndex() {

ListIterator<Munitions> iter = list.listIterator(6);

**while** (iter.hasNext()) {

Assert.*assertNotNull*(iter.next());

}

**while** (iter.hasPrevious()) {

Assert.*assertNotNull*(iter.previous());

}

}

/\*\*

\* Test the method remove(int).

\*/

@Test

**public** **void** testRemoveByIndex() {

Assert.*assertNull*(list.remove(100));

Assert.*assertTrue*(list.get(5).equals(list.remove(5)));

Assert.*assertEquals*(14, list.size());

}

/\*\*

\* Test the method remove(Object).

\*/

@Test

**public** **void** testRemoveObject() {

Assert.*assertFalse*(list.remove(**null**));

Assert.*assertTrue*(list.remove(**new** Munitions(2, 3, "iron")));

Assert.*assertEquals*(14, list.size());

}

/\*\*

\* Test the method removeAll(Collection<?>).

\*/

@Test

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

DoubleList sample = **new** DoubleList(**new** Munitions(2, 3, "iron"));

sample.add(**new** Munitions(**new** Munitions(3, 4, "iron")));

Assert.*assertTrue*(list.removeAll(sample));

Assert.*assertFalse*(list.removeAll(sample));

Assert.*assertEquals*(13, list.size());

}

/\*\*

\* Test the method retainAll(Collection<?>).

\*/

@Test

**public** **void** testRetainsAll() {

DoubleList sample = **new** DoubleList(list.subList(1, 5));

sample.add(**new** Munitions(4, 24, "cuprum"));

Assert.*assertTrue*(list.retainAll(sample));

Assert.*assertEquals*(4, list.size());

}

/\*\*

\* Test the method set(int, Munitions).

\*/

@Test

**public** **void** testSet() {

Assert.*assertNull*(list.set(100, **null**));

Assert.*assertTrue*(list.get(5).equals(

list.set(5, **new** Munitions(4, 3, "cuprum"))));

Assert.*assertTrue*(list.get(5).equals(**new** Munitions(4, 3, "cuprum")));

}

/\*\*

\* Test the method subList(int, int).

\*/

@Test

**public** **void** testSubList() {

Assert.*assertTrue*(list.subList(1, 1).size() == 0);

List<Munitions> sample = list.subList(1, 5);

Assert.*assertTrue*(sample.size() == 4);

ListIterator<Munitions> iter = sample.listIterator();

**while** (iter.hasNext()) {

Assert.*assertNotNull*(iter.next());

}

}

/\*\*

\* Test the method toArray().

\*/

@Test

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

Object[] sample = list.toArray();

Assert.*assertEquals*(15, sample.length);

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

Assert.*assertNotNull*(sample[i]);

}

}

/\*\*

\* Test the method toArray(T[]).

\*/

@Test

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

Munitions[] sample1 = **new** Munitions[5];

sample1 = list.toArray(sample1);

Assert.*assertEquals*(15, sample1.length);

**for** (**int** i = 0; i < sample1.length; i++) {

Assert.*assertNotNull*(sample1[i]);

}

Munitions[] sample2 = **new** Munitions[20];

sample2 = list.toArray(sample2);

Assert.*assertEquals*(20, sample2.length);

**for** (**int** i = 0; i < sample2.length - list.size(); i++) {

Assert.*assertNotNull*(sample1[i]);

}

}

}

Методи класу DoubleList, у яких за допомогою конструкції try{}catch(){} були оброблені виключні ситуації:

/\*\*

\* Inserts the object of class Munitions at the specified position in this

\* list.

\*/

@Override

**public** **void** add(**int** index, Munitions element) {

**if** (index <= **this**.size()) {

**if** (index == 0) {

setFirst(**new** Cell(**null**, first, element));

**try** {

getFirst().getNext().setPrevious(getFirst());

} **catch** (NullPointerException e) {

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

}

} **else** {

ListIterator<Munitions> iter = **this**.listIterator();

**while** (iter.hasNext()) {

iter.next();

**if** (iter.nextIndex() == index) {

iter.add(element);

}

}

}

}

}

/\*\*

\* Inserts the specified element into the list.

\*/

@Override

**public** **void** add(Munitions arg0) {

Cell a = **new** Cell(present, present.getNext(), arg0);

**try** {

a.getNext().setPrevious(a);

} **catch** (NullPointerException e) {

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

} **finally** {

present.setNext(a);

present = present.getNext();

}

}

/\*\*

\* Removes from the list the last element that was returned by

\* next() or previous().

\*/

@Override

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

**if** (present == first) {

**try** {

setFirst(**new** Cell(**null**, present.getNext().getNext(),

present.getNext().getData()));

getFirst().getNext().setPrevious(getFirst());

present = getFirst();

} **catch** (NullPointerException e) {

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

setFirst(**null**);

}

} **else** {

**try** {

present.getPrevious().setNext(present.getNext());

present.getNext().setPrevious(present.getPrevious());

} **catch** (NullPointerException e) {

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

present.getPrevious().setNext();

} **finally** {

present = present.getPrevious();

}

}

}

/\*\*

\* **@return** an array containing all of the elements in this list in proper

\* sequence.

\*/

@Override

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

**if** (a **instanceof** Munitions[]) {

**try** {

System.*arraycopy*(toArray(), 0, a, 0, size());

} **catch** (IndexOutOfBoundsException e) {

a = (T[]) **new** Munitions[size()];

System.*arraycopy*(toArray(), 0, a, 0, size());

}

}

**return** a;

}

1. **Висновок**

В результаті виконання лабораторної роботи я ознайомився із механізмом обробки виключних ситуацій та тестуванням в мові програмування Java. За допомогою конструкції try{}catch(){} були оброблені виключні ситуації типу NullPointerException та IndexOutOfBoundsException.

За допомогою бібліотеки JUnit було виконане тестування методів класу DoubleList. Результати тестування позитивні.