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 №7

З дисципліни «Об’єктно-орієнтоване програмування»

Тема: «Обробка виключних ситуацій та основи тестування в мові програмування Java»

Виконав:

студент групи ІВ-71

Мазан Я. В.

Номер залікової книжки:

7109

Перевірив:

Подрубайло О. О.

Київ 2018

1. Варіант завдання.

Номер залікової книжки — 7109

Завдання

1. Модифікувати класи з попередніх лабораторних робіт (лабораторні роботи №5 та №6) 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.

2. Код програми

Файл Main.java:

```
/*
 * @(#)Lab7.java 1.0 30/05/18
 *
 * Copyright (c) 2018 Yan Mazan
 */
public class Main {


    public static void main(String[] args) throws MyException {
        /**
         * Initialising vegetables
         */
        Vegetable cucumber = null;
        Vegetable onion = null;
        Vegetable cabbage = null;
        Vegetable selera = null;
        Vegetable k = null;
        Vegetable m = null;
        /**
         * Giving value to vegetables and testing them on wrong input
         */
        try {
            cucumber = new Vegetable("Огірок", 16);
```

```
    onion = new Vegetable("Цибуля", 40);
    cabbage = new Vegetable("Капуста", 25);
    selera = new Vegetable("Селера", 16);
    k = new Vegetable("R", -1874545);
    m = new Vegetable("Pine", -1000);
} catch (MyException e) {
    e.printStackTrace();
}
/**
 * Adding vegetables to my collection salad
 */
VegetablesCollection<Vegetable> salad = new VegetablesCollection();
try {
    salad.add(m);
    salad.add(cucumber);
    salad.add(onion);
    salad.add(cabbage);
    salad.add(selera);

} catch (NullPointerException e) {
    System.out.println("null exception");
} catch (ArrayIndexOutOfBoundsException e2) {
    System.out.println("ArrayIndexOutOfBoundsException");
}

/**
 * Print salad
 */
System.out.println("My salad: ");
for (Vegetable i: salad) {
    System.out.println(i);
}

/**
 * Find vegetables with nutrition in selected range
```

```
         */
        int min_nutrition = 10;
        int max_nutrition = 30;
        System.out.println("\nSalad elements with nutrition between 10 and 30: ");
        for (Vegetable i: salad) {
            if (min_nutrition <= i.nutrition && max_nutrition >= i.nutrition) {
                System.out.println(i);
            }
        }
    }
}
```

Файл MyException.java:

```
/**
 * Class of my exception which is used when vegetable
 * is initialised with wrong input data
 */
public class MyException extends Exception {
    public MyException() {
        System.out.println("Wrong input data!");
    }
}
```

Файл TestIterator.java:

```
/**
 * Testing my iterator inside VegetablesCollection
 */
import org.junit.Before;
import org.junit.Test;
import java.util.Iterator;

import static org.junit.Assert.*;

public class TestIterator {

    /**
     * Initialising variables for testing
```

```
 */
Iterator testIterator;
VegetablesCollection testCollection;
private Vegetable a;
private Vegetable b;
private Vegetable c;

/**
 * Initialising variables for testing
 */
@Before
public void setUp() throws MyException{
    a = new Vegetable("Помідор", 12);
    b = new Vegetable("Огірок", 8);
    testCollection = new VegetablesCollection();
    testCollection.add(a);
    testCollection.add(b);
    testIterator = testCollection.iterator();
}

/**
 * Testing iterator's method hasNext
 */
@Test
public void testHasNext(){
    assertTrue(testIterator.hasNext());
    testIterator.next();
    assertFalse(testIterator.hasNext());
}

/**
 * Testing iterator's method Next
 */
@Test
public void testNext(){
```

```
        assertSame(b,testIterator.next());
    }
}
```

Файл IestVegetablesCollection.java:

```
/**
 * Testing of my VegetablesCollection
 */

import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
public class TestVegetablesCollection {

    private VegetablesCollection testCollection;
    private Vegetable a;
    private Vegetable b;

    /**
     * Method that initialises everything necessary before each testing
     * @throws MyException
     */
    @Before
    public void setUp() throws MyException{
        a = new Vegetable("Помідор", 12);
        b = new Vegetable("Огірок", 8);
        testCollection = new VegetablesCollection();
    }

    /**
     * Testing on IndexOutOfBoundsException
     */
    @Test
    public void testIndexOutOfBoundsException(){
        assertTrue(testCollection.remove(5));
    }
```

```
/**
 * Testing on NulPointerException
 */
@Test
public void testNullPointerException() {
    assertTrue(testCollection.add(null));
}

/**
 * Testing of method clear()
 */
@Test
public void testClear(){
    testCollection.add(a);
    testCollection.add(b);
    testCollection.clear();
    assertEquals(0, testCollection.size());
}

/**
 * Testing of method size()
 */
@Test
public void testSize(){
    testCollection.add(a);
    testCollection.add(b);
    assertEquals(2, testCollection.size());
}

/**
 * Testing of method isEmpty()
 */
@Test
public void testIsEmpty(){
```

```
    assertTrue(testCollection.isEmpty());
    testCollection.add(a);
    testCollection.add(b);
    assertFalse(testCollection.isEmpty());
}

/**
 * Testing of method contains()
 */
@Test
public void testContains(){
    testCollection.add(a);
    assertFalse(testCollection.isEmpty());
    assertTrue(testCollection.contains(a));
    assertFalse(testCollection.contains(b));
}

/**
 * Testing of method containsAll()
 */
@Test
public void testContainsAll(){
    VegetablesCollection testCollection2 = new VegetablesCollection();
    testCollection2.add(a);
    testCollection.add(a);
    testCollection.add(b);
    assertTrue(testCollection.containsAll(testCollection2));
    assertFalse(testCollection2.containsAll(testCollection));
}

/**
 * Testing of method toArray()
 */
@Test
public void testToArray(){
```

```
    Vegetable[] testArray = new Vegetable[2];
    testArray[0] = a;
    testArray[1] = b;
    testCollection.add(a);
    testCollection.add(b);
    assertSame(testCollection.toArray(), testArray);
}

/**
 * Testing of method add()
 */
@Test
public void testAdd(){
    assertTrue(testCollection.add(a));
}

/**
 * Testing of method remove()
 */
@Test
public void testRemove(){
    testCollection.add(a);
    testCollection.add(b);
    testCollection.remove(b);
    assertFalse(testCollection.contains(b));
}

/**
 * Testing of method addAll()
 */
@Test
public void testAddAll(){
    VegetablesCollection testCollection2 = new VegetablesCollection();
    testCollection2.add(a);
    testCollection.add(b);
```

```
        testCollection.addAll(testCollection2);
        assertTrue(testCollection.contains(b));
    }

    /**
     * Testing of method retainAll()
     */
    @Test
    public void testRetainAll(){
        VegetablesCollection testCollection2 = new VegetablesCollection();
        testCollection2.add(a);
        testCollection.add(a);
        testCollection.add(b);
        testCollection.retainAll(testCollection2);
        assertFalse(testCollection.contains(b));
    }

    /**
     * Testing of method removeAll()
     */
    @Test
    public void testRemoveAll(){
        VegetablesCollection testCollection2 = new VegetablesCollection();
        testCollection2.add(a);
        testCollection.add(a);
        testCollection.add(b);
        testCollection.removeAll(testCollection2);
        assertFalse(testCollection.contains(a));
    }
}
```

Файл Vegetable.java:

```
/** Class Vegetablen defines a vegetable with its name and nutrition.
 * VegetablesCollection implements interface Set
 */
public class Vegetable {
```

```
    public String name = "";
    public int nutrition = 0;

    /**
     * Constructor of empty vegetable
     */
    public Vegetable(){}

    /**
     * Standard constructor of vegetable
     */
    public Vegetable(String name, int calories_value) throws MyException {
        if (calories_value<0) {
            throw new MyException();
        }
        else {
            this.name = name;
            this.nutrition = calories_value;
        }
    }

    /**
     * Method to print attributes of the vegetable
     * @return void
     */
    public String toString() {
        return "Овоч: " + name + "; Поживність: " + nutrition + " кал";
    }

}
```

Файл VegetablesCollection.java:

```
/*
 * @(#)VegetablesCollection.java 1.0 30/05/18
 *
 * Copyright (c) 2018 Yan Mazan
```

```
 */
import java.util.Collection;
import java.util.Set;
import java.util.Arrays;
import java.util.Iterator;

/** Class VegetablesCollection creates collection of vegetables.
 * VegetablesCollection implements interface Set
 *
 * @version 1.0 5 May 2018
 * @author Yan Mazan
 * @since 1.0
 */
public class VegetablesCollection<T> implements Set<Vegetable> {

    /**
     * @param size
     * number of elements in collection
     * @param increasePercent
     * percent, on which we increase array of elements
     * @param elements
     * list of elements of collection
     */
    private int size;
    private double increasePercent = 1.3;
    private Object[] elements = new Object[15];

    /**
     * Constructor of empty collection
     */
    public VegetablesCollection(){
        size = 0;
    }

    /**
```

```
 * Constructor of collection with one element
 */
public VegetablesCollection(Vegetable o){
    size = 1;
    elements[0] = o;
}

/**
 * Constructor of collection, that includes elements
 * of VegetablesCollection collection
 */
public VegetablesCollection(Collection<? extends Vegetable> o){
    size = 0;
    addAll(o);
}

/**
 * Private method, which increases size of elements
 * of collection array
 * @return void
 */
private void increaseSize(){
    Object[] temporary = elements;
    elements = new Object[(int)(elements.length*increasePercent)];
    size = 0;
    for(Object t:temporary){
        add((Vegetable)t);
    }
}

/**
 * Overridden standard methods of Set collection
 *
 * Method that clears this collection from its elements
 * @return void
```

```
 */
@Override
public void clear(){
    for (int i = 0; i<elements.length; i++){
        elements[i] = null;
    }
    size = 0;
}

/**
 * Method that returns number of elements in collection
 * @return size of collection
 */
@Override
public int size(){
    return size;
}

/**
 * Method that checks out is the collection empty
 * @return true if collection is empty, else false
 */
@Override
public boolean isEmpty(){
    return size==0;
}

/**
 * Method that checks out does the collection contain defined element
 * @return true if collection contains the element, else false
 */
@Override
public boolean contains(Object o){
    for (int i = 0; i<elements.length; i++){
        if (elements[i] == o){
```

```
            return true;
        }
    }
    return false;
}

/**
 * Method that checks out does the collection contain
 * all elements of given collection
 * @return true if collection contains the elements, else false
 */
@Override
public boolean containsAll(Collection<?> c){
    MyIterator iterator = (MyIterator)c.iterator();
    while (iterator.hasNext()) {
        if (!contains(iterator.next())){
            return false;
        }
    }
    return true;
}

/**
 * Method that transmits collection into array
 * @return array
 */
@Override
public Object[] toArray(){
    return Arrays.copyOf(elements, size);
}

/**
 * Method that transmits collection into defined type array
 * @return array
 */
```

```
@Override
public <E> E[] toArray(E[] a) {
    if (a.length < size) {
        return (E[]) Arrays.copyOf(elements, size, a.getClass());
    }
    System.arraycopy(elements, 0, a, 0, size);
    if (a.length > size) {
        a[size] = null;
    }
    return a;

}

/**
 * Method that adds an element into collection
 * @return true
 */
@Override
public boolean add(Vegetable e){
    if (!contains(e)) {
        for (int i = 0; i < elements.length; i++) {
            if (elements[i] == null) {
                size++;
                elements[i] = e;
                return true;
            }
        }
        increaseSize();
        for (int i = 0; i < elements.length; i++) {
            if (elements[i] == null) {
                size++;
                elements[i] = e;
                return true;
            }
        }
```

```
    }
    return false;
}

/**
 * Private method that returns last not null element
 * of data structure of collection
 * @return int index of last not null element
 */
private int lastNotEmpty(){
    int i = elements.length-1;
    while (elements[i]==null) { i--; }
    return i+1;
    }

/**
 * Method that removes element from collection
 * @return true if the element was removed, else false
 */
@Override
public boolean remove(Object o){
    if(size==0){
        return false;
    }
    for (int i = 0; i < elements.length; i++) {
        if (elements[i]==o){
            size--;
            //to prevent exist of empty elements not in the end of array
            elements[i]=elements[lastNotEmpty()];
            elements[lastNotEmpty()]= null;
            return true;
        }
    }
    return false;
}
```

```
/**
 * Method that adds all the elements from the given collection
 * @return true
 */
@Override
public boolean addAll(Collection<? extends Vegetable> c){
    MyIterator iterator = (MyIterator)c.iterator();
    while (iterator.hasNext()) {
        for (int i = 0; i < elements.length; i++) {
            if (elements[i] == null) {
                add((Vegetable)iterator.next());
            }
        }
        increaseSize();
    }
    return true;
}

/**
 * Method that removes all the elements not belonging
 * to given collection
 * @return true if any elements were removed, else false
 */
@Override
public boolean retainAll(Collection<?> c){
    boolean flag = false;
        for (Object i: elements) {
            if (!c.contains(i)){
                flag |= remove(i);
            }
    }
    return flag;
}
```

```
/**
 * Method that removes all the elements belonging
 * to given collection
 * @return true if any elements were removed, else false
 */
@Override
public boolean removeAll(Collection<?> c){
    boolean flag = false;
    for (Object o:c){
        flag |= remove(o);
    }
    return flag;
}

/**
 * Method that returns iterator of the collection
 * @return iterator of collection
 */
@Override
public Iterator<Vegetable> iterator(){
    return new MyIterator();
}

/**
 * Defining an iterator to the collection
 */
private class MyIterator implements Iterator {
    private int index = 0;

    @Override
    public boolean hasNext() {
        return index<lastNotEmpty();
    }

    @Override
```

```
        public Object next() {
            while (hasNext()){
                if(elements[index] == null) {
                    index++;
                }
                else{
                    return elements[index++];
                }
            }
            return null;
        }
    }
}
```

3. UML-діаграма класів