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 №6

З дисципліни «Об’єктно-орієнтоване програмування»

Тема: «Робота з колекціями 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

$C_2 = 1$

$C_3 = 2$

C_2	Інтерфейс
0	List
1	Set

C_3	Внутрішня структуру колекції
0	Масив із початковою кількістю елементів 10 та збільшенням кількості елементів на 50%
1	Список
2	Масив із початковою кількістю елементів 15 та збільшенням кількості елементів на 30%

2. Код програми

Файл Main.java:

```
public class Main {
    public static void main(String[] args) {
        Vegetable tomato = new Vegetable("Помідор", 18);
        Vegetable cucumber = new Vegetable("Огірок", 16);
        Vegetable onion = new Vegetable("Цибуля", 40);
        Vegetable cabbage = new Vegetable("Капуста", 25);
        Vegetable carrot = new Vegetable("Морква", 41);
        Vegetable potato = new Vegetable("Картопля", 77);
        Vegetable Renji = new Vegetable("Ренджі", 0);
        Vegetable radish = new Vegetable("Редька", 15);
        Vegetable selera = new Vegetable("Селера", 16);
        VegetablesCollection<Vegetable> salad = new VegetablesCollection<>();
        VegetablesCollection<Vegetable> salad2 = new VegetablesCollection<>(onion);
        salad2.add(radish);
        salad2.add(selera);
        salad2.add(potato);
        salad.add(tomato);
        salad.add(cucumber);
        salad.add(cabbage);
        salad.add(carrot);
        salad.add(onion);
        salad.add(potato);
        salad.add(potato);
        salad.add(potato);
```

```
        salad.add(potato);
        salad.add(potato);
        salad.add(potato);
        salad.remove(tomato);
        salad.add(Renji);
        VegetablesCollection<Vegetable> salad3 = new VegetablesCollection<>(salad2);
        salad3.remove(radish);
        salad3.remove(selera);
        System.out.println("Салат");
        for (Vegetable i: salad) {
            i.print();
        }
        System.out.println("Салат 2");
        for (Vegetable i: salad2) {
            i.print();
        }
        System.out.println("Салат 3");
        for (Vegetable i: salad3) {
            i.print();
        }
        System.out.println("Чи входять всі елементи salad3 в salad2: " + salad2.containsAll(salad3));
        System.out.println("Чи входять всі елементи salad3 в salad: " + salad.containsAll(salad3));
        salad.removeAll(salad2);
        System.out.println("З салату видалено всі інгредієнти салату 2");
        for (Vegetable i: salad) {
            i.print();
        }
        salad.addAll(salad3);

        System.out.println("До салату додано всі інгредієнти салату 3");
        for (Vegetable i: salad) {
            i.print();
        }
        salad.retainAll(salad3);
        System.out.println("З салату видалено все, що не входить у салат 3");
        for (Vegetable i: salad) {
            i.print();
        }
        System.out.printf("Розміри салатів:\n1-ший: %5d\n2-гий: %5d\n3-тій:
%5d\n",salad.size(),salad2.size(),salad3.size());
    }
}
```

Файл Vegetable.java:

```
public class Salad {
   Vegetable[] salad_elements;
   public Salad(Vegetable... vegetable) {
      salad_elements = vegetable;
   }
   public void print() {
      System.out.println("Салат: ");
      System.out.printf("%s %41s", "Овоч:", "Калорійність овоча:\n");
      for (Vegetable i: salad_elements) {
         i.print();
      }
   }
}
```

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) {
      try {
         this.name = name;
         this.nutrition = calories_value;
      }
      catch (Exception exc) {
         System.out.println("Ви ввели неправильні дані для описання овоча");
      }
   }
   /**
    * Method to print attributes of the vegetable
    * @return void
```

```
     */
    public void print() {
        System.out.printf("%10s; %20d кал\n", name, nutrition);
    }
}
```

Файл VegetablesCollection.java:

```
/*
 * @(#)VegetablesCollection.java 1.0 5/05/2018
 *
 * 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;
```

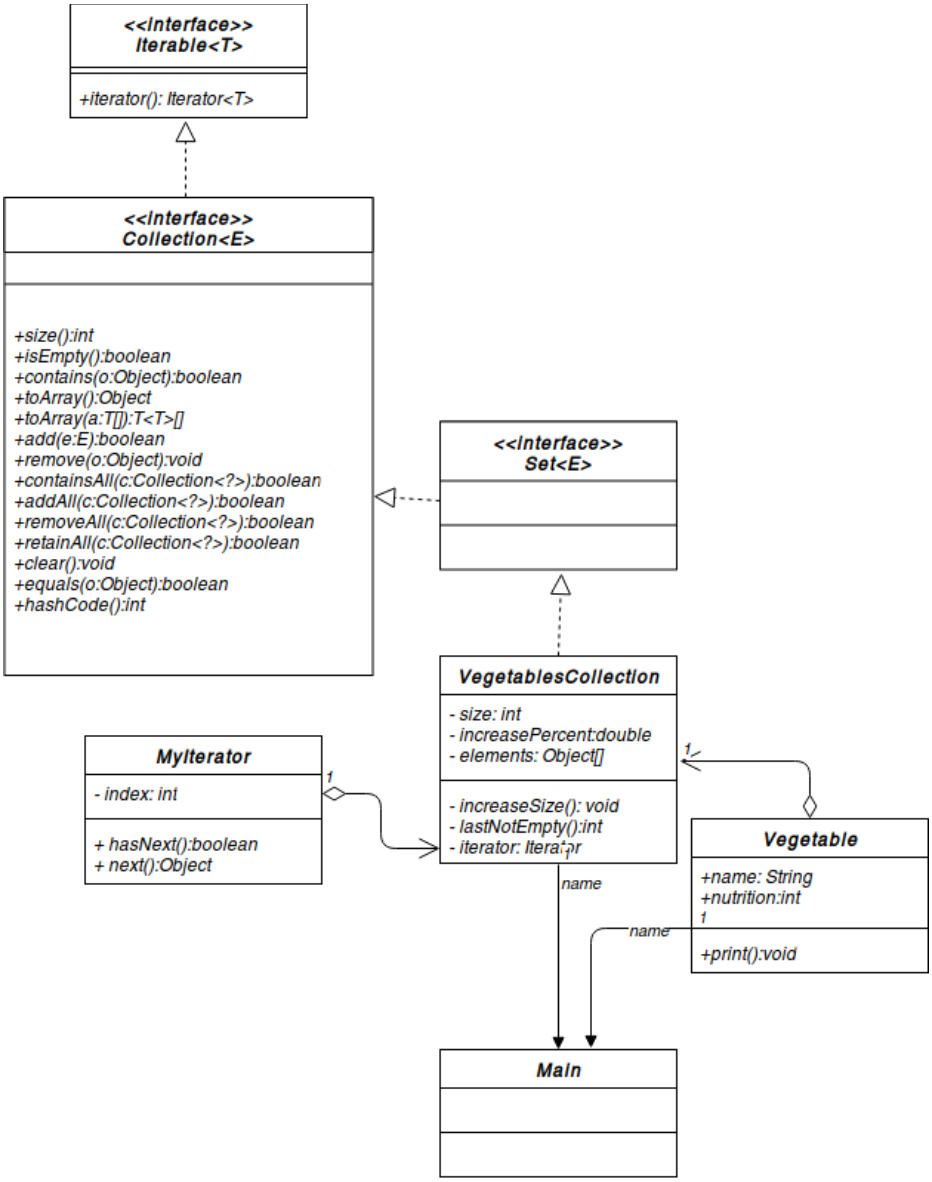
```
            return true;
        }
      }
   }
   return false;
}
/**
 * Private method that returns last not null element
 * of data structure of collection
 * @return int index of last not null element
 */
private int lastNotEmpty(){
   int i = elements.length-1;
   while (elements[i]==null) {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



4. Висновок

Під час виконання даної лабораторної роботи в мене виникли деякі складнощі з тим, які методи мені необнідно визначати та яку структуру даних використовувати доцільніше. На жаль, довелось використовувати масив, хоча хеш-таблиця була би зручнішою для використання для Set.