НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №8**

з дисципліни **«**Основи технологій програмування**»**

Варіант: «6412»

Виконав:

студент 2 курсу

група ІП-64

Лейзьо Сергій Іванович

Перевірив:

Подрубайло Олександр Олександрович

Київ – 2018

**Завдання**

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

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

**1.** Створити клас, що складається з методів збереження до файлу та зчитування з файлу колекції (лабораторна робота No6), як єдиного об’єкту; як послідовності

об’єктів узагальненого класу (лабораторна робота No5); 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.

**Програмний код**

**Cucumber**

package com.company;  
  
public class Cucumber extends Vegetable{  
 public Cucumber(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность огурца: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес огурца: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
 }

**Feta**

package com.company;  
  
public class Feta extends Vegetable{  
 public Feta(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность феты: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес феты: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**Main**

package com.company;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 /\*  
 Salad salad=new Salad();  
 salad.Salad();  
 System.out.println("\nKaлорийность салата: " + salad.CalSalad());  
 System.out.println("Как вы хотите отсортировать овощи? 1-по калориям 2-по массе");  
 Scanner sort=new Scanner(System.in);  
 int number=sort.nextInt();  
 salad.Sort(number);  
 System.out.println("Введите диапазон калорийности от a до b:");  
 System.out.print("a:");  
 Scanner from=new Scanner(System.in);  
 int a=from.nextInt();  
 System.out.print("b:");  
 Scanner to=new Scanner(System.in);  
 int b=to.nextInt();  
 salad.Search(a,b);  
  
 VegetableList vegetableList=new VegetableList();  
 Vegetable a=new Cucumber(1,1,"a");  
 Vegetable b=new Cucumber(2,2,"b");  
 Vegetable c=new Cucumber(3,3,"c");  
 Vegetable d=new Cucumber(3,3,"c");  
 vegetableList.add(a);  
 vegetableList.add(b);  
 vegetableList.add(c);  
 //System.out.println(vegetableList.get(0));  
 // System.out.println(vegetableList.size());  
 // System.out.println(vegetableList.isEmpty());  
 // System.out.println(vegetableList.contains(d));  
 // Vegetable list2[] = new Vegetable[vegetableList.size()];  
 //list2 = vegetableList.toArray(list2);  
 //for(Vegetable number : list2){  
 // System.out.println("Number = " + number);  
 // }  
 //vegetableList.remove(1);  
 //System.out.println(vegetableList.get(1));  
 ArrayList<Vegetable> deliveryCities = new ArrayList<>();  
  
 deliveryCities.add(d);  
 deliveryCities.add(c);  
 vegetableList.addAll(deliveryCities);  
 System.out.println(vegetableList.get(4));  
 \*/  
 }  
}

**Onion**

package com.company;  
  
public class Onion extends Vegetable{  
 public Onion(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность лука: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес лука: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**RedPepper**

package com.company;  
  
public class RedPepper extends Vegetable{  
 public RedPepper(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность красного перца: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес красного перца: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**Salad**

package com.company;  
  
import java.util.Arrays;  
import java.util.Comparator;  
  
public class Salad{  
 Vegetable param[]={new Cucumber(16,100,"Cucumber"),  
 new RedPepper(20,110,"RedPepper"),  
 new Tomato(180,101,"Tomato"),  
 new Feta(264,200,"Feta"),  
 new Onion(35,90,"Onion")};  
 public void Salad(){  
 for(int i=0;i<param.length;i++){  
 this.param[i].Weight();  
 this.param[i].Caloricity();  
 }  
 }  
 public double CalSalad(){  
 double cal=0;  
 for(int i=0;i<param.length;i++){  
 cal=cal+param[i].caloricity;  
 }  
 return cal;  
 }  
 public void Sort(int number)  
 {  
 switch (number){  
 case 1: Arrays.*sort*(param, Comparator.*comparingDouble*(o -> o.getCaloricity()));  
 System.*out*.println("Отсортировано по калориям: ");  
 for(int i=0;i<param.length;i++){  
 System.*out*.print(param[i].getCaloricity()+" ");  
 System.*out*.println(param[i].toString());  
 }  
 break;  
 case 2: Arrays.*sort*(param, Comparator.*comparingDouble*(o -> o.getWeight()));  
 System.*out*.println("Отсортировано по весу: ");  
 for(int i=0;i<param.length;i++){  
 System.*out*.print(param[i].getWeight()+" ");  
 System.*out*.println(param[i].toString());  
 }  
 break;  
 default: System.*out*.println("Вы ввели не то число!");  
 }  
 }  
 public void Search(int a,int b){  
 System.*out*.println("В этот диапазон входят: ");  
 for(int i=0;i<param.length;i++){  
 if((param[i].getCaloricity()>=a)&&(param[i].getCaloricity()<=b)){  
 System.*out*.println(param[i].toString());  
 }  
 }  
 }  
}

**Tomato**

package com.company;  
  
public class Tomato extends Vegetable{  
 public Tomato(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность помидора: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес помидора: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**Vegetable**

package com.company;  
  
import java.io.Serializable;  
import java.util.Objects;  
  
abstract class Vegetable implements Serializable{  
 protected double caloricity;  
 protected double weight;  
 protected String vegetables;  
 abstract public void Caloricity();  
 abstract public void Weight();  
 abstract public double getWeight();  
 abstract public double getCaloricity();  
 abstract public String toString();  
  
 public Vegetable(double caloricity, double weight, String vegetables) {  
 this.caloricity = caloricity;  
 this.weight = weight;  
 this.vegetables = vegetables;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Vegetable)) return false;  
 Vegetable vegetable = (Vegetable) o;  
 return getClass().equals(o.getClass())  
 && vegetable.weight== weight  
 && vegetable.caloricity == caloricity  
 && Objects.*equals*(vegetables, vegetable.vegetables);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(weight, caloricity, vegetables);  
 }  
}

**VegetableList**

package com.company;  
  
import java.io.Serializable;  
import java.util.Collection;  
import java.util.Iterator;  
import java.util.List;  
import java.lang.reflect.Array;  
import java.util.ListIterator;  
  
public class VegetableList implements List<Vegetable>, Serializable {  
 private int size = 0; //размер List  
  
 private ListNode head;  
 private ListNode tail;  
  
 //Пустой конструктор  
 public VegetableList() {  
 }  
  
 public VegetableList(Vegetable vegetable) {  
 add(vegetable);  
 }  
  
 public VegetableList(Collection<? extends Vegetable> c) {  
 addAll(c);  
 }  
  
 //Класс для хранения узлов  
 private class ListNode implements Serializable {  
  
 Vegetable vegetable;  
 ListNode next;  
 ListNode prev;  
  
 ListNode(Vegetable vegetable) {  
 this.vegetable = vegetable;  
 }  
  
 ListNode(Vegetable vegetable, ListNode next, ListNode prev) {  
 this.vegetable = vegetable;  
 this.next = next;  
 this.prev = prev;  
 }  
 }  
  
 //Возвразает размер VegetableList  
 @Override  
 public int size() {  
 return size;  
 }  
  
 //Возвращает пуста ли коллекция  
 @Override  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 //Возвращает true, если этот список содержит указанный элемент  
 @Override  
 public boolean contains(Object o) {  
 if (head == null) {  
 return false;  
 }  
 for (ListNode data = head; data != null; data = data.next) {  
 if (o.equals(data.vegetable)) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 @Override  
 public Iterator<Vegetable> iterator() {  
 return new Iterator<Vegetable>() {  
 private int position = 1;  
  
 @Override  
 public boolean hasNext() {  
 if (position <= size()) {  
 return true;  
 }  
 return false;  
 }  
  
 @Override  
 public Vegetable next() {  
 Vegetable data = get(position);  
 position++;  
 return data;  
 }  
  
 @Override  
 public void remove() {  
 VegetableList.this.remove(position);  
 }  
 };  
 }  
  
 @Override  
 public Object[] toArray() {  
 Object[] result = new Object[size];  
 int number = 0;  
 for (ListNode data = head; data != null; data = data.next)  
 result[number++] = data.vegetable;  
 return result;  
 }  
  
 //Возвращает массив, содержащий все элементы в этом списке  
 @SuppressWarnings("unchecked")  
 @Override  
 public <T> T[] toArray(T[] array) {  
 if (array.length < size)  
  
 array = (T[]) Array.*newInstance*(  
 array.getClass().getComponentType(), size);  
 int number = 0;  
 Object[] result = array;  
 for (ListNode data = head; data != null; data = data.next)  
 result[number++] = data.vegetable;  
  
 if (array.length > size) {  
 array[size] = null;  
 }  
 return array;  
 }  
  
 //Добавляет элемент в список  
 @Override  
 public boolean add(Vegetable vegetable) {  
 //Добавление в конец списка  
 ListNode newNode = new ListNode(vegetable);  
 if (head == null) {  
 head = newNode;  
 tail = newNode;  
 } else {  
 newNode.prev = tail;  
 tail.next = newNode;  
 tail = newNode;  
 }  
 size++;  
 /\*  
 //Добавление в начало списка  
 ListNode newNode = new ListNode(vegetable);  
 if(head == null){  
 head = newNode;  
 tail = newNode;  
 }  
 else{  
 newNode.next = head;  
 head.prev = newNode;  
 head = newNode;  
 }  
 size++;\*/  
 return true;  
 }  
  
 //Удаляет элемент, возвращая его содержимое  
 @Override  
 public boolean remove(Object o) {  
 if (head == null) {  
 return false;  
 }  
 for (ListNode data = head; data != null; data = data.next) {  
 if (o.equals(data.vegetable)) {  
 unlink(data);  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private void unlink(ListNode data) {  
 final ListNode next = data.next;  
 final ListNode prev = data.prev;  
  
 if (prev == null) {  
 head = next;  
 } else {  
 prev.next = next;  
 data.prev = null;  
 }  
  
 if (next == null) {  
 tail = prev;  
 } else {  
 next.prev = prev;  
 data.next = null;  
 }  
 data.vegetable = null;  
 size--;  
 }  
  
 //Возвращает, содержатся ли все элементы c в коллекции  
 @Override  
 public boolean containsAll(Collection<?> c) {  
 for (Object o : c)  
 if (contains(o)) {  
 return true;  
 }  
 return false;  
 }  
  
 //Добавляет все элементы в указанной коллекции до конца  
 @Override  
 public boolean addAll(Collection<? extends Vegetable> c) {  
 for (Vegetable vegetable : c) {  
 add(vegetable);  
 }  
 return true;  
 }  
  
 //Добавляет все элементы в указанной коллекции до конца начиная с index  
 @Override  
 public boolean addAll(int index, Collection<? extends Vegetable> c) {  
 if (index == size) {  
 addAll(c);  
 return true;  
 }  
 if (c.isEmpty()) {  
 return false;  
 }  
 VegetableList listAdd = new VegetableList(c);  
 final ListNode newHead = listAdd.head;  
 final ListNode newTail = listAdd.tail;  
 if (index == 0) {  
 newTail.next = head;  
 newHead.prev = head.prev;  
 head.prev = newTail;  
 head = newHead;  
 } else {  
 ListNode temp = getElement(index);  
 final ListNode prev = temp.prev;  
  
 prev.next = newHead;  
 newHead.prev = prev;  
  
 temp.prev = newTail;  
 newTail.next = temp;  
 }  
 size += c.size();  
 return true;  
 }  
  
 //Узнаём после какого элемента нам нужно вставлять коллекцию  
 private ListNode getElement(int index) {  
 ListNode needEl=head;  
 for (int i=1;i<index;i++){  
 needEl=needEl.next;  
 }  
 return needEl;  
 }  
  
 //Удаляет все элементы из коллекции  
 @Override  
 public boolean removeAll(Collection<?> c) {  
 boolean logicValue = false;  
 for (Object o : c) {  
 logicValue = logicValue || remove(o);  
 }  
 return logicValue;  
 }  
 //Удаление элементов из списка, которые не содержатся в указанной коллекции.  
 @Override  
 public boolean retainAll(Collection<?> c) {  
 if(c == null)  
 {  
 throw new NullPointerException("Сollection is null");  
 }  
 Iterator itr = iterator(); //перебираем список  
 boolean found = false;  
 while(itr.hasNext())  
 {  
 if(!c.contains(itr.next()))  
 {  
 itr.remove();  
 found = true;  
 }  
 }  
 return found;  
 }  
 //Обнуляет список  
 @Override  
 public void clear() {  
 head=tail=null;  
 size=0;  
 }  
 @Override  
 public Vegetable get(int index) {  
 return getElement(index).vegetable;  
 }  
 //Устанавливает новый овощ по индексу, возвращая старый  
 @Override  
 public Vegetable set(int index, Vegetable element) {  
 ListNode data = getElement(index);  
 Vegetable oldVegetable = data.vegetable;  
 data.vegetable = element;  
 return oldVegetable;  
 }  
 //Добавляет элемент по индексу  
 @Override  
 public void add(int index, Vegetable element) {  
 if (index == size) {  
 add(element);  
 }  
 else {  
 ListNode data = getElement(index);  
 final ListNode prev = data.prev;  
 final ListNode newNode = new ListNode(element,data,prev);  
 data.prev = newNode;  
 if (prev == null)  
 head = newNode;  
 else  
 prev.next = newNode;  
 size++;  
 }  
 }  
 //Удаляет элемент по индексу и возвращает его  
 @Override  
 public Vegetable remove(int index) {  
 ListNode removeEll=head;  
 for(int i=0;i<index;i++){  
 removeEll=removeEll.next;  
 }  
 Vegetable v=removeEll.vegetable;//Определяем овощ  
 unlink(removeEll);  
 return v;  
 }  
 //Возвращает индекс первого вхождения указанного элемента  
 //в этом списке или -1, если этот список не содержит элемент.  
 @Override  
 public int indexOf(Object o) {  
 int index = 0;  
 for (ListNode data = head; data != null; data = data.next) {  
 if (o.equals(data.vegetable)) {  
 return index;  
 }  
 index++;  
 }  
 return -1;  
 }  
 //Возвращает индекс последнего вхождения указанного элемента  
 //в этом списке или -1, если этот список не содержит элемент.  
 @Override  
 public int lastIndexOf(Object o) {  
 int index = size-1;  
 for (ListNode data = tail; data != null; data = data.prev) {  
 if (o.equals(data.vegetable)) {  
 return index;  
 }  
 index--;  
 }  
 return -1;  
 }  
 @Override  
 public ListIterator<Vegetable> listIterator() {  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public ListIterator<Vegetable> listIterator(int index) {  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public List<Vegetable> subList(int fromIndex, int toIndex) {  
 throw new UnsupportedOperationException();  
 }  
}

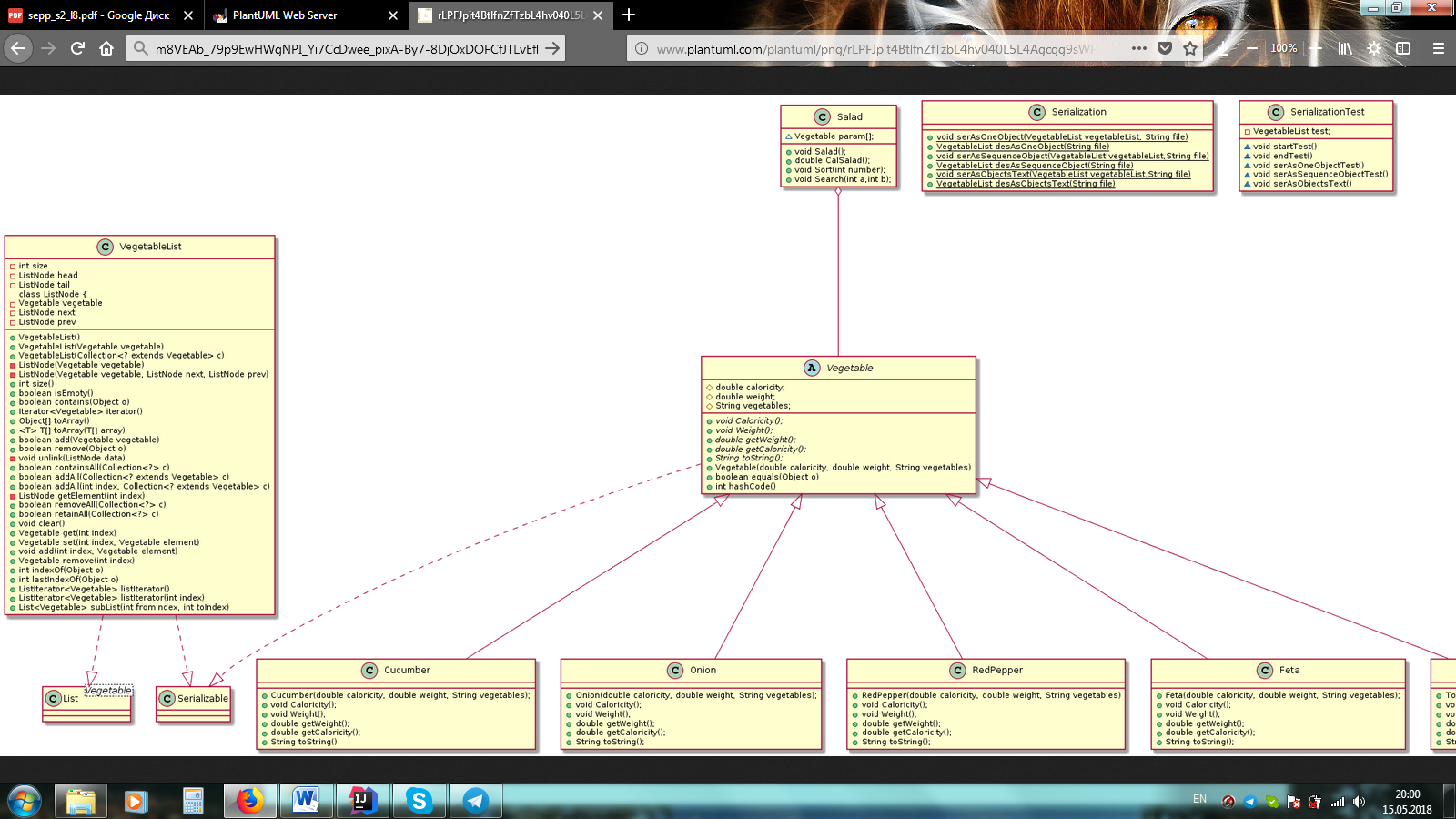
**Serialization**

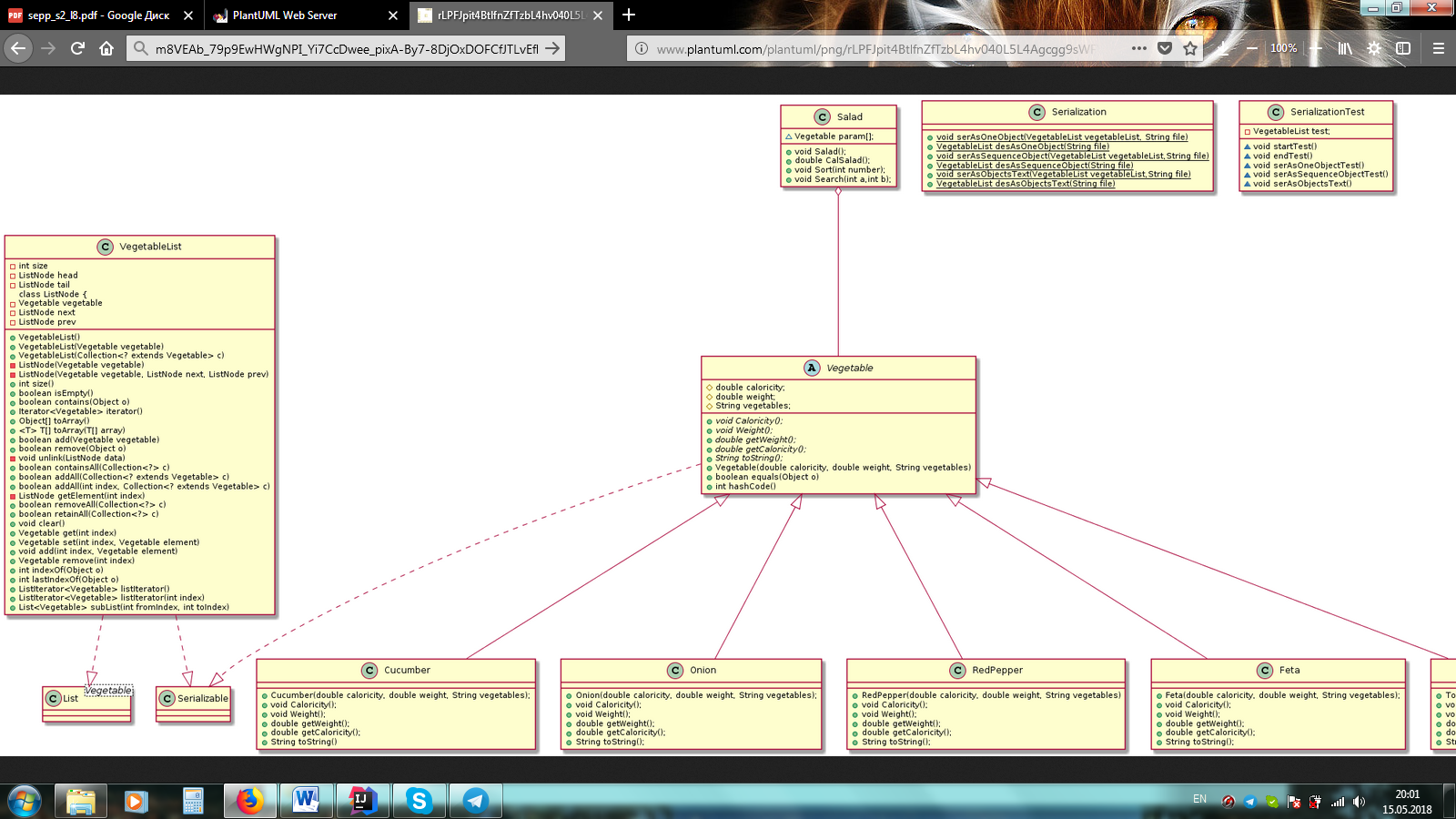
package com.company;  
  
import java.io.\*;  
import java.util.Scanner;  
  
public class Serialization {  
  
 //Сохранение в файл и считывание с файла Коллекции, как единого объекта  
 public static void serAsOneObject(VegetableList vegetableList, String file) throws IOException {  
 try (var oos = new ObjectOutputStream(new FileOutputStream(file))) {  
 oos.writeObject(vegetableList);  
 oos.close();  
 }  
 }  
 public static VegetableList desAsOneObject(String file) throws IOException, ClassNotFoundException{  
 try (var ois = new ObjectInputStream(new FileInputStream(file))) {  
 return (VegetableList) ois.readObject();  
 }  
 }  
  
 //Сохранение в файл и считывание, как последовательность объектов обобщенного класса  
 public static void serAsSequenceObject(VegetableList vegetableList,String file) throws IOException {  
 try (var ous = new ObjectOutputStream(new FileOutputStream(file))) {  
 ous.writeInt(vegetableList.size());  
 for(var vegetable:vegetableList) {  
 ous.writeObject(vegetable);  
 }  
 ous.close();  
 }  
  
 }  
 public static VegetableList desAsSequenceObject(String file) throws IOException, ClassNotFoundException {  
 try (var ois = new ObjectInputStream(new FileInputStream(file))) {  
 var sizeOfObjects=ois.readInt();  
 var vegetableList=new VegetableList();  
 for(var i = 0; i<sizeOfObjects; i++){  
 vegetableList.add((Vegetable) ois.readObject());  
 }  
 ois.close();  
 return vegetableList;  
 }  
 }  
  
 //Сохранение в файл и считывание, как последовательность объектов в виде текста  
 public static void serAsObjectsText(VegetableList vegetableList,String file) throws IOException {  
 try(var pw = new PrintWriter(new FileOutputStream(file)))  
 {  
 pw.println("Size of VegetableList : "+"["+vegetableList.size()+"]");  
 for (var vegetable:vegetableList){  
 pw.println("Class name : "+"["+vegetable.getClass().getSimpleName()+"]");  
 pw.println(" caloricity: "+"["+vegetable.getCaloricity()+"]");  
 pw.println(" weight: "+"["+vegetable.getWeight()+"]");  
 pw.println(" vegetables: "+"["+vegetable.toString()+"]");  
 }  
 pw.close();  
 }  
 }  
 public static VegetableList desAsObjectsText(String file) throws IOException,ClassNotFoundException{  
 try(var sc = new Scanner(new File(file))){  
 var vegetableList=new VegetableList();  
  
 var sSize = sc.nextLine();  
 sSize = sSize.substring(sSize.indexOf('[')+1,sSize.indexOf(']'));  
 int size=Integer.*valueOf*(sSize);  
  
 for (var i = 0; i<size; i++) {  
  
 var className = sc.nextLine();  
 className = className.substring(className.indexOf('[')+1,className.indexOf(']'));  
  
 var sCaloricity = sc.nextLine();  
 sCaloricity = sCaloricity.substring(sCaloricity.indexOf('[')+1,sCaloricity.indexOf(']'));  
 double calorisity=Double.*valueOf*(sCaloricity);  
  
 var sWeight = sc.nextLine();  
 sWeight= sWeight.substring(sWeight.indexOf('[')+1,sWeight.indexOf(']'));  
 double weight=Double.*valueOf*(sWeight);  
  
 var toString = sc.nextLine();  
 toString = toString.substring(toString.indexOf('[')+1,toString.indexOf(']'));  
  
 if (className.equals(Cucumber.class.getSimpleName())) {  
 vegetableList.add(new Cucumber(calorisity,weight,toString));  
 }  
 else if(className.equals(RedPepper.class.getSimpleName())) {  
 vegetableList.add(new RedPepper(calorisity,weight,toString));  
 }  
 else if(className.equals(Tomato.class.getSimpleName())) {  
 vegetableList.add(new Tomato(calorisity,weight,toString));  
 }  
 else if (className.equals(Feta.class.getSimpleName())) {  
 vegetableList.add(new Feta(calorisity,weight,toString));  
 }  
 else if (className.equals(Onion.class.getSimpleName())) {  
 vegetableList.add(new Onion(calorisity,weight,toString));  
 }  
 else throw new ClassNotFoundException(className);  
 }  
 sc.close();  
 return vegetableList;  
 }  
 }  
}

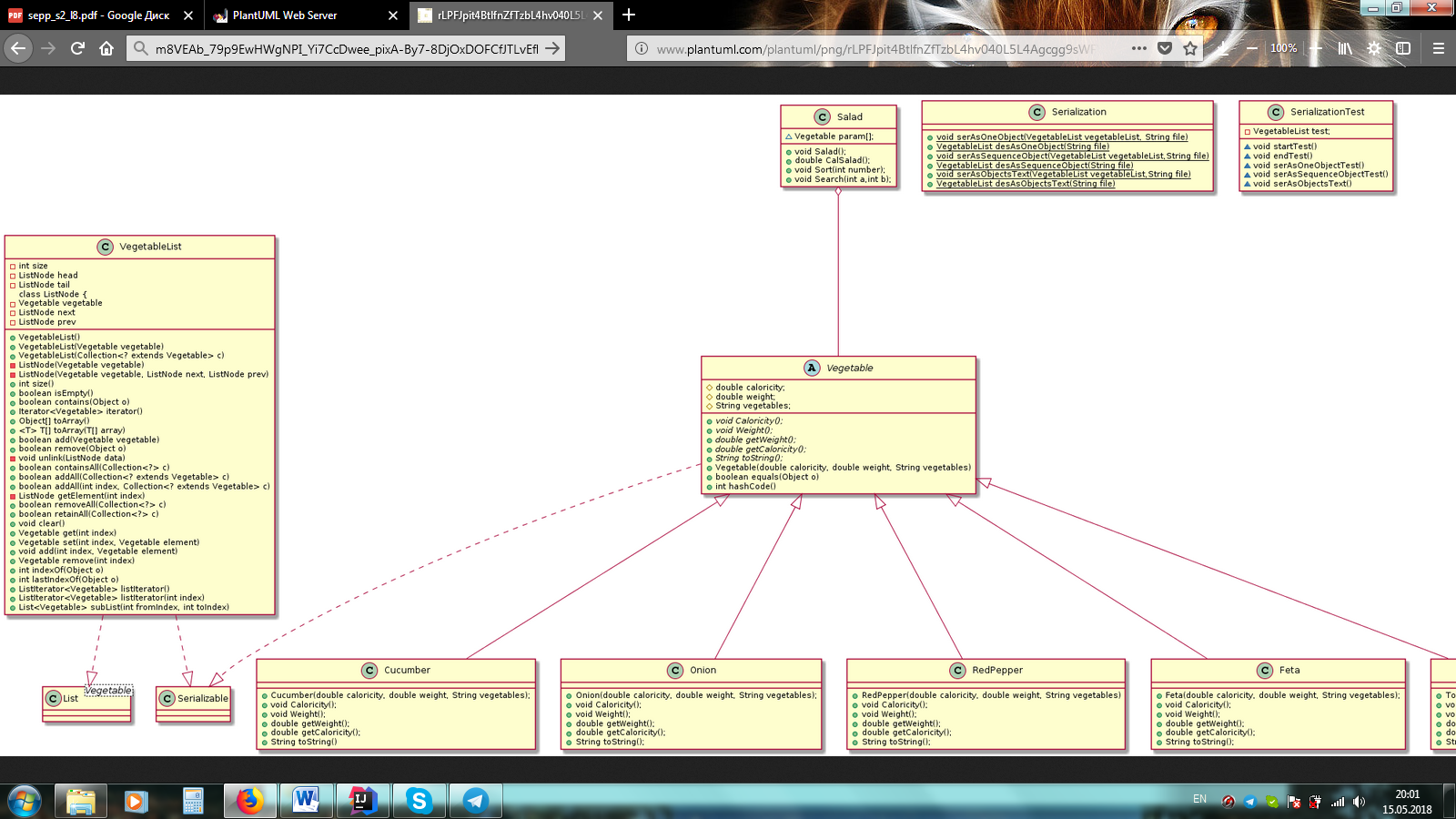
**SerializationTest**

package com.company;  
  
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import java.io.IOException;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class SerializationTest {  
  
 private VegetableList test;  
  
 @BeforeEach  
 void startTest() {  
 test = new VegetableList();  
 Vegetable vegetables[] = {new Cucumber(16,100,"Cucumber"),  
 new RedPepper(20,110,"RedPepper"),  
 new Tomato(180,101,"Tomato"),  
 new Feta(264,200,"Feta"),  
 new Onion(35,90,"Onion")};  
 for (var vegetable : vegetables) {  
 test.add(vegetable);  
 }  
 }  
  
 @AfterEach  
 void endTest() {  
 test.clear();  
 }  
  
 @Test  
 void serAsOneObjectTest() {  
 try {  
 Serialization.*serAsOneObject*(test,"serAsOneObject.bin");  
 var checkupSAOO = Serialization.*desAsOneObject*("serAsOneObject.bin");  
 var arrTest = test.toArray();  
 var arrCheckupTest = checkupSAOO.toArray();  
 for (var i = 0; i < arrTest.length; i++) {  
 *assertEquals*(arrTest[i], arrCheckupTest[i]);  
 }  
 *assertEquals*(test.size(), test.size());  
 } catch (IOException | ClassNotFoundException ex) {  
 System.*out*.println("Ошибка при сериализации объекта");  
 ex.printStackTrace();  
 }  
 }  
  
 @Test  
 void serAsSequenceObjectTest() {  
 try {  
 Serialization.*serAsSequenceObject*(test,"serAsSequenceObject.bin");  
 var checkupSASO = Serialization.*desAsSequenceObject*("serAsSequenceObject.bin");  
 *assertEquals*(test.size(), checkupSASO.size());  
 var arrTest = test.toArray();  
 var arrCheckupTest = checkupSASO.toArray();  
 for (var i = 0; i < arrCheckupTest.length; i++) {  
 *assertEquals*(arrTest[i], arrCheckupTest[i]);  
 }  
 } catch (IOException | ClassNotFoundException ex) {  
 System.*out*.println("Ошибка при сериализации объекта");  
 ex.printStackTrace();  
 }  
 }  
  
 @Test  
 void serAsObjectsText() {  
 try {  
 Serialization.*serAsObjectsText*(test,"serAsObjectsText.bin");  
 var checkupSAOT = Serialization.*desAsObjectsText*("serAsObjectsText.bin");  
 var arrTest = test.toArray();  
 var arrCheckupTest = checkupSAOT.toArray();  
 for (var i = 0; i < arrTest.length; i++) {  
 *assertEquals*(arrTest[i], arrCheckupTest[i]);  
 }  
 *assertEquals*(test.size(), test.size());  
 } catch (IOException |ClassNotFoundException ex) {  
 System.*out*.println("Ошибка при сериализации объекта");  
 ex.printStackTrace();  
 }  
 }  
}

**Дiаграма класiв**







**Висновок**

У готовій лабораторній роботі №8 створено 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. Написано JUnit-тести для перевірки працездатності усіх методів та виключних ситуацій класу Serialization. Код відповідає стандартам JCC.