МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ 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»

на тему: *«Робота з колекціями в мові програмування Java.»*

Виконав:

студент гр.ІО-31

Долинний О.В.

Перевірив:

Невдащенко М.В.

Київ 2014

**Лабораторна робота №7**

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

**Мета:** Здобуття навичок у створенні власних та використанні стандартних колекцій в мові програмування Java.

1. ***Варіант завдання***

Номер залікової книжки  *3110*

Інтерфейс: *C2 = 3110 mod 2 = 0* List

Внутрішня структуру колекції *C3 = 3110 mod 3 = 2* Двозв’язний список

Створити клас, що описує типізовану колекцію (типом колекції є узагальнений клас з лабораторної роботи №6), що реалізує заданий варіантом інтерфейс та має задану внутрішню структуру. Реалізувати всі методи інтерфейсу, а також створити не менше ніж 3 конструктори (1 – порожній, 2 – в який передається 1 об’єкт узагальненого класу, 3 – в який передається стандартна колекція об’єктів). Всі початкові дані задаються у виконавчому методі. Код повинен відповідати стандартам JCC та бути детально задокументований.

1. ***Опис алгоритму програми***

Програма реалізує колекцію із внутрішньою структурою двозв’язний список. Клас Vegetables – узагальнений клас із лабораторної роботи №6, у класі ListElement описується елемент списку, в якому міститься об’єкт класу Vegetables та посилання на наступний та попередній елементи списку. Клас VegList реалізує методи інтерфейсу List.

1. ***Програмний код***

Class VegList

/\*

\* @(#)VegList.java 1.0 14/05/14

\*

\* Copyright (c) 2014 Alex Dolinniy

\*/

**package** laba\_7;

**import** java.util.Collection;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.ListIterator;

**import** laba\_7.ListElement;

/\*\* Class VegList creates collection of vegetables.

\* Class VegList implements interface List

\* with structure of doubly linked list.

\*

\* **@version** 1.0 14 May 2014

\* **@author** Alex Dolinniy

\* **@since** 1.0

\*/

**public** **class** VegList **implements** List<Vegetables> {

/\*\*

\* **@param** size

\* size of collection

\* **@param** first

\* first element of collection

\* **@param** last

\* last element of collection

\*/

**private** **int** size;

**private** ListElement first = **new** ListElement(**null**);

**private** ListElement last = **new** ListElement(**null**);

/\*\*

\* Empty constructor of VegList

\*/

**public** VegList() {

first = **new** ListElement(**null**);

last = **new** ListElement(**null**);

}

/\*\*

\* Constructor which have one parameter of class ListElement

\* **@param** buf object of class ListElement

\*/

**public** VegList(Vegetables buf) {

**this**();

add(buf);

}

/\*\*

\* Constructor with collection as a parameter

\* **@param** coll collection

\*/

**public** VegList(Collection<? **extends** Vegetables> coll) {

**this**();

addAll(coll);

}

/\*\*

\* Method returns size of collection.

\* **@return** size of collection

\*/

@Override

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

**return** size;

}

/\*\*

\* Method adds object of Vegetables to the end of list.

\* **@return** true

\*/

@Override

**public** **boolean** add(Vegetables buf) {

**if**(size == 0) {

first.setData(buf);

last = first;

}

**else** {

last.setNext(**new** ListElement(buf));

last.getNext().setPrevious(last);

last = last.getNext();

}

size++;

**return** **true**;

}

/\*\*

\* Method adds object of Vegetables to the defined place in list.

\*/

@Override

**public** **void** add(**int** index, Vegetables buf) {

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

first.setPrevious(**new** ListElement(buf));

first.getPrevious().setNext(first);

first = first.getPrevious();

}

**else** {

**int** i = 0;

ListElement current = first;

**for**( i= 0; i < index; i++)

current=current.getNext();

current.getPrevious().setNext(**new** ListElement(buf));

current.getPrevious().getNext().setPrevious(current.getPrevious());

current.setPrevious(current.getPrevious().getNext());

current.getPrevious().setNext(current);

}

size++;

}

/\*\*

\* Method adds collection to the defined place in list.

\* **@return** true

\*/

@Override

**public** **boolean** addAll(**int** index, Collection <? **extends** Vegetables> coll) {

**if**(index < 0)

index = 0;

**if** (index >= **this**.size())

**for** (**int** i = 0; i < coll.size(); i++)

add(((VegList) coll).get(i));

**else**

**for** (**int** i = 0; i < coll.size(); i++)

add(index++, ((VegList) coll).get(i));

**return** **true**;

}

/\*\*

\* Method adds collection to the end of list.

\* **@return** true

\*/

@Override

**public** **boolean** addAll(Collection<? **extends** Vegetables> coll) {

**for** (**int** i = 0; i < coll.size(); i++)

add(((VegList) coll).get(i));

**return** **true**;

}

/\*\*

\* Method clears collection.

\*/

@Override

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

last = **new** ListElement(**null**);

first = last;

size = 0;

}

/\*\*

\* Method checks is element present in collection

\* **@return** true if element is present in collection, else false

\*/

@Override

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

**return** indexOf(buf) >= 0;

}

/\*\*

\* Method checks are all elements of transmissible collection

\* present in our collection

\* **@return** true if all elements of transmissible collection

\* are present in our collection, else false.

\*/

@Override

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

**boolean** result = **true**;

**for**(**int** i =0; i < coll.size(); i++)

**if**(!**this**.contains(((VegList)coll).get(i)))

result = **false**;

**return** result;

}

/\*\*

\* Method checks is collection empty

\* **@return** true if collection is empty, else false

\*/

@Override

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

**return** size == 0;

}

/\*\*

\* Method removes element from collection.

\* **@return** true if element was removed from collection,

\* else false

\*/

@Override

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

ListElement current = first;

**boolean** result = **false**;

**if**(current.getData().getInformation().equals(

((Vegetables)buf).getInformation())) {

first = current.getNext();

first.setPrevious(**null**);

result = **true**;

size--;

}

**else** {

**while**(current.getNext() != **null**) {

ListElement next = current.getNext();

**if**(next.getData().getInformation().equals(

((Vegetables)buf).getInformation())) {

current.setNext(next.getNext());

current.getNext().setPrevious(current);

size--;

result = **true**;

}

**else**

current = current.getNext();

}

}

**return** result;

}

/\*\*

\* Method removes element from separate place in collection.

\* **@return** value of removed element

\*/

@Override

**public** Vegetables remove(**int** index) {

Vegetables result = **null**;

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

first.getNext().setPrevious(**null**);

first = first.getNext();

}

**if** ((index != 0)&&(index != **this**.size() - 1)) {

ListElement current = first.getNext();

**for**(**int** i = 1; i < index; i++)

current=current.getNext();

result = current.getData();

current.getPrevious().setNext(current.getNext());

current.getNext().setPrevious(current.getPrevious());

}

**if** ( index == **this**.size() - 1) {

last.getPrevious().setNext(**null**);

last = last.getPrevious();

}

size--;

**return** result;

}

/\*\*

\* Method removes all elements of transmissible collection

\* in our collection.

\* **@return** true

\*/

@Override

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

**boolean** result = **true**;

**for**(**int** i = 0; i < coll.size(); i++)

**if** (**this**.indexOf(((VegList)coll).get(i)) != -1)

**this**.remove(**this**.indexOf(((VegList)coll).get(i)));

**else**

result = **false**;

**return** result; }

/\*\*

\* Method removes all elements of our collection that

\* are not present in transmissible collection.

\* **@return** true if our collection has all elements

\* of transmissible collection, else returns false.

\*/

@Override

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

**for**(**int** i = **this**.size() - 1; i >= 0; i--)

**if** (((VegList)coll).indexOf(**this**.get(i)) == -1)

**this**.remove(i);

**if** (**this**.containsAll(coll))

**return** **true**;

**else**

**return** **false**; }

/\*\*

\* Method returns element on defined place in collection.

\* **@return** value of element

\*/

@Override

**public** Vegetables get(**int** index) {

Vegetables result = **null**;

**int** i = 0;

ListElement current = first;

**for**(i = 0; i < index; i++)

current = current.getNext();

result = current.getData();

**return** result;

}

/\*\*

\* Method sets element on defined place in collection.

\* **@return** value of changed element

\*/

@Override

**public** Vegetables set(**int** index, Vegetables buf) {

Vegetables result = **null**;

**int** i = 0;

ListElement current = first;

**for**(i = 0; i < index; i++)

current = current.getNext();

result = current.getData();

current.setData(buf);

**return** result;

}

/\*\*

\* Method transmits collection to array.

\* **@return** array of vegetables

\*/

@Override

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

Vegetables[] a = **new** Vegetables[size];

**for**(**int** i = 0; i < size; i++)

a[i] = **this**.get(i);

**return** a;

}

/\*\*

\* Method transmits collection to defined array.

\* **@return** array of vegetables

\*/

@Override

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

Vegetables[] a = **new** Vegetables[arr.length];

**for**(**int** i = 0; ((i < size) && (i < arr.length)); i++)

a[i] = (Vegetables) **this**.get(i);

**for**(**int** i = size; i < arr.length; i++)

a[i] = **null**;

**return** a;

}

/\*\*

\* Method returns collection from fromIndex to toIndex.

\* **@return** collection of vegetables from fromIndex to toIndex.

\*/

@Override

**public** VegList subList(**int** fromIndex, **int** toIndex) {

VegList result = **new** VegList();

**for**(**int** i = fromIndex; i < toIndex; i++)

result.add(**this**.get(i));

**return** result;

}

/\*\*

\* Method returns first entry of element.

\* **@return** first entry of element if it is present in collection,

\* else returns false.

\*/

@Override

**public** **int** indexOf(Object buf) {

**int** result = -1;

**for** (**int** i = 0; (i < size) && (result == -1); i++)

**if** (**this**.get(i).getInformation().equals(**new** ListElement(

(Vegetables)buf).getData().getInformation()))

result = i;

**return** result;

}

/\*\*

\* Method returns last entry of element.

\* **@return** last entry of element if it is present in collection,

\* else returns false.

\*/

@Override

**public** **int** lastIndexOf(Object buf) {

**int** result = -1;

**for** (**int** i = size - 1; (i >= 0) && (result == -1); i--)

**if** (**this**.get(i).getInformation().equals(**new** ListElement(

(Vegetables)buf).getData().getInformation()))

result = i;

**return** result;

}

@Override

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

**return** **new** InIterator();

}

@Override

**public** Iterator<Vegetables> iterator(**int** index) {

**return** **new** InIterator(index);

}

/\*\*

\* Method create Iterator to iterate over the collection

\* **@return** required Iterator.

\*/

@Override

**private** **class** InIterator **implements** Iterator<Vegetables> {

**private** **int** currentIndex = 0;

**public** InIterator() {

currentIndex = 0;

}

**public** InIterator(**int** index) {

currentIndex = index;

}

@Override

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

**return** currentIndex < size;

}

@Override

**public** Vegetables next() {

**return** (Vegetables) get(currentIndex++);

}

@Override

**public** **int** nextIndex() {

**return** currentIndex + 1;

}

@Override

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

}

}

@Override

**public** ListIterator<Vegetables> listIterator() {

**return** **new** InnerIterator();

}

@Override

**public** ListIterator<Vegetables> listIterator(**int** index) {

**return** **new** InnerIterator(index);

}

/\*\*

\* Method create ListIterator to iterate over the collection

\* **@return** required ListIterator.

\*/

**private** **class** InnerIterator **implements** ListIterator<Vegetables> {

**private** **int** currentIndex = 0;

**public** InnerIterator() {

currentIndex = 0;

}

**public** InnerIterator(**int** index) {

currentIndex = index;

}

@Override

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

**return** currentIndex < size;

}

@Override

**public** Vegetables next() {

**return** (Vegetables) get(currentIndex++);

}

@Override

**public** **boolean** hasPrevious() {

**return** currentIndex >= 0;

}

@Override

**public** Vegetables previous() {

**return** (Vegetables) get(currentIndex--);

}

@Override

**public** **int** nextIndex() {

**return** currentIndex + 1;

}

@Override

**public** **int** previousIndex() {

**return** currentIndex - 1;

}

@Override

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

}

@Override

**public** **void** set(Vegetables veg) {

VegList.**this**.set(currentIndex, veg);

}

@Override

**public** **void** add(Vegetables veg) {

VegList.**this**.add(currentIndex, veg);

}

}

}

Class ListElement

/\*

\* @(#)ListElement.java 1.0 14/05/14

\*

\* Copyright (c) 2014 Alex Dolinniy

\*/

**package** laba\_7;

**import** laba\_7.Vegetables;

/\*\* Class ListElement creates object vegetable.

\* Class has getters, setters and constructor

\* and method that return information about vegetable.

\*

\* **@version** 1.0 14 May 2014

\* **@author** Alex Dolinniy

\* **@since** 1.0

\*/

**public** **class** ListElement {

/\*\*

\* **@param** previous

\* previous element

\* **@param** next

\* next element

\* **@param** data

\* field of object Vegetables

\*/

**private** ListElement previous;

**private** ListElement next;

**private** Vegetables data;

/\*\*

\* Constructor of ListElement

\*/

**public** ListElement(Vegetables inData){

setData(inData);

setNext(**null**);

setPrevious(**null**);

}

/\*\*

\* **@return** data

\*/

**public** Vegetables getData() {

**return** data;

}

/\*\*

\* **@param** data

\*/

**public** **void** setData(Vegetables inData) {

data = inData;

}

/\*\*

\* **@return** next element

\*/

**public** ListElement getNext() {

**return** next;

}

/\*\*

\* **@param** next

\*/

**public** **void** setNext(ListElement inNext) {

next = inNext;

}

/\*\*

\* **@return** previous element

\*/

**public** ListElement getPrevious() {

**return** previous;

}

/\*\*

\* **@param** previous

\*/

**public** **void** setPrevious(ListElement inPrevious) {

previous = inPrevious;

}

}

Class Vegetables

/\*

\* @(#)Vegetables.java 1.0 14/05/14

\*

\* Copyright (c) 2014 Alex Dolinniy

\*/

**package** laba\_7;

/\*\* Class Vegetables creates object vegetable.

\* Class has getters, setters and constructor

\* and method that return information about vegetable.

\*

\* **@version** 1.0 14 May 2014

\* **@author** Alex Dolinniy

\* **@since** 1.0

\*/

**public** **class** Vegetables {

/\*\*

\* **@param** name

\* name of vegetable

\* **@param** weight

\* weight of vegetables

\* **@param** caloricity

\* caloricity of vegetables

\*/

**private** String name;

**private** **double** weight;

**private** **double** caloricity;

/\*\*

\* Constructor of Vegetable

\*/

**public** Vegetables (String inName, **double** inWeight, **double** inCaloricity) {

setName(inName);

setWeight(inWeight);

setCaloricity(inCaloricity);

}

/\*\*

\* **@return** name

\*/

**public** String getName() {

**return** name;

}

/\*\*

\* **@return** weight

\*/

**public** **double** getWeight() {

**return** weight;

}

/\*\*

\* **@return** caloricity

\*/

**public** **double** getCaloricity() {

**return** caloricity;

}

/\*\*

\* **@param** name

\*/

**public** **void** setName (String inName) {

name = inName;

}

/\*\*

\* **@param** weight

\*/

**public** **void** setWeight (**double** inWeight) {

weight = inWeight;

}

/\*\*

\* **@param** caloricity

\*/

**public** **void** setCaloricity (**double** inCaloricity) {

caloricity = inCaloricity;

}

/\*\*

\* **@return** information about vegetable in salad

\*/

**public** String getInformation() {

**return** "Name:" + name + " Weight:" + String.*valueOf*(weight) +

" Caloricity:" + String.*valueOf*(caloricity);

}

}

***Висновок***

Під час лабораторної роботи я здобув навички у створенні власних колекцій в мові програмування Java та покращив навички документування коду.

Я створив клас VegList що описує типізовану колекцію, внутрішньою структурою якої є двозв’язний список. Клас реалізує інтерфейс List. Я створив 3 конструктори - пустий, з одним параметром типу Vegetables та з параметром - колекцією.

Для реалізації ітераторів я створив внутрішні класи InIterator який реалізовує інтерфейс Iterator та InnerIterator який реалізовує інтерфейс ListIterator.

Код був оформлений і задокументований відповідно до стандартів JCC.