НТУУ «КПИ»

ФИВТ

Кафедра ВТ

Лабораторная работа №7

Выполнил:

студент группы ИВ-34

Мозговой Иван

Проверил:

Невдащенко М. В.

Номер зачетной книжки:

3416

Киев 2014г.

Определение задания:

3416 mod 2 = 0 => интерфейс List;

3416 mod 3 = 2 => Двусвязный список;

Описание программы:

В данной программе используется 8 классов. 5 для хранения информации о тарифах, 1 для описи коллекции, 1 для геттеров и сеттеров, 1 для хранении информации о тарифах. Работа каждого класса описана в коде.

Код программы:

**import** java.util.ArrayList;

/\*\*

\*

\* **@version** 1.1 29 May 2014

\* **@author** Ivan Mozgovyi

\*

\*/

/\*\*

\* Это исполнительный класс.

\*

\*/

**public** **class** Main {

**public** **static** **void** main (String [] args){

TarifList t = **new** TarifList();

TarifList tt = **new** TarifList(**new** UnlimInternetTalkText(0, 0, 0, 0, 0, "egg"));

ArrayList<Atnt> al = **new** ArrayList<Atnt>();

al.add(**new** UnlimInternetTalkText(0, 0, 0, 0, 0, "swag"));

TarifList ttt = **new** TarifList(al);

t.addAll(tt);

t.addAll(ttt);

}

}

/\*\*

\*

\* **@author** Ivan

\* класс для хранения ввода и вывода информации о тарифах

\*

\*/

**public** **class** Atnt {

**private** **int** payPerMonth;

**private** **double** textPrice;

**private** **double** talkPrice;

**private** **int** internetPrice;

**private** **int** customersOnThisTariff;

**private** String name;

**public** Atnt(**int** payPerMonth, **double** textPrice, **double** talkPrice, **int** internetPrice, **int** customersOnThisTariff, String name){

**this**.customersOnThisTariff=customersOnThisTariff;

**this**.internetPrice=internetPrice;

**this**.name=name;

**this**.payPerMonth=payPerMonth;

**this**.talkPrice=talkPrice;

**this**.textPrice=talkPrice;

}

**public** **double** getPriceTalk(){

**return** talkPrice;

}

**public** **void** setPrice(String s) {

name = s;

}

**public** **double** getPriceText(){

**return** textPrice;

}

**public** **double** getPriceInternet(){

**return** internetPrice;

}

**public** **int** getCustomers(){

**return** customersOnThisTariff;

}

**public** **int** getPricePerMonth(){

**return** payPerMonth;

}

**public** String getNameOfTariff(){

**return** name;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (obj **instanceof** Atnt) {

Atnt a = (Atnt) obj;

**if** (a.getCustomers() != getCustomers() || !a.getNameOfTariff().equals(getNameOfTariff()) || a.getPriceTalk() != getPriceTalk() || a.getPriceText() != getPriceText() || a.getPriceInternet() != getPriceInternet() || a.getPricePerMonth() != getPricePerMonth() ) {

**return** **false**;

}

**return** **true**;

} **else** {

**return** **false**;

}

}

}

/\*\*

\*

\* **@author** Ivan

\* класс для оплаты тарифов.

\*

\*/

**public** **class** EverythingPayed **extends** Atnt {

**public** EverythingPayed(**int** payPerMonth, **double** textPrice, **double** talkPrice, **int** internetPrice, **int** customersOnThisTariff, String name){

**super**(payPerMonth, textPrice, talkPrice, internetPrice, customersOnThisTariff, name);

}

}

/\*\*

\*

\*

\* **@author** Ivan

\* класс геттеров и сеттеров

\*

\*/

**public** **class** Node {

**private** Atnt a;

**private** Node next;

**private** Node preview;

**public** Node() {

}

**public** Node(Atnt t) {

a = t;

}

**public** **void** setNext(Node n) {

next = n;

}

**public** Node getNext() {

**return** next;

}

**public** **void** setPreviev(Node n) {

preview = n;

}

**public** Node getPreview() {

**return** preview;

}

**public** **void** setData(Atnt t) {

a = t;

}

**public** Atnt getData() {

**return** a;

}

@Override

**public** **boolean** equals(Object obj) {

**return** getData().equals(((Node) obj).getData());

}

}

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.ListIterator;

/\*\*

\*

\*

\* **@author** Ivan

\* класс который описывает коллекцию

\*

\*/

**public** **class** TarifList **implements** List<Atnt> {

**private** Node firstNode = **null**;

**int** size = 0;

**public** TarifList() {

}

**public** TarifList(Atnt t) {

firstNode = **new** Node(t);

size++;

}

**public** TarifList(Collection<Atnt> c) {

addAll(c);

}

@Override

**public** **boolean** add(Atnt arg0) {

Node tNode = firstNode;

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

tNode = tNode.getNext();

}

tNode.setNext(**new** Node(arg0));

**return** **true**;

}

@Override

**public** **void** add(**int** arg0, Atnt arg1) {

**if** (arg0 == 0) {

firstNode.setPreviev(**new** Node(arg1));

firstNode.getPreview().setNext(firstNode);

firstNode = firstNode.getPreview();

} **else** {

**int** i = 0;

Node tNode = firstNode;

**for** (i = 0; i < arg0; i++)

tNode = tNode.getNext();

tNode.getPreview().setNext(**new** Node(arg1));

tNode.getPreview().getNext().setPreviev(tNode.getPreview());

tNode.setPreviev(tNode.getPreview().getNext());

tNode.getPreview().setNext(tNode);

}

size++;

}

@Override

**public** **boolean** addAll(Collection<? **extends** Atnt> arg0) {

Object[] a = arg0.toArray();

firstNode = **new** Node((Atnt)a[0]);

size++;

Node tNode = firstNode;

**for** (**int** i = 1; i < a.length; i++) {

tNode.setNext(**new** Node((Atnt)a[i]));

Node pNode = tNode;

tNode = tNode.getNext();

tNode.setPreviev(pNode);

size++;

}

**return** **true**;

}

@Override

**public** **boolean** addAll(**int** arg0, Collection<? **extends** Atnt> arg1) {

Object o[] = arg1.toArray();

**if** (arg0 < 0)

arg0 = 0;

**if** (arg0 >= **this**.size())

**for** (**int** i = 0; i < arg1.size(); i++)

add((Atnt) o[i]);

**else**

**for** (**int** i = 0; i < arg1.size(); i++)

add(arg0++, ((TarifList) arg1).get(i));

**return** **true**;

}

@Override

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

firstNode = **null**;

size = 0;

}

@Override

**public** **boolean** contains(Object arg0) {

**return** indexOf(arg0) > -1;

}

@Override

**public** **boolean** containsAll(Collection<?> arg0) {

Object o[] = arg0.toArray();

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

**for** (**int** i = 0; i < arg0.size(); i++)

**if** (!**this**.contains(o[i]))

result = **false**;

**return** result;

}

@Override

**public** Atnt get(**int** arg0) {

Node tNode = firstNode;

**for** (**int** i = 0; i < arg0 - 1; i++) {

tNode = tNode.getNext();

}

**return** tNode.getData();

}

@Override

**public** **int** indexOf(Object arg0) {

Node tNode = firstNode;

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

**if** (arg0.equals(tNode.getData())) {

**return** i;

}

tNode = tNode.getNext();

}

**return** -1;

}

@Override

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

**return** size == 0;

}

@Override

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

**return** **new** Iterator<Atnt>() {

Node e = firstNode;

**int** i = 0;

@Override

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

**return** e.getNext() != **null**;

}

@Override

**public** Atnt next() {

i++;

e = e.getNext();

**return** e.getData();

}

@Override

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

TarifList.**this**.remove(i);

}

};

}

@Override

**public** **int** lastIndexOf(Object arg0) {

**int** result = -1;

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

**if** (arg0.equals(get(i)))

result = i;

**return** result;

}

**private** **class** LI **implements** ListIterator<Atnt> {

**int** index;

**public** LI(**int** i) {

index = i;

}

@Override

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

TarifList.**this**.add(index, arg0);

}

@Override

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

**return** (index < size - 1 && index > -1);

}

@Override

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

**return** (index > 0 && index < size);

}

@Override

**public** Atnt next() {

**return** TarifList.**this**.get(index + 1);

}

@Override

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

**if** (hasNext()) {

**return** index + 1;

} **else** {

**return** -1;

}

}

@Override

**public** Atnt previous() {

**return** TarifList.**this**.get(index - 1);

}

@Override

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

**if** (hasPrevious()) {

**return** index - 1;

} **else** {

**return** -1;

}

}

@Override

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

TarifList.**this**.remove(index);

}

@Override

**public** **void** set(Atnt arg0) {

TarifList.**this**.set(index, arg0);

}

}

@Override

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

**return** **new** LI(0);

}

@Override

**public** ListIterator<Atnt> listIterator(**int** arg0) {

**return** **new** LI(arg0);

}

@Override

**public** **boolean** remove(Object arg0) {

Node tNode = firstNode;

**while** (tNode != **null**) {

**if** (tNode.equals(arg0)) {

**if** (tNode.getPreview() != **null** && tNode.getNext() != **null**) {

tNode.getPreview().setNext(tNode.getNext());

tNode.getNext().setPreviev(tNode.getPreview());

} **else** {

**if** (tNode.getPreview() == **null**) {

firstNode = tNode.getNext();

firstNode.setPreviev(**null**);

} **else** **if** (tNode.getNext() == **null**)

tNode.getPreview().setNext(**null**);

}

}

tNode = tNode.getNext();

}

**return** **false**;

}

@Override

**public** Atnt remove(**int** arg0) {

Atnt result = **null**;

**if** (arg0 == 0) {

firstNode.getNext().setPreviev(**null**);

firstNode = firstNode.getNext();

}

**if** (arg0 != 0) {

Node tNode = firstNode.getNext();

**for** (**int** i = 1; i < arg0; i++)

tNode = tNode.getNext();

result = tNode.getData();

tNode.getPreview().setNext(tNode.getNext());

tNode.getNext().setPreviev(tNode.getPreview());

}

size--;

**return** result;

}

@Override

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

Object[] o = c.toArray();

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

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

**if** (**this**.indexOf(o[i]) != -1)

**this**.remove(**this**.indexOf(o[i]));

**else**

result = **false**;

**return** result;

}

@Override

**public** **boolean** retainAll(Collection<?> arg0) {

Node tNode = firstNode;

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

Node pNode = tNode;

**if** (!arg0.contains(tNode.getData())) {

remove(i);

i--;

}

tNode = pNode.getNext();

}

**return** **true**;

}

@Override

**public** Atnt set(**int** arg0, Atnt arg1) {

Node tNode = firstNode;

**for** (**int** i = 0; i < arg0 - 1; i++) {

tNode = tNode.getNext();

}

Atnt a = tNode.getData();

tNode.setData(arg1);

**return** a;

}

@Override

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

**return** size;

}

@Override

**public** List<Atnt> subList(**int** arg0, **int** arg1) {

TarifList result = **new** TarifList();

**for** (**int** i = arg0; i < arg1; i++)

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

**return** result;

}

@Override

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

Object[] o = **new** Object[size];

Node tNode = firstNode;

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

o[i] = tNode.getData();

tNode = tNode.getNext();

}

**return** o;

}

@SuppressWarnings("unchecked")

@Override

**public** <T> T[] toArray(T[] arg0) {

**if** (arg0.length < size)

**return** (T[]) Arrays.*copyOf*(toArray(), size, arg0.getClass());

System.*arraycopy*(toArray(), 0, arg0, 0, size);

**if** (arg0.length > size)

arg0[size] = **null**;

**return** arg0;

}

}

/\*\*

\*

\* **@author** Ivan

\* класс для хранения тарифа UnlimInternetTalkText

\*

\*/

**public** **class** UnlimInternetTalkText **extends** Atnt{

**public** UnlimInternetTalkText(**int** payPerMonth, **double** textPrice, **double** talkPrice, **int** internetPrice, **int** customersOnThisTariff, String name){

**super**(payPerMonth, textPrice, talkPrice, internetPrice, customersOnThisTariff, name);

}

}

/\*\*

\*

\* **@author** Ivan

\*класс для хранения тарифа unlimTalkPayedTextInternet.

\*

\*/

**public** **class** unlimTalkPayedTextInternet **extends** Atnt{

**public** unlimTalkPayedTextInternet(**int** payPerMonth, **double** textPrice, **double** talkPrice, **int** internetPrice, **int** customersOnThisTariff, String name){

**super**(payPerMonth, textPrice, talkPrice, internetPrice, customersOnThisTariff, name);

}

}

/\*\*

\*

\* **@author** Ivan

\* класс для хранения тарифа UnlimTextPayedTalkInternet.

\*

\*/

**public** **class** UnlimTextPayedTalkInternet **extends** Atnt{

**public** UnlimTextPayedTalkInternet(**int** payPerMonth, **double** textPrice, **double** talkPrice, **int** internetPrice, **int** customersOnThisTariff, String name){

**super**(payPerMonth, textPrice, talkPrice, internetPrice, customersOnThisTariff, name);

}

}

/\*\*

\*

\* **@author** Ivan

\* класс для хранения тарифа UnlimTextTalkPayedInternet.

\*

\*/

**public** **class** UnlimTextTalkPayedInternet **extends** Atnt{

**public** UnlimTextTalkPayedInternet(**int** payPerMonth, **double** textPrice, **double** talkPrice, **int** internetPrice, **int** customersOnThisTariff, String name){

**super**(payPerMonth, textPrice, talkPrice, internetPrice, customersOnThisTariff, name);

}

}

Выводы:

Здесь ссылки в каждом узле указывают на предыдущий и на последующий узел в списке. По двусвязному списку можно передвигаться в любом направлении — как к началу, так и к концу. В этом списке проще производить удаление и перестановку элементов, так как всегда известны адреса тех элементов списка, указатели которых направлены на изменяемый элемент.