Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №6

Виконав:

студент групи ІО-22

Коломієць О.

Перевірив:

Невдащенко М. В.

Номер залікової книжки:

2209

Київ 2013р.

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

**Мета:** Здобуття навичок у створенні власних та використанні стандартних

колекцій в мові програмування Java.

1) Визначення формули :

2209 mod 2 = 1 => C2 : Set

2209 mod 3 = 1 => C3 : Список

2) Опис програми

Програма реалізує колекцію із внутрішньою структурою список. Клас Travel – узагальнений клас із лабораторної роботи №5, у клас Node описується вузол списку в якому міститься об’єкт класу Travel та посилання на наступну вершину. Клас TravelSet реалізує методи інтерфейсу Set.

3) Код програми

/\*\*

\* Clas Travel from Lab\_5

\* **@author** Kolomiets Oleh

\*

\*/

**public** **class** Travel {

/\*\*

\* **@param** d

\* time of travel

\* **@param** f

\* type of food

\*/

**public** Travel(**int** d, **int** f){

days = d;

food = f;

}

/\*\* gets days of travel

\* **@return** days

\*/

**public** **int** getDays(){

**return** days;

}

/\*\* gets type of food

\* **@return**

\*/

**public** **int** getFood(){

**return** food;

}

/\*\*Sets days of travel

\* **@param** days

\*/

**public** **void** setDays(**int** days) {

**this**.days = days;

}

/\*\*

\* sets type of food

\* **@param** food

\*/

**public** **void** setFood(**int** food) {

**this**.food = food;

}

/\*

\* @Override

\*/

**public** String toString(){

**return** "days of travel = " + days + " - food = " + food;

}

/\*\*

\* Days field

\*/

**private** **int** days;

/\*\*

\* Food field

\*/

**private** **int** food;

}

/\*\*

\* Node of list

\* **@author** Kolomiets Oleh

\*

\*/

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

/\*\*

\* data field

\*/

**private** Travel data;

/\*\*

\* nextNode field

\*/

**private** Node nextNode;

/\*\*

\* Constructor of Node with data parameter

\* **@param** data object of Travel class

\*/

**public** Node(Travel data){

**this**.data = data;

}

/\*\*

\* Constructor of Node with data and link to the next node parameters

\* **@param** data object of Travel class

\* **@param** node object of Node class

\*/

**public** Node(Travel data, Node node){

**this**.data = data;

**this**.nextNode = node;

}

/\*\*

\* gets data

\* **@return** data

\*/

**public** Travel getData() {

**return** data;

}

/\*\*

\* sets data

\* **@param** data

\*/

**public** **void** setData(Travel data) {

**this**.data = data;

}

/\*\*

\* gets NextNode

\* **@return** NextNode

\*/

**public** Node getNextNode() {

**return** nextNode;

}

/\*\* sets NextNode

\* **@param** nextNode

\*/

**public** **void** setNextNode(Node nextNode) {

**this**.nextNode = nextNode;

}

}

**import** java.util.Collection;

**import** java.util.Iterator;

**import** java.util.Set;

/\*\*

\* Class which implements Set through list data structure

\* **@author** Kolomiets Oleh

\*

\*/

**public** **class** TravelSet **implements** Set<Travel>{

/\*\*

\* size field

\*/

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

/\*\*

\* link for the first node

\*/

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

/\*\*

\* link for the last node

\*/

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

/\*\*

\* empty constructor

\*/

**public** TravelSet(){

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

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

}

/\*\*

\* constructor which have one parameter of class Travel

\* **@param** t object of class Travel

\*/

**public** TravelSet(Travel t){

add(t);

}

/\*\*

\* constructor with collection as parameter

\* **@param** a collection

\*/

**public** TravelSet(Collection<Travel> a ){

addAll(a);

}

/\* adds element to set.

\* adds element to the end of set

\*/

@Override

**public** **boolean** add(Travel a) {

**if**(a == **null**)

**throw** **new** NullTravelPointerException();

**if**(!contains(a)){

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

first.setData(a);

last = first;

}

**else**{

last.setNextNode(**new** Node(a));

last = last.getNextNode();

}

size++;

**return** **true**;

}

**else**

**return** **false**;

}

/\*

\* adds collection to set.

\* iterates over collection and adds elements to the end of set

\*/

@Override

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

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

**for**(Travel t : arg0){

**if**(add(t))

result = **true**;

**else**

result = **false**;

}

**return** result;

}

/\*

\* clears the set

\*/

@Override

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

first = last = **new** Node(**null**);

size = 0;

}

/\*

\* Checks is element present in collection

\* return true if it is, else returns false

\*/

@Override

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

**for**(Node curr = first; curr != **null**; curr = curr.getNextNode())

**if**(o.equals(curr.getData()))

**return** **true**;

**return** **false**;

}

/\*

\* Checks is collection contains in collection

\* return true if it is, else returns false

\*/

@Override

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

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

**for**(Iterator<?> iterator = coll.iterator(); iterator.hasNext();){

**if**(contains(iterator.next()))

result = **true**;

}

**return** result;

}

/\*

\* Checks is collection empty

\* return true if it is, else returns false

\*/

@Override

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

**return** size == 0;

}

/\*

\* create Iterator to iterate over the collection

\*/

@Override

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

Iterator<Travel> it = **new** Iterator<Travel>(){

Node temp = **new** Node(**null**);

Node current = first;

@Override

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

**if** (current.getNextNode() == **null**)

**return** **false**;

**else**

**return** **true**;

}

@Override

**public** Travel next() {

Travel result = **null**;

**if**(hasNext()){

temp = current;

current = current.getNextNode();

result = temp.getData();

}

**return** result;

}

@Override

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

}

};

**return** it;

}

/\*

\* Remove element from set if it there

\*/

@Override

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

Node current = first;

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

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

first = current.getNextNode();

result = **true**;

size--;

}

**else**{

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

Node next = current.getNextNode();

System.*out*.println("next" + next.getData());

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

current.setNextNode(next.getNextNode());

size--;

//System.out.println(" " + current.getNextNode());

last = current;

result = **true**;

}

**else**

current = current.getNextNode();

}

}

**return** result;

}

/\*

\* Remove collection from set if it there

\*/

@Override

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

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

**for**(Iterator<?> iterator = coll.iterator(); iterator.hasNext();){

Object obj = (Object)iterator.next();

result = remove(obj);

}

**return** result;

}

@Override

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

**return** **false**;

}

/\*

\* returne size of the set

\*/

@Override

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

**return** size;

}

/\*

\* convert set to array

\* return array of Object

\*/

@Override

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

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

**int** i = 0;

**for**(Iterator<?> iterator = iterator(); iterator.hasNext();){

Object obj = (Object)iterator.next();

a[i] = obj;

i++;

}

**return** a;

}

/\*

\* convert set to array of given type

\* return array

\*/

@SuppressWarnings("unchecked")

@Override

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

**return** (T[])toArray();

}

}

4) Висновок

Під час лабораторної роботи я здобув навички у створенні власних та використанні стандартних колекцій в мові програмування Java та покращив навички

документування коду.