МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ”ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра інформаційних

систем та мереж

Звіт з лабораторної роботи №5

з дисципліни "Технології програмування в розподілених інформаційних системах"

на тему:

**"Використання хмарних сервісів з використанням Google App Engine"**

Варіант №24

Виконав:

Студент групи КН-38

Стасишин О.З.

Прийняв:

доцент каф. ІСМ, к.т.н.

Щербак С. С.

ЛЬВІВ – 2018

**Мета роботи:** набуття практичних навичок опрацювання таких динамічних структур як зв’язні списки і дерева в системах Firebase/Google App Engine.

**Завдання на роботу:**

Розробити програми, які виконують операції вказані в індивідуальному та загальному завданні. Загальне. Інтерфейс програми для роботи зі списком/деревом повинен забезпечувати ввід і побудову списку/дерева та збереження/відтворення свого стану з бд Firebase. Крім того:

* Кожен елемент списку містить інформаційне поле(атрибут) деякого простого типу: символ, стрічка, число.
* Всі операції над динамічними структурами повинні супроводжуватись відповідним виводом на екран.
* В контрольних прикладах забезпечити опрацювання структур з 1000 елементами (заповнення можливе випадковими даними).

**Індивідуальне завдання:**



**Текст програми на мові Java:**

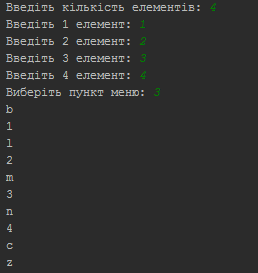
**BTree.java**

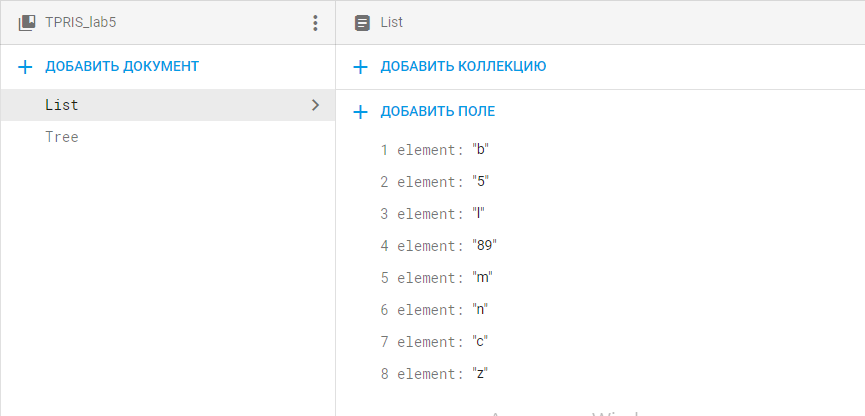
import com.google.api.core.ApiFuture;  
import com.google.cloud.firestore.DocumentReference;  
import com.google.cloud.firestore.WriteResult;  
  
import java.util.HashMap;  
import java.util.LinkedList;  
import java.util.Map;  
import java.util.Queue;  
import java.util.concurrent.ExecutionException;  
  
public class BTree {  
 private BTNode root;  
  
  
 public BTree(int value) {  
 this.root = new BTNode(value);  
  
 /\*for(int i = 0; i < 2000; i++){  
 insertBST((int)(Math.random() \* 10 + 50));  
 }\*/  
  
  
 insertBST(35);  
 insertBST(5);  
 insertBST(1);  
 insertBST(20);  
 insertBST(99);  
 insertBST(4);  
 insertBST(17);  
 insertBST(31);  
 }  
  
  
 public void addDataBaseTree(DocumentReference documentReference){  
 try {  
 Map<String, Object> data = new HashMap<String, Object>();  
 BTree tree = this;  
 Queue<Integer> q = levelOrderTraversalOpps(tree);  
  
 Object [] treeArray = q.toArray();  
 for(int i = 0; i < treeArray.length; i++){  
 System.*out*.println(treeArray[i] + " ");  
 data.put("Element " + i , treeArray[i]);  
 }  
 ApiFuture<WriteResult> result = documentReference.set(data);  
 result.get().getUpdateTime();  
 }catch(InterruptedException interrupted){  
 interrupted.printStackTrace();  
 }catch (ExecutionException e){  
 e.printStackTrace();  
 }  
 }  
  
 public BTree insertBST(int value) {  
 BTree tree = this;  
 tree.root = addBST(tree.root, value);  
 return tree;  
 }  
  
 private BTNode addBST(BTNode currentNode, int value) {  
  
 if (currentNode == null) {  
 return new BTNode(value);  
 }  
 else {  
 if (value <= currentNode.value) {  
 currentNode.left = addBST(currentNode.left, value);  
 }  
 else {  
 currentNode.right = addBST(currentNode.right, value);  
 }  
 }  
 return currentNode;  
 }  
  
  
 public Queue levelOrderTraversalOpps(BTree root) {  
 Queue<BTNode> queue=new LinkedList<BTNode>();  
 Queue<Integer> queue1 = new LinkedList<Integer>();  
 queue.add(root.root);  
 while(!queue.isEmpty())  
 {  
 BTNode tempNode=queue.poll();  
 if(tempNode.left!=null) {  
 queue.add(tempNode.left);  
 queue1.add(tempNode.left.value);  
 }  
 if(tempNode.right!=null) {  
 queue.add(tempNode.right);  
 queue1.add(tempNode.right.value);  
 }  
 }  
 return queue1;  
 }  
  
 /\*public Queue levelOrderTraversal(BTree root) {  
 Queue<BTNode> queue=new LinkedList<BTNode>();  
 queue.add(root.root);  
 while(!queue.isEmpty())  
 {  
 BTNode tempNode=queue.poll();  
 System.out.print(tempNode.value + " ");  
 if(tempNode.left!=null)  
 queue.add(tempNode.left);  
 if(tempNode.right!=null)  
 queue.add(tempNode.right);  
 }  
 return queue;  
 }\*/  
 public void printLinebyLine(BTree root){  
 if(root == null){  
 return;  
 }  
 final Queue<BTNode> queue = new LinkedList<BTNode>();  
 queue.add(root.root);  
 while(!queue.isEmpty()){  
 final int size = queue.size();  
 for (int i = 0; i < size; i++) {  
 BTNode node = queue.remove();  
 System.*out*.print(node.value + " ");  
 if(node.left != null){  
 queue.add(node.left);  
 }  
 if(node.right != null){  
 queue.add(node.right);  
 }  
 }  
 System.*out*.println();  
 }  
 }  
  
  
 public boolean isLeaf(BTNode node) {  
 return node.left == null && node.right == null;  
  
 }  
  
  
 public int howManyLeaves() {  
 return howManyLeaves(root);  
 }  
  
 private int howManyLeaves(BTNode node) {  
 if (node == null) {  
 return 0;  
 }  
 else if (isLeaf(node)) {  
 System.*out*.println("Лист: " + node.value);  
 return 1;  
 }  
 return howManyLeaves(node.left) + howManyLeaves(node.right);  
 }  
  
  
 private class BTNode {  
 int value;  
 BTNode left;  
 BTNode right;  
  
 public BTNode(int value) {  
 this.value = value;  
 left = null;  
 right = null;  
 }  
  
 }  
  
}

**ListParam.java**

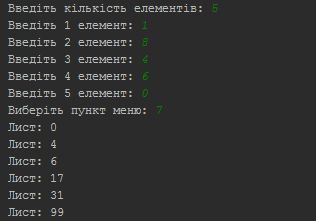
import com.google.api.core.ApiFuture;  
import com.google.cloud.firestore.DocumentReference;  
import com.google.cloud.firestore.WriteResult;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.ExecutionException;  
  
public class ListParam {  
  
 private ArrayList<Object> list = new ArrayList<Object>();  
  
 public ArrayList<Object> getListParam() {  
 return list;  
 }  
  
 ListParam(){  
  
 /\*for(int i = 0; i < 2000; i++){  
 list.add((char)Math.random() \* 10 + 50);  
 }\*/  
  
 list.add("b");  
 list.add("l");  
 list.add("m");  
 list.add("n");  
 list.add("c");  
 list.add("z");  
 }  
  
 public void addDataBase(DocumentReference documentReference) {  
 try {  
 Map<String, Object> data = new HashMap<String, Object>();  
 int N = list.size();  
 Object[] dequeArr = list.toArray();  
 for (int i = 0; i < N; i++) {  
 data.put((i + 1) + " element", dequeArr[i]);  
 }  
 ApiFuture<WriteResult> result = documentReference.set(data);  
 result.get().getUpdateTime();  
 }catch(InterruptedException interrupted){  
 interrupted.printStackTrace();  
 }catch (ExecutionException e){  
 e.printStackTrace();  
 }  
 }  
}

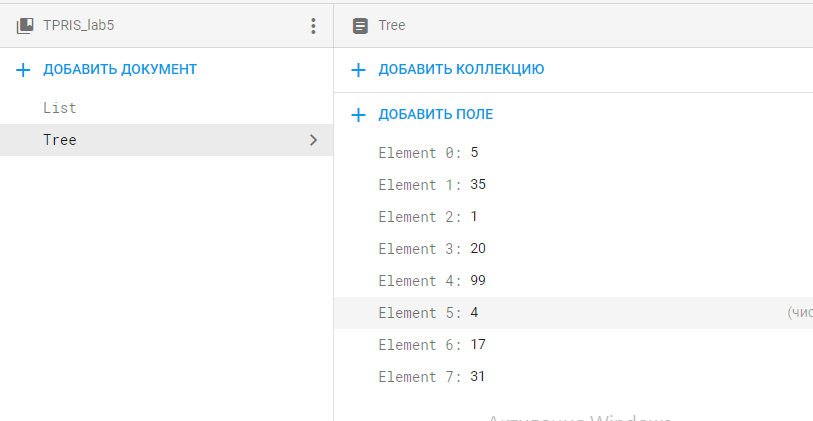
**Результати виконання програми:**





*Рис. 1. Робота зі списком*





*Рис. 2. Робота з деревом*

**Висновок:** програма призначена для роботи зі списком, деревом та Firebase. Програма реалізує усі необхідні завдання та зберігає/відтворює свій стан з БД. На цій лабораторній роботі було набуто навичок моделювання зв’язаних динамічних структур даних, як список та дерево і роботи з ними в середовищі Firebase/Google App Engine. Програма написана на мові Java у середовищі IntelliJ.