Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной инженерии и компьютерной техники

**Лабораторная работа №4**

по дисциплине

«Многопоточное программирование»

**Выполнил**

Ореховский А.,

группа P3417

**Преподаватель**

Доронин О.В.

Санкт-Петербург

2020

# Листинг кода

## Реализация lock free

import java.util.concurrent.atomic.AtomicMarkableReference;

public class SetImpl<T extends Comparable<T>> implements Set<T> {

    static class Tuple<T> {

        public final T prev;

        public final T curr;

        public Tuple(T x, T y) {

            this.prev = x;

            this.curr = y;

        }

    }

    static class Node<T> {

        public final T value;

        public final AtomicMarkableReference<Node<T>> nextReference;

        public Node(T value, Node<T> next) {

            this.value = value;

            nextReference = new AtomicMarkableReference<>(next, false);

        }

    }

    private final Node<T> head = new Node<>(null, null);

    @Override

    public boolean add(T value) {

        while (true) {

            Tuple<Node<T>> tuple = find(value);

            Node<T> prev = tuple.prev;

            Node<T> curr = tuple.curr;

            if (curr != null && curr.value.equals(value)) {

                return false;

            }

            Node<T> tail = new Node<>(value, curr);

            if (prev.nextReference.compareAndSet(curr, tail, false, false)) {

                return true;

            }

        }

    }

    @Override

    public boolean remove(T value) {

        while (true) {

           Tuple<Node<T>> tuple = find(value);

           Node<T> prev = tuple.prev;

           Node<T> curr = tuple.curr;

            if (curr == null || curr.value.compareTo(value) != 0) {

                return false;

            }

            Node<T> next = curr.nextReference.getReference();

            if (!curr.nextReference.attemptMark(next, true)) {

                continue;

            }

            prev.nextReference.compareAndSet(curr, next, false, false);

            return true;

        }

    }

    @Override

    public boolean contains(T value) {

        Node<T> curr = head.nextReference.getReference();

        while (curr != null && curr.value.compareTo(value) < 0) {

            curr = curr.nextReference.getReference();

        }

        return curr != null && curr.value.compareTo(value) == 0 && !curr.nextReference.isMarked();

    }

    @Override

    public boolean isEmpty() {

        Node<T> curr = head.nextReference.getReference();

        while (curr != null) {

            if(curr.nextReference.isMarked()) {

                curr = curr.nextReference.getReference();

            }

            else {

                return false;

            }

        }

        return true;

    }

    private Tuple<Node<T>> find(T value) {

        retry:

        while (true) {

            Node<T> prev = head;

            Node<T> curr = head.nextReference.getReference();

            Node<T> tmp;

            while (curr != null) {

                tmp = curr.nextReference.getReference();

                if (curr.nextReference.isMarked()) {

                    if (!prev.nextReference.compareAndSet(curr, tmp, false, false)) {

                        continue retry;

                    }

                } else {

                    if (curr.value.compareTo(value) >= 0) {

                        return new Tuple<>(prev, curr);

                    }

                    prev = curr;

                }

                curr = tmp;

            }

            return new Tuple<>(prev, null);

        }

    }

}

## Тесты

import org.junit.Test;

import java.util.HashSet;

import java.util.Random;

import static org.junit.Assert.\*;

public class SetTest {

    @Test

    public void add() {

        String value = "test";

        Set<String> set = new SetImpl<>();

        assertTrue(set.add(value));

        assertTrue(set.contains(value));

    }

    @Test

    public void doNotAdd() {

        String value = "test";

        Set<String> set = new SetImpl<>();

        assertTrue(set.add(value));

        assertTrue(set.contains(value));

        assertFalse(set.add(value));

    }

    @Test

    public void remove() {

        String value = "test";

        Set<String> set = new SetImpl<>();

        set.add(value);

        assertTrue(set.remove(value));

        assertFalse(set.contains(value));

    }

    @Test

    public void doNotRemove() {

        String value = "test";

        Set<String> set = new SetImpl<>();

        assertFalse(set.remove(value));

        assertFalse(set.contains(value));

    }

    @Test

    public void contains() {

        String value = "test";

        Set<String> set = new SetImpl<>();

        set.add(value);

        assertTrue(set.contains(value));

    }

    @Test

    public void doNotContains() {

        String addedValue = "added";

        String nonAddedValue = "non added";

        Set<String> set = new SetImpl<>();

        set.add(addedValue);

        assertFalse(set.contains(nonAddedValue));

    }

    @Test

    public void isEmpty() {

        Set<String> set = new SetImpl<>();

        assertTrue(set.isEmpty());

    }

    @Test

    public void isNotEmpty() {

        Set<String> set = new SetImpl<>();

        set.add("test");

        assertFalse(set.isEmpty());

    }

    @Test

    public void hashSetCompare() {

        java.util.Set<Integer> hashSet = new HashSet<>();

        Set<Integer> manualSet = new SetImpl<>();

        Random random = new Random();

        for (int i = 0; i < 10; i++) {

            int value = random.nextInt(Integer.MAX\_VALUE);

            assertEquals(hashSet.add(value), manualSet.add(value));

            assertTrue(manualSet.contains(value));

            assertFalse(manualSet.isEmpty());

        }

        for (Integer value : hashSet) {

            assertTrue(manualSet.contains(value));

            assertTrue(manualSet.remove(value));

            assertFalse(manualSet.contains(value));

        }

        assertTrue(manualSet.isEmpty());

    }

}

# Итоги

По окончании данной ЛР, я применил на практике знания об алгоритме lock free, а так же научился использовать элементы синхронизации на Java.