Suhyoon Bae

Lab 06

CIS 120 Data Structures

2020-04-07

Lab 06

Extending the Generic List ADT

**1. The Word doc containing the test cases, for both methods.**

**\*testi\_isEmpty**

package test;

import static org.junit.Assert.\*;

import java.awt.List;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import method.UnsortedLists;

public class testisEmpty {

public static UnsortedLists<Integer> list;

protected final static int DEFCAP = 100; // default capacity

protected static int origCap; // original capacity

@Before

public void setUp() throws Exception {

list = new UnsortedLists<Integer>(DEFCAP);

origCap = DEFCAP;

UnsortedLists.numElements = 0;

}

@Test

public void testisEmpty\_emptylist() {

assertEquals(true, list.isEmpty());

}

@Test

public void testisEmpty\_oneelement() {

list.add(3);

assertEquals(false, list.isEmpty());

}

@Test

public void testisEmpty\_manyelements() {

list.add(3);

list.add(10);

assertEquals(false,list.isEmpty());

}

}

**\*test\_removeAll**

package test;

import static org.junit.Assert.\*;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import org.omg.CosNaming.NamingContextExtPackage.AddressHelper;

import method.UnsortedLists;

public class testRemoveAll {

static UnsortedLists<Integer> list;

protected final static int DEFCAP = 100; // default capacity

protected static int origCap; // original capacity

@BeforeClass

public static void setUpBeforeClass() throws Exception {

list = new UnsortedLists<Integer>(100);

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

@Before

public void setUp() throws Exception {

list = new UnsortedLists<Integer>(DEFCAP);

origCap = DEFCAP;

UnsortedLists.numElements = 0;

}

@Test

public void test\_emptyList() {

assertEquals(0, list.removeAll(2));

}

@Test

public void testRemove\_manyelemetns() {

list.add(2);

list.add(2);

list.add(20);

list.add(2);

assertEquals(3, list.removeAll(2));

}

@Test

public void testRemoveAll\_oneelement() {

list.add(2);

list.add(10);

assertEquals(1, list.removeAll(2));

}

@Test

public void testRemoveAll\_elementdne() {

list.add(3);

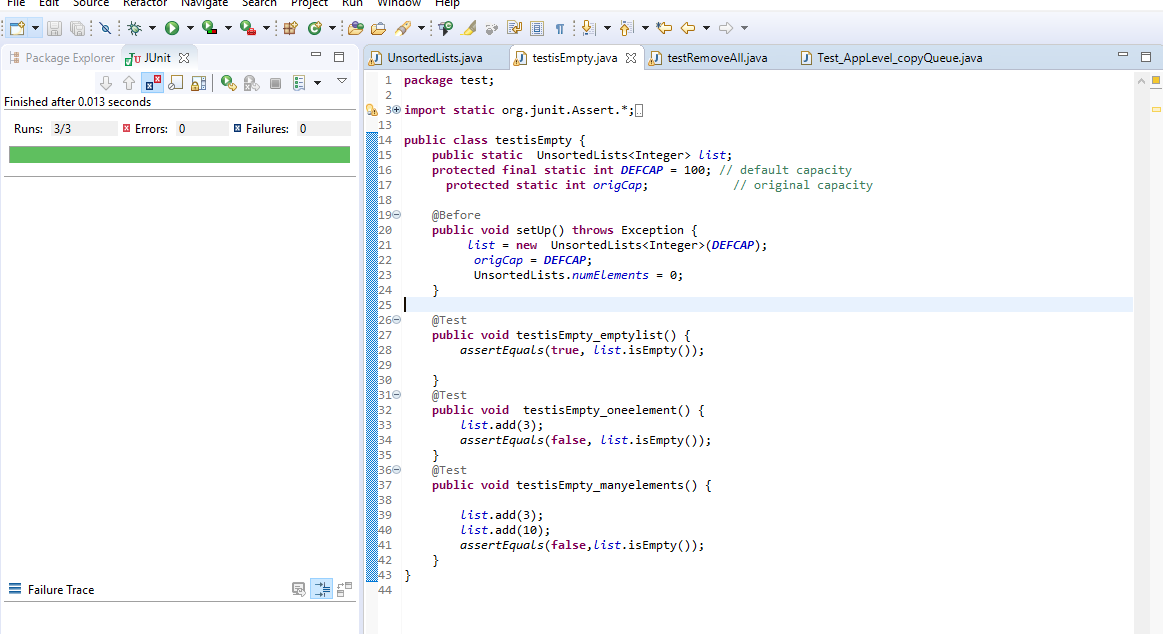
list.add(10);

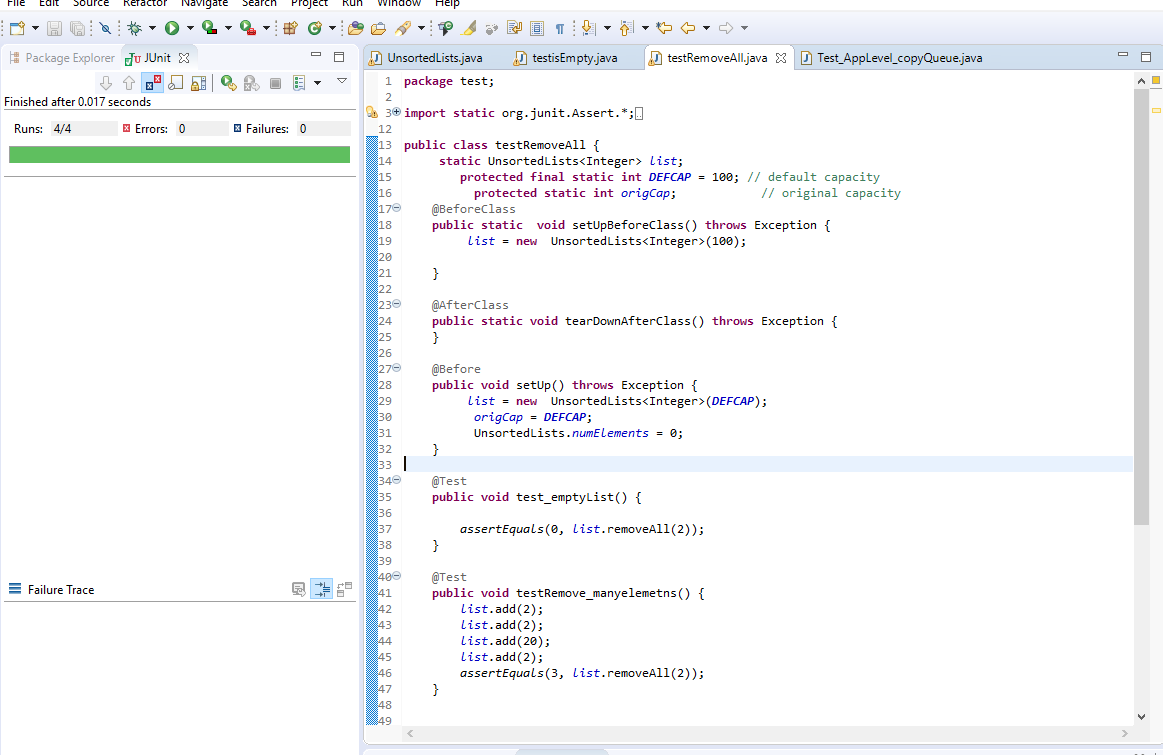
assertEquals(0, list.removeAll(9));

}

}

**2. Screen shots of the testing window, copied into a Word doc, demonstrating successful completion of the tests.**





**3. big-O algorithmic analysis for each of isEmpty() and removeAll(). Pay particular attention to the impact upon the result by removeAll() perhaps repeatedly calling remove().**

isEmpty() is O(N) because it has to be checked more than one time to check if the array is empty or not.

removeAll() is O(1) because the elements in the array will be removed at once.