Suhyoon Bae

Lab 03

CIS 120 Data Structures

2020-02-20

Lab 03

Generic Stack ADT

Page(s) File

1 Code printouts - StackInterface

2 Code printouts - ArrayStack, LinkedStack

3-5 The classes from the testStack package

6-8 Screen shots of unit test

9 Big-O efficiencies, test plan

**2. Code Printouts (mono-font, from Eclipse, use line numbering)**

**a. StackInterface.java (yellow highlighter around the code you added)**

package stackPackage;

public interface StackInterface<T>

{

void pop() throws StackUnderflowException;

// Throws StackUnderflowException if this stack is empty,

// otherwise removes top element from this stack.

T top() throws StackUnderflowException;

// Throws StackUnderflowException if this stack is empty,

// otherwise returns top element from this stack.

boolean isEmpty();

// Returns true if this stack is empty, otherwise returns false

**b. ArrayStack.java (print only the methods you added)**

**public** String toString(){

String result = "";

**if** (isEmpty()){

result = "Empty Stack";

}

**else** {

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

result = result +stack[i].toString() +"|";

}

result = "bottom|"+result+"top";

}

**return** result;

}

**public** **int** sizeIs(){

**return** topIndex+1;

}

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

topIndex = -1;

}

**c. LinkedStack.java (print only the methods you added)**

**public** String toString(){

String result = "";

**if** (top == **null**){

result ="Empty Stack";

}

**else** {

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

result = (top.getInfo()).toString() + "|" + result;

top = top.getLink();

}

result = "bottom|"+result+"top";

}

**return** result;

}

**public** **int** sizeIs(){

**int** size = 0;

**if** (top !=**null**){

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

size++;

top = top.getLink();

}

}

**return** size;

}

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

top = **null**; }

**d. The classes from the testStack package(test\_toString(), test\_sizeIs(), test\_clear() )**

**test\_toString()**

**package** testStack;

**import** **static** org.junit.Assert.\*;

**import** org.junit.After;

**import** org.junit.Assert;

**import** org.junit.Before;

**import** org.junit.Test;

**import** stackPackage.ArrayStack;

**import** stackPackage.LinkedStack;

**public** **class** Test\_toString {

/\*

\* Class to test the toString method added to the Stack ADT of Lab03

\*

\* test sizeIs on an empty stack a stack with one element a stack with many (but

\* less than full) elements and a "full" ArrayStack (not applicable to Linked

\* Stack - comment it out)

\*/

ArrayStack<Integer> stk1;

@Before

**public** **void** setUp() **throws** Exception {

stk1 = **new** ArrayStack<Integer>(5);

}

/\*

\* LinkedStack stk;

\*

\* @Before public void setUp() throws Exception {

\* stk = new LinkedStack<Integer>(); }

\*/

@Test

**public** **void** test\_toString\_on\_an\_emptyStack() {

Assert.*assertEquals*("Empty Stack", stk1.toString());

}

@Test

**public** **void** test\_toString\_on\_a\_stack\_with\_1\_element() {

stk1.push(5);

Assert.*assertEquals*("bottom|5|top", stk1.toString());

}

@Test

**public** **void** test\_toString\_on\_a\_stack\_with\_multiple\_elements() {

stk1.push(5);stk1.push(4);stk1.push(3);

Assert.*assertEquals*("bottom|5|4|3|top", stk1.toString());

}

// comment this test OUT when testing LinkedStack

@Test

**public** **void** test\_toString\_on\_a\_full\_stack() {

stk1.push(1);stk1.push(2);stk1.push(3);stk1.push(4);stk1.push(5);

Assert.*assertEquals*("bottom|1|2|3|4|5|top", stk1.toString());

}

@After

**public** **void** tearDown() **throws** Exception {

}

}

**test\_sizeIs()**

**package** testStack;

**import** **static** org.junit.Assert.\*;

**import** org.junit.After;

**import** org.junit.Assert;

**import** org.junit.Before;

**import** org.junit.Test;

**import** stackPackage.ArrayStack;

**import** stackPackage.LinkedStack;

/\*

\* Class to test the sizeIs method added to the ArrayStack ADT of Lab03

\*

\* test sizeIs on an empty stack

\* a stack with one element

\* a stack with multiple elements

\* and a "full" ArrayStack (not applicable to Linked Stack - comment it out)

\*/

**public** **class** Test\_sizeIs {

// ArrayStack stk;

//

// @Before

// public void setUp() throws Exception {

//

// stk = new ArrayStack<Integer>(5);

//

// }

LinkedStack stk;

@Before

**public** **void** setUp() **throws** Exception {

stk = **new** LinkedStack<Integer>();

}

@Test

**public** **void** test\_sizeIs\_on\_an\_emptyStack() {

Assert.*assertEquals*(0, stk.sizeIs());

}

@Test

**public** **void** test\_sizeIs\_on\_a\_stack\_with\_1\_element() {

stk.push(5);

Assert.*assertEquals*(1, stk.sizeIs());

}

@Test

**public** **void** test\_sizeIs\_on\_a\_stack\_with\_multiple\_elements() {

stk.push(5);

stk.push(4);

stk.push(3);

Assert.*assertEquals*(3, stk.sizeIs());

}

// comment this test OUT when testing LinkedStack

// @Test

// public void test\_sizeIs\_on\_a\_full\_stack() {

// stk.push(1); stk.push(2); stk.push(3); stk.push(4); stk.push(5);

//

// Assert.assertEquals(5, stk.sizeIs());

// }

}

**test\_clear()**

**package** testStack;

**import** **static** org.junit.Assert.\*;

**import** org.junit.After;

**import** org.junit.Assert;

**import** org.junit.Before;

**import** org.junit.Test;

**import** stackPackage.ArrayStack;

**import** stackPackage.LinkedStack;

**public** **class** Test\_clear {

// ArrayStack<Integer> stk;

//

// @Before

// public void setUp() throws Exception {

// stk = new ArrayStack<Integer>(5);

// }

LinkedStack<Integer> stk;

@Before

**public** **void** setUp() **throws** Exception {

stk = **new** LinkedStack<Integer>();

}

@Test

**public** **void** test\_clear\_on\_an\_emptyStack() {

stk.clear();

Assert.*assertEquals*(**true**, stk.isEmpty());

}

@Test

**public** **void** test\_clear\_on\_a\_stack\_with\_1\_element() {

stk.push(14);

stk.clear();

Assert.*assertEquals*(**true**, stk.isEmpty());

}

@Test

**public** **void** test\_clear\_on\_a\_stack\_with\_multiple\_elements() {

stk.push(5); stk.push(4); stk.push(3);

stk.clear();

Assert.*assertEquals*(**true**, stk.isEmpty());

}

/\*

\* @Test public void test\_clear\_on\_a\_full\_stack() { stk.push(1); stk.push(2);

\* stk.push(3); stk.push(4); stk.push(5); stk.clear(); Assert.assertEquals(true,

\* stk.isEmpty()); }

\*/

@After

**public** **void** tearDown() **throws** Exception {

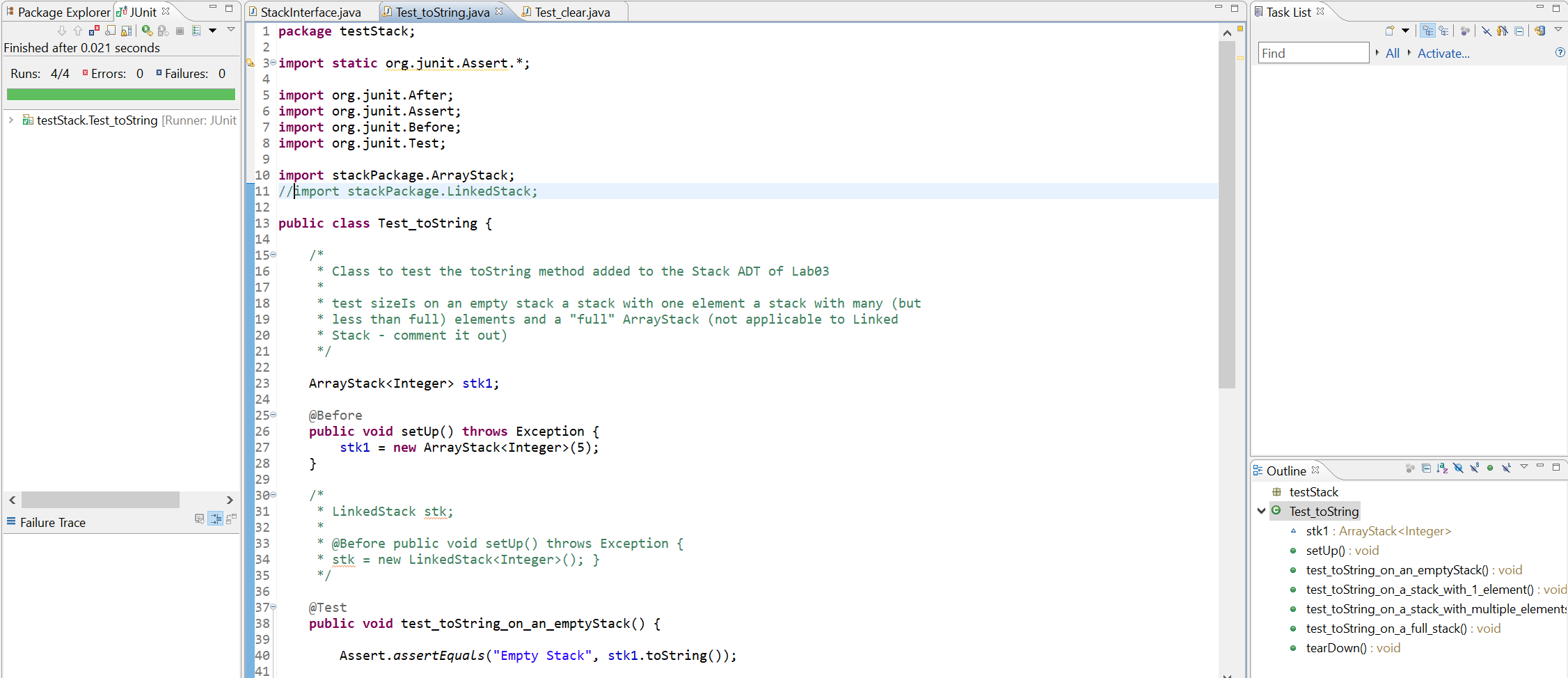
}

}

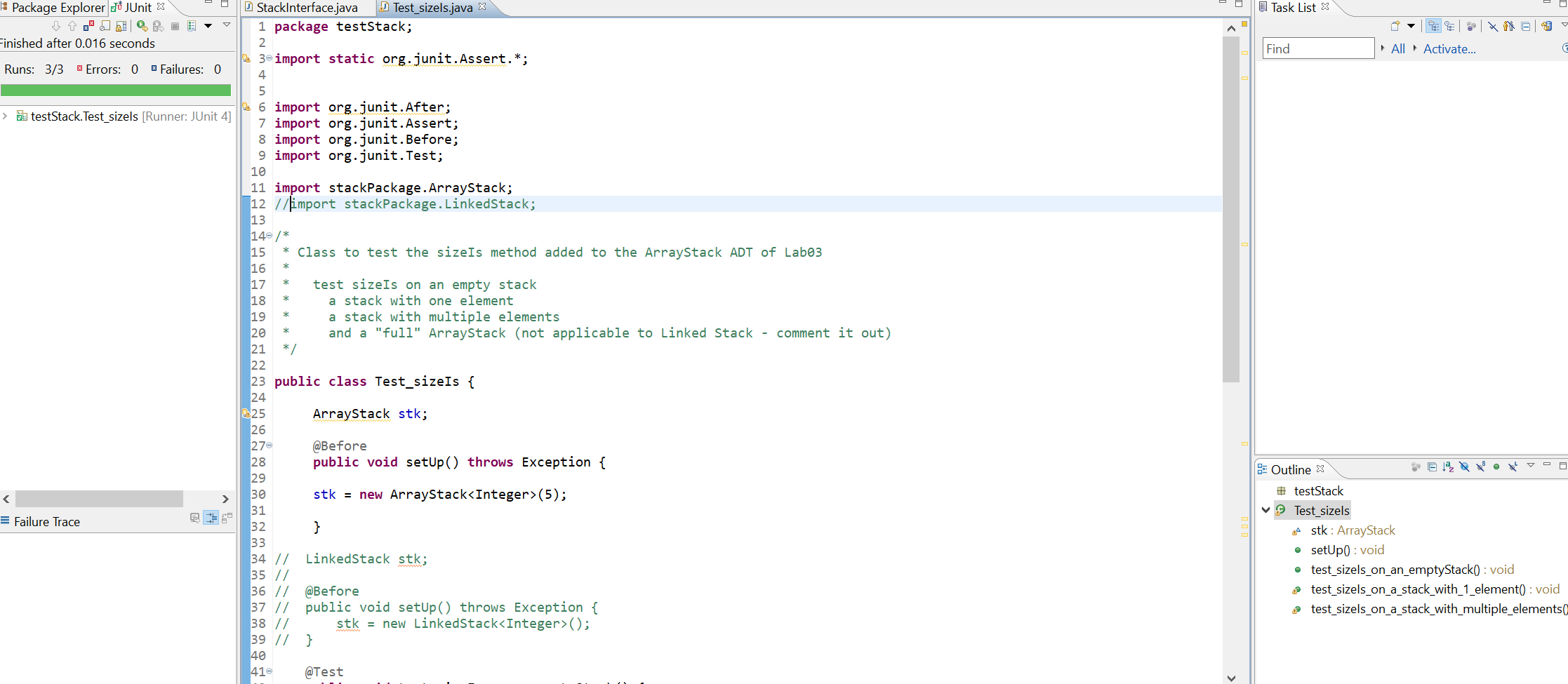
**3. Screen shots of the console window, copied into a Word doc, demonstrating successful completion of**

**a. The toString(), sizeIs(), and clear() unit tests of the Array based Stack.**

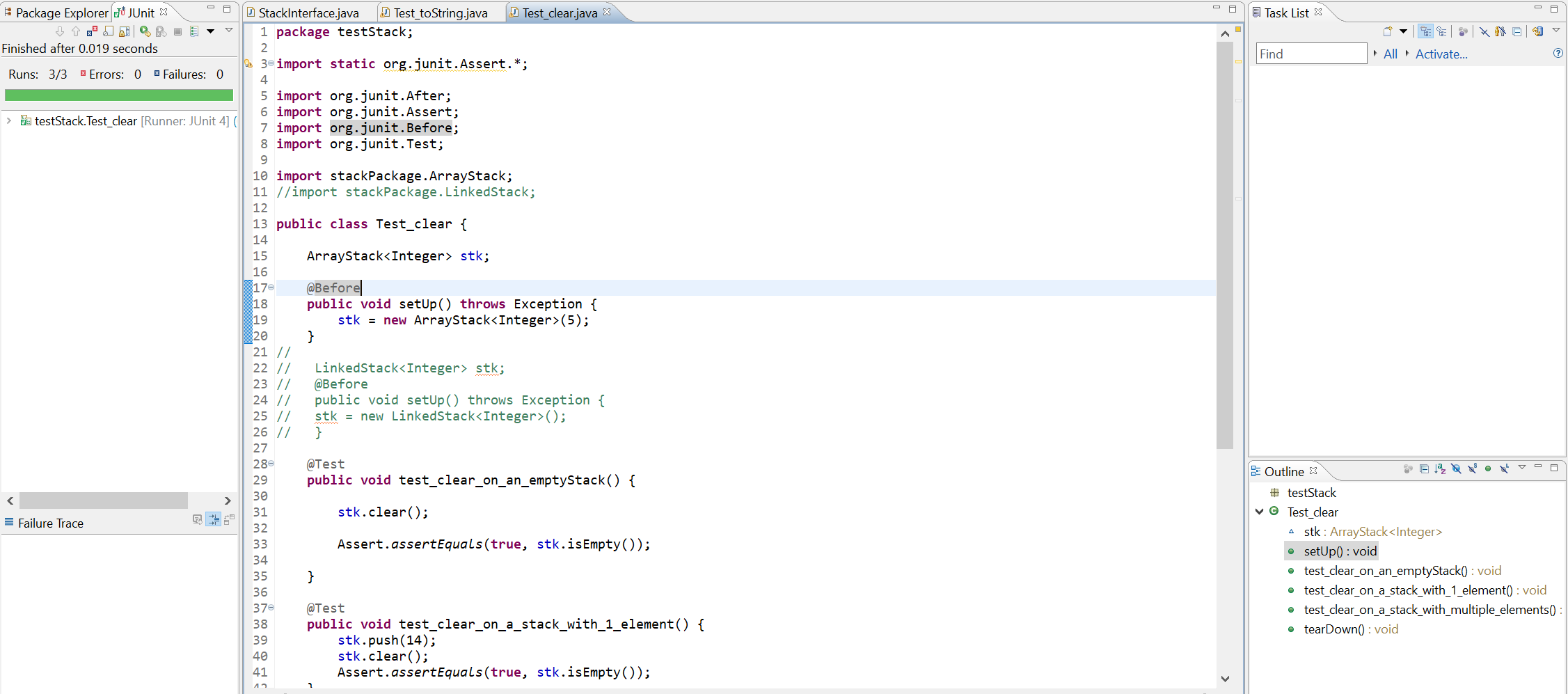
toString()



sizeIs()

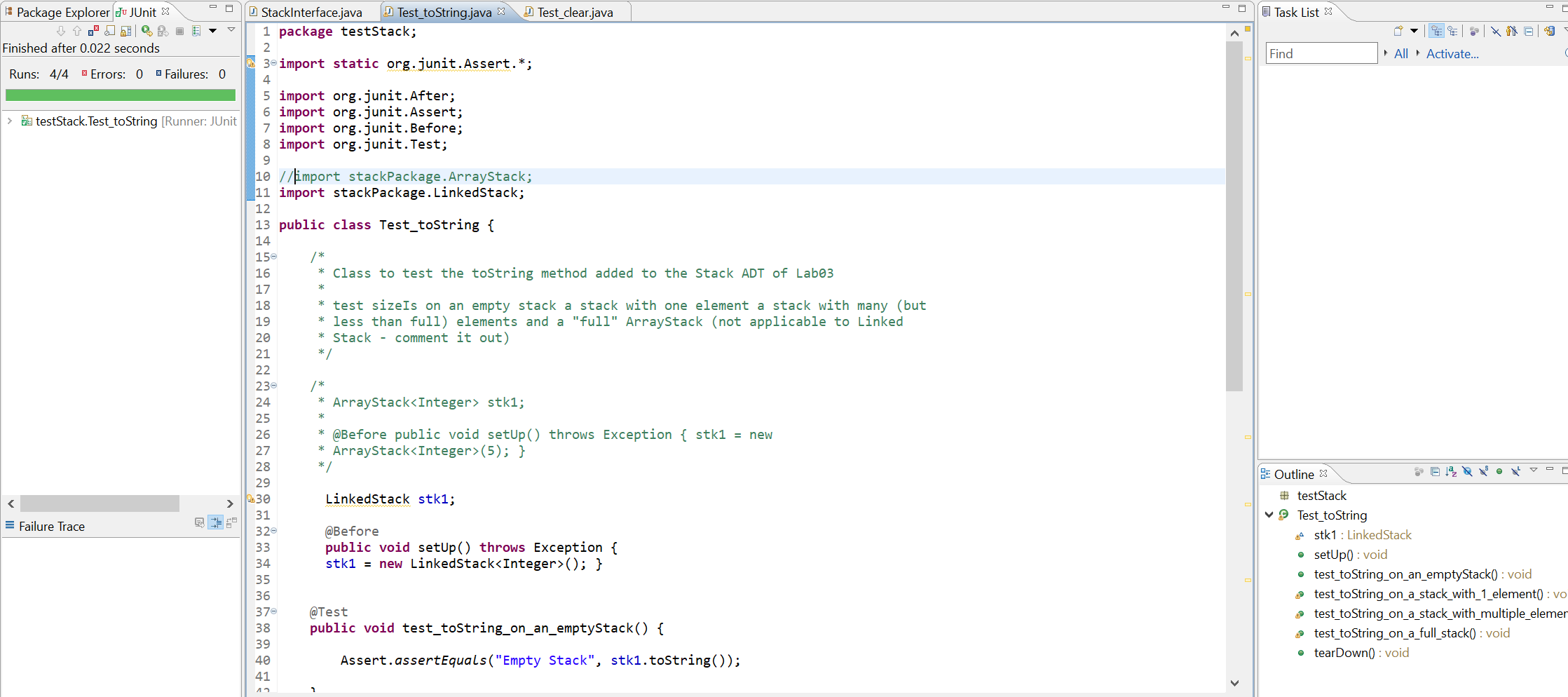


clear()

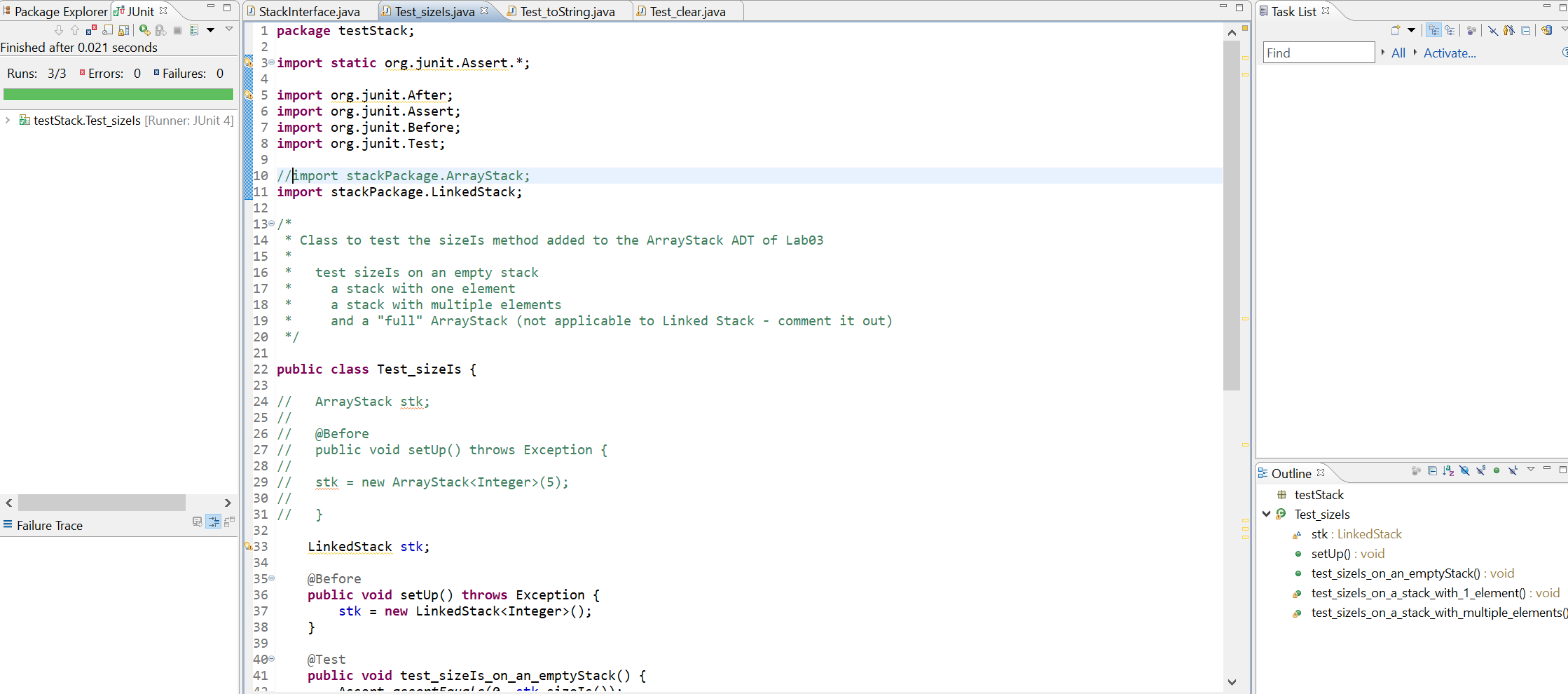


**b. The toString(), sizeIs(), and clear() unit tests of the Linked based Stack.**

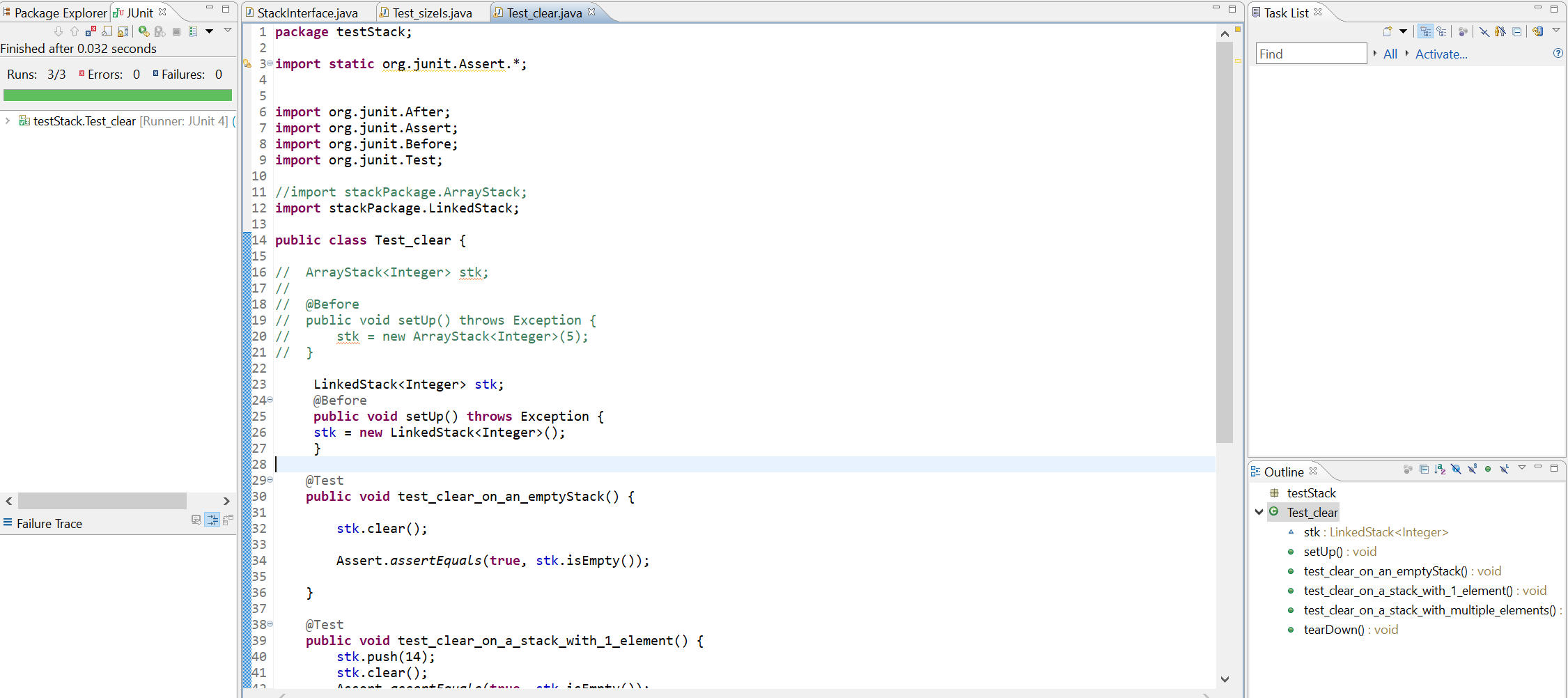
toString()



sizeIs()



clear()



**4. The Word doc containing the test plan and the table of** **Big-O efficiencies (copied in from below).**

|  |  |  |  |
| --- | --- | --- | --- |
| operation | Constructor, observer, or transformer? | Big-O for Array-Based | Big-O for Reference-Based |
| constructor | Constructor | O(1) | O(1) |
| push | Transformer | O(1) | O(1) |
| pop | Transformer | O(1) | O(1) |
| top | Observer | O(1) | O(1) |
| isEmpty | Observer | O(1) | O(1) |
| isFull | Observer | O(1) | O(1) |
| sizeIs | Observer | O(1) | O(N) |
| clear | Transformer | O(1) | O(1) |
| toString | Observer | O(N) | O(N) |

Test Plan

|  |  |  |
| --- | --- | --- |
| Operation to be Tested and Description of Action | Input Value | Expected Output |
| toString()  invoke when empty  push 1 item and invoke  push 3 items and invoke | 5  5,8,12 | “Empty Stack”  “bottom|5|top”  Bottom|5|8|12|top |
| sizeIs()  invoke when empty  push 1item and invoke  push 3 items and invoke | 5  10,5,5 | 1  3 |
| Clear()  Invoke when empty  Push 1 item and invoke |  | Stack is empty  Stack is empty |