SWE 637 Homework 5

Shaeq Khan

This assignment was completed using the Code coverage plugin in NetBeans 6.9.1

The NetBeans Code Coverage Plugin provides an interactive way to see testing coverage results within the NetBeans IDE. This enables developers to quickly identify the portions of Java code that aren't covered by their unit tests. The plugin is part of the standard distribution. Of course you need to have tests in your project for it to work. This plugin uses Emma which was suggested on the class website. I tried to run Emma via command line initially but was unable to do so successfully.

I chose to test the **java.util.Stack** class for this homework. The code for the class was taken from docjar.com and is pasted below –

```
/*
 * SWE 637 Homework 5
 * Source code for java.util.Stack picked for the homework
 * Link http://www.docjar.com/html/api/java/util/Stack.java.html
 *
 * @author skhan27
 */


/*
      * Copyright (c) 1994, 2010, Oracle and/or its affiliates. All rights reserved.
      * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
      *
      * This code is free software; you can redistribute it and/or modify it
      * under the terms of the GNU General Public License version 2 only, as
      * published by the Free Software Foundation.  Oracle designates this
      * particular file as subject to the "Classpath" exception as provided
      * by Oracle in the LICENSE file that accompanied this code.
      *
      * This code is distributed in the hope that it will be useful, but WITHOUT
      * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
      * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
      * version 2 for more details (a copy is included in the LICENSE file that
      * accompanied this code).
      *
      * You should have received a copy of the GNU General Public License version
      * 2 along with this work; if not, write to the Free Software Foundation,
      * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
      *
      * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
      * or visit www.oracle.com if you need additional information or have any
      * questions.
      */

     //package java.util;

     /**
      * The <code>Stack</code> class represents a last-in-first-out
      * (LIFO) stack of objects. It extends class <tt>Vector</tt> with five
      * operations that allow a vector to be treated as a stack. The usual
```

```
 * <tt>push</tt> and <tt>pop</tt> operations are provided, as well as a
 * method to <tt>peek</tt> at the top item on the stack, a method to test
 * for whether the stack is <tt>empty</tt>, and a method to <tt>search</tt>
 * the stack for an item and discover how far it is from the top.
 * <p>
 * When a stack is first created, it contains no items.
 *
 * <p>A more complete and consistent set of LIFO stack operations is
 * provided by the {@link Deque} interface and its implementations, which
 * should be used in preference to this class.  For example:
 * <pre>   {@code
 *   Deque<Integer> stack = new ArrayDeque<Integer>();}</pre>
 *
 * @author  Jonathan Payne
 * @since   JDK1.0
 */


//headers included
import java.util.Vector;
import java.util.EmptyStackException;

public
class Stack<E> extends Vector<E> {

    /**
     * Creates an empty Stack.
     */
    public Stack() {
    }


    /**
     * Pushes an item onto the top of this stack. This has exactly
     * the same effect as:
     * <blockquote><pre>
     * addElement(item)</pre></blockquote>
     *
     * @param   item   the item to be pushed onto this stack.
     * @return  the <code>item</code> argument.
     * @see     java.util.Vector#addElement
     */
    public E push(E item) {
        addElement(item);

        return item;
    }


    /**
     * Removes the object at the top of this stack and returns that
     * object as the value of this function.
     *
     * @return  The object at the top of this stack (the last item
     *          of the <tt>Vector</tt> object).
     * @throws  EmptyStackException  if this stack is empty.
     */
    public synchronized E pop() {
        E       obj;
        int     len = size();
```

```
    obj = peek();
    removeElementAt(len - 1);

    return obj;
}


/**
 * Looks at the object at the top of this stack without removing it
 * from the stack.
 *
 * @return  the object at the top of this stack (the last item
 *          of the <tt>Vector</tt> object).
 * @throws  EmptyStackException  if this stack is empty.
 */
public synchronized E peek() {
    int     len = size();

    if (len == 0)
        throw new EmptyStackException();
    return elementAt(len - 1);
}


/**
 * Tests if this stack is empty.
 *
 * @return  <code>true</code> if and only if this stack contains
 *          no items; <code>false</code> otherwise.
 */
public boolean empty() {
    return size() == 0;
}


/**
 * Returns the 1-based position where an object is on this stack.
 * If the object <tt>o</tt> occurs as an item in this stack, this
 * method returns the distance from the top of the stack of the
 * occurrence nearest the top of the stack; the topmost item on the
 * stack is considered to be at distance <tt>1</tt>. The <tt>equals</tt>
 * method is used to compare <tt>o</tt> to the
 * items in this stack.
 *
 * @param   o   the desired object.
 * @return  the 1-based position from the top of the stack where
 *          the object is located; the return value <code>-1</code>
 *          indicates that the object is not on the stack.
 */
public synchronized int search(Object o) {
    int i = lastIndexOf(o);

    if (i >= 0) {
        return size() - i;
    }
    return -1;
}
```

```
/*
 * @author skhan27
 * @returns the contents of the Stack from top to bottom
 */
public String toString(){
    String s = "";
    for(int i = 0; i < size(); i++)
        s = s + get(i);

    return s;
}

/** use serialVersionUID from JDK 1.0.2 for interoperability */
private static final long serialVersionUID = 1224463164541339165L;   }
```

The **test cases** I wrote for this code is as follows –

```
/*
 * SWE 637 Homework 5
 * Test cases for java.util.Stack
 *
 * @author skhan27
 */

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.EmptyStackException;


public class StackTest {

    Stack s = new Stack();

    //push one item on the stack
    @Test
    public void testPushString() {
        Object o = s.push("A string");
        assertEquals(s.toString(), o.toString());
    }

    //push some numbers on the stack
    @Test
    public void testPushNumbers(){

        for(int i = 0; i < 5; i++){
            Object temp = s.push(i);
        }

        assertEquals(s.toString(), "01234");

    }

    //push an item on the stack and then remove it
    @Test
    public void testPop(){

        Object pu = s.push(3);
        Object po = s.pop();

        assertEquals(po.toString(), "3");
    }
```

```
//push several items on the stack and pop one item
@Test
public void testPopWithItems(){

    for(int i = 0; i < 5; i++){
        Object temp = s.push(i);
    }

    Object po = s.pop();

    //Item popped on the top of the list is 4
    assertEquals(po.toString(), "4");

    //the stack after the item is popped does not contain the item
    assertEquals(s.toString(), "0123");
}

//test throws an exception when pop is called on an empty stack
@Test(expected = EmptyStackException.class)
public void testPopException(){
    Object po = s.pop();
}

//test to peek at the top of the stack
@Test
public void testPeekWithItems(){

    for(int i = 0; i < 5; i++){
        Object temp = s.push(i);
    }

    Object pe = s.peek();

    //Item peeked on the top of the list is 4
    assertEquals(pe.toString(), "4");

    //the stack after the item is peeked at remains the same
    assertEquals(s.toString(), "01234");
}

//test to get an exception when we try to peek at an empty stack
@Test(expected = EmptyStackException.class)
public void testPeekException(){
    Object pe = s.peek();
}

//test to check an empty and nonempty stack
@Test
public void testEmpty(){

    assertTrue(s.empty());

    Object pu = s.push("something");
    assertFalse(s.empty());
}
```

```
    //test search method when the object is not in the stack
    @Test
    public void testSearchObjectNotPresent(){

        for(int i = 0; i < 5; i++){
            Object temp = s.push(i);
        }

        assertEquals(s.search(9), -1);
    }

    //test search method when one instance of object is present
    @Test
    public void testSearchOneObjectPresent(){

        for(int i = 0; i < 5; i++){
            Object temp = s.push(i);
        }

        assertEquals(s.search(3), 2);
    }

    //test search method when multiple instance of object is present
    @Test
    public void testSearchMultipleObjectPresent(){

        s.push("a"); s.push("b"); s.push("c"); s.push("a"); s.push("d");

        assertEquals(s.search("a"), 2);
    }


}
```

Screen shots of visual representation of code coverage after the plugin is activated

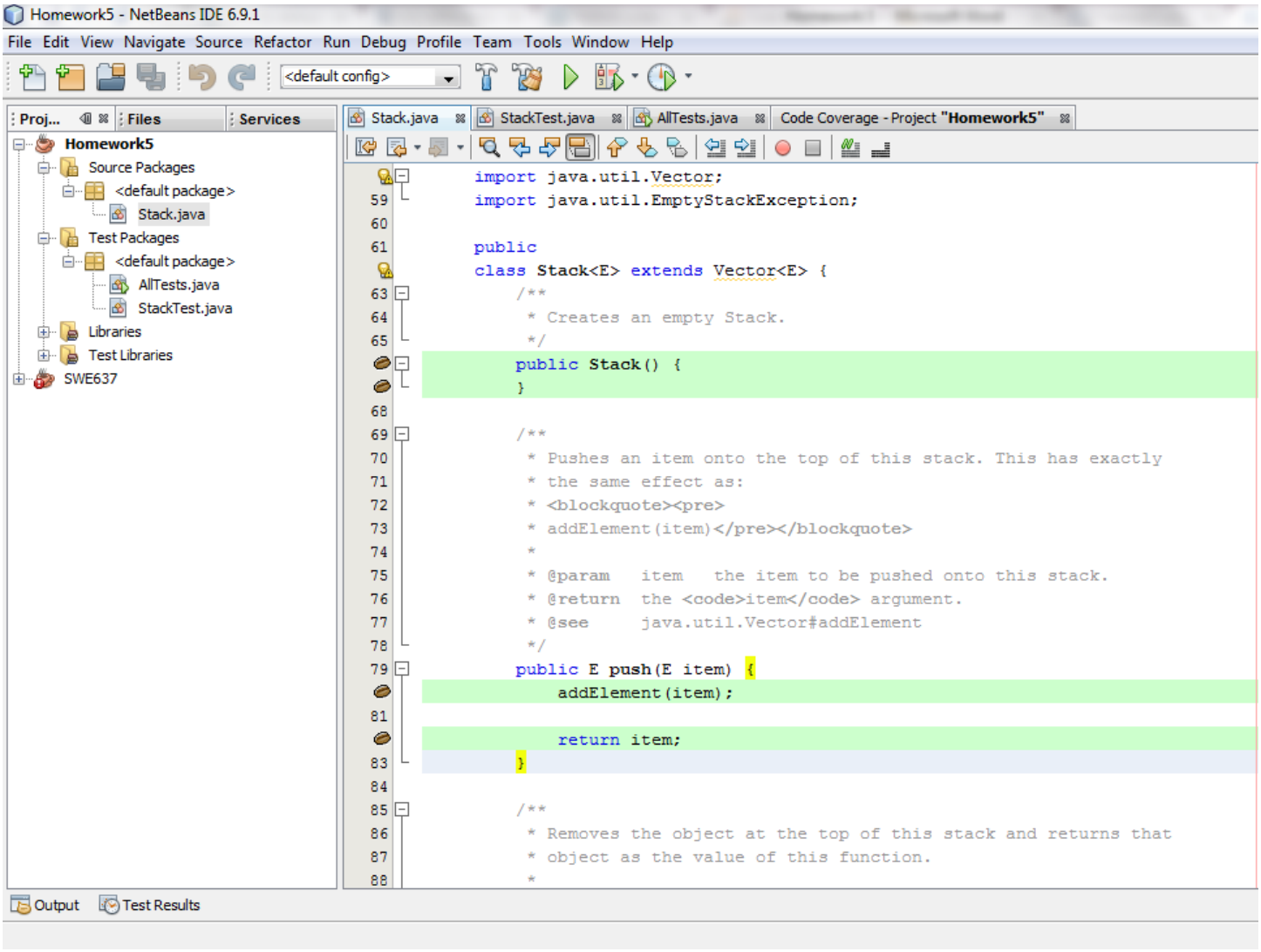

Figure 1 – Constructor and push method shown with coverage

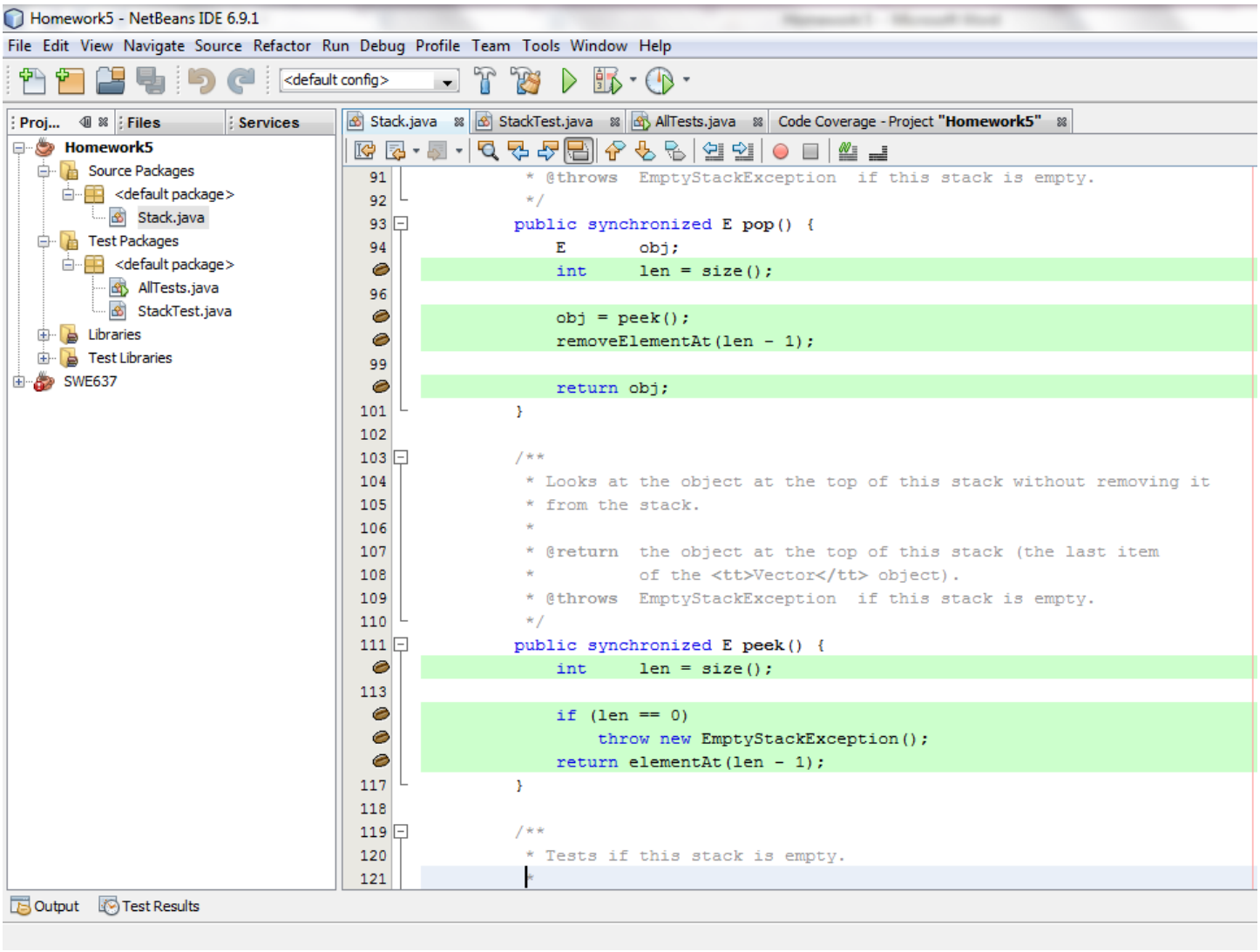

Figure 2 – pop and peek method shown with coverage

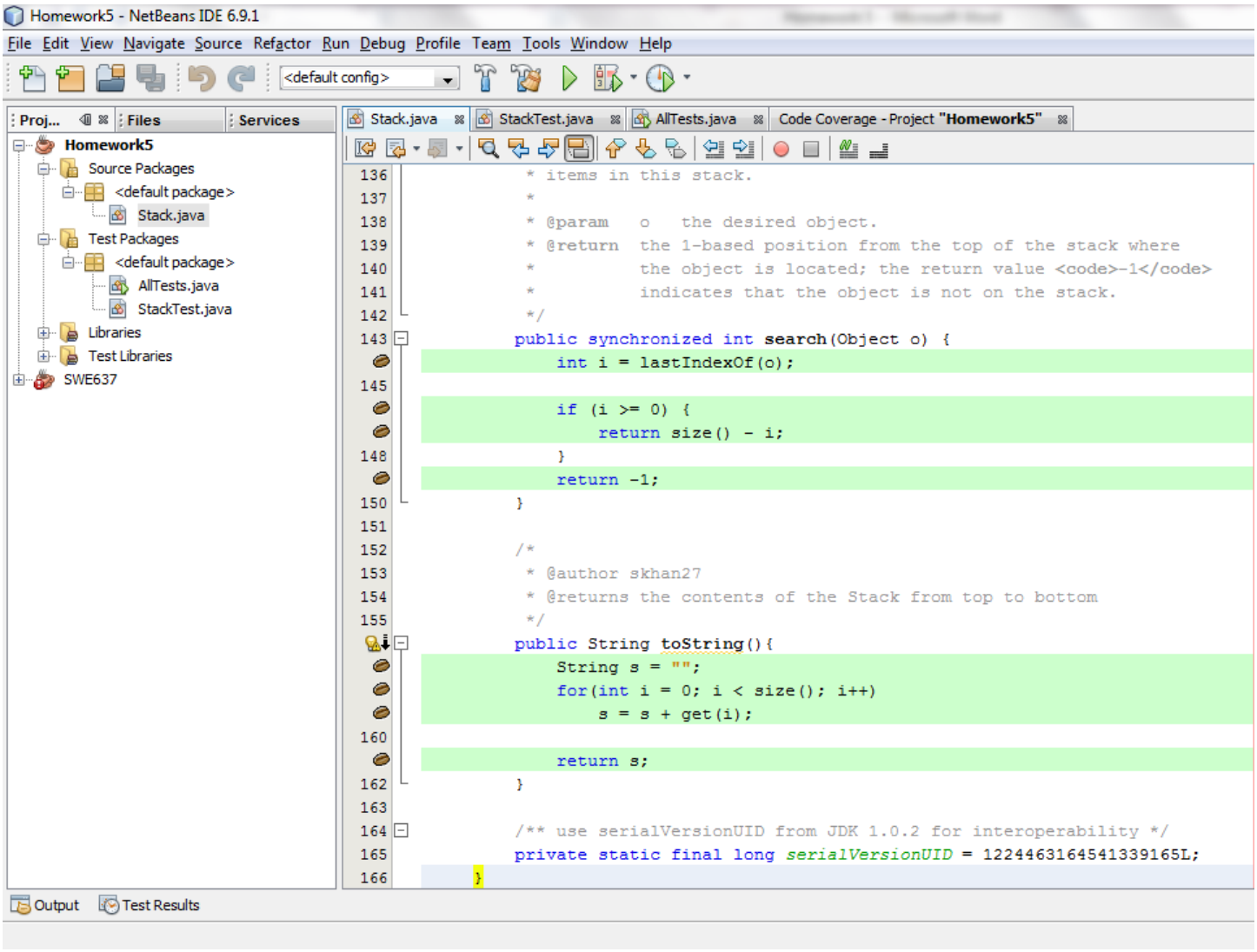

Figure 3 – search and toString(self implemented)shown with coverage

All tests pass for the code

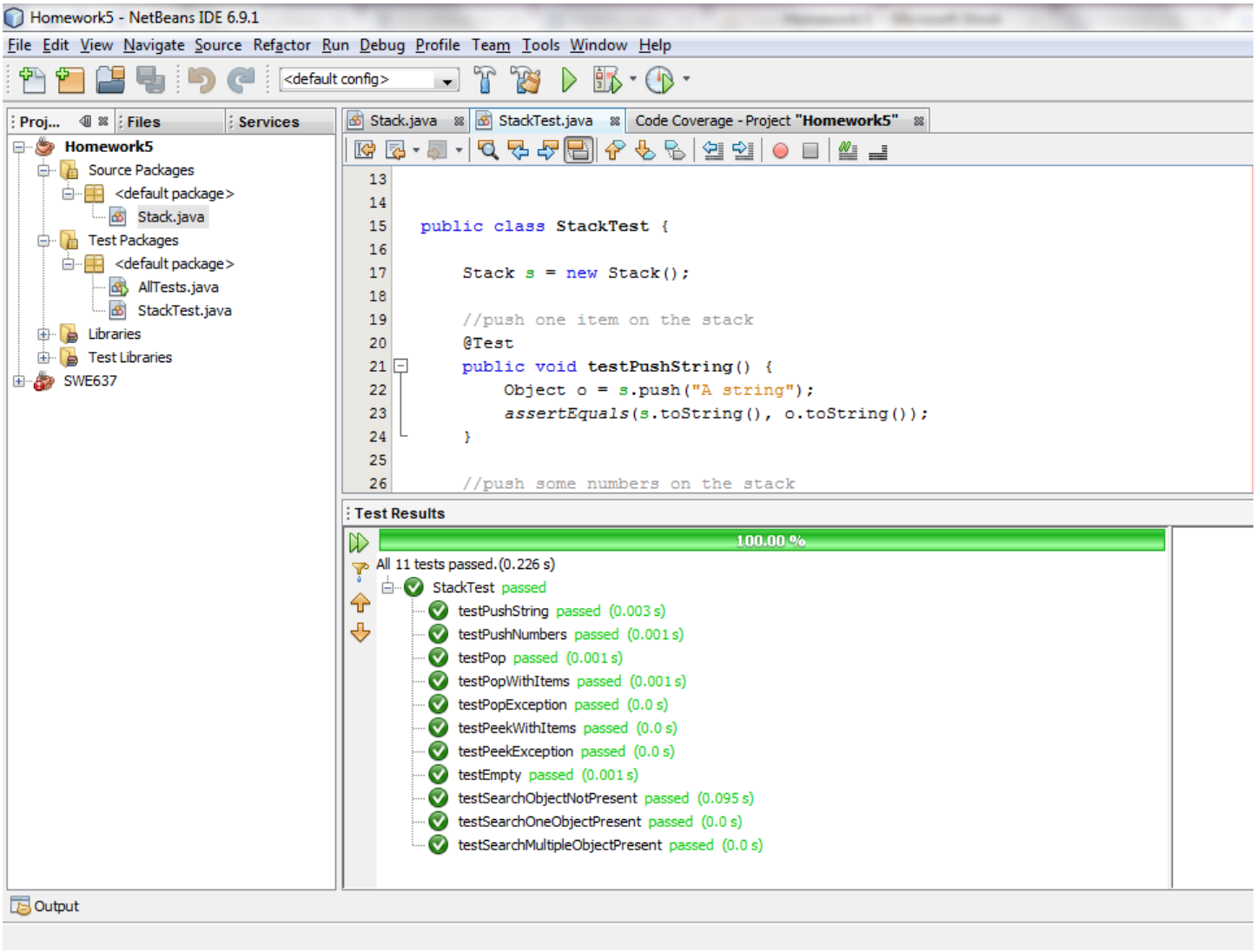

Figure 4 – Tests passed shown for java.util.Stack

Code coverage report generated by NetBeans