A Declaration on Plagiarism

| | |
|---|---|
| **Name:** | Archana Kalapgar |
| **Student Number:** | 19210184 |
| **Programme:** | MCM19-20 |
| **Module Code:** | CA650 |
| **Assignment Title:** | Software Process Quality |
| **Submission Date:** | 11 Feb 2020 |
| **Module Coordinator:** | Renaat Verbruggen |

Name: Archana Kalapgar Date: 11-02-2020

# CA650 XUnit Assignment

## Abstract

A set of classes in java that implements the Abstract Data Type - the Stack. Created and added integers to a Stack used push, pop, isEmpty and peek methods. Used JUnit (version4) to create a test suite for testing your class.

## 1. Stack Code [2 page]

```
package com.dcu.assignment;

public class Stack {
	private int arr[];
	private int top;
	private static int DEFAULT_CAPACITY = 3;

	public Stack(int size) {
		arr = new int[size];
		top = -1;
	}

	public Stack() {
		arr = new int[DEFAULT_CAPACITY];
		top = -1;
	}

	public void push(int x) {
		if (isFull()) {
			System.out.println("OverFlow\nProgram Terminated\n");
			throw new StackOverflowError();
		} else {
			System.out.println("Inserting " + x);
			arr[++top] = x;
		}
	}

	public int pop() {
		if (isEmpty()) {
			System.out.println("UnderFlow\nProgram Terminated");
			return -1;
		}
		System.out.println("Removing " + peek());
		return arr[top--];
	}
	public int peek() {

		if (!isEmpty())
			return arr[top];
		else
			return -1;
	}

	public int size() {
		return top + 1;
	}

	public Boolean isEmpty() {
		return top == -1;

	}

	public Boolean isFull() {
		return top == arr.length - 1;
	}

	public int testing() {
		return 1;
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Stack s = new Stack(3);
		s.push(1);
		s.push(2);
		s.pop();
		s.pop();

		s.push(3);

		System.out.println("Top element is: " + s.peek());
		System.out.println("Stack size is " + s.size());

		s.pop();

		if (s.isEmpty())
			System.out.println("Stack is Empty");
		else
			System.out.println("Stack is Not Empty");

	}

}
```

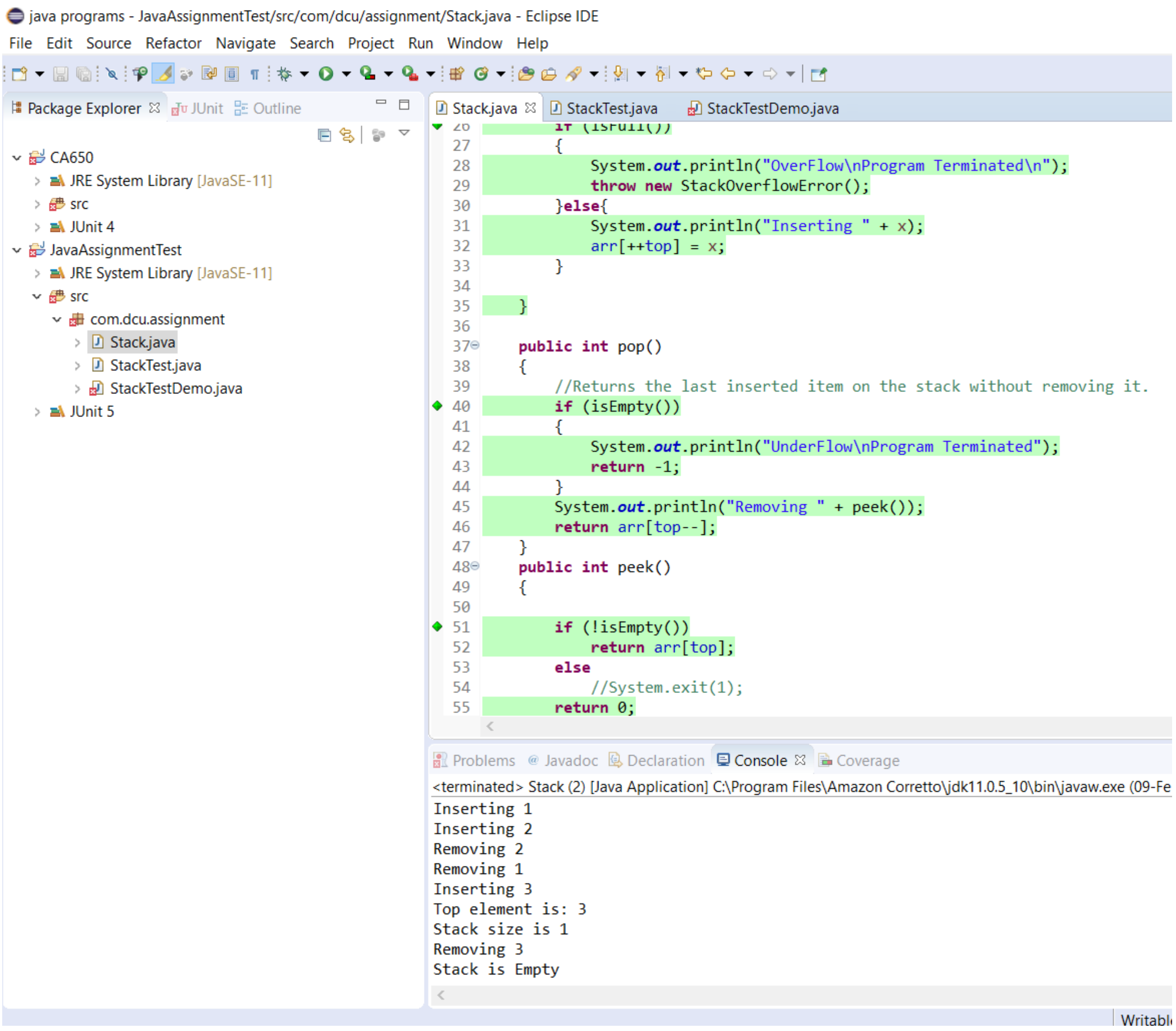
**2. JUnit Code [2 page]**

```
package com.dcu.assignment;

import static org.junit.Assert.*;

import java.util.NoSuchElementException;

import org.junit.Test;

public class StackTest {

	private static int DEFAULT_TEST_CAPACITY = 3;
	Stack emtyStack = new Stack(DEFAULT_TEST_CAPACITY);

	@Test
	public void testIsEmpty() {
		assertTrue("not isEmpty after construction", emtyStack.isEmpty());
		emtyStack.push(1);
		assertFalse("isEmpty after push", emtyStack.isEmpty());
		emtyStack.pop();
		assertTrue("not isEmpty after push/pop", emtyStack.isEmpty());
	}

	@Test
	public void testPushEmpty() {
		assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
		assertEquals(-1, emtyStack.peek());
		emtyStack.push(1);
		assertEquals(1, emtyStack.pop());
	}

	@Test(expected = StackOverflowError.class)
	public void testPushStackisFull() {
		assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
		emtyStack.push(1);
		emtyStack.push(2);
		emtyStack.push(3);
		// Adding Extra Element
		emtyStack.push(4);
	}

	@Test
	public void testPopNormal() {
		assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
		emtyStack.push(1);
		assertEquals( 1, emtyStack.pop());
	}

	@Test
	public void testPopStackisEmpty() {
		assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
		int data = emtyStack.peek();
		assertEquals(data, -1);
	}

	@Test
	public void testPopStackisFull() {
		assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
		emtyStack.push(1);
		assertEquals( 1, emtyStack.pop()); // pop on 1st
		emtyStack.push(2);
		assertEquals( 2, emtyStack.pop());
		emtyStack.push(3);
		assertEquals( 3, emtyStack.pop());
	}

	@Test
	public void testPeekNormal() {
		assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
		emtyStack.push(1);

		assertEquals( 1, emtyStack.peek());
		emtyStack.push(2);
		assertEquals(2, emtyStack.peek());
		emtyStack.push(3);
		assertEquals(3, emtyStack.peek());
		emtyStack.pop();
		assertEquals(2, emtyStack.peek());
		emtyStack.pop();
		assertEquals(1, emtyStack.peek());
		emtyStack.pop();
		emtyStack.peek();
		assertEquals(-1, emtyStack.peek());
	}

	@Test
	public void testPeekStackisEmpty() {
		assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
		int data = emtyStack.peek();
		assertEquals(data, -1);
	}

}
```

## 3. Sample Run. [2 pages]

*Screenshot or image of code execution*

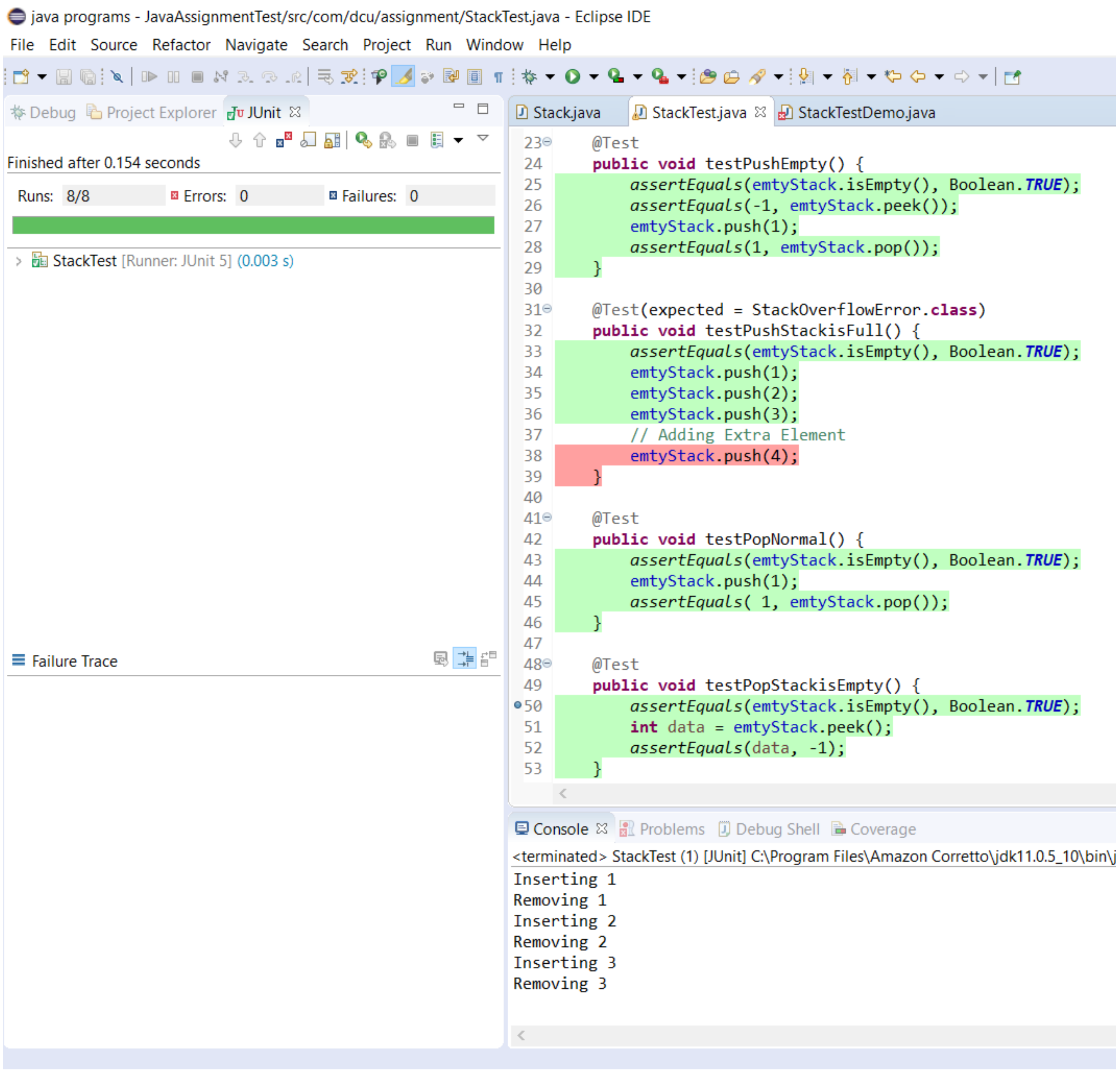
*i. Screenshot of Main Stack Code:*



```java
        if (isFull())
        {
            System.out.println("OverFlow\nProgram Terminated\n");
            throw new StackOverflowError();
        }else{
            System.out.println("Inserting " + x);
            arr[++top] = x;
        }

    }

    public int pop()
    {
        //Returns the last inserted item on the stack without removing it.
        if (isEmpty())
        {
            System.out.println("UnderFlow\nProgram Terminated");
            return -1;
        }
        System.out.println("Removing " + peek());
        return arr[top--];
    }
    public int peek()
    {

        if (!isEmpty())
            return arr[top];
        else
            //System.exit(1);
        return 0;
```

```
<terminated> Stack (2) [Java Application] C:\Program Files\Amazon Corretto\jdk11.0.5_10\bin\javaw.exe (09-Fe
Inserting 1
Inserting 2
Removing 2
Removing 1
Inserting 3
Top element is: 3
Stack size is 1
Removing 3
Stack is Empty
```

*ii.        Screenshot of Test Suit:*



java programs - JavaAssignmentTest/src/com/dcu/assignment/StackTest.java - Eclipse IDE

Finished after 0.154 seconds

Runs: 8/8    Errors: 0    Failures: 0

StackTest [Runner: JUnit 5] (0.003 s)

```java
    @Test
    public void testPushEmpty() {
        assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
        assertEquals(-1, emtyStack.peek());
        emtyStack.push(1);
        assertEquals(1, emtyStack.pop());
    }

    @Test(expected = StackOverflowError.class)
    public void testPushStackisFull() {
        assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
        emtyStack.push(1);
        emtyStack.push(2);
        emtyStack.push(3);
        // Adding Extra Element
        emtyStack.push(4);
    }

    @Test
    public void testPopNormal() {
        assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
        emtyStack.push(1);
        assertEquals( 1, emtyStack.pop());
    }

    @Test
    public void testPopStackisEmpty() {
        assertEquals(emtyStack.isEmpty(), Boolean.TRUE);
        int data = emtyStack.peek();
        assertEquals(data, -1);
    }
```

```
<terminated> StackTest (1) [JUnit] C:\Program Files\Amazon Corretto\jdk11.0.5_10\bin\j
Inserting 1
Removing 1
Inserting 2
Removing 2
Inserting 3
Removing 3
```

**4. Errors, faults and failures discovered [1 page]**
*Critically analyse the outcome of your code.*

Test Cases:

1. testIsEmpty
   a. Checks if the stack is empty
   b. Push integer if the stack is empty
   c. Checks if stack is not empty after push
   d. Pop integer
   e. Checks if stack is empty after pop

2. testPushEmpty
   a. Checks if the stack is empty
   b. Push integer if the stack is empty

3. testPushStackisFull
   a. As stack size is 3, we have to push 3 integers to make it full stack
   b. **Errors:** It didn't allow to insert 4th integer in testing
   c. **Faults:** `expected = StackOverflowError.class (in junit test case)`
   d. **Failures:** Unless we mention the StackOverflowError.class, our Junit code was bound to failures as we cannot insert integer in a full stack

4. testPopNormal
   a. Check if stack is empty
   b. Push integer
   c. assertEquals checks and does the pop operation

5. testPopStackisEmpty
   a. assertEquals checks if stack is empty
   b. **Errors:** Junit does not allow to pop if there is no integer in the stack
   c. **Faults:** `int data = emtyStack.peek();`
   d. **Failures:** Unless we mention the above code, Junit code will exit from the system which result in failure. After inserting the code, It returns -1

6. testPopStackisFull
   a. assertEquals checks if stack is empty
   b. push three integers in the stack to make it full
   c. asserEquals checks and pop integer one by one from the stack

7. testPeekNormal
   a. assertEquals checks if stack is empty
   b. push integers to the stack
   c. asserEqual peek the integer from the stack

8. testPeekStackisEmpty
   a. assertEquals checks if stack is empty

b. **Errors:** Junit does not allow to peek if there is no integer in the stack

**c. Faults:** `int data = emtyStack.peek();`

**d. Failures:** Unless we mention the above code, Junit code does not work. It returns -1

## References

1. https://junit.org/junit5/

2. https://www.java.com/en/

3. https://www.techiedelight.com/stack-implementation-in-java/

4. https://www.cs.cmu.edu/~pattis/15-1XX/15-200/handouts/junitlab/index.html