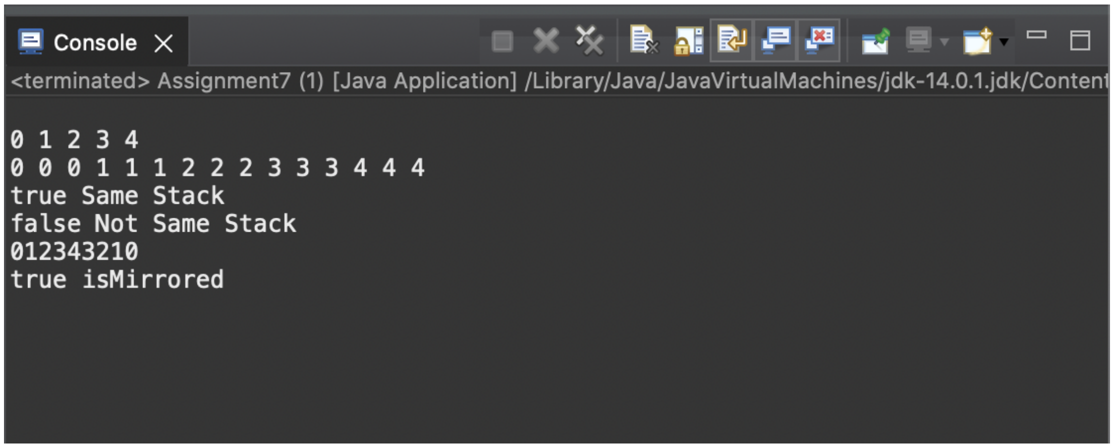
Student Name: **Natalia Reeck Zanini** Course: **CS 211**

Instructor: **Craig Niiyama**

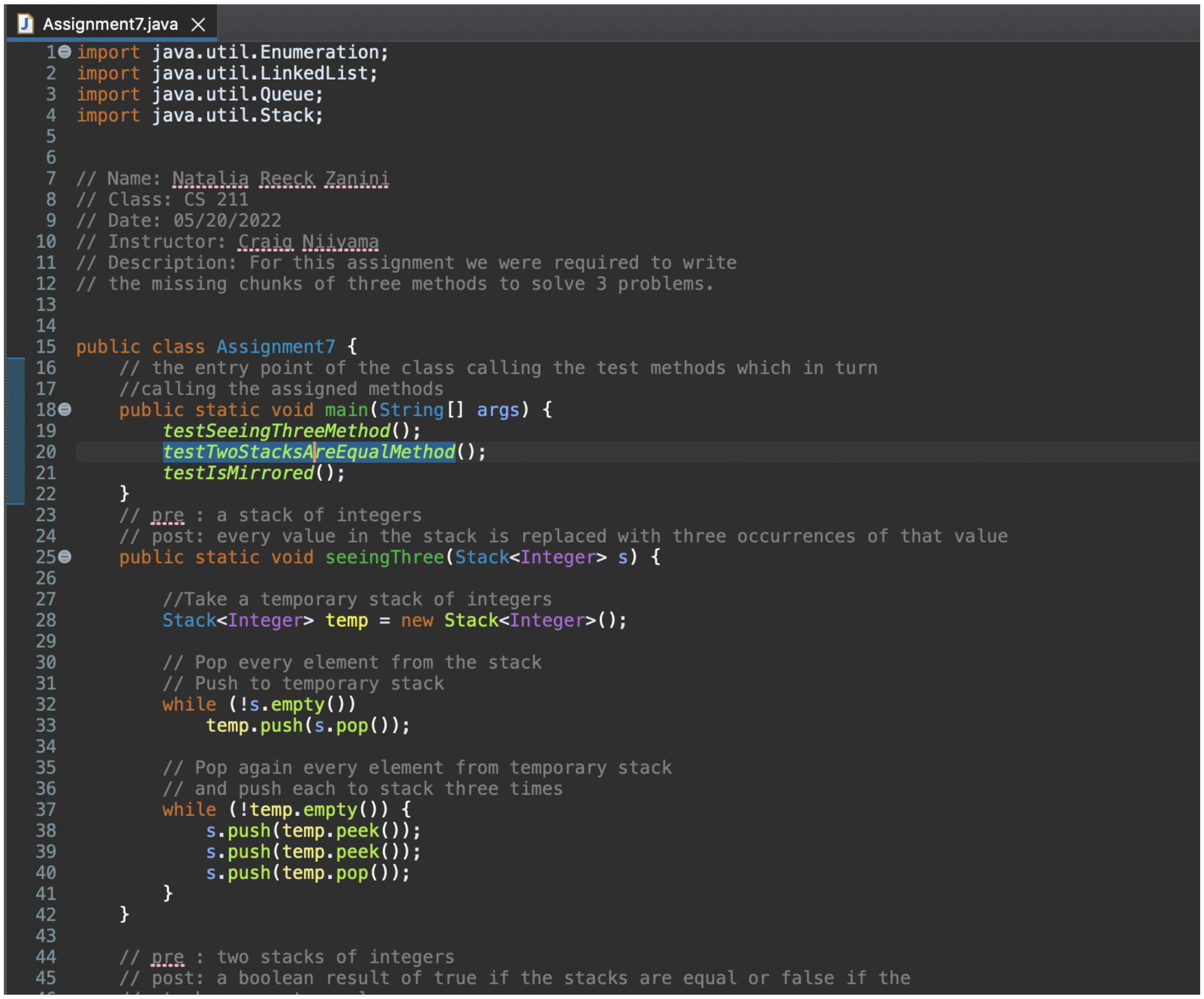
QA Document for Assignment 7

Console Screenshot:

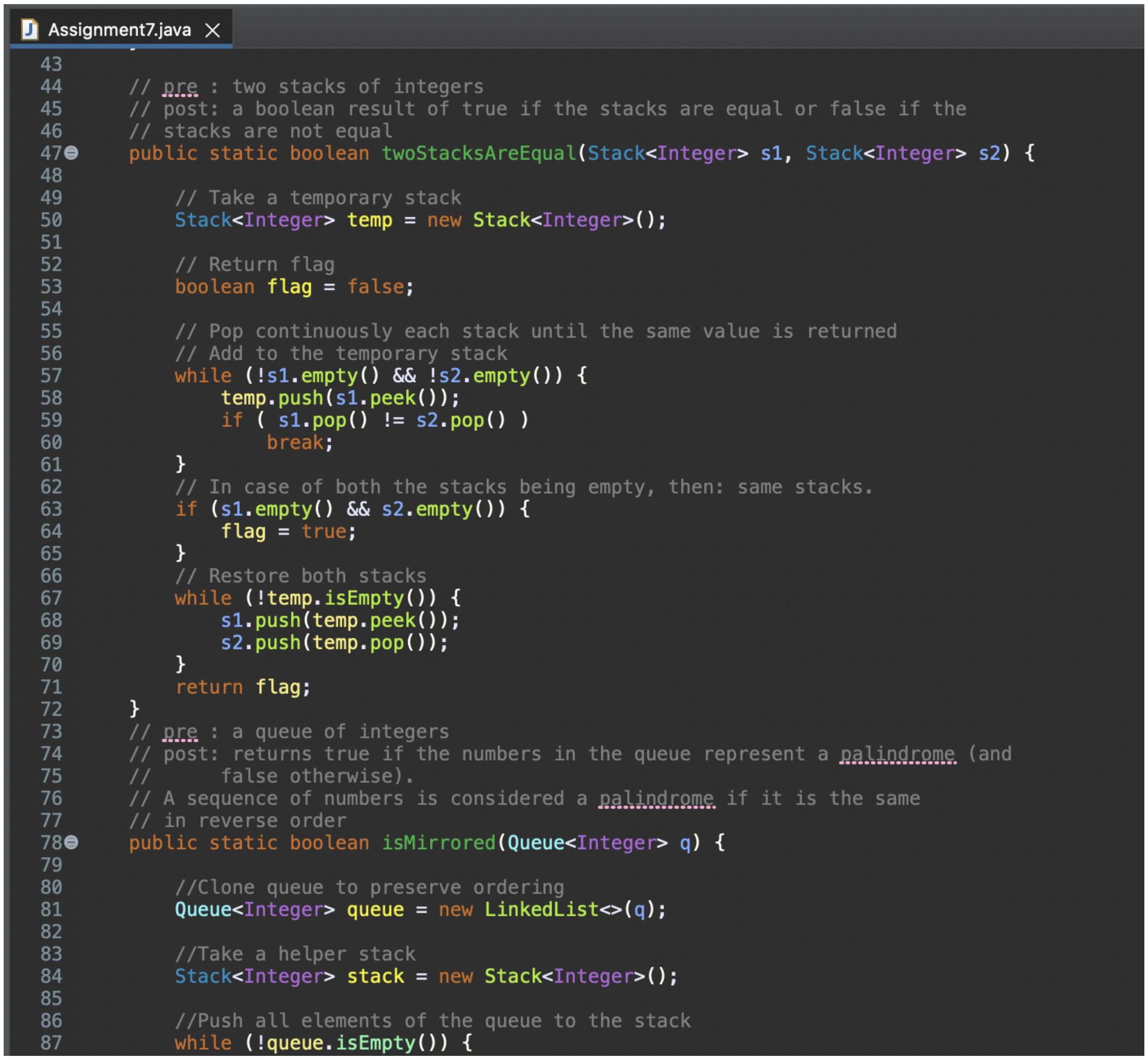


```
<terminated> Assignment7 (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Content

0 1 2 3 4
0 0 0 1 1 1 2 2 2 3 3 3 4 4 4
true Same Stack
false Not Same Stack
012343210
true isMirrored
```

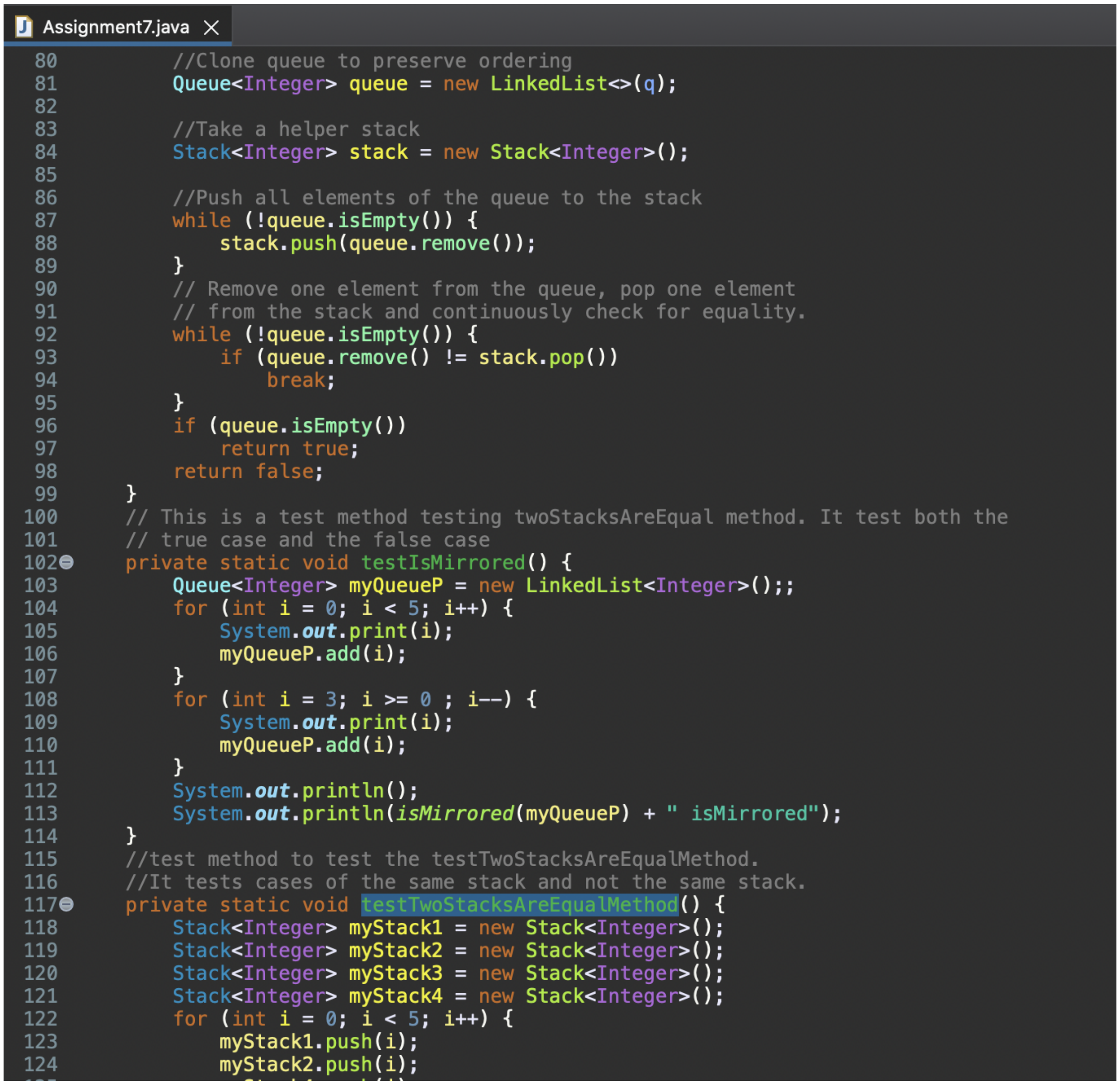
Screenshots of code:



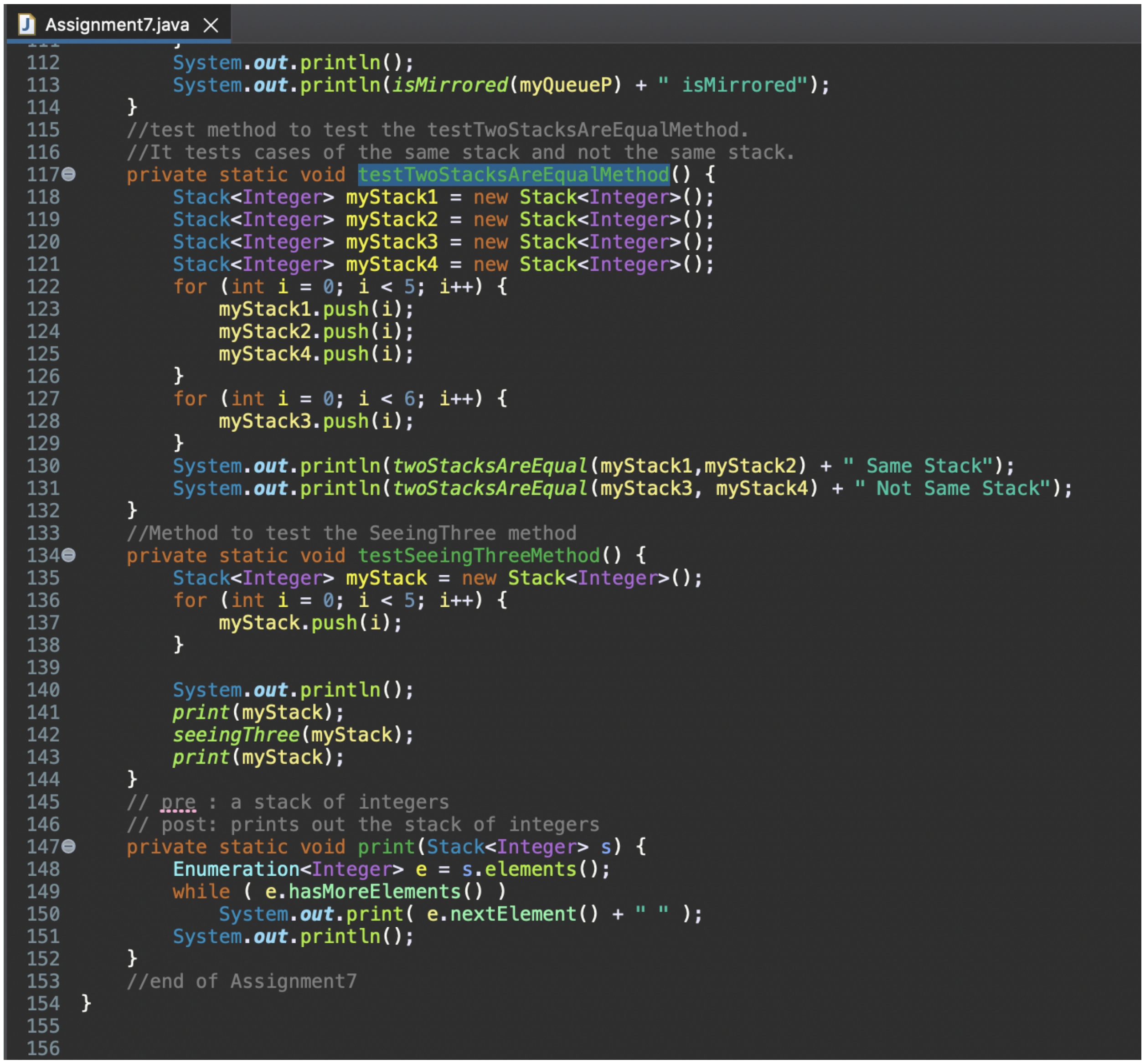
```java
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;


// Name: Natalia Reeck Zanini
// Class: CS 211
// Date: 05/20/2022
// Instructor: Craig Niiyama
// Description: For this assignment we were required to write
// the missing chunks of three methods to solve 3 problems.


public class Assignment7 {
    // the entry point of the class calling the test methods which in turn
    //calling the assigned methods
    public static void main(String[] args) {
        testSeeingThreeMethod();
        testTwoStacksAreEqualMethod();
        testIsMirrored();
    }
    // pre : a stack of integers
    // post: every value in the stack is replaced with three occurrences of that value
    public static void seeingThree(Stack<Integer> s) {

        //Take a temporary stack of integers
        Stack<Integer> temp = new Stack<Integer>();

        // Pop every element from the stack
        // Push to temporary stack
        while (!s.empty())
            temp.push(s.pop());

        // Pop again every element from temporary stack
        // and push each to stack three times
        while (!temp.empty()) {
            s.push(temp.peek());
            s.push(temp.peek());
            s.push(temp.pop());
        }
    }

    // pre : two stacks of integers
    // post: a boolean result of true if the stacks are equal or false if the
```


Assignment7.java

```java

    // pre : two stacks of integers
    // post: a boolean result of true if the stacks are equal or false if the
    // stacks are not equal
    public static boolean twoStacksAreEqual(Stack<Integer> s1, Stack<Integer> s2) {

        // Take a temporary stack
        Stack<Integer> temp = new Stack<Integer>();

        // Return flag
        boolean flag = false;

        // Pop continuously each stack until the same value is returned
        // Add to the temporary stack
        while (!s1.empty() && !s2.empty()) {
            temp.push(s1.peek());
            if ( s1.pop() != s2.pop() )
                break;
        }
        // In case of both the stacks being empty, then: same stacks.
        if (s1.empty() && s2.empty()) {
            flag = true;
        }
        // Restore both stacks
        while (!temp.isEmpty()) {
            s1.push(temp.peek());
            s2.push(temp.pop());
        }
        return flag;
    }
    // pre : a queue of integers
    // post: returns true if the numbers in the queue represent a palindrome (and
    //       false otherwise).
    // A sequence of numbers is considered a palindrome if it is the same
    // in reverse order
    public static boolean isMirrored(Queue<Integer> q) {

        //Clone queue to preserve ordering
        Queue<Integer> queue = new LinkedList<>(q);

        //Take a helper stack
        Stack<Integer> stack = new Stack<Integer>();

        //Push all elements of the queue to the stack
        while (!queue.isEmpty()) {
```


Assignment7.java

```
        //Clone queue to preserve ordering
        Queue<Integer> queue = new LinkedList<>(q);

        //Take a helper stack
        Stack<Integer> stack = new Stack<Integer>();

        //Push all elements of the queue to the stack
        while (!queue.isEmpty()) {
            stack.push(queue.remove());
        }
        // Remove one element from the queue, pop one element
        // from the stack and continuously check for equality.
        while (!queue.isEmpty()) {
            if (queue.remove() != stack.pop())
                break;
        }
        if (queue.isEmpty())
            return true;
        return false;
    }
    // This is a test method testing twoStacksAreEqual method. It test both the
    // true case and the false case
    private static void testIsMirrored() {
        Queue<Integer> myQueueP = new LinkedList<Integer>();;
        for (int i = 0; i < 5; i++) {
            System.out.print(i);
            myQueueP.add(i);
        }
        for (int i = 3; i >= 0 ; i--) {
            System.out.print(i);
            myQueueP.add(i);
        }
        System.out.println();
        System.out.println(isMirrored(myQueueP) + " isMirrored");
    }
    //test method to test the testTwoStacksAreEqualMethod.
    //It tests cases of the same stack and not the same stack.
    private static void testTwoStacksAreEqualMethod() {
        Stack<Integer> myStack1 = new Stack<Integer>();
        Stack<Integer> myStack2 = new Stack<Integer>();
        Stack<Integer> myStack3 = new Stack<Integer>();
        Stack<Integer> myStack4 = new Stack<Integer>();
        for (int i = 0; i < 5; i++) {
            myStack1.push(i);
            myStack2.push(i);
```



```java
        System.out.println();
        System.out.println(isMirrored(myQueueP) + " isMirrored");
    }
    //test method to test the testTwoStacksAreEqualMethod.
    //It tests cases of the same stack and not the same stack.
    private static void testTwoStacksAreEqualMethod() {
        Stack<Integer> myStack1 = new Stack<Integer>();
        Stack<Integer> myStack2 = new Stack<Integer>();
        Stack<Integer> myStack3 = new Stack<Integer>();
        Stack<Integer> myStack4 = new Stack<Integer>();
        for (int i = 0; i < 5; i++) {
            myStack1.push(i);
            myStack2.push(i);
            myStack4.push(i);
        }
        for (int i = 0; i < 6; i++) {
            myStack3.push(i);
        }
        System.out.println(twoStacksAreEqual(myStack1,myStack2) + " Same Stack");
        System.out.println(twoStacksAreEqual(myStack3, myStack4) + " Not Same Stack");
    }
    //Method to test the SeeingThree method
    private static void testSeeingThreeMethod() {
        Stack<Integer> myStack = new Stack<Integer>();
        for (int i = 0; i < 5; i++) {
            myStack.push(i);
        }

        System.out.println();
        print(myStack);
        seeingThree(myStack);
        print(myStack);
    }
    // pre : a stack of integers
    // post: prints out the stack of integers
    private static void print(Stack<Integer> s) {
        Enumeration<Integer> e = s.elements();
        while ( e.hasMoreElements() )
            System.out.print( e.nextElement() + " " );
        System.out.println();
    }
    //end of Assignment7
}
```