Project 2

Matthew Mucha

CMSC 203

Professor Thai

Approach:

The first step in implementing Project 2 was to create generic data structures that followed the given interfaces, such as MyQueue and MyStack. These classes had essential features like size tracking, dequeuing/popping, and enqueueing/pushing, and they used an ArrayList for effective element storage. Robust techniques for determining fullness, emptiness, and string conversion were emphasized. Three essential activities were the focus of the Notation utility class: evaluatePostfix, which evaluates postfix expressions, infix to postfix conversion (infixToPostfix), and postfix to infix conversion (postfixToInfix). Using the generic queue (MyQueue) to convert infix to postfix allowed for a more efficient workflow, according to the designated algorithms for every task. Exception classes, which cover situations like incorrect notations and stack/queue overflow/underflow, were developed to handle mistakes properly. thorough examination that covers both legitimate and illegitimate phrases.

What I learned:

I now have a better knowledge of how to create generic data structures, particularly generic stacks (MyStack) and queues (MyQueue), thanks to this project. Stack and queue data structures were practically used through the conversion algorithms for evaluating postfix expressions, converting postfix to infix, and converting postfix to postfix. I also became better at handling exceptions and making sure the implemented classes are resilient.

Issues:

One difficulty I ran into was effectively handling the conversion methods, particularly when it came to operator precedence and parenthesis. The first challenges were related to making sure the conversion operations were right and managing exceptions properly.

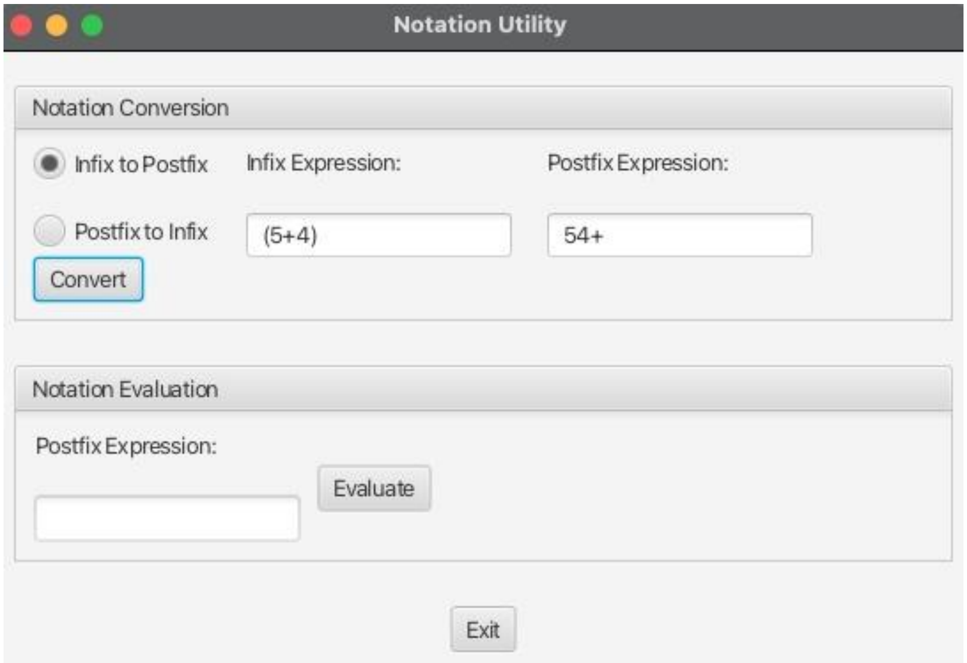
What I would of done differently:

With hindsight, I would have handled parenthesis in a more methodical manner. The implementation may have been sped up with a more thorough design for handling parenthetical edge circumstances. Additionally, testing and debugging would have been easier if the conversion methods had been divided into smaller, testable parts.

Application for the future:

The ideas acquired throughout this study have broad applications in several contexts. It will be helpful to comprehend and use generic data structures when creating more intricate applications that need algorithmic operations and data manipulation. Compiler design fundamentals include conversion methods for various notations, which may be expanded to handle increasingly complex mathematical expressions.

Infix to Postfix:



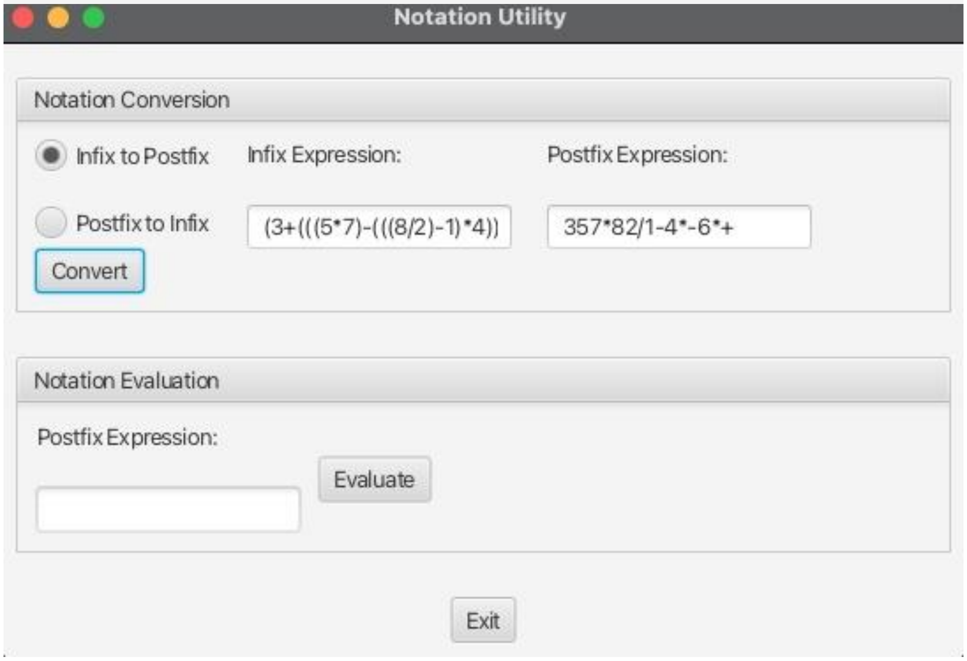
● Infix to Postfix

○ Postfix to Infix

Infix Expression: (5+4)

Postfix Expression: 54+

Postfix Expression:



● Infix to Postfix

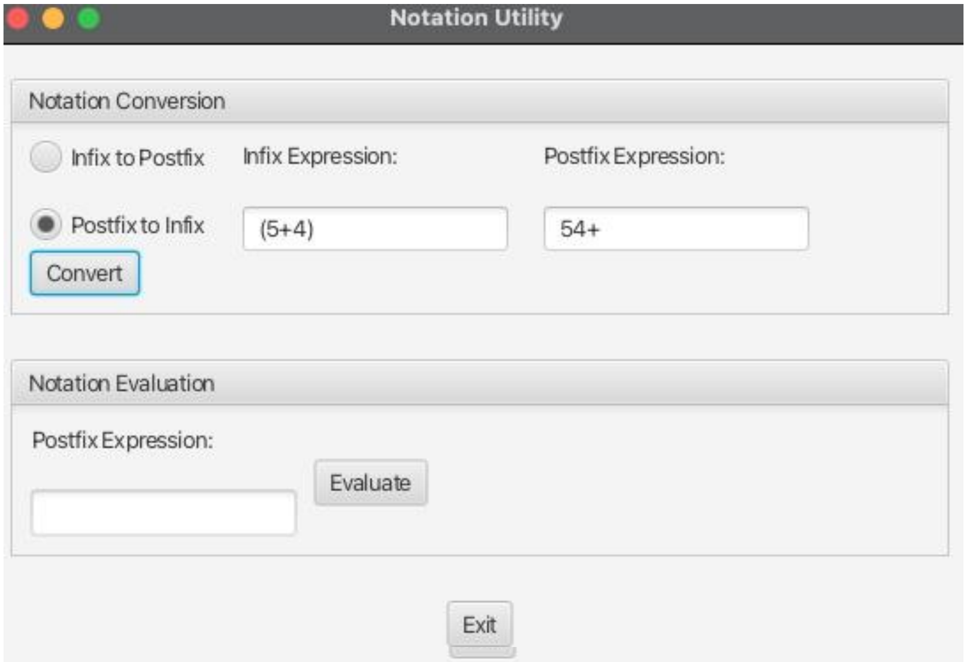
○ Postfix to Infix

Infix Expression: (3+(((5*7)-(((8/2)-1)*4))

Postfix Expression: 357*82/1-4*-6*+

Postfix Expression:

Postfix to infix:

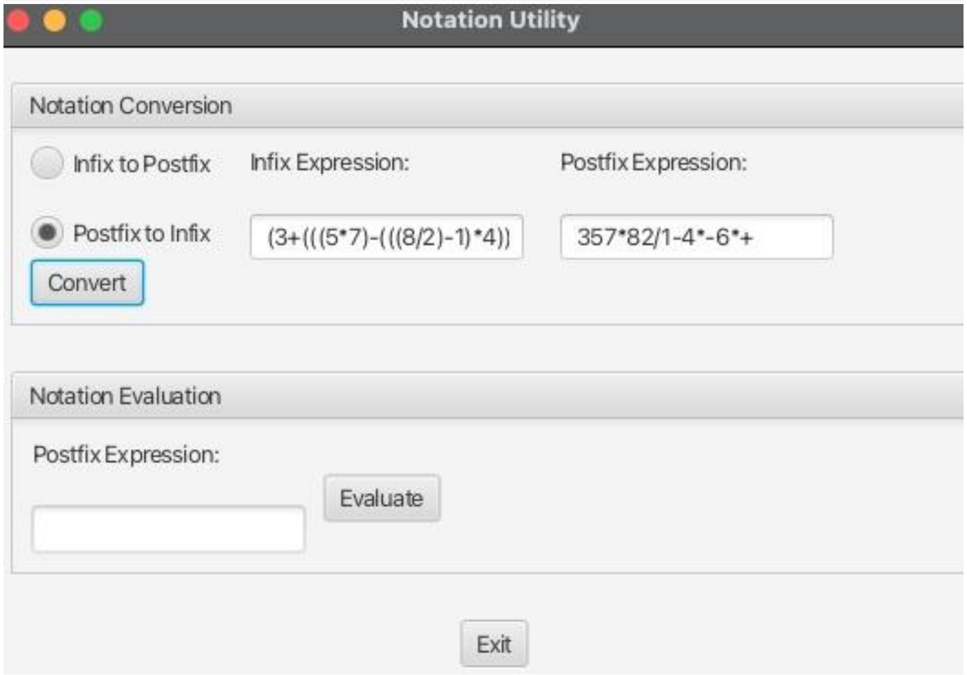


○ Infix to Postfix

● Postfix to Infix

Infix Expression: (5+4)

Postfix Expression: 54+



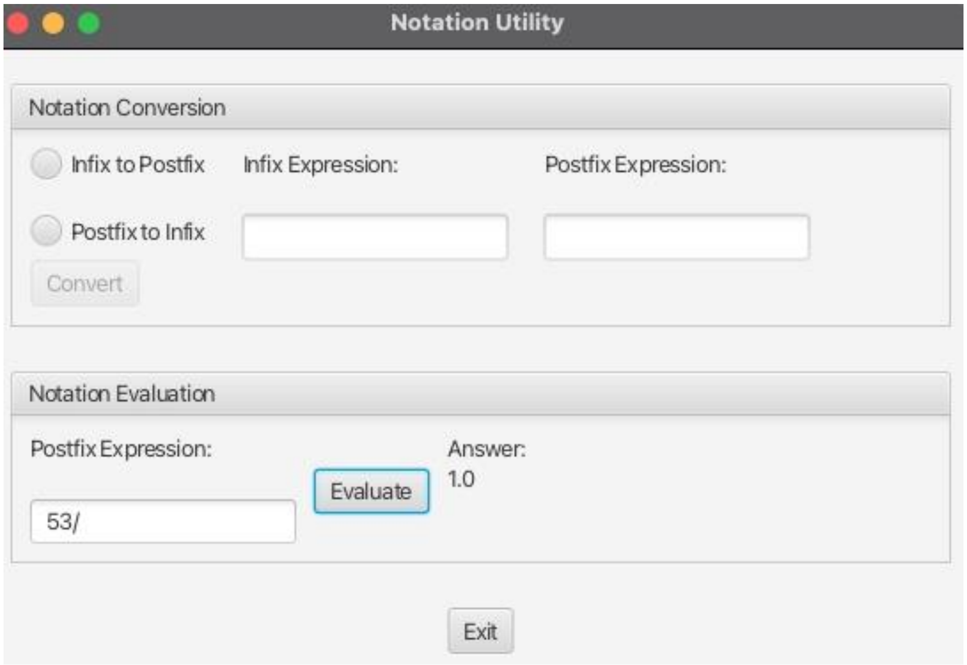
○ Infix to Postfix

● Postfix to Infix

Infix Expression: (3+(((5*7)-(((8/2)-1)*4))

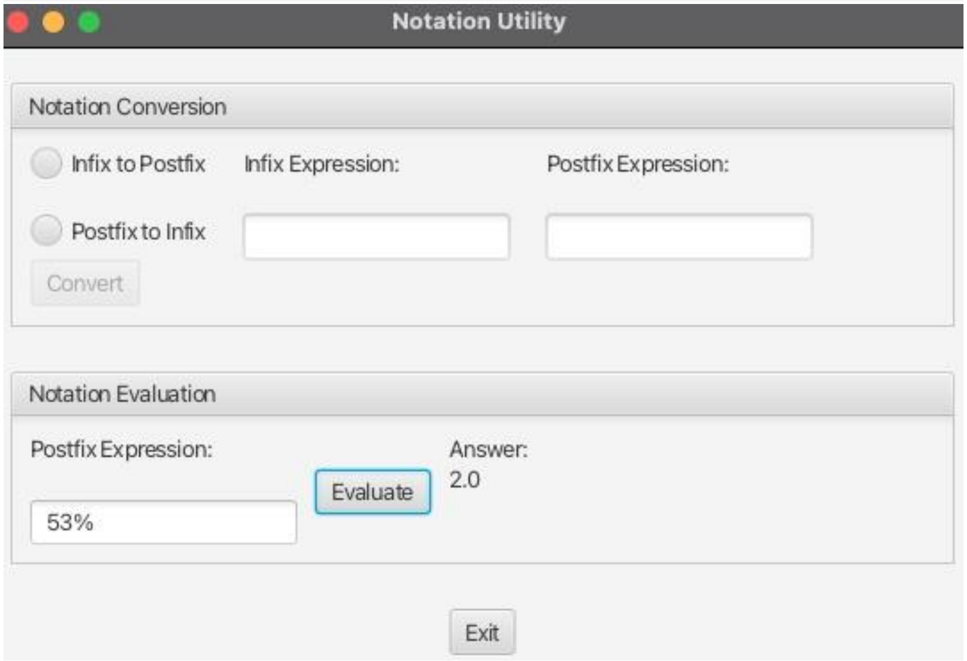
Postfix Expression: 357*82/1-4*-6*+

Postfix Expression:



Postfix Expression: 53/

Answer: 1.0

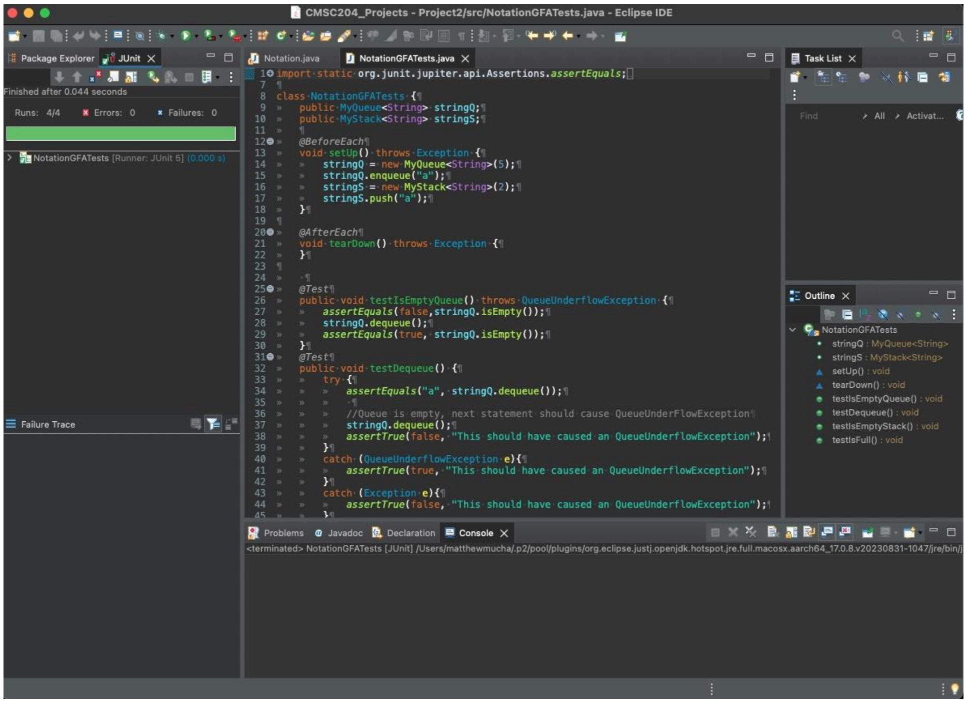


Postfix Expression: 53%

Answer: 2.0

Test cases:

NotationGFATests:



CMSC204_Projects - Project2/src/NotationGFATests.java - Eclipse IDE

Finished after 0.044 seconds

Runs: 4/4 Errors: 0 Failures: 0

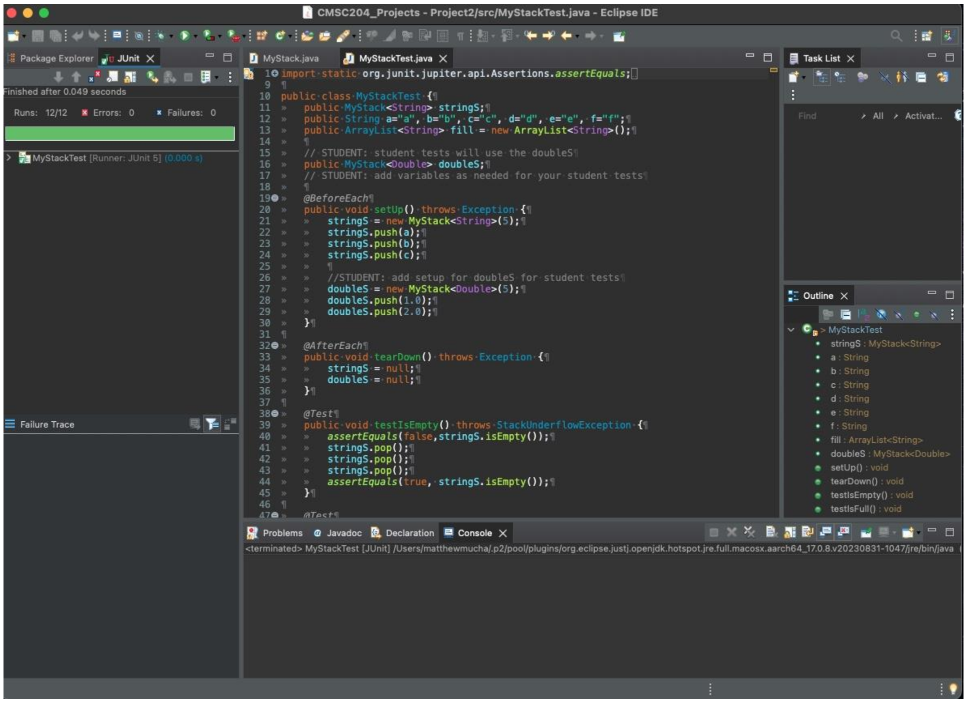
NotationGFATests [Runner: JUnit 5] (0.000 s)

```java
import static org.junit.jupiter.api.Assertions.assertEquals;

class NotationGFATests {
    public MyQueue<String> stringQ;
    public MyStack<String> stringS;
    
    @BeforeEach
    void setUp() throws Exception {
        stringQ = new MyQueue<String>(5);
        stringQ.enqueue("a");
        stringS = new MyStack<String>(2);
        stringS.push("a");
    }

    @AfterEach
    void tearDown() throws Exception {
    }

     
    @Test
    public void testIsEmptyQueue() throws QueueUnderflowException {
        assertEquals(false,stringQ.isEmpty());
        stringQ.dequeue();
        assertEquals(true, stringQ.isEmpty());
    }
    @Test
    public void testDequeue() {
        try {
            assertEquals("a", stringQ.dequeue());
             
            //Queue is empty, next statement should cause QueueUnderFlowException
            stringQ.dequeue();
            assertTrue(false, "This should have caused an QueueUnderflowException");
        }
        catch (QueueUnderflowException e){
            assertTrue(true, "This should have caused an QueueUnderflowException");
        }
        catch (Exception e){
            assertTrue(false, "This should have caused an QueueUnderflowException");
        }
```

```
NotationGFATests
    stringQ : MyQueue<String>
    stringS : MyStack<String>
    setUp() : void
    tearDown() : void
    testIsEmptyQueue() : void
    testDequeue() : void
    testIsEmptyStack() : void
    testIsFull() : void
```

```
<terminated> NotationGFATests [JUnit] /Users/matthewmucha/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.8.v20230831-1047/jre/bin/j
```

MyStackTest:



CMSC204_Projects - Project2/src/MyStackTest.java - Eclipse IDE

Finished after 0.049 seconds

Runs: 12/12 Errors: 0 Failures: 0

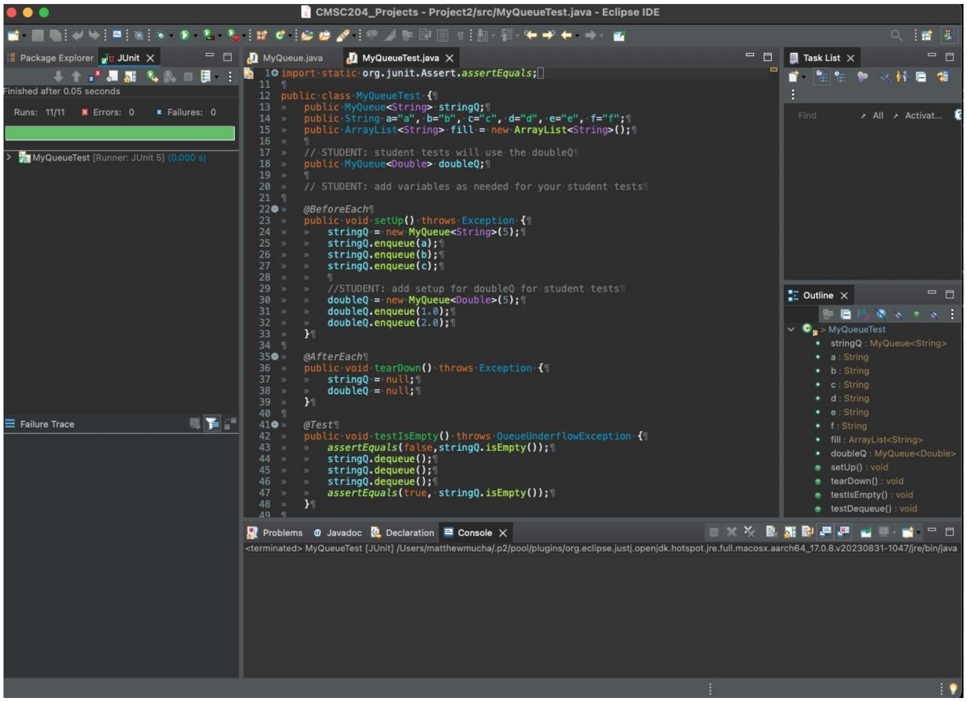
MyStackTest [Runner: JUnit 5] (0.000 s)

```java
import static org.junit.jupiter.api.Assertions.assertEquals;

public class MyStackTest {
    public MyStack<String> stringS;
    public String a="a", b="b", c="c", d="d", e="e", f="f";
    public ArrayList<String> fill = new ArrayList<String>();
    
    // STUDENT: student tests will use the doubleS
    public MyStack<Double> doubleS;
    // STUDENT: add variables as needed for your student tests
    
    @BeforeEach
    public void setUp() throws Exception {
        stringS = new MyStack<String>(5);
        stringS.push(a);
        stringS.push(b);
        stringS.push(c);
        
        //STUDENT: add setup for doubleS for student tests
        doubleS = new MyStack<Double>(5);
        doubleS.push(1.0);
        doubleS.push(2.0);
    }

    @AfterEach
    public void tearDown() throws Exception {
        stringS = null;
        doubleS = null;
    }

    @Test
    public void testIsEmpty() throws StackUnderflowException {
        assertEquals(false,stringS.isEmpty());
        stringS.pop();
        stringS.pop();
        stringS.pop();
        assertEquals(true, stringS.isEmpty());
    }

    @Test
```

```
> MyStackTest
    stringS : MyStack<String>
    a : String
    b : String
    c : String
    d : String
    e : String
    f : String
    fill : ArrayList<String>
    doubleS : MyStack<Double>
    setUp() : void
    tearDown() : void
    testIsEmpty() : void
    testIsFull() : void
```

```
<terminated> MyStackTest [JUnit] /Users/matthewmucha/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.8.v20230831-1047/jre/bin/java (
```

MyQueueTest:



CMSC204_Projects - Project2/src/MyQueueTest.java - Eclipse IDE

Finished after 0.05 seconds

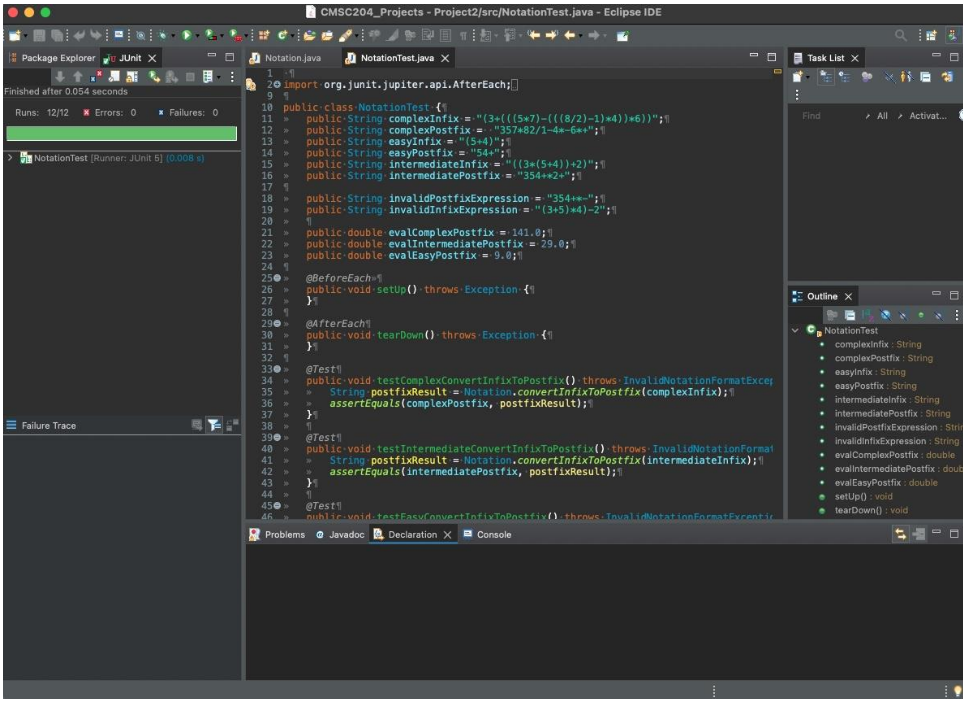
Runs: 11/11 Errors: 0 Failures: 0

MyQueueTest [Runner: JUnit 5] (0.000 s)

```java
import static org.junit.Assert.assertEquals;

public class MyQueueTest {
    public MyQueue<String> stringQ;
    public String a="a", b="b", c="c", d="d", e="e", f="f";
    public ArrayList<String> fill = new ArrayList<String>();

    // STUDENT: student tests will use the doubleQ
    public MyQueue<Double> doubleQ;

    // STUDENT: add variables as needed for your student tests

    @BeforeEach
    public void setUp() throws Exception {
        stringQ = new MyQueue<String>(5);
        stringQ.enqueue(a);
        stringQ.enqueue(b);
        stringQ.enqueue(c);

        //STUDENT: add setup for doubleQ for student tests
        doubleQ = new MyQueue<Double>(5);
        doubleQ.enqueue(1.0);
        doubleQ.enqueue(2.0);
    }

    @AfterEach
    public void tearDown() throws Exception {
        stringQ = null;
        doubleQ = null;
    }

    @Test
    public void testIsEmpty() throws QueueUnderflowException {
        assertEquals(false,stringQ.isEmpty());
        stringQ.dequeue();
        stringQ.dequeue();
        stringQ.dequeue();
        assertEquals(true, stringQ.isEmpty());
    }
```

```
<terminated> MyQueueTest [JUnit] /Users/matthewmucha/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.8.v20230831-1047/jre/bin/java
```

NotationTest:



CMSC204_Projects - Project2/src/NotationTest.java - Eclipse IDE

Finished after 0.054 seconds

Runs: 12/12 Errors: 0 Failures: 0

NotationTest [Runner: JUnit 5] (0.008 s)

```java

import org.junit.jupiter.api.AfterEach;

public class NotationTest {
    public String complexInfix = "(3+(((5*7)-(((8/2)-1)*4))*6))";
    public String complexPostfix =  "357*82/1-4*-6*+";
    public String easyInfix = "(5+4)";
    public String easyPostfix = "54+";
    public String intermediateInfix = "((3*(5+4))+2)";
    public String intermediatePostfix = "354+*2+";

    public String invalidPostfixExpression = "354+*-";
    public String invalidInfixExpression = "(3+5)*4)-2";

    public double evalComplexPostfix = 141.0;
    public double evalIntermediatePostfix = 29.0;
    public double evalEasyPostfix = 9.0;

    @BeforeEach
    public void setUp() throws Exception {
    }

    @AfterEach
    public void tearDown() throws Exception {
    }

    @Test
    public void testComplexConvertInfixToPostfix() throws InvalidNotationFormatExce
        String postfixResult = Notation.convertInfixToPostfix(complexInfix);
        assertEquals(complexPostfix, postfixResult);
    }

    @Test
    public void testIntermediateConvertInfixToPostfix() throws InvalidNotationFormat
        String postfixResult = Notation.convertInfixToPostfix(intermediateInfix);
        assertEquals(intermediatePostfix, postfixResult);
    }

    @Test
    public void testEasyConvertInfixToPostfix() throws InvalidNotationFormatExceptio
```