XUnit Assignment

Name: Róisín Ní Bhriain
Student Number: 23269640
Email: roisin.nibhriain3@mail.dcu.ie
Program: MCM Secure Software Engineering
Module Code: CA650
Submission: XUnit Assignment
Word Count: 429 (without code)

Róisín Ní Bhriain - 23269640

Signed Name: Róisín Ní Bhriain
Date: 09-02-2024

Reflection

Introduction

For this assignment I decided to use JUnit testing. I had some experience using JUnit before but never JUnit 5. I first created the methods mentioned in the assignment specification and then began testing both the initialisation and the normal functionality. Then I added some edge cases to make sure these were all covered. I fixed any faults I came across as I went along and continued running the tests. The main faults I came across were:

First Problem

The initialisation test discovered that the remove function had a fault where the wrong logic was being used in an if statement and creating a failure where the tail variable was being updated when it shouldn't have been as there were no values in the queue. This in turn was causing an error in the test where the howMany function was returning an incorrect value and the test was failing to be equal to zero (instead was equal to -1).

Second Problem

The second problem discovered was in the testAdding test. This discovered that the add function had a fault where there was no check to ensure that the queue was not full. THis created a failure where the tail was being used as an index that was out of bounds. This caused an error message to be displayed.

Third Problem

The third problem was discovered with the remove test. The fault was that there was no check to ensure that the queue wasn't empty before removing an item from the queue. This caused a failure where the tail was being decremented when it shouldn't have been. This error showed up in the test when attempting to add to the queue after it had just had too many items removed.

Conclusion

Testing the code was very helpful in discovering faults that I hadn't realised were there. As the code was simple I presumed I wouldn't have many faults but it was quicker in discovering them than debugging manually. Using the @before and @after functionality in JUnit was also something I had never before and it was useful for the testing in this case to set up and tear down an empty Queue object before each test. I also hadn't used any of the assertThrows functionality (which used to be expected in JUnit 4). It was useful to make sure that any exceptions were thrown. Overall testing was very helpful to make sure the queue class that I created worked the way I expected. It involved more testing than I expected considering it was such a small piece of code.

Queue Code

```
package Queue;

public class Queue {

        final int DEFAULT_SIZE = 10;
        private int tail = -1;
        private Object [] list;

        Queue () {
                create( DEFAULT_SIZE );
        }

        public void create ( int maxNum ) {
                list = new Object[ maxNum ];
                tail = -1;
        }

        public int maxSize () {
                return list.length;
        }

        public void add ( Object newObj ) {

                if ( tail + 1 < maxSize() ) {
                        tail++;

                        list[ tail ] = newObj;
                } else {
                        System.out.print( "Queue is full!" );
                }
        }

        public Object remove () {

                if ( !isEmpty() ) {
                        Object current = list[ 0 ];
                        for ( int i = 0; i < tail ; i++ ) {
                                list[ i ] = list[ i + 1 ];
                        }

                        tail--;

                        return current;
                }

                System.out.print("Queue was empty!");


                return null;

        }
        public int howMany () {
                return tail + 1;
        }

        public Boolean isEmpty () {

                if ( howMany() == 0     ) {

                        return true;

                }
                return false;
        }

}
```

XUnit Code

```
package Queue;

import static org.junit.jupiter.api.Assertions.*;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import org.junit.*;
import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Test;

class QueueTest {
        Queue queue;

        @BeforeEach
        void setUp () {
                queue = new Queue();
        }

        @Test
        void testInitialisation() {
                assertTrue(  queue.maxSize() == 10, "Length is equal to default size" );

                assertTrue( queue.remove() == null, "Make sure there is null returned with
remove function" );

                assertTrue( queue.howMany() == 0, "Make sure there is nothing in the list"
);

                assertTrue( queue.isEmpty(), "Make sure list returns empty" );

        }
```

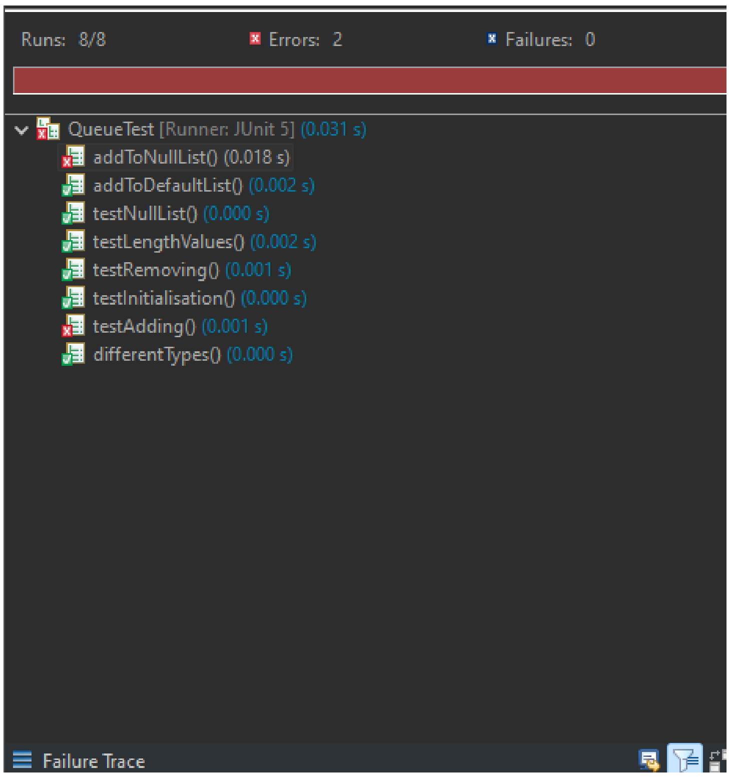
```
	// This test sets the list max num to 0
	@Test
	void testNullList () {
		queue.create( 0 );

		assertTrue(  queue.maxSize() == 0, "Length is equal to 0" );

		assertTrue( queue.remove() == null, "Make sure there is null returned with
remove function" );

		assertTrue( queue.howMany() == 0, "Make sure there is nothing in the list"
);

		assertTrue( queue.isEmpty(), "Make sure list returns empty" );

	}

	@Test
	void addToNullList () {
		queue.create( 0 );

		ByteArrayOutputStream outContent = new ByteArrayOutputStream();
		System.setOut(new PrintStream(outContent));

		Object obj = new Object();
		queue.add( obj );

		String expectedOutput  = "Queue is full!" ;

		// Do the actual assertion.
		assertEquals(expectedOutput, outContent.toString());

	}

	@Test
	void addToDefaultList () {

		Object obj = new Object();
		queue.add( obj );

		assertTrue( queue.remove() == obj, "ensure object added is same as object
removed");

	}

	@Test
	void testLengthValues () {

		assertThrows( NegativeArraySizeException.class, () -> {
			queue.create( -1 );
		});

		assertThrows( OutOfMemoryError.class, () -> {
			queue.create( Integer.MAX_VALUE );
```

```
                    });

                    queue.create( 20 );

                    assertTrue(  queue.maxSize() == 20, "Length is equal to 20" );

                    assertTrue( queue.remove() == null, "Make sure there is null returned with
remove function" );

                    assertTrue( queue.howMany() == 0, "Make sure there is nothing in the list"
);

                    assertTrue( queue.isEmpty(), "Make sure list returns empty" );
          }

          @Test
          void testAdding () {
                    queue.create( 5 );

                    for (int i = 0; i < 5; i++ ) {
                              Object obj = new Object();
                              queue.add( obj );
                              System.out.println("hello");
                              assertTrue( queue.howMany() == i + 1, "Make sure list returns
correct number of elements" );
                    }

                    assertTrue( !queue.isEmpty(), "Make sure queue does not return empty" );

                    Object obj = new Object();
                    queue.add( obj );

                    ByteArrayOutputStream outContent = new ByteArrayOutputStream();
                    System.setOut(new PrintStream(outContent));

                    queue.add( obj );

                    String expectedOutput  = "Queue is full!" ;

                    // Do the actual assertion.
                    assertEquals(expectedOutput, outContent.toString());
          }

          @Test
          void testRemoving () {
                    queue.create( 3 );

                    queue.add("1");
                    queue.add("2");
                    queue.add("3");

                    assertTrue( queue.remove() == "1", "Make sure first object is removed" );
                    assertTrue( queue.howMany() == 2, "make sure correct number in queue"
);
```

```
                assertTrue( queue.remove() == "2", "Make sure first object is removed" );
                assertTrue( queue.howMany() == 1, "make sure correct number in queue"
);
                assertTrue( queue.remove() == "3", "Make sure first object is removed" );
                assertTrue( queue.howMany() == 0, "make sure correct number in queue"
);
                assertTrue( queue.isEmpty(), "Make sure queue is empty" );

                ByteArrayOutputStream outContent = new ByteArrayOutputStream();
                System.setOut(new PrintStream(outContent));

                queue.remove();

                String expectedOutput  = "Queue was empty!" ;

                assertEquals(expectedOutput, outContent.toString());

        }

        @Test
        void differentTypes () {
                queue.create( 3 );

                Object obj = new Object();
                queue.add(obj);
                queue.add("1");
                queue.add(14);

                assertTrue( queue.remove() == obj, "Make sure first object is removed" );
                assertTrue( queue.remove() == "1", "Make sure first object is removed" );
                assertTrue( queue.remove().equals(14), "Make sure first object is removed"
);

        }

        @Test
        void removeAndAdd () {
                queue.create( 3 );

                ByteArrayOutputStream outContent = new ByteArrayOutputStream();
                System.setOut(new PrintStream(outContent));

                assertTrue( queue.remove() == null, "Make sure first object is null" );

                String expectedOutput  = "Queue was empty!" ;

                assertEquals(expectedOutput, outContent.toString());

                queue.add(1);
                assertTrue( queue.remove().equals(1), "Make sure first object has been
added correctly" );

        }
        @AfterEach
        void tearDown () {
                queue = null;
        }

}
```

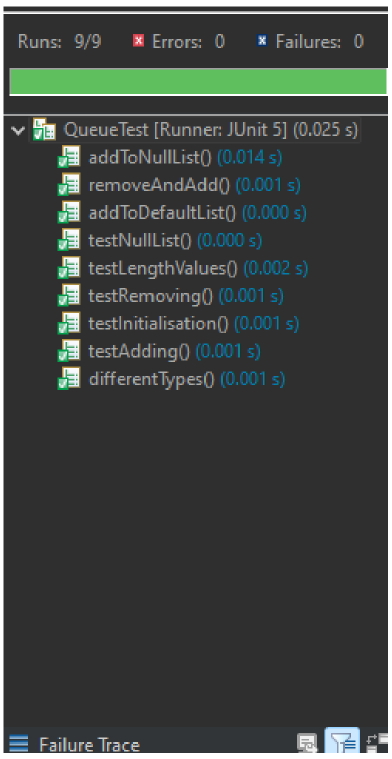
Sample Runs

With Some Errors:



```
Runs: 8/8          Errors: 2          Failures: 0

QueueTest [Runner: JUnit 5] (0.031 s)
    addToNullList() (0.018 s)
    addToDefaultList() (0.002 s)
    testNullList() (0.000 s)
    testLengthValues() (0.002 s)
    testRemoving() (0.001 s)
    testInitialisation() (0.000 s)
    testAdding() (0.001 s)
    differentTypes() (0.000 s)
```

Final Run:



```
Runs: 9/9    Errors: 0    Failures: 0

QueueTest [Runner: JUnit 5] (0.025 s)
    addToNullList() (0.014 s)
    removeAndAdd() (0.001 s)
    addToDefaultList() (0.000 s)
    testNullList() (0.000 s)
    testLengthValues() (0.002 s)
    testRemoving() (0.001 s)
    testInitialisation() (0.001 s)
    testAdding() (0.001 s)
    differentTypes() (0.001 s)
```