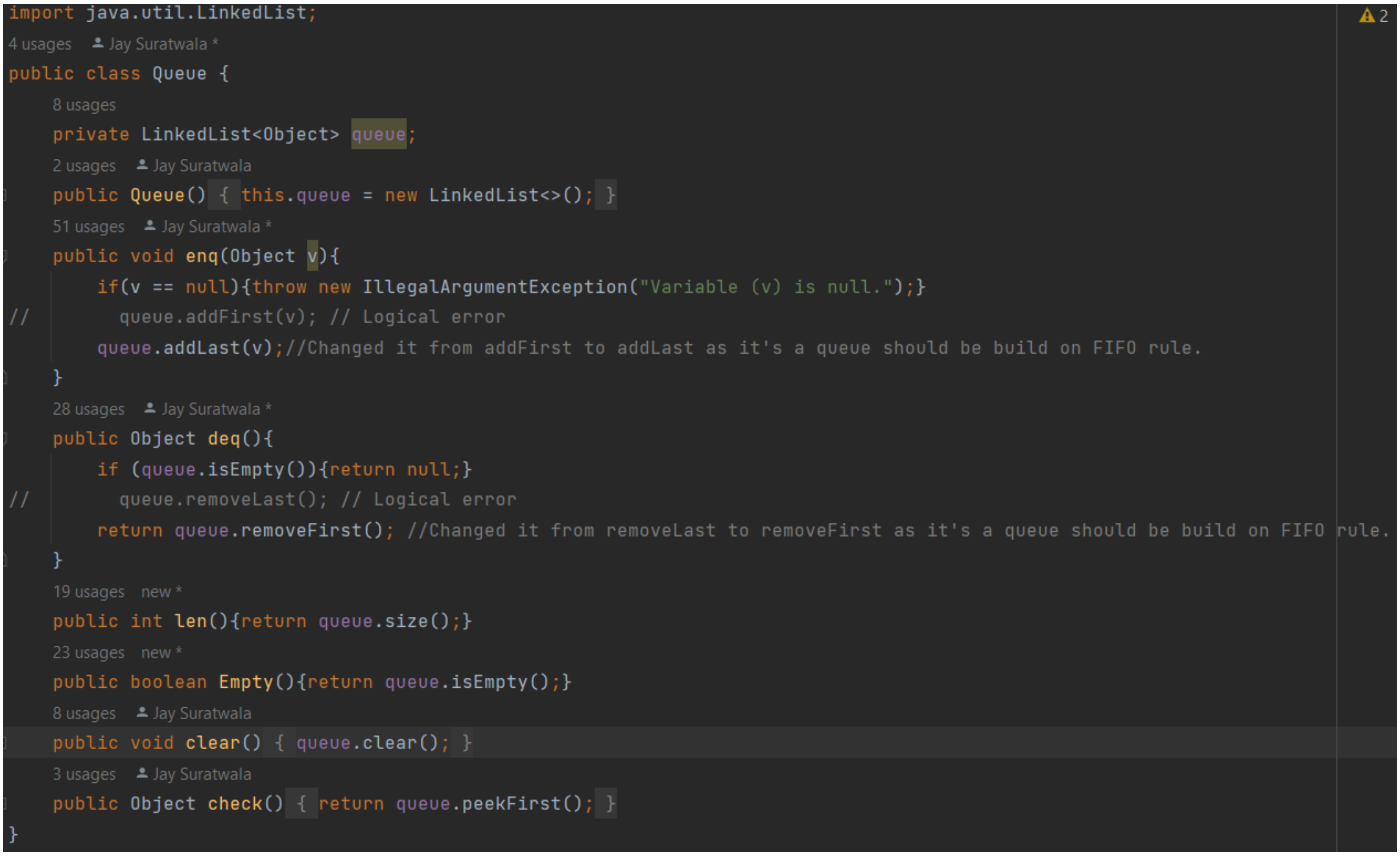
CSC 1137 XUnit Assignment

Jay Suratwala, 11103

Msc Secure Software Engineering(MCM 1), DCU

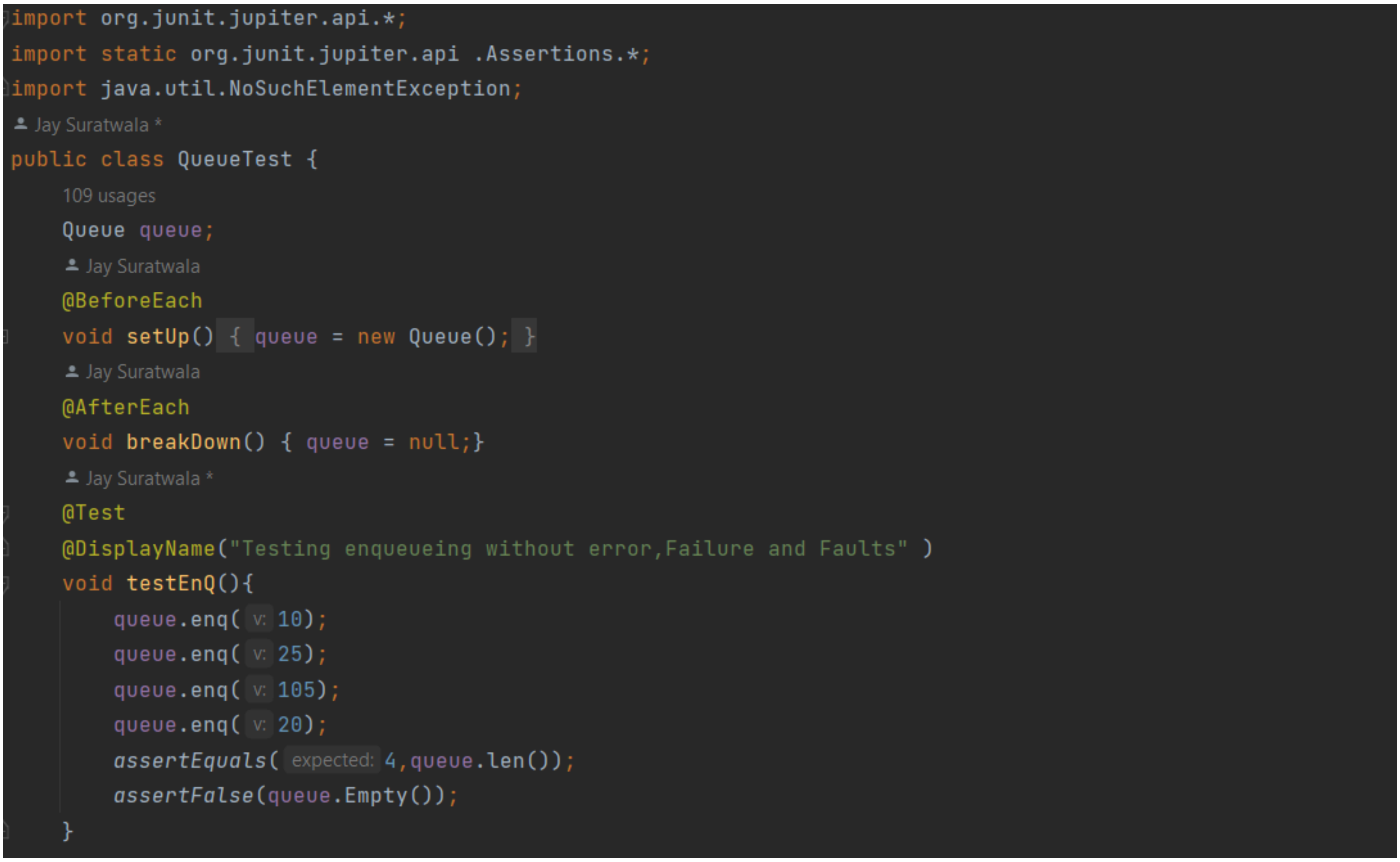
1. Queue.java Class Code (Java)



```
import java.util.LinkedList;
4 usages   Jay Suratwala *
public class Queue {
    8 usages
    private LinkedList<Object> queue;
    2 usages   Jay Suratwala
    public Queue() { this.queue = new LinkedList<>(); }
    51 usages   Jay Suratwala *
    public void enq(Object v){
        if(v == null){throw new IllegalArgumentException("Variable (v) is null.");}
//          queue.addFirst(v); // Logical error
        queue.addLast(v);//Changed it from addFirst to addLast as it's a queue should be build on FIFO rule.
    }
    28 usages   Jay Suratwala *
    public Object deq(){
        if (queue.isEmpty()){return null;}
//          queue.removeLast(); // Logical error
        return queue.removeFirst(); //Changed it from removeLast to removeFirst as it's a queue should be build on FIFO rule.
    }
    19 usages   new *
    public int len(){return queue.size();}
    23 usages   new *
    public boolean Empty(){return queue.isEmpty();}
    8 usages   Jay Suratwala
    public void clear() { queue.clear(); }
    3 usages   Jay Suratwala
    public Object check() { return queue.peekFirst(); }
}
```

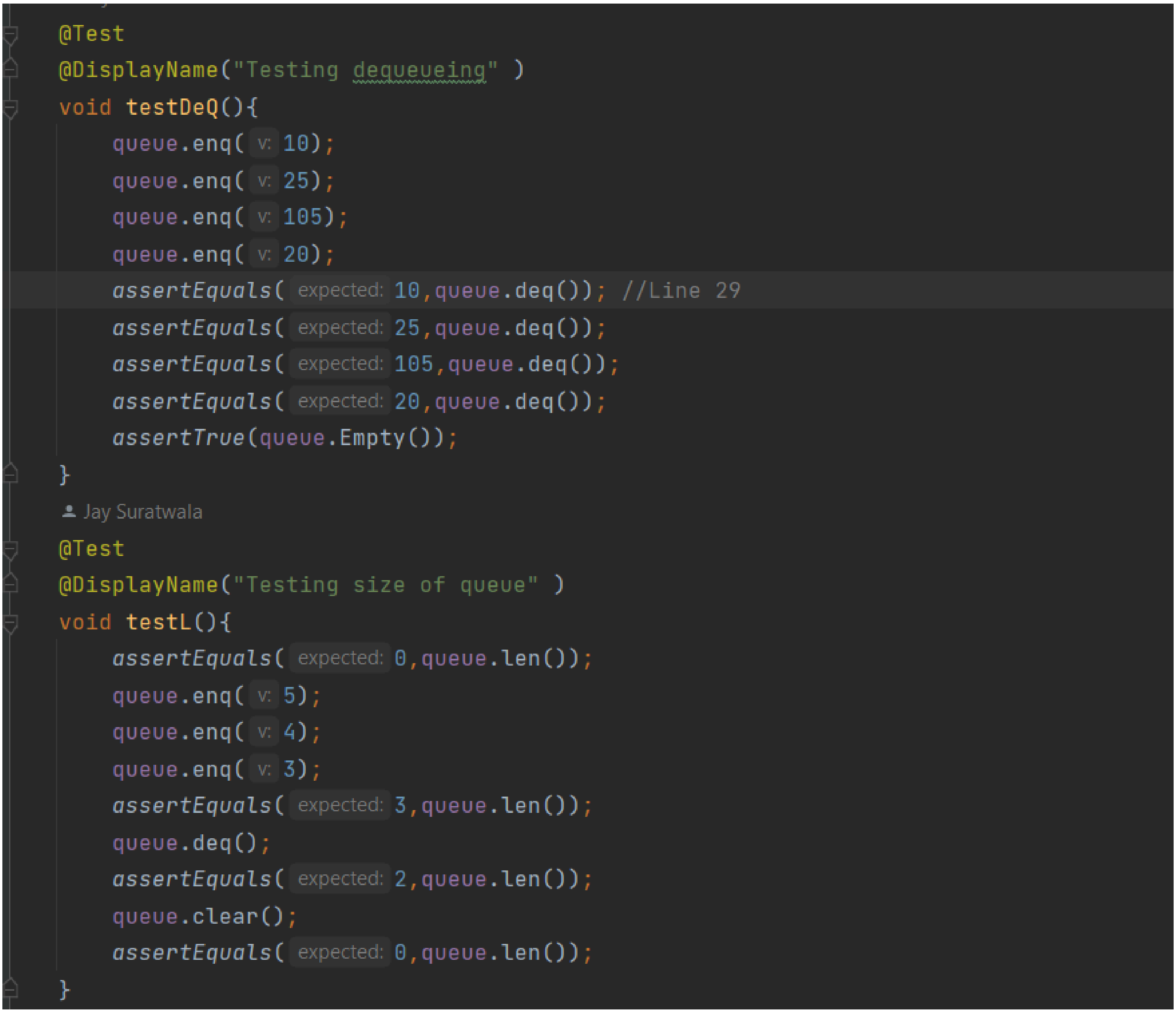
Fig 1.1

2. QueueTest.java Class Code (JUnit 5)



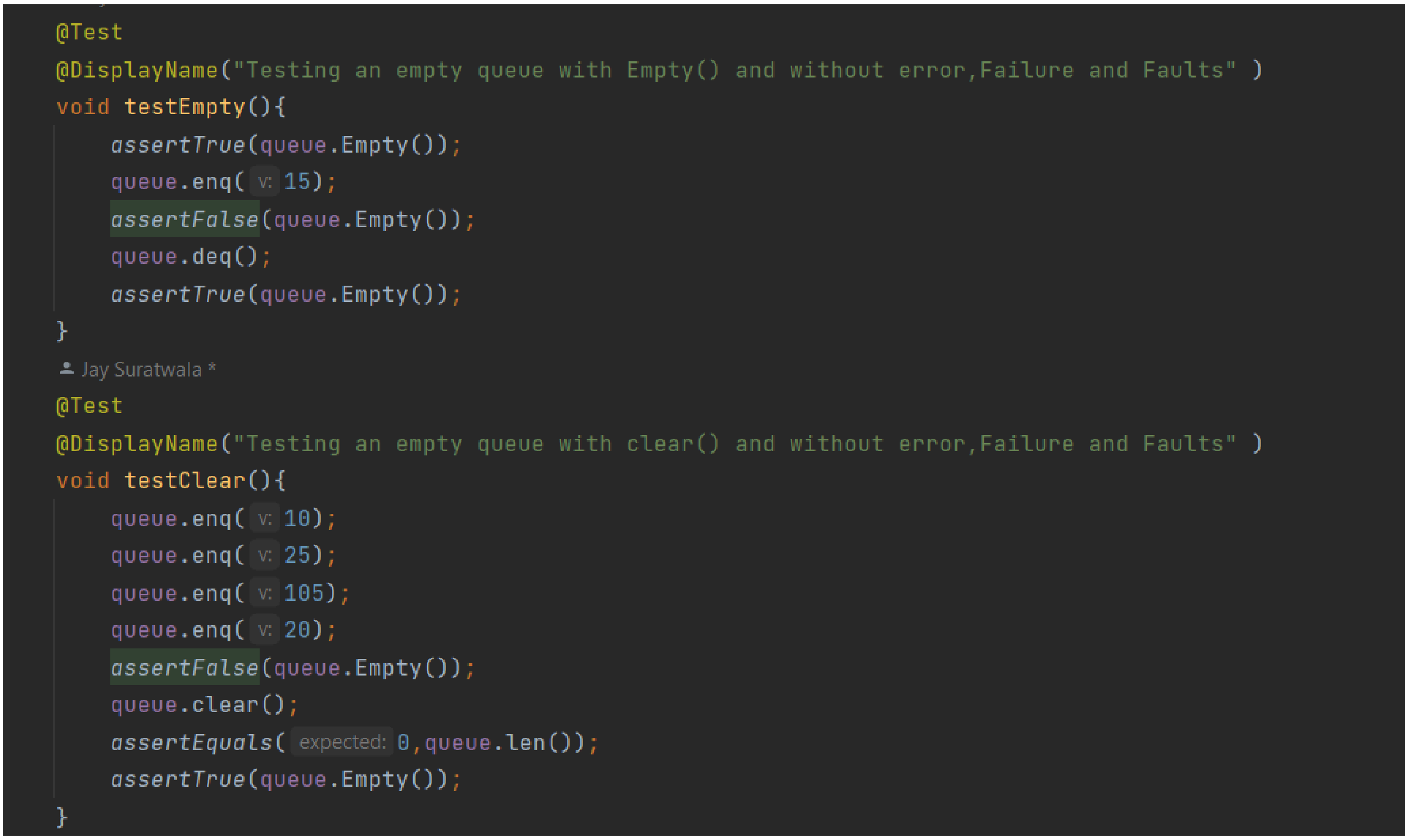
```
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api .Assertions.*;
import java.util.NoSuchElementException;
Jay Suratwala *
public class QueueTest {
    109 usages
    Queue queue;
    Jay Suratwala
    @BeforeEach
    void setUp() { queue = new Queue(); }
    Jay Suratwala
    @AfterEach
    void breakDown() { queue = null;}
    Jay Suratwala *
    @Test
    @DisplayName("Testing enqueueing without error,Failure and Faults" )
    void testEnQ(){
        queue.enq( v: 10);
        queue.enq( v: 25);
        queue.enq( v: 105);
        queue.enq( v: 20);
        assertEquals( expected: 4,queue.len());
        assertFalse(queue.Empty());
    }
```

Fig 2.1



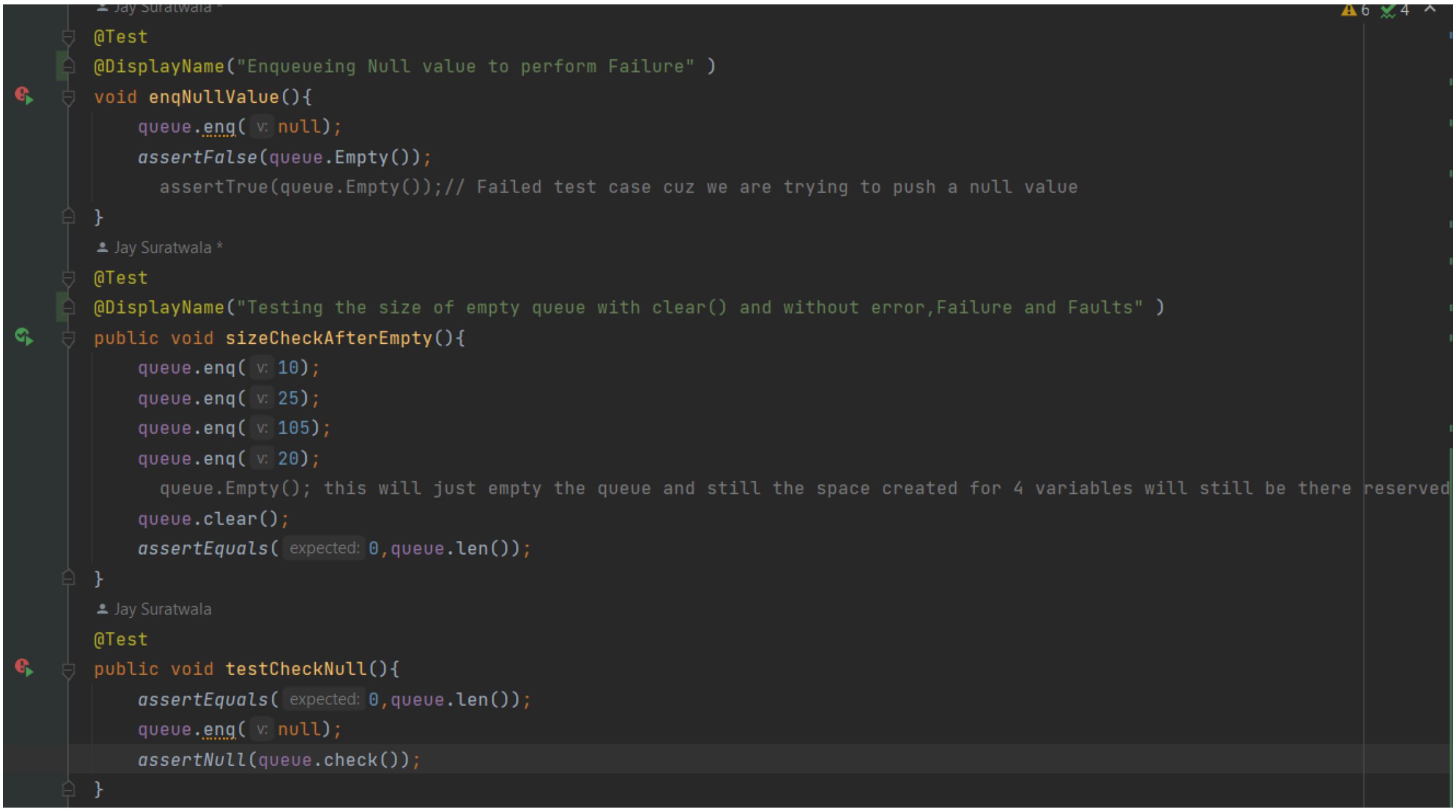
```
    @Test
    @DisplayName("Testing dequeueing" )
    void testDeQ(){
        queue.enq( v: 10);
        queue.enq( v: 25);
        queue.enq( v: 105);
        queue.enq( v: 20);
        assertEquals( expected: 10,queue.deq()); //Line 29
        assertEquals( expected: 25,queue.deq());
        assertEquals( expected: 105,queue.deq());
        assertEquals( expected: 20,queue.deq());
        assertTrue(queue.Empty());
    }
    Jay Suratwala
    @Test
    @DisplayName("Testing size of queue" )
    void testL(){
        assertEquals( expected: 0,queue.len());
        queue.enq( v: 5);
        queue.enq( v: 4);
        queue.enq( v: 3);
        assertEquals( expected: 3,queue.len());
        queue.deq();
        assertEquals( expected: 2,queue.len());
        queue.clear();
        assertEquals( expected: 0,queue.len());
    }
```

Fig 2.2



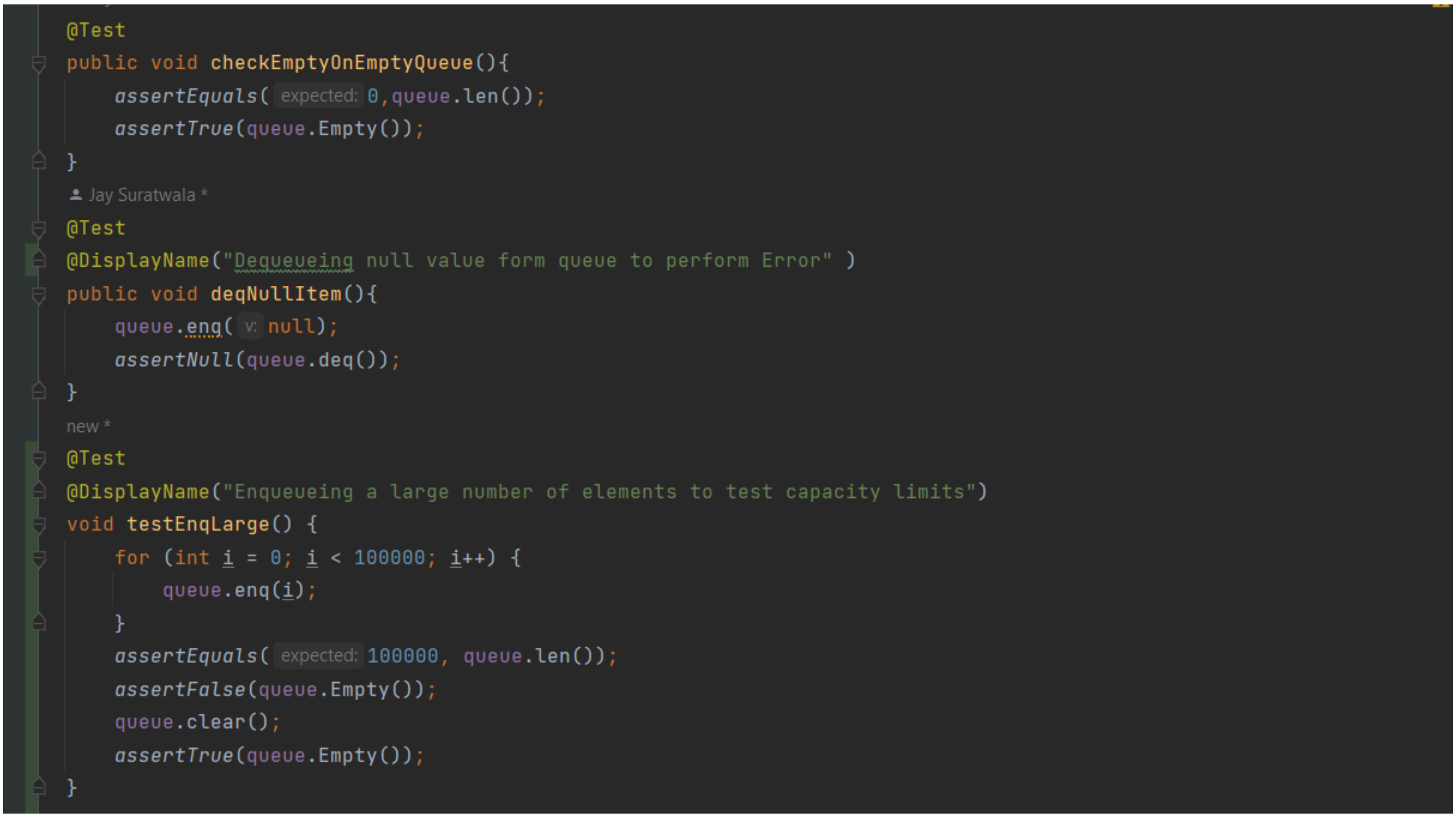
```
@Test
@DisplayName("Testing an empty queue with Empty() and without error,Failure and Faults" )
void testEmpty(){
    assertTrue(queue.Empty());
    queue.enq( v: 15);
    assertFalse(queue.Empty());
    queue.deq();
    assertTrue(queue.Empty());
}
Jay Suratwala *
@Test
@DisplayName("Testing an empty queue with clear() and without error,Failure and Faults" )
void testClear(){
    queue.enq( v: 10);
    queue.enq( v: 25);
    queue.enq( v: 105);
    queue.enq( v: 20);
    assertFalse(queue.Empty());
    queue.clear();
    assertEquals( expected: 0,queue.len());
    assertTrue(queue.Empty());
}
```

Fig 2.3



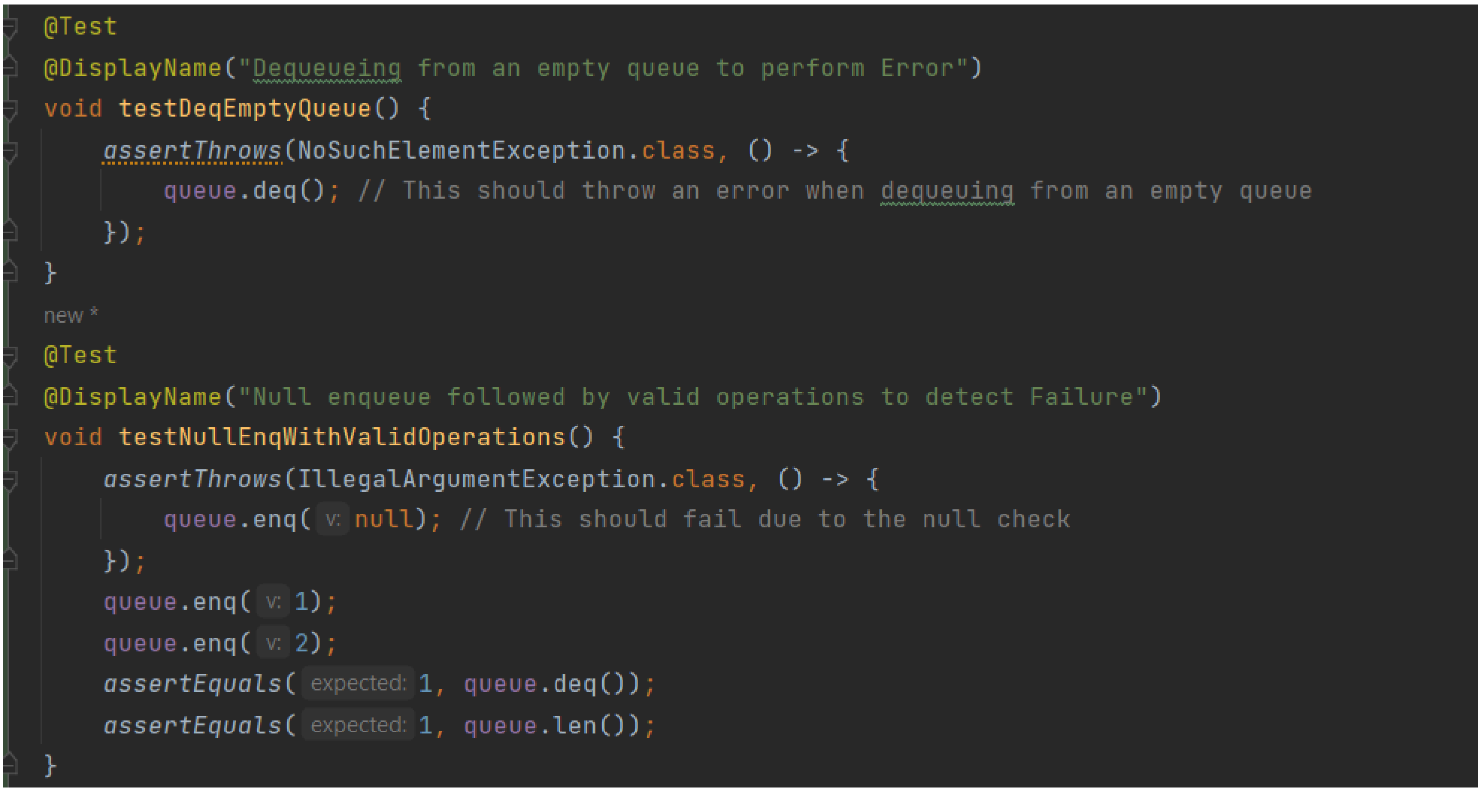
```
Jay Suratwala *
@Test
@DisplayName("Enqueueing Null value to perform Failure" )
void enqNullValue(){
    queue.enq( v: null);
    assertFalse(queue.Empty());
      assertTrue(queue.Empty());// Failed test case cuz we are trying to push a null value
}
Jay Suratwala *
@Test
@DisplayName("Testing the size of empty queue with clear() and without error,Failure and Faults" )
public void sizeCheckAfterEmpty(){
    queue.enq( v: 10);
    queue.enq( v: 25);
    queue.enq( v: 105);
    queue.enq( v: 20);
      queue.Empty(); this will just empty the queue and still the space created for 4 variables will still be there reserved
    queue.clear();
    assertEquals( expected: 0,queue.len());
}
Jay Suratwala
@Test
public void testCheckNull(){
    assertEquals( expected: 0,queue.len());
    queue.enq( v: null);
    assertNull(queue.check());
}
```

Fig 2.4



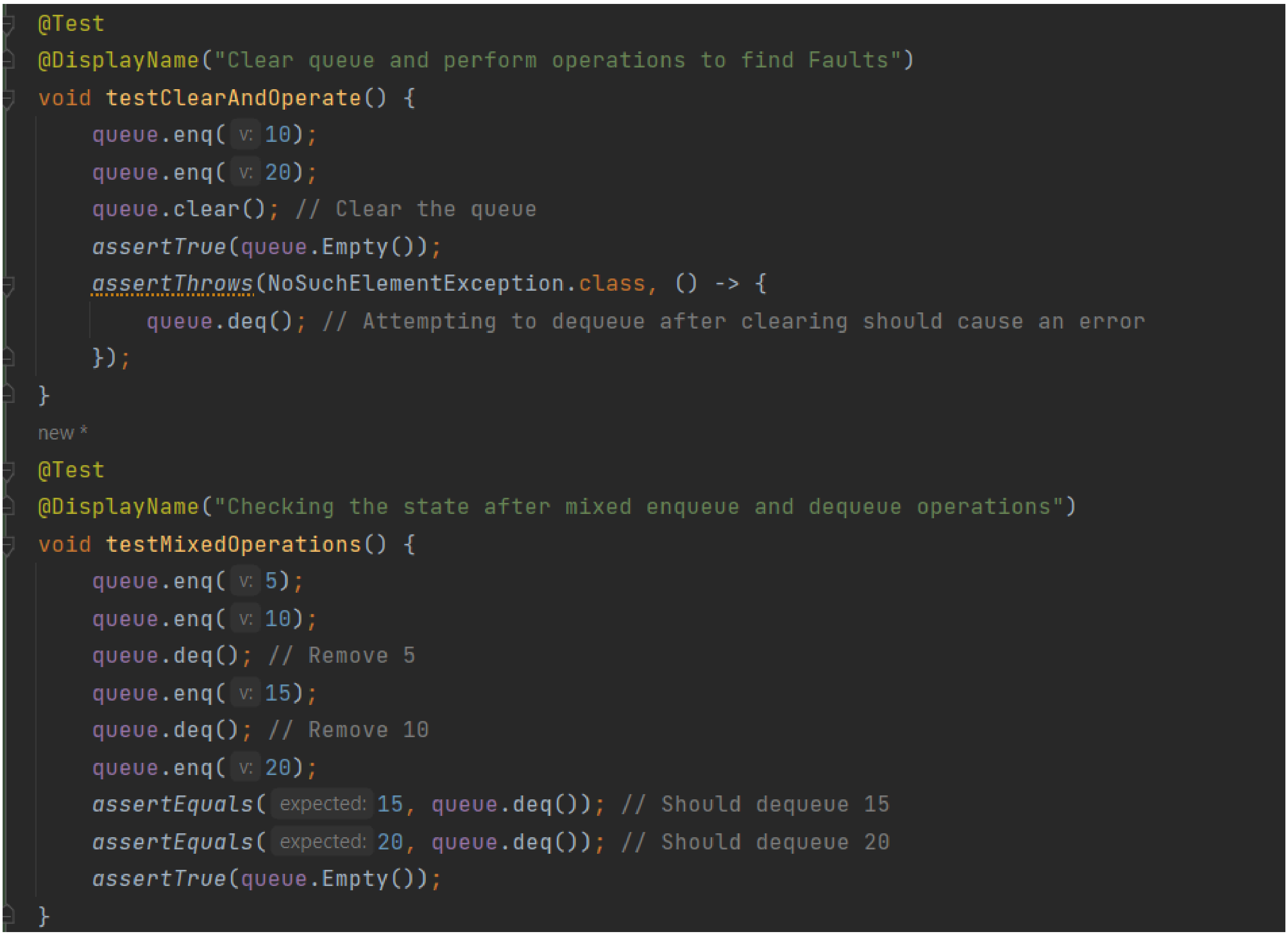
```
@Test
public void checkEmptyOnEmptyQueue(){
    assertEquals( expected: 0,queue.len());
    assertTrue(queue.Empty());
}
 Jay Suratwala *
@Test
@DisplayName("Dequeueing null value form queue to perform Error" )
public void deqNullItem(){
    queue.enq( v: null);
    assertNull(queue.deq());
}
new *
@Test
@DisplayName("Enqueueing a large number of elements to test capacity limits")
void testEnqLarge() {
    for (int i = 0; i < 100000; i++) {
        queue.enq(i);
    }
    assertEquals( expected: 100000, queue.len());
    assertFalse(queue.Empty());
    queue.clear();
    assertTrue(queue.Empty());
}
```

Fig 2.5



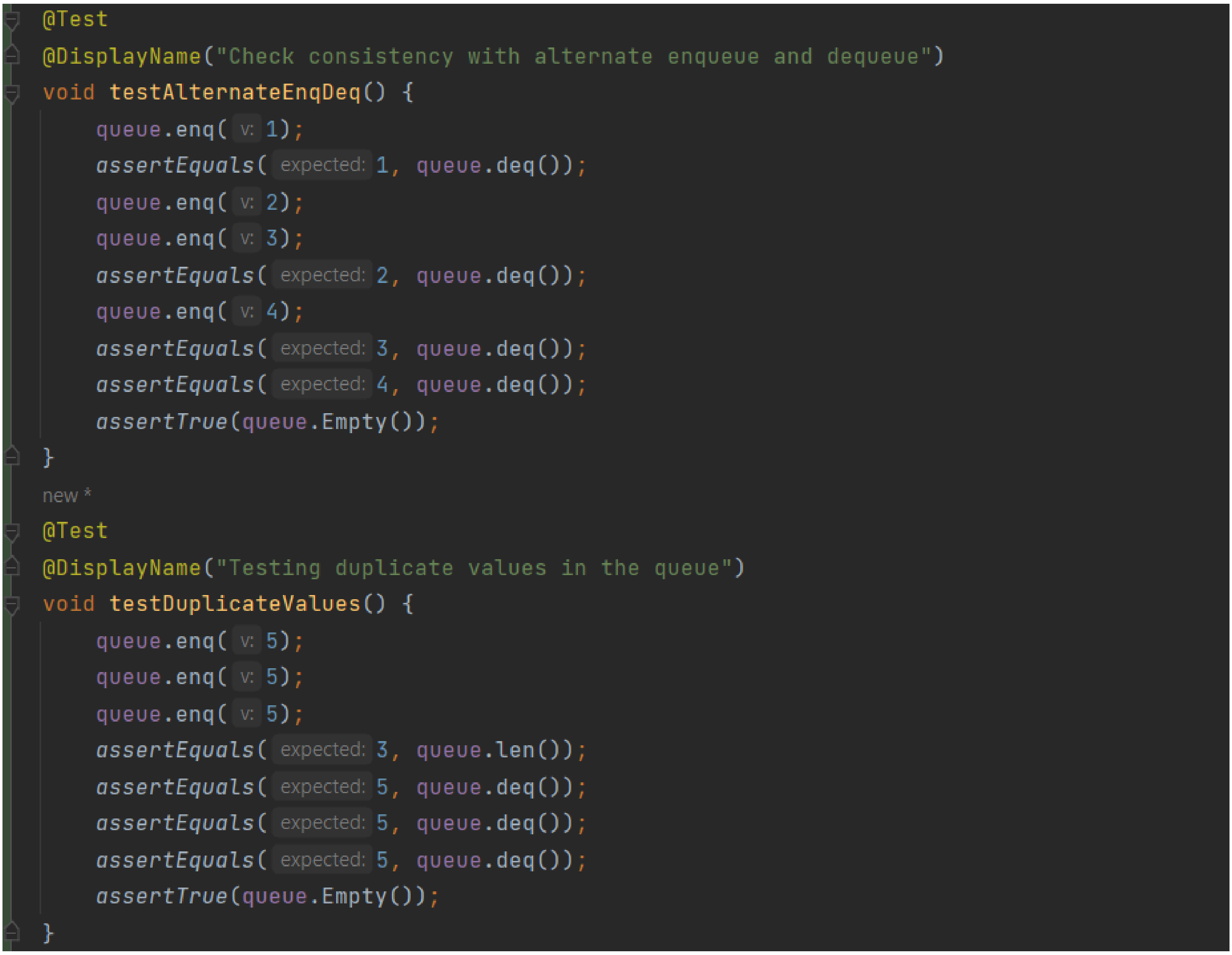
```
@Test
@DisplayName("Dequeueing from an empty queue to perform Error")
void testDeqEmptyQueue() {
    assertThrows(NoSuchElementException.class, () -> {
        queue.deq(); // This should throw an error when dequeuing from an empty queue
    });
}
new *
@Test
@DisplayName("Null enqueue followed by valid operations to detect Failure")
void testNullEnqWithValidOperations() {
    assertThrows(IllegalArgumentException.class, () -> {
        queue.enq( v: null); // This should fail due to the null check
    });
    queue.enq( v: 1);
    queue.enq( v: 2);
    assertEquals( expected: 1, queue.deq());
    assertEquals( expected: 1, queue.len());
}
```

Fig 2.6



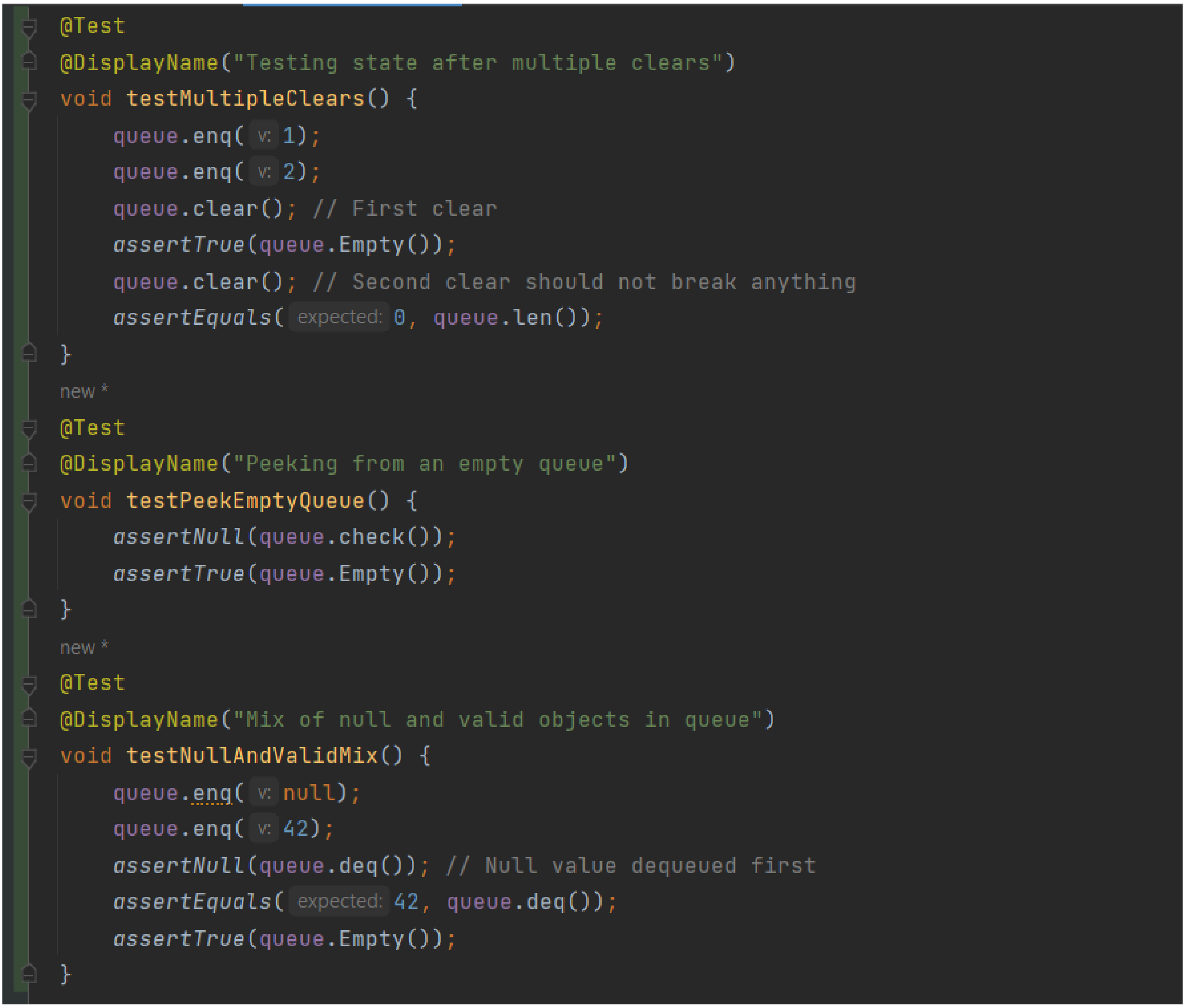
```
@Test
@DisplayName("Clear queue and perform operations to find Faults")
void testClearAndOperate() {
    queue.enq( v: 10);
    queue.enq( v: 20);
    queue.clear(); // Clear the queue
    assertTrue(queue.Empty());
    assertThrows(NoSuchElementException.class, () -> {
        queue.deq(); // Attempting to dequeue after clearing should cause an error
    });
}
new *
@Test
@DisplayName("Checking the state after mixed enqueue and dequeue operations")
void testMixedOperations() {
    queue.enq( v: 5);
    queue.enq( v: 10);
    queue.deq(); // Remove 5
    queue.enq( v: 15);
    queue.deq(); // Remove 10
    queue.enq( v: 20);
    assertEquals( expected: 15, queue.deq()); // Should dequeue 15
    assertEquals( expected: 20, queue.deq()); // Should dequeue 20
    assertTrue(queue.Empty());
}
```

Fig 2.7



```
@Test
@DisplayName("Check consistency with alternate enqueue and dequeue")
void testAlternateEnqDeq() {
    queue.enq( v: 1);
    assertEquals( expected: 1, queue.deq());
    queue.enq( v: 2);
    queue.enq( v: 3);
    assertEquals( expected: 2, queue.deq());
    queue.enq( v: 4);
    assertEquals( expected: 3, queue.deq());
    assertEquals( expected: 4, queue.deq());
    assertTrue(queue.Empty());
}
new *
@Test
@DisplayName("Testing duplicate values in the queue")
void testDuplicateValues() {
    queue.enq( v: 5);
    queue.enq( v: 5);
    queue.enq( v: 5);
    assertEquals( expected: 3, queue.len());
    assertEquals( expected: 5, queue.deq());
    assertEquals( expected: 5, queue.deq());
    assertEquals( expected: 5, queue.deq());
    assertTrue(queue.Empty());
}
```

Fig 2.8

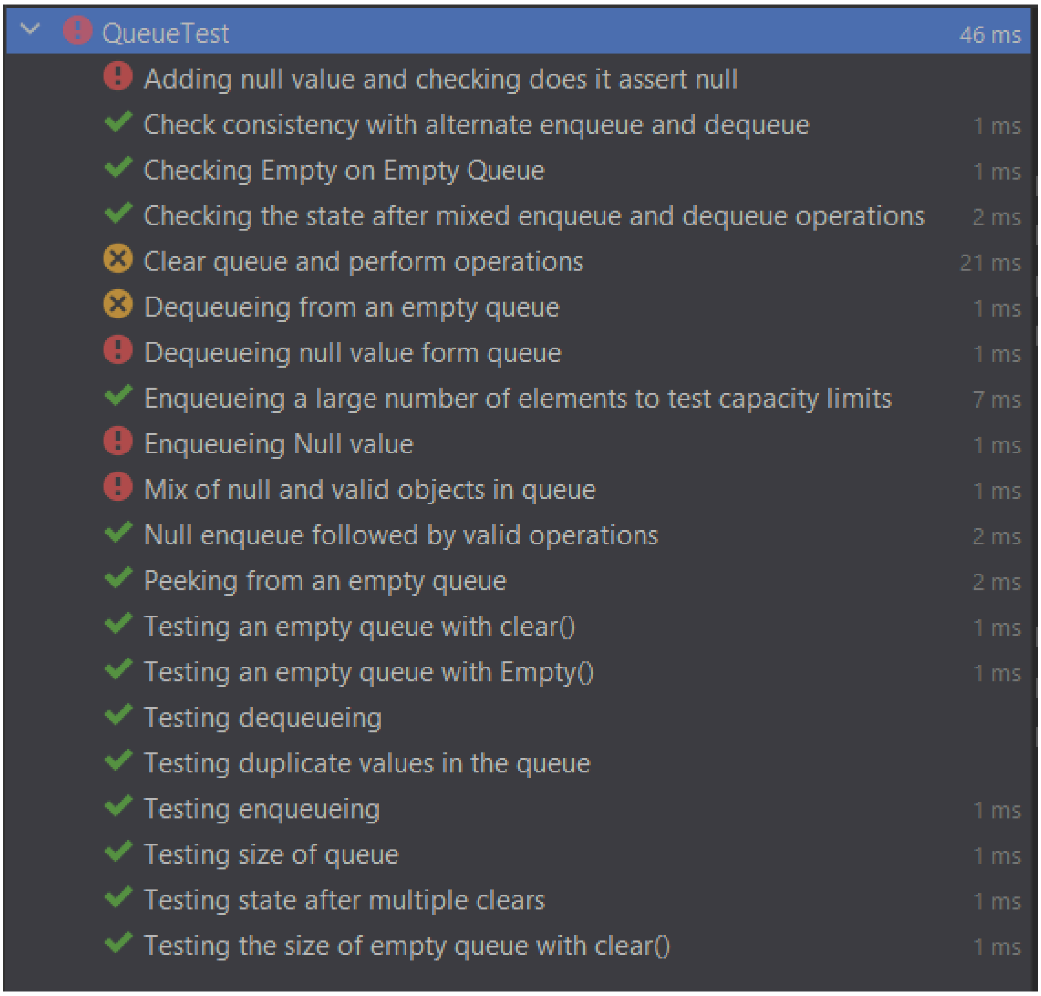


```
@Test
@DisplayName("Testing state after multiple clears")
void testMultipleClears() {
    queue.enq( v: 1);
    queue.enq( v: 2);
    queue.clear(); // First clear
    assertTrue(queue.Empty());
    queue.clear(); // Second clear should not break anything
    assertEquals( expected: 0, queue.len());
}
new *
@Test
@DisplayName("Peeking from an empty queue")
void testPeekEmptyQueue() {
    assertNull(queue.check());
    assertTrue(queue.Empty());
}
new *
@Test
@DisplayName("Mix of null and valid objects in queue")
void testNullAndValidMix() {
    queue.enq( v: null);
    queue.enq( v: 42);
    assertNull(queue.deq()); // Null value dequeued first
    assertEquals( expected: 42, queue.deq());
    assertTrue(queue.Empty());
}
```

Fig 2.9

3. Sample Output of Tests

- QueueTest.java

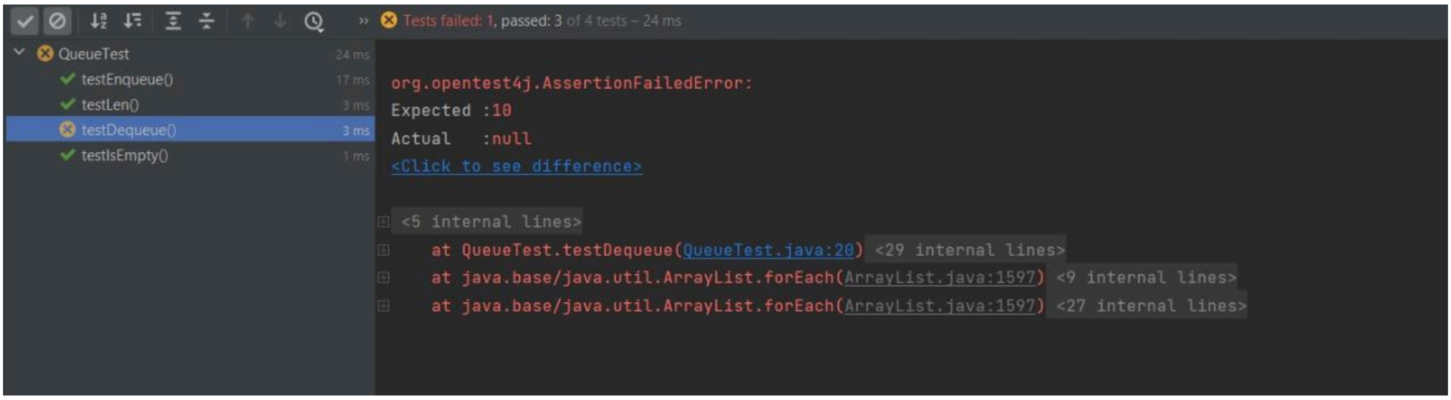


```
QueueTest                                                                46 ms
    ! Adding null value and checking does it assert null
    ✓ Check consistency with alternate enqueue and dequeue                1 ms
    ✓ Checking Empty on Empty Queue                                        1 ms
    ✓ Checking the state after mixed enqueue and dequeue operations       2 ms
    ☒ Clear queue and perform operations                                  21 ms
    ☒ Dequeueing from an empty queue                                       1 ms
    ! Dequeueing null value form queue                                    1 ms
    ✓ Enqueueing a large number of elements to test capacity limits       7 ms
    ! Enqueueing Null value                                               1 ms
    ! Mix of null and valid objects in queue                              1 ms
    ✓ Null enqueue followed by valid operations                           2 ms
    ✓ Peeking from an empty queue                                         2 ms
    ✓ Testing an empty queue with clear()                                 1 ms
    ✓ Testing an empty queue with Empty()                                 1 ms
    ✓ Testing dequeueing
    ✓ Testing duplicate values in the queue
    ✓ Testing enqueueing                                                  1 ms
    ✓ Testing size of queue                                               1 ms
    ✓ Testing state after multiple clears                                 1 ms
    ✓ Testing the size of empty queue with clear()                        1 ms
```

Fig 3.1

4. Reflection on Errors, Faults, and Failures

In the course of implementing and testing the Queue class, several challenges were encountered. These included compilation errors, logical faults, and test failures, which provided valuable learning opportunities and insights into the importance of robust software testing. First we write down the Queue class . Then we write down the Que Following is the table of test cases and their reasoning behind why errors, failures and faults were encountered.

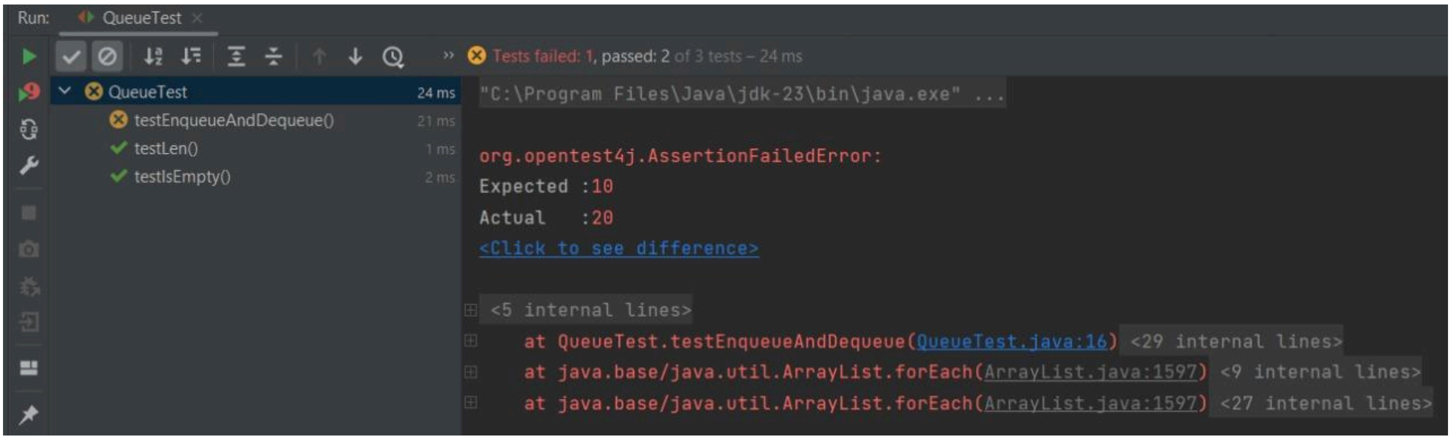


```
☒ Tests failed: 1, passed: 3 of 4 tests – 24 ms

QueueTest              24 ms
  ✓ testEnqueue()      17 ms
  ✓ testLen()           3 ms
  ☒ testDequeue()       3 ms
  ✓ testIsEmpty()       1 ms

org.opentest4j.AssertionFailedError:
Expected :10
Actual   :null
<Click to see difference>

<5 internal lines>
    at QueueTest.testDequeue(QueueTest.java:20) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
```

Error (Logical Error in testDequeue() now replaced with testDeq() method)

Location: assertEquals(10,queue.deq()); (Currently at line 29 refer Fig 2.2)

Explanation: The logical error occurred because of not using the Queue.java functions as they were meant to be. Before the line assertEquals(10,queue.deq()) another command which is queue.isEmpty() was used and resulted in having an empty with null value in it whereas according to the code there should be 10 as the expected value.



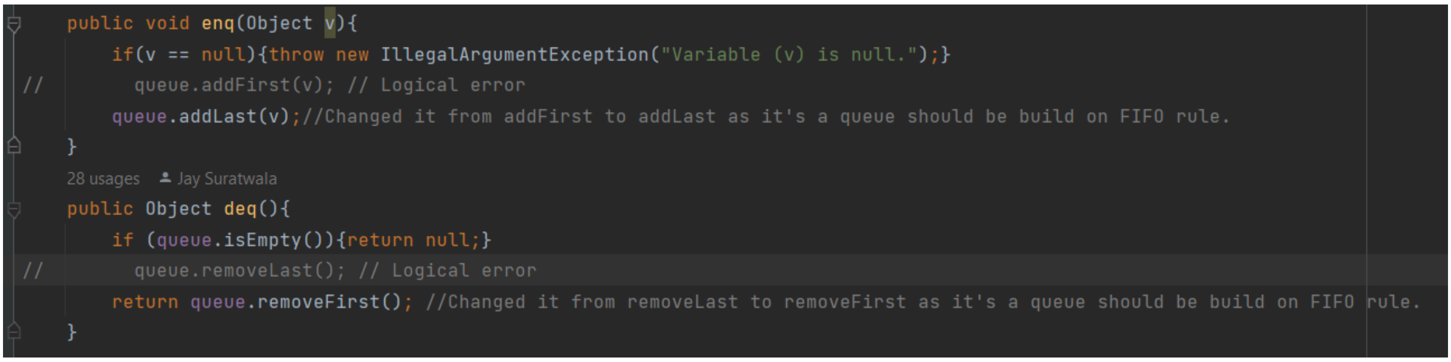
```
QueueTest                          24 ms
  testEnqueueAndDequeue()          21 ms
  testLen()                         1 ms
  testIsEmpty()                     2 ms
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...

org.opentest4j.AssertionFailedError:
Expected :10
Actual   :20
<Click to see difference>

<5 internal lines>
    at QueueTest.testEnqueueAndDequeue(QueueTest.java:16) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
```

Error (Logical Error)

Location: The method testEnqueueAndDequeue() is now replaced with testEnq() and testDeq; (refer Fig 2.1 & 2.2)

Explanation: Here an attempt to merge testEnqueue() and testDequeue was made to reduce the number of lines from code and improve the efficiency as a result some code statements got overlapped. Due to this the testEnqueueAndDequeue() method was splitted in 2 small methods namely testEnq() and testDeq().



```
public void enq(Object v){
    if(v == null){throw new IllegalArgumentException("Variable (v) is null.");}
//      queue.addFirst(v); // Logical error
    queue.addLast(v);//Changed it from addFirst to addLast as it's a queue should be build on FIFO rule.
}
28 usages  Jay Suratwala
public Object deq(){
    if (queue.isEmpty()){return null;}
//      queue.removeLast(); // Logical error
    return queue.removeFirst(); //Changed it from removeLast to removeFirst as it's a queue should be build on FIFO rule.
}
```

Error (Logical Error in enq() and deq() method)

Location: queue.addFirst(v); (Commented line in the image)

Explanation: The logical error is in using addFirst() instead of addLast() for the enqueue operation in the enq() method. A queue should follow the FIFO (First-In-First-Out) principle, but addFirst() behaves like a stack (LIFO). This is an error in the algorithm's logic. This same thing applies for remove.Last() and remove.First() .

Adding null value and checking does it assert null

Error: Passing a null value to the enq() method

Location: Queue.java → enq() method, where if (v == null) throws java.lang.IllegalArgumentException: Variable (v) is null.

Explanation: The enq() method checks for null and throws an exception when it encounters a null value. This is a deliberate error handling mechanism in the code, but the test case should assert that this exception is thrown, which it fails to do.

Clear queue and perform operations

Fault: The clear() method does not properly reset the internal state of the queue.

Location: Queue.java → clear() method.

Explanation: After calling clear() , operations on the queue should behave as if it were empty. However, when a subsequent dequeue operation is performed, the expected exception (java.util.NoSuchElementException) is not thrown, indicating a fault in the way the clear() method resets the internal state of the queue.

Dequeuing from an empty queue

Error: Failing to throw the expected java.util.NoSuchElementException when dequeuing from an empty queue.

Location: Queue.java → deq() method.

Explanation: The deq() method should check if the queue is empty and throw NoSuchElementException when an element is dequeued. The absence of this exception indicates an error in the exception handling logic.

Dequeuing null value from queue

Fault: The queue allows storing and dequeuing null values.

Location: Queue.java → deq() method.

Explanation: Dequeuing a null value is considered a fault because it indicates the queue contains invalid data. The e() method should ensure that null values are not enqueued in the first place or are properly handled.

Enqueueing Null value

Failure: The enq() method throws java.lang.IllegalArgumentException: Variable (v) is null. when a null value is enqueued.

Location: Queue.java → enq() method.

Explanation: This failure occurs because the enq() method has a validation check that prevents null values from being added to the queue. The test case should expect this exception and handle it, but failing to do so results in a test failure.

Mix of null and valid objects in queue

Fault: The queue throws java.lang.IllegalArgumentException when a null value is enqueued, disrupting the normal operation.

Location: Queue.java → enq() method.

Explanation: The queue does not support mixed data (valid objects and null values). Attempting to enqueue a null value alongside valid data causes the exception to be thrown and the test to fail.

5. Conclusion

In this document, a complete implementation of a Queue Abstract Data Type (ADT) in Java, along with a JUnit 5 test suite to ensure its correctness. The reflection section provided a detailed analysis of the errors, faults, and failures encountered, as well as the corrective measures taken.

Through rigorous testing and careful debugging, the Queue class was refined to meet the expected FIFO behavior. This experience has emphasized the importance of unit testing and thorough error handling in software development. By adopting these practices, we ensure the development of more reliable and resilient systems.

6. References

[1] https://junit.org/junit5/
[2] https://github.com/ms5589/Queue-implementation-and-Testing
[3]https://github.com/Vikrant100/CA650-Software-Process-Quality/blob/main/CA650%20assignment/assignment%201%20report%20unit%20testing.pdf