@Nested Tests

This lesson demonstrates how to use @Nested annotation to configure the test hierarchy.

WE'LL COVER THE FOLLOWING

@Nested Tests

`@Nested` annotation provides tests creator more functionality to show the relationship among several groups of tests.

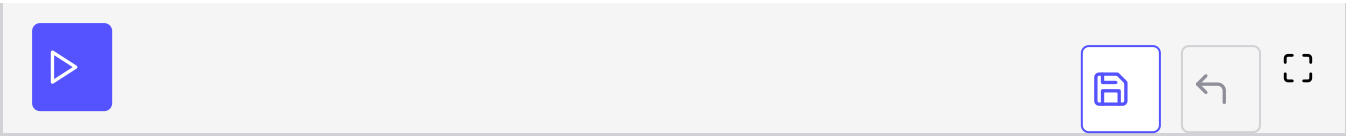
This relationship is achieved by providing nested classes to the main test class. But, by default nested classes don't participate in test execution. In order to provide testing capabilities to nested classes `@Nested` annotation is used.

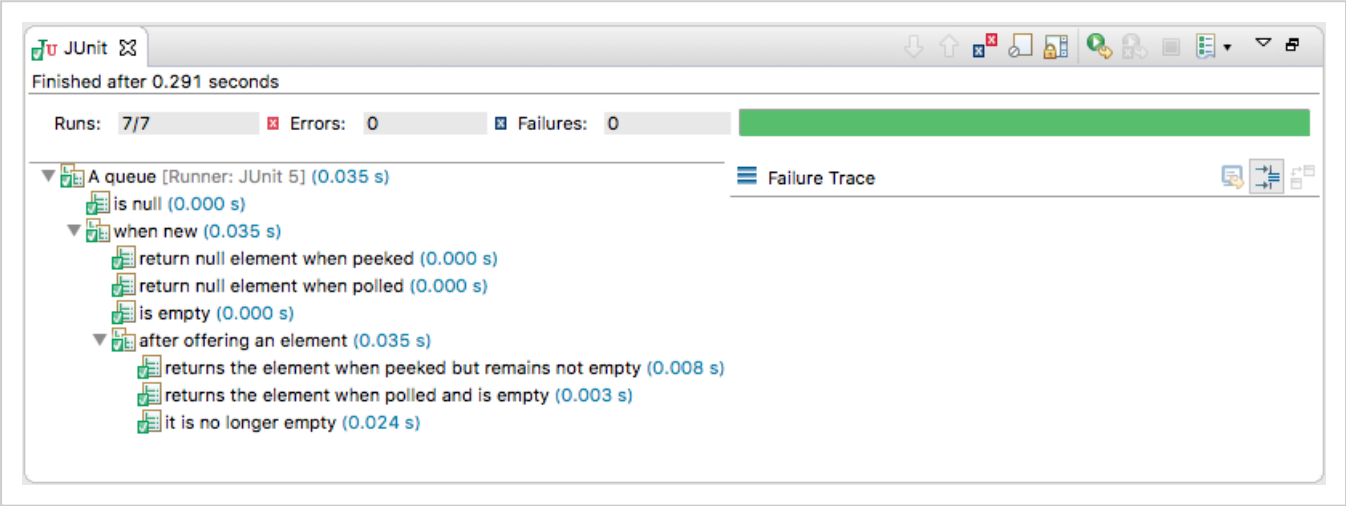
Let's take a look at a demo.

```
package io.educative.junit5;

import static org.junit.Assert.assertNull;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.LinkedList;
import java.util.Queue;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;

@DisplayName("A queue")
public class TestingAQueueDemo {
        Queue<String> queue; // A Queue of String

    @Test
    @DisplayName("is null")
    void isNotInstantiated() {
```

```
    assertNull(queue);
}


@Nested
@DisplayName("when new")
class WhenNew {

    @BeforeEach
    void createNewStack() {
        queue = new LinkedList<>();
    }

    @Test
    @DisplayName("is empty")
    void isEmpty() {
        assertTrue(queue.isEmpty());
    }

    @Test
    @DisplayName("return null element when polled")
    void returnNullWhenPolled() {
        assertNull(queue.poll());
    }

    @Test
    @DisplayName("return null element when peeked")
    void returnNullWhenPeeked() {
            assertNull(queue.peek());
    }

    @Nested
    @DisplayName("after offering an element")
    class AfterOffering {

        String anElement = "an element";

        @BeforeEach
        void offerAnElement() {
            queue.offer(anElement);
        }

        @Test
        @DisplayName("it is no longer empty")
        void isNotEmpty() {
            assertFalse(queue.isEmpty());
        }

        @Test
        @DisplayName("returns the element when polled and is empty")
        void returnElementWhenPolled() {
            assertEquals(anElement, queue.poll());
            assertTrue(queue.isEmpty());
        }

        @Test
        @DisplayName("returns the element when peeked but remains not empty")
        void returnElementWhenPeeked() {
            assertEquals(anElement, queue.peek());
            assertFalse(queue.isEmpty());
        }
    }
}
```

```
}
```





Explanation

On running `TestingAQueueDemo.java` as JUnit Test case, the output generated is demonstrated in the above image.

`@Nested` annotation helps us writing test cases in a more descriptive way. Nested classes enforce the idea of Behaviour Driven Development. It separates our test class in logical groupings. It helps us in readability of test cases.

In the next chapter, we will discuss about JUnit 5 Integration with Maven.