@Nested Tests

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

We'll cover the following

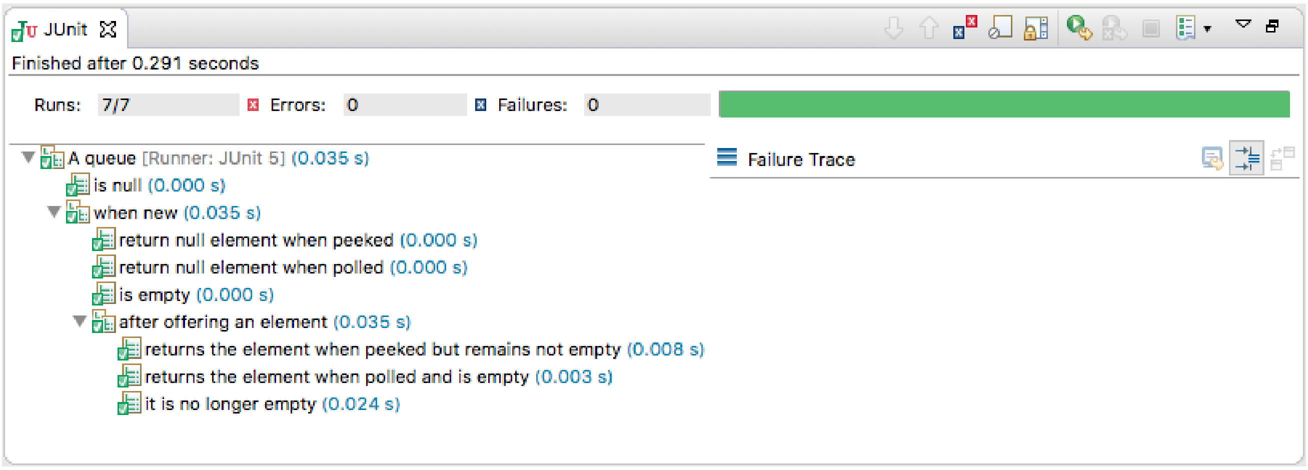
@Nested Tests

`@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
import static org.junit.jupite
import static org.junit.jupite
import static org.junit.jupite

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

import org.junit.jupiter.api.B
import org.junit.jupiter.api.D
import org.junit.jupiter.api.N
import org.junit.jupiter.api.T

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

    @Test
    @DisplayName("is null")
    void isNotInstantiated() {
        assertNull(queue);
    }

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

        @BeforeEach
        void createNewStack()
```



```
JUnit
Finished after 0.291 seconds
Runs: 7/7    Errors: 0    Failures: 0

A queue [Runner: JUnit 5] (0.035 s)                          Failure Trace
  is null (0.000 s)
  when new (0.035 s)
    return null element when peeked (0.000 s)
    return null element when polled (0.000 s)
    is empty (0.000 s)
    after offering an element (0.035 s)
      returns the element when peeked but remains not empty (0.008 s)
      returns the element when polled and is empty (0.003 s)
      it is no longer empty (0.024 s)
```

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.