@BeforeEach and @AfterEach Annotation

This lesson demonstrates working of two Lifecycle methods annotated with - @BeforeEach and @AfterEach Annotation.

@Test and Constructor

In Junit 5 for each test, a new instance of test class is created. So, for e.g. if a class has two `@Test` methods than two instances of test class will be created, one for each test. Thus, the constructor of the test class will be called as many times as there is the number of `@Test` methods. Let's look at the demo:-

```java
package io.educative.junit5;

import org.junit.jupiter.api.Test;

public class LifecycleTest {

    public LifecycleTest() {
        System.out.println("LifecycleTest - Constructor got executed !!!");
    }

    @Test
    public void testOne() {
        System.out.println("LifecycleTest - testOne() got executed !!!");
    }

    @Test
    public void testTwo() {
        System.out.println("LifecycleTest - testTwo() got executed !!!");
    }

}
```
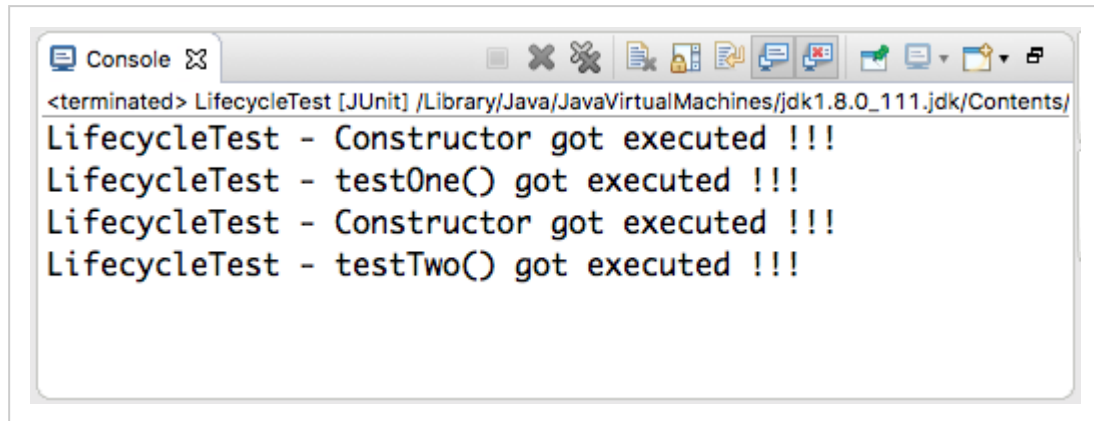
```
LifecycleTest - Constructor got executed !!!
LifecycleTest - testOne() got executed !!!
LifecycleTest - Constructor got executed !!!
LifecycleTest - testTwo() got executed !!!
```

Output of the test

Here, you can see constructor got called two times because there were two `@Test` annotated methods. Thus, it proves that each test runs in its own test class instance.

@BeforeEach and @AfterEach

Methods annotated with `@BeforeEach` and `@AfterEach` as the name suggests are called before each and after each `@Test` methods. So, if in a test class there are two @Test methods, the @BeforeEach method will be called twice, just before each @Test method and similarly, the @AfterEach method will be called twice, just after each @Test method. Let's look at demo taking previous test class:-

```
package io.educative.junit5;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class LifecycleTest {

        public LifecycleTest() {
                System.out.println("LifecycleTest - Constructor got executed !!!");
        }

        @BeforeEach
        public void beforeEach() {
                System.out.println("LifecycleTest - beforeEach() got executed !!!");
        }

        @Test
        public void testOne() {
                System.out.println("LifecycleTest - testOne() got executed !!!");
        }
```

```
        @Test
        public void testTwo() {
                System.out.println("LifecycleTest - testTwo() got executed !!!");

        }

        @AfterEach
        public void afterEach() {
                System.out.println("LifecycleTest - afterEach() got executed !!!");
        }

}
```



```
Console ⊠                              ☐ ✕ ✕  ⧉ ⧉ ⧉ ⧉ ⧉   ⧉ ▾ ⧉ ▾ ⧉
<terminated> LifecycleTest [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/
LifecycleTest - Constructor got executed !!!
LifecycleTest - beforeEach() got executed !!!
LifecycleTest - testOne() got executed !!!
LifecycleTest - afterEach() got executed !!!
LifecycleTest - Constructor got executed !!!
LifecycleTest - beforeEach() got executed !!!
LifecycleTest - testTwo() got executed !!!
LifecycleTest - afterEach() got executed !!!
```

Output of the test

As Junit test class has two `@Test` methods, it executes each of the test methods in a separate instance of the test class. Thus, it picks the first `@Test` method in random order and -

1. It initializes the test class by calling its constructor.

2. After that, it executes the `@BeforeEach` annotated method.

3. After that, it executes the `@Test` method.

4. After that, it executes the `@AfterEach` annotated method.

5. Then it picks the second `@Test` method and executes again from step 1 to 4.

Usually, when we have common setup logic across various test cases, the common initialization code is placed in the `@BeforeEach` annotated method and cleaned up in `@AfterEach` annotated method.

In the next lesson, we will look into `@BeforeAll()` and `@AfterAll` lifecycle callbacks.