TestInfo parameter

This lesson demonstrates how to use dependency injection (TestInfo Parameter) in Constructor and Test methods.

We'll cover the following

ParameterResolver

In previous JUnit versions, there was **limited/no** support to allow parameters in test constructors or methods. One of the major changes in JUnit 5 Jupiter was that both test constructors and test methods are now allowed to have parameters. These parameters provide metadata to constructors and test methods. Thus, allows for greater flexibility and enables Dependency Injection for test methods and constructors.

In Junit 5, there is an interface in `org.junit.jupiter.api.extension` package by name, `ParameterResolver`. This interface defines the API for test extensions that wish to dynamically resolve parameters at runtime.

There are different types of ParameterResolver's in Junit 5. Generally, if a test constructor or a `@Test`, `@BeforeEach`, `@AfterEach`, `@BeforeAll`, `@AfterAll` or `@TestFactory` method accepts a parameter, the parameter must be resolved at runtime by a registered `ParameterResolver`.

Read more - https://junit.org/junit5/docs/5.3.2/api/org/junit/jupiter/api/extension/package-summary.html for `org.junit.jupiter.api.extension`

https://junit.org/junit5/docs/5.3.2/api/org/junit/jupiter/api/extension/ParameterResolver.html for `ParameterResolver`.

TestInfo Parameter

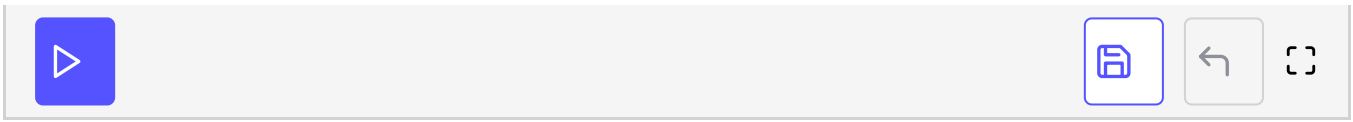
TestInfoParameterResolver - It is a built-in ParameterResolver. If a method parameter is of type `TestInfo`, it signifies that `TestInfoParameterResolver` will supply an instance of TestInfo corresponding to the current test as the value for the parameter.

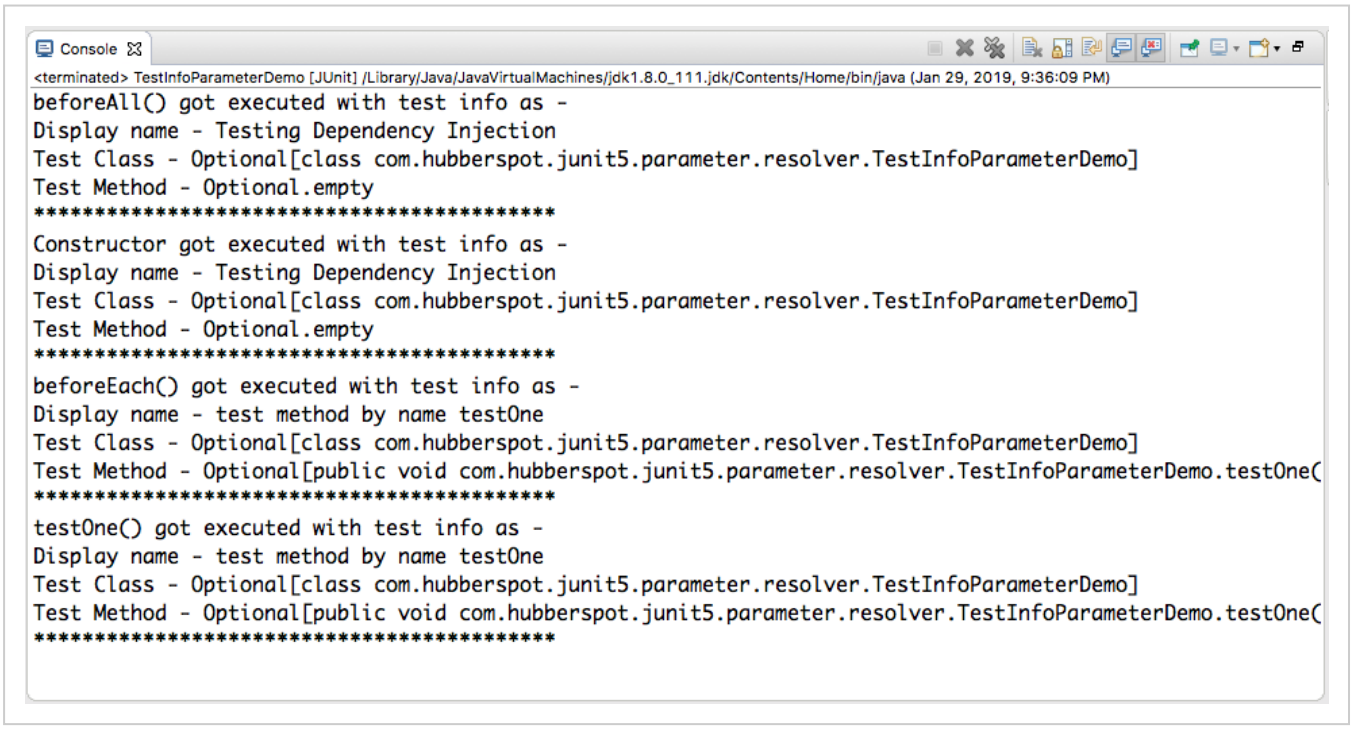
The `TestInfo` as the parameter can then be used to retrieve information or metadata about the current test, such as the test's display name, the test class, the test method, or associated tags. If `@DisplayName` annotation is used on test methods then the custom name associated with test methods is retrieved or else technical name, which is the actual name of the test class or test method.

The following demonstrates how to have `TestInfo` injected into a test constructor, @Test and lifecycle methods:-

```
package io.educative.junit5;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInfo;

@DisplayName("Testing Dependency Injection")
public class TestInfoParameterDemo {

	@BeforeAll
	public static void beforeAll(TestInfo testInfo) {
		System.out.println("beforeAll() got executed with test info as - ");
		System.out.println("Display name - " + testInfo.getDisplayName());
		System.out.println("Test Class - " + testInfo.getTestClass());
		System.out.println("Test Method - " + testInfo.getTestMethod());
		System.out.println("*****************************************");
	}

	public TestInfoParameterDemo(TestInfo testInfo) {
		System.out.println("Constructor got executed with test info as - ");
		System.out.println("Display name - " + testInfo.getDisplayName());
		System.out.println("Test Class - " + testInfo.getTestClass());
		System.out.println("Test Method - " + testInfo.getTestMethod());
		System.out.println("*****************************************");
	}

	@BeforeEach
	public void beforeEach(TestInfo testInfo) {
		System.out.println("beforeEach() got executed with test info as - ");
		System.out.println("Display name - " + testInfo.getDisplayName());
		System.out.println("Test Class - " + testInfo.getTestClass());
		System.out.println("Test Method - " + testInfo.getTestMethod());
		System.out.println("*****************************************");
	}
```

```
        @Test
        @DisplayName("test method by name testOne")

        public void testOne(TestInfo testInfo) {
                System.out.println("testOne() got executed with test info as - ");
                System.out.println("Display name - " + testInfo.getDisplayName());
                System.out.println("Test Class - " + testInfo.getTestClass());
                System.out.println("Test Method - " + testInfo.getTestMethod());
                System.out.println("*****************************************");
        }

}
```





Code Explanation

The above output of the program demonstrates how `TestInfo` parameter can be injected into test constructor and test methods.

It gives information to test methods and constructor about the test class and test methods.

Here, you can see `@BeforeAll` annotated method and constructor. It prints `@DisplayName` value of test class and also prints test class fully qualified name. Similarly, `TestInfo` provides the same info in `@AfterAll` annotated method.

Since for every new @Test method a new instance of test class is created, therefore in `@BeforeEach`, `@AfterEach` and `@Test` annotated method, it prints `@DisplayName` value of test method and also prints test class fully qualified

name.

In the next chapter, we will be studying about JUnit 5 Assumptions.