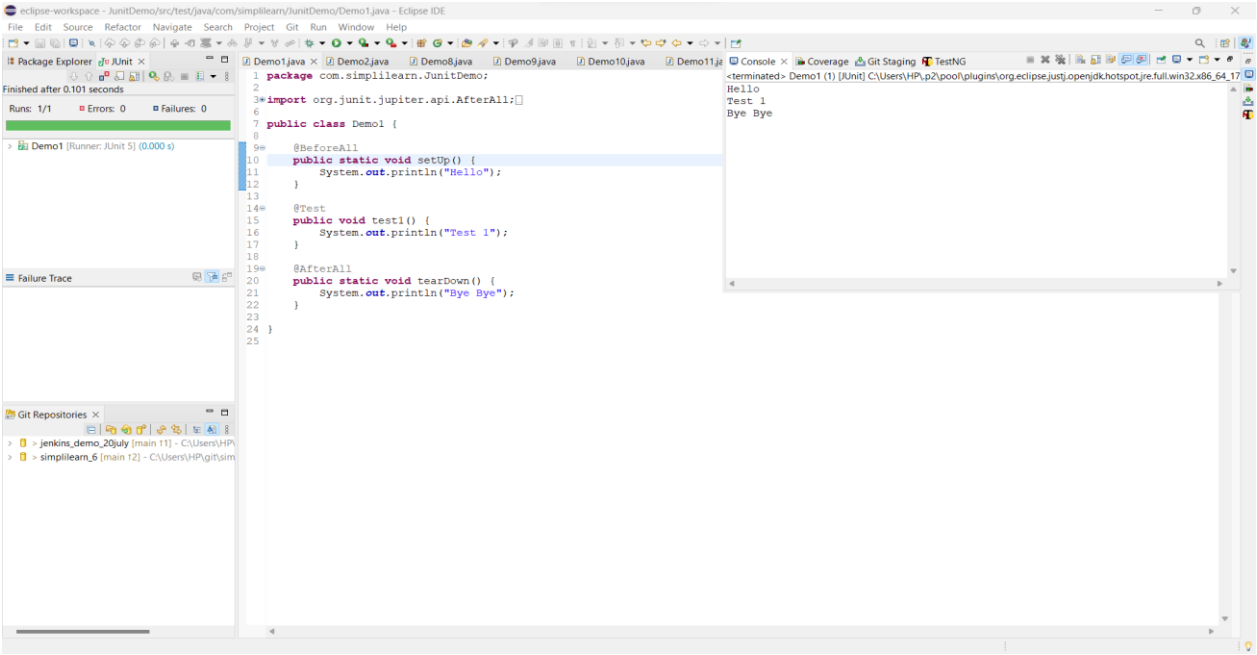
LifeCycle Methods:

```
package com.simplilearn.JunitDemo;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
public class Demo1 {
	@BeforeAll
	public static void setUp() {
		System.out.println("Hello");
	}
	@Test
	public void test1() {
		System.out.println("Test 1");
	}
	@AfterAll
	public static void tearDown() {
		System.out.println("Bye Bye");
	}
}
```

Output:

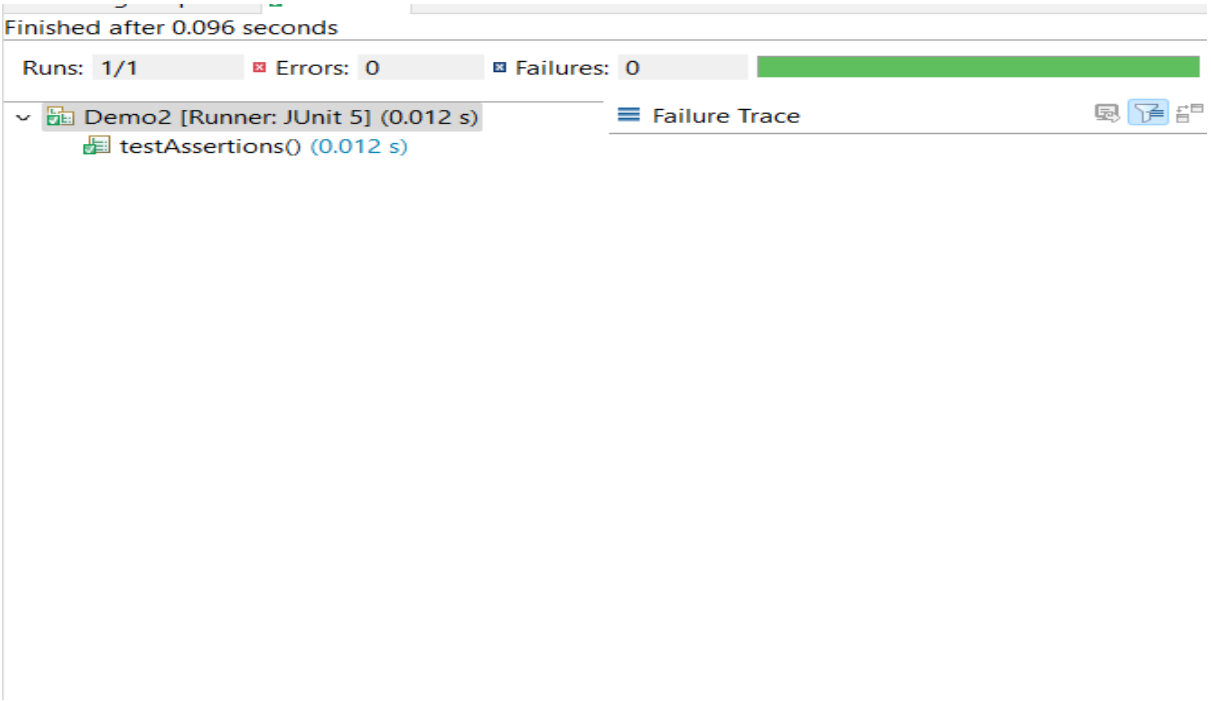


To Demonstrate Assertions:

```
package com.simplilearn.JunitDemo;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertArrayEquals;

import static org.junit.jupiter.api.Assertions.*;

public class Demo2 {

        @Test

        public void testAssertions() {

                String str1 = new String("abc");

                String str2 = new String("abc");

                String str3 = null;

                String str4 = "abc";

                String str5 = "abc";


                int val1 = 5;

                int val2 = 6;


                String[] expectedArray = {"one", "two", "three"};

                String[] resultArray = {"one", "two", "three"};
```

```
        //Check that two objects are equal
        assertEquals(str1, str2);

        //Check that a condition is true
        assertTrue(val1 < val2);

        //Check that a condition is false
        assertFalse(val1 > val2);

        //Check that an object is not null
        assertNotNull(str1);

        //Check that an object is null
        assertNull(str3);

        //Check of two object references point to the same object
        assertSame(str4, str5);

        //Check if two object references do not point to the same object
        assertNotSame(str1, str3);

        //Check if 2 arrays are equal
        assertArrayEquals(expectedArray, resultArray);
    }
}
```

Output:



To demonstrate how tests are disabled:

```
package com.simplilearn.JunitDemo;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
public class Demo3 {
@BeforeAll
	public static void setUp() {
		System.out.println("Hello");
	}
	@Test
	public void test1() {
		System.out.println("Today");
	}
	@Disabled
	@Test
```

```
        public void test2() {
                System.out.println("is");
        }
        @Test
        public void test3() {
                System.out.println("Friday");
        }
        @AfterAll
        public static void tearDown() {
                System.out.println("Bye Bye");
        }
}
```

Output:

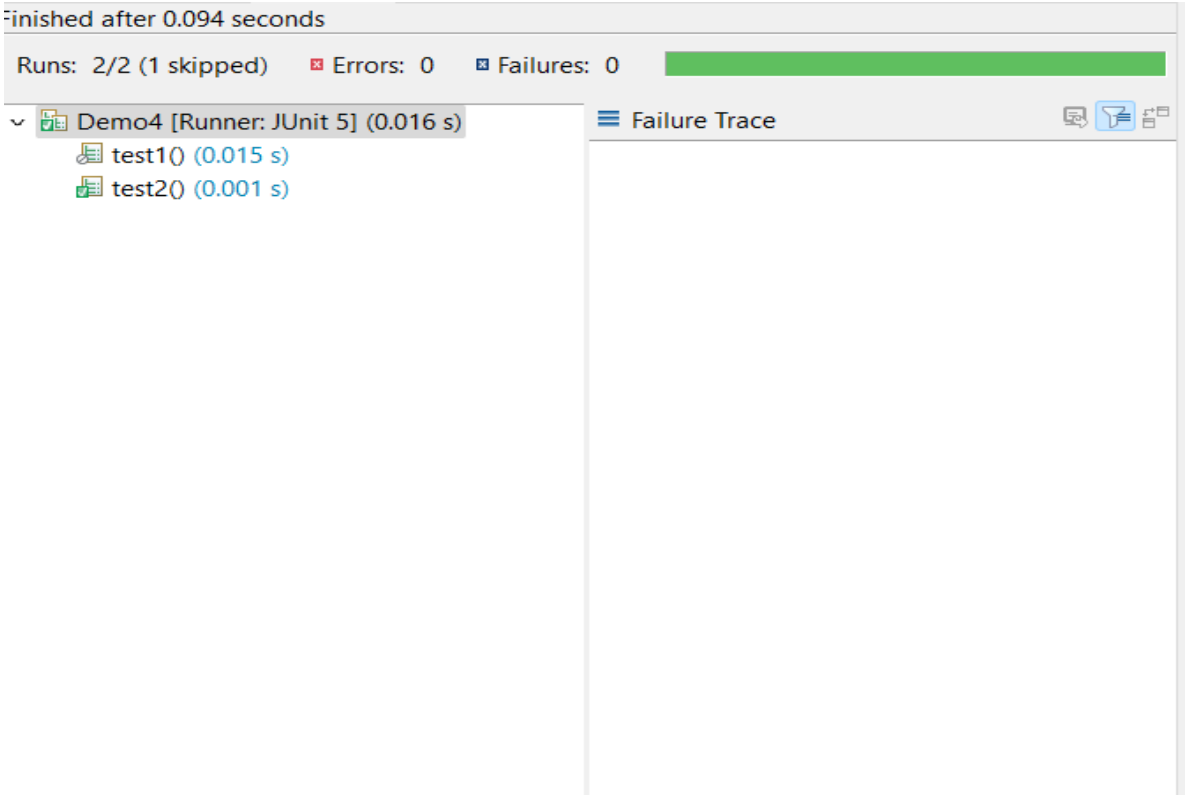
```
Console ×  Coverage  Git Staging  TestNG
<terminated> Demo3 (1) [JUnit] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17
Hello
Today
Friday
Bye Bye
```

To demonstrate assumptions in JUnit:

```
package com.simplilearn.JunitDemo;
import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.Assumptions;
import org.junit.jupiter.api.Test;
public class Demo4 {
```

```
@Test
public void test1() {
        //assertTrue("abc".contains("a"));
        Assumptions.assumeTrue("abc".contains("z"));
        System.out.println("Test1");
}
@Test
public void test2() {
        Assumptions.assumingThat("abc".contains("z"), ()-> {
                System.out.println("Friday");
        });
}}
```

Output:

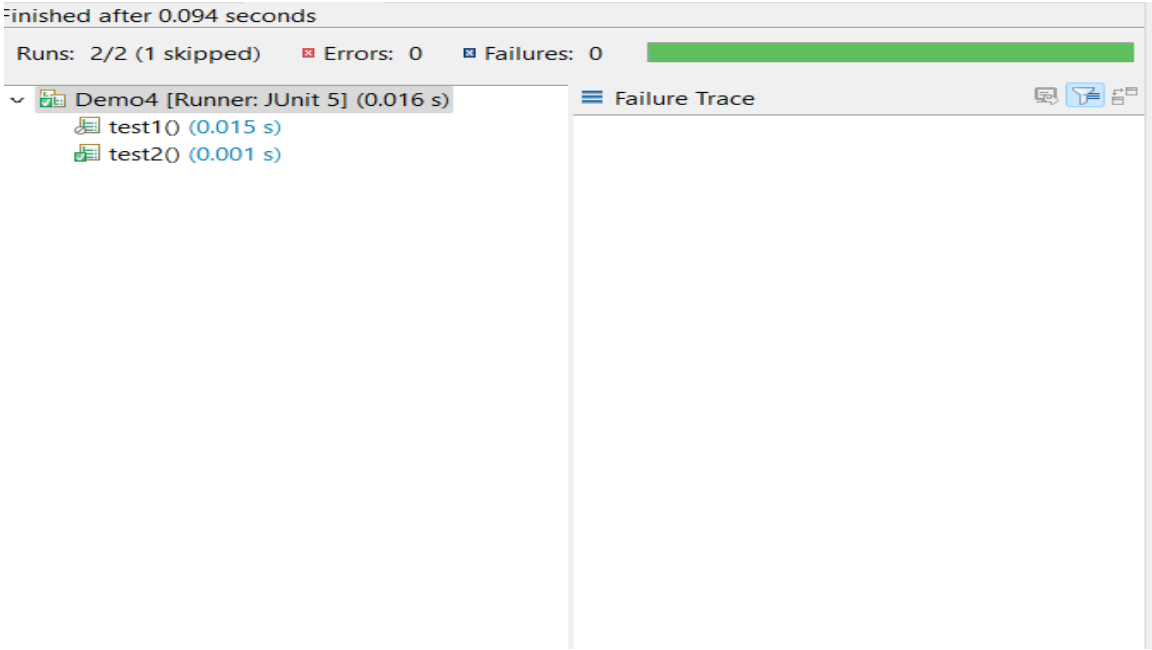


To demonstrate test interfaces and default methods in JUnit:

```
package com.simplilearn.JunitDemo;
```

```
public interface Demo5 {
public void day();
        public default void month() {
                System.out.println("It is July");
        }
}
```

Output:

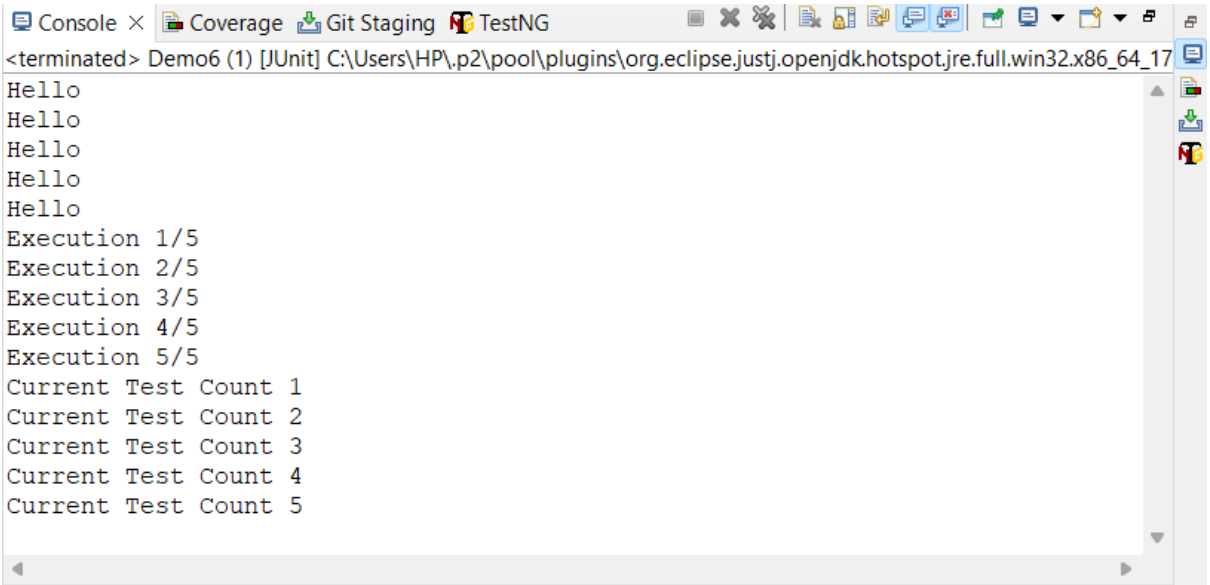


To demonstrate how tests are repeated in JUnit:

```
package com.simplilearn.JunitDemo;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.RepetitionInfo;
import org.junit.jupiter.api.TestInfo;
public class Demo6 {
        @RepeatedTest(5)
        public void test1() {
                System.out.println("Hello");
```

```
	}
	@RepeatedTest(value = 5, name = "{displayName} {currentRepetition}/{totalRepetitions}")
	@DisplayName("Execution")
	public void test2(TestInfo testinfo) {
		System.out.println(testinfo.getDisplayName());
	}
	@RepeatedTest(5)
	public void test3(RepetitionInfo repetitionInfo) {
		System.out.println("Current Test Count " + repetitionInfo.getCurrentRepetition());
	}
}
```

Output:

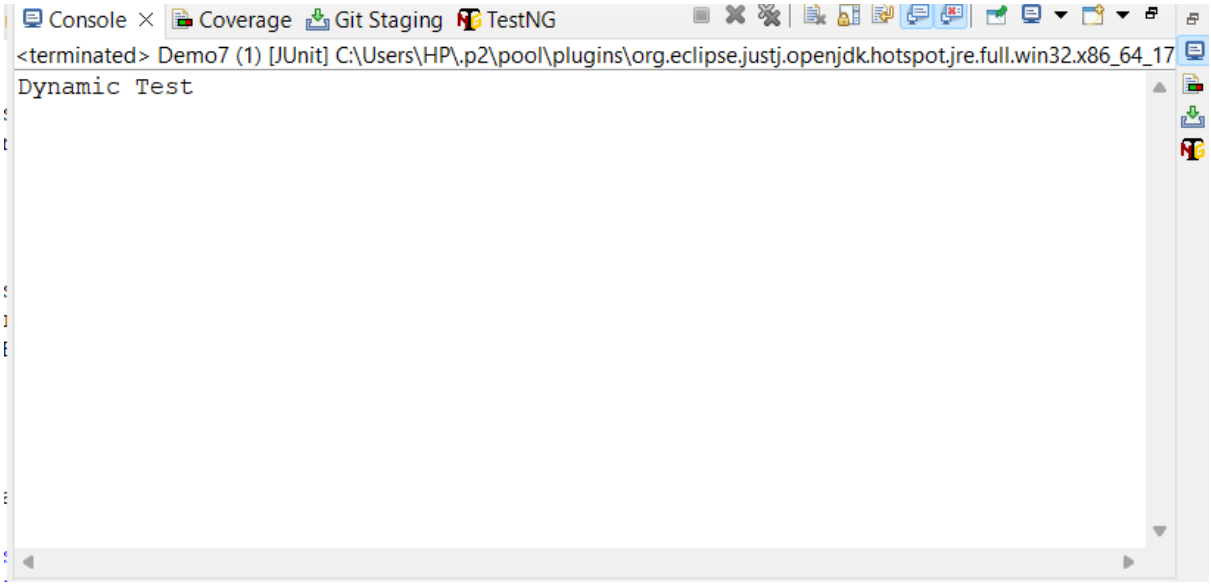


To demonstrate how dynamic tests are created in JUnit:

```
package com.simplilearn.JunitDemo;

import static org.junit.jupiter.api.Assertions.assertTrue;

import static org.junit.jupiter.api.DynamicTest.dynamicTest;
```

```
import java.util.Arrays;
import java.util.Collection;

import org.junit.jupiter.api.DynamicTest;
import org.junit.jupiter.api.TestFactory;
import org.junit.jupiter.api.function.Executable;
public class Demo7 {
	@TestFactory
	public Collection<DynamicTest> dynamicTests() {
		return Arrays.asList(
				dynamicTest("Simple Test", () -> assertTrue(true)),
				dynamicTest("Executable Class", new MyExecutable()),
				dynamicTest("Exception Executable", () -> {throw new
Exception("Exception Example");}),
				dynamicTest("Simple Test 2", () -> assertTrue(true)));
	}

}
class MyExecutable implements Executable {
	@Override
	public void execute() {
		System.out.println("Dynamic Test");
	}
}
```

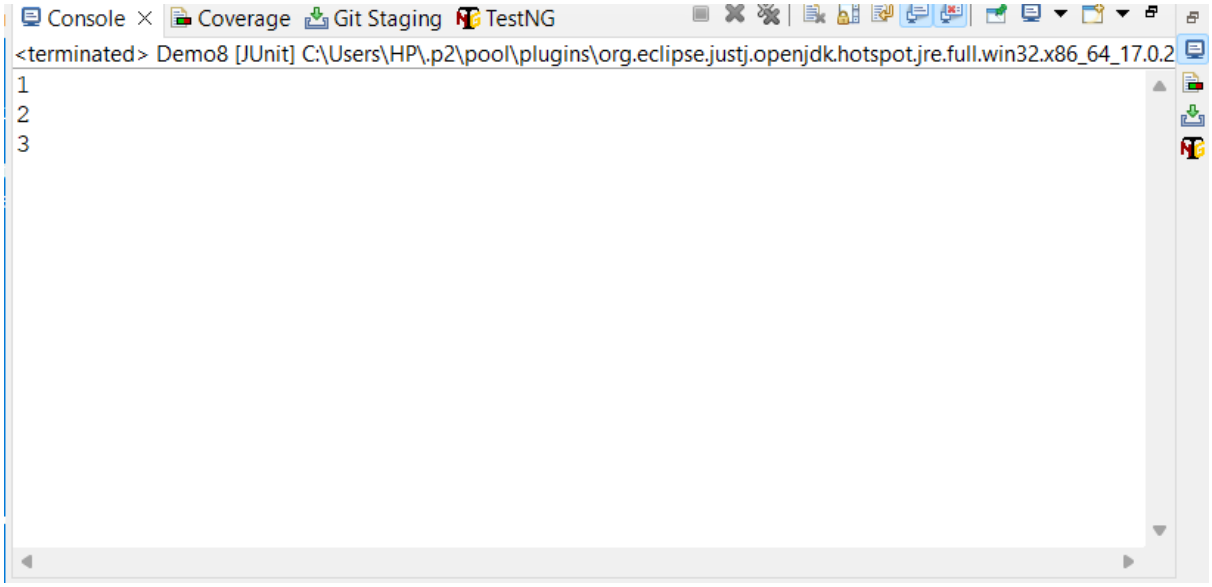
Output:



To demonstrate how parameterized tests are created in JUnit:

```
package com.simplilearn.JunitDemo;

import static org.junit.Assert.assertTrue;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

public class Demo8 {

	@ParameterizedTest

	@ValueSource(ints = {1,2,3})

	public void test1(int i) {

		System.out.println(i);

	}


	@ParameterizedTest

	@ValueSource(strings = {"4", "5", "6"})

	public void test2(String s) {

		assertTrue(Integer.parseInt(s) < 6);

	}

}
```

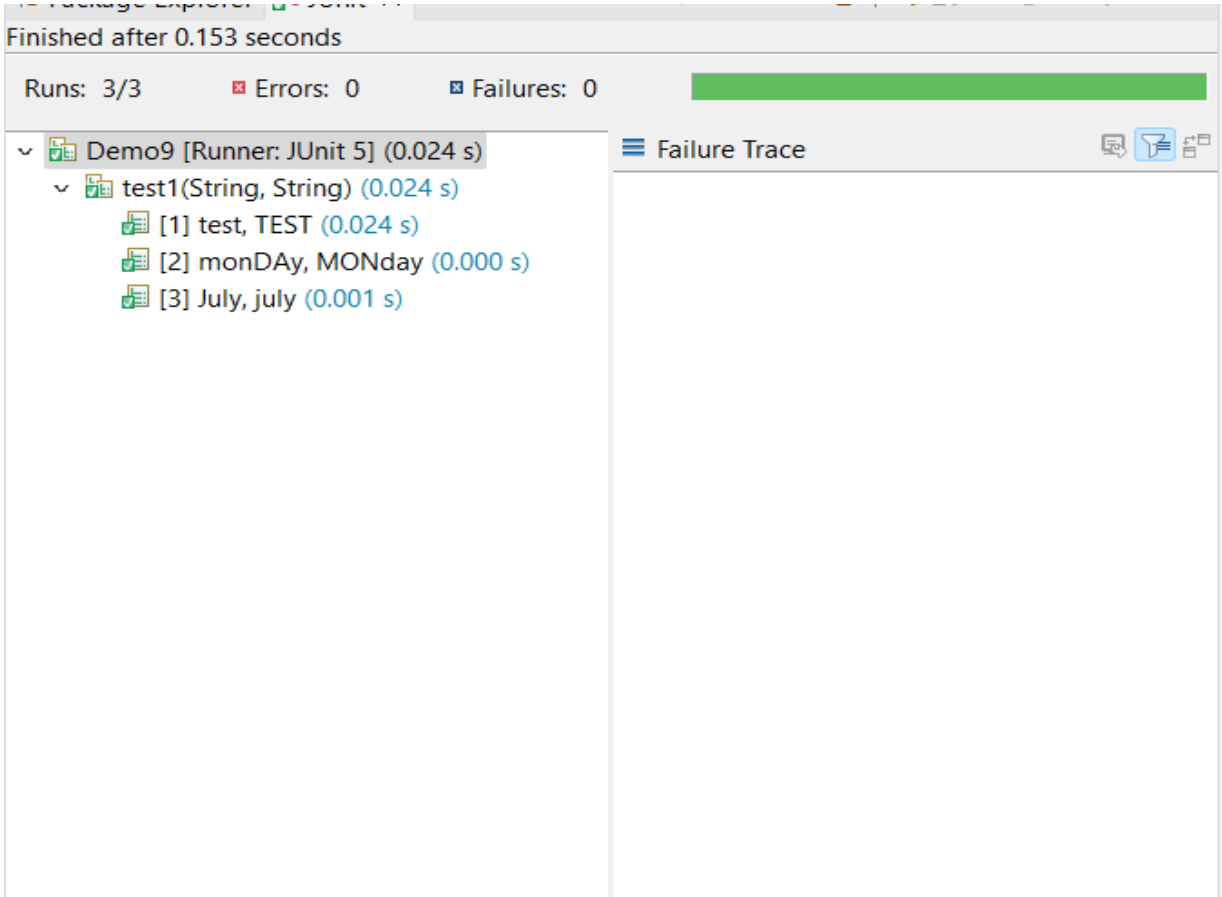
Output:



To demonstrate how argument sources are used in JUnit:

```
package com.simplilearn.JunitDemo;

import static org.junit.Assert.assertEquals;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;


public class Demo9 {

	@ParameterizedTest

	@CsvSource({"test, TEST", "monDAy, MONday", "July, july"})

	public void test1(String actual, String expected) {

		String actualValue = actual.toLowerCase();

		String expectedValue = expected.toLowerCase();

		assertEquals(actualValue, expectedValue);

	}

}
```

Output:

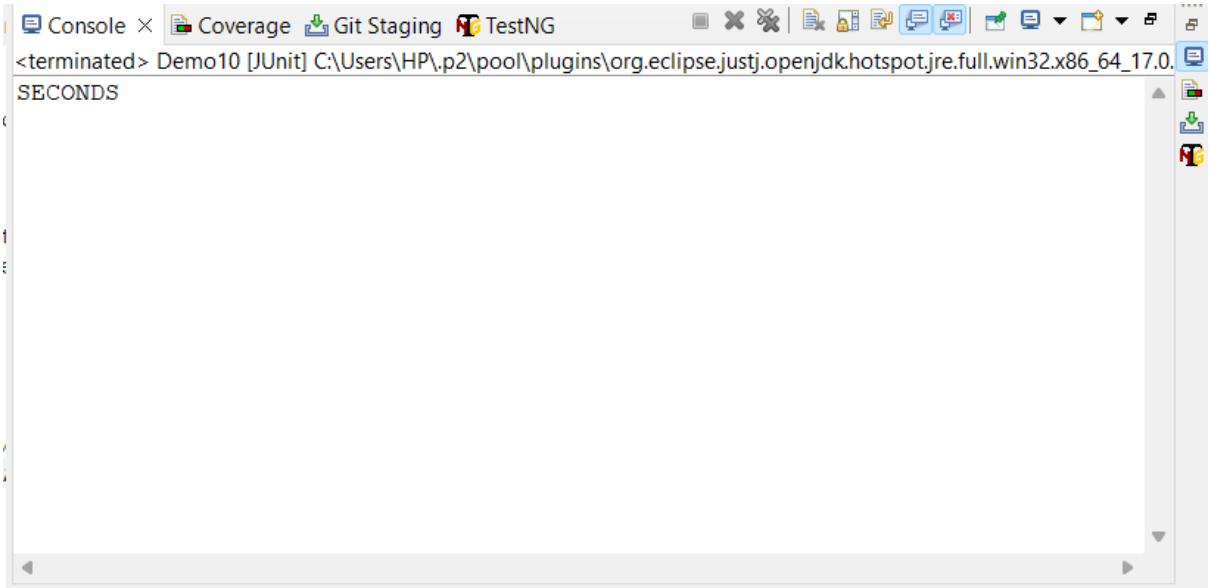


To demonstrate how argument conversions are used in JUnit:

```
package com.simplilearn.JunitDemo;

import static org.junit.Assert.assertNotNull;

import java.util.concurrent.TimeUnit;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

public class Demo10 {

	@ParameterizedTest

	@ValueSource(strings = "SECONDS")

	public void test1(TimeUnit arg) { //Implicit Conversion}

		System.out.println(arg.name());

		assertNotNull(arg.name());

	}
```

```
}
```

Output:

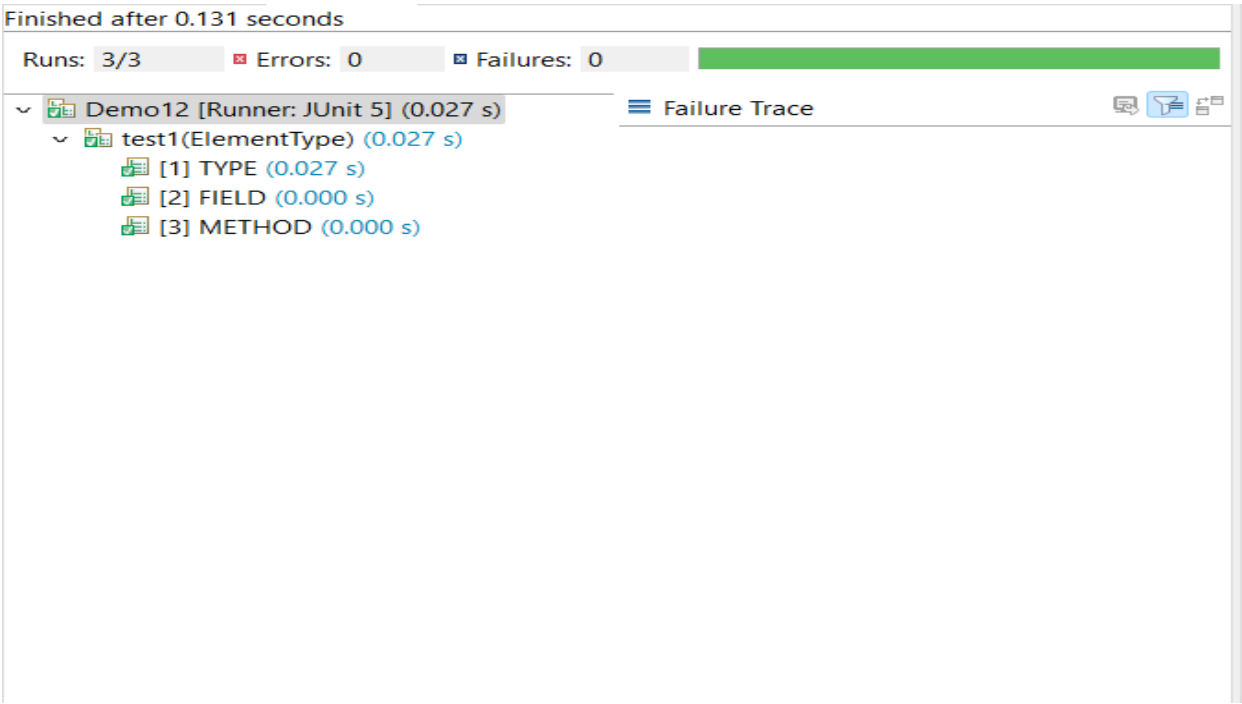



```
package com.simplilearn.JunitDemo;
import static org.junit.Assert.assertNotNull;
import java.util.concurrent.TimeUnit;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.converter.ArgumentConverter;
import org.junit.jupiter.params.converter.ConvertWith;
import org.junit.jupiter.params.provider.EnumSource;

public class Demo11 {
	@ParameterizedTest
	@EnumSource(TimeUnit.class)//Explicit Conversion
	public void test2(@ConvertWith(ArgumentConverter.class)String arg) {
		assertNotNull(TimeUnit.valueOf(arg));
	}

}
```

To demonstrate extension points:

```
package com.simplilearn.JunitDemo;

import static org.junit.Assert.assertTrue;

import java.lang.annotation.ElementType;

import java.util.EnumSet;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.EnumSource;


public class Demo12 {

	@ParameterizedTest

	@EnumSource(value = ElementType.class, names = {"TYPE", "METHOD", "FIELD" })

	public void test1(ElementType et) {

		assertTrue(EnumSet.of(ElementType.FIELD, ElementType.TYPE,
ElementType.METHOD).contains(et));

	}

}
```

Output:



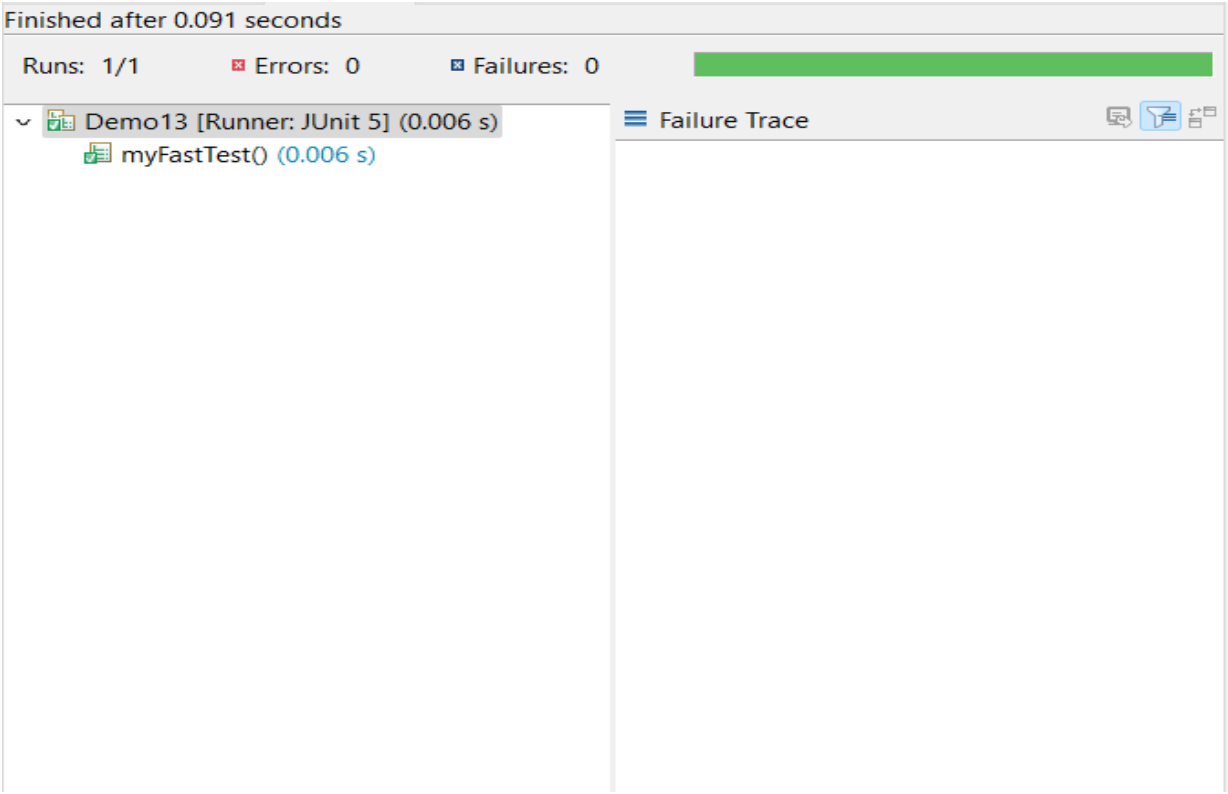
To demonstrate meta-annotation:

```
package com.simplilearn.JunitDemo;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

import org.junit.jupiter.api.Tag;

import org.junit.jupiter.api.Test;


@Target({ElementType.TYPE, ElementType.METHOD})

@Retention(RetentionPolicy.RUNTIME)

@Tag("fast")

@Test

public @interface Fast {


}
```

Now create another class

```
package com.simplilearn.JunitDemo;

public class Demo13 {

	@Fast

	public void myFastTest() {


	}

}
```

Output:

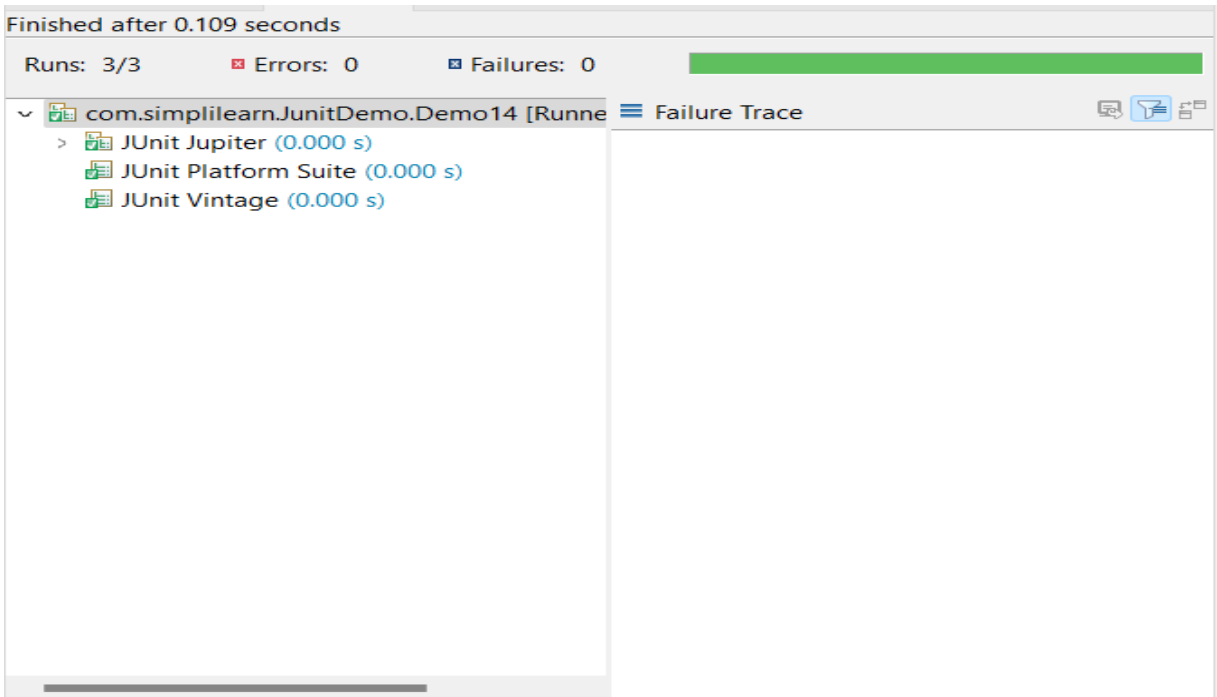


To demonstrate how tests with tags are included or excluded:

```
package com.simplilearn.JunitDemo;

import org.junit.jupiter.api.Tag;

import org.junit.jupiter.api.Test;

import org.junit.platform.runner.JUnitPlatform;

import org.junit.platform.suite.api.IncludeTags;

import org.junit.runner.RunWith;


@RunWith(JUnitPlatform.class)

@IncludeTags("production")

public class Demo14 {

        @Test

        @Tag("development")

        @Tag("production")
```

```
        public void test1() {

        }
        @Test
        @Tag("development")
        public void test2() {

        }
        @Test
        @Tag("development")
        public void test3() {

        }

}
```

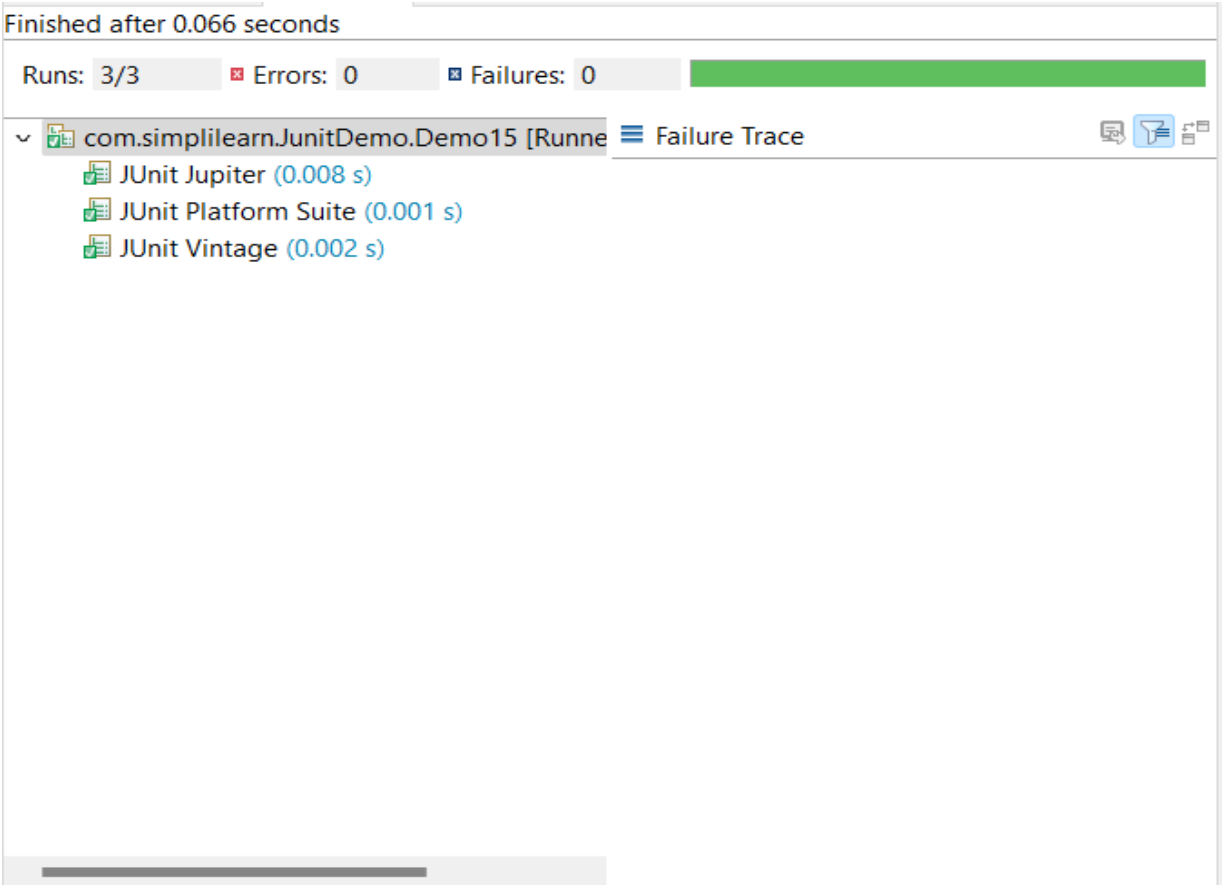
Output:



For Excluded Tags:

```
package com.simplilearn.JunitDemo;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;
import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.ExcludeTags;
import org.junit.runner.RunWith;

@RunWith(JUnitPlatform.class)
@ExcludeTags("production")
public class Demo15 {
	@Test
	@Tag("development")
	@Tag("production")
	public void test1() {

	}

	@Test
	@Tag("production")
	public void test2() {

	}

	@Test
	@Tag("production")
	public void test3() {

	}}
```

Output:


Finished after 0.066 seconds
Runs: 3/3
Errors: 0
Failures: 0
com.simplilearn.JunitDemo.Demo15 [Runne
Failure Trace
JUnit Jupiter (0.008 s)
JUnit Platform Suite (0.001 s)
JUnit Vintage (0.002 s)