Junit 5

First Steps

We Cover

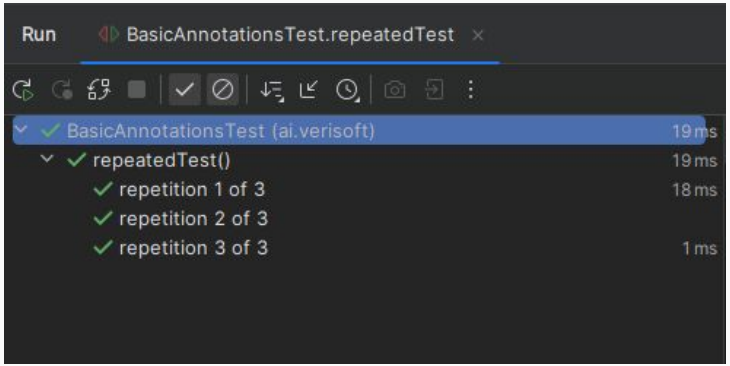
- Basic Annotations
- Assertions
- Life Cycle Annotations

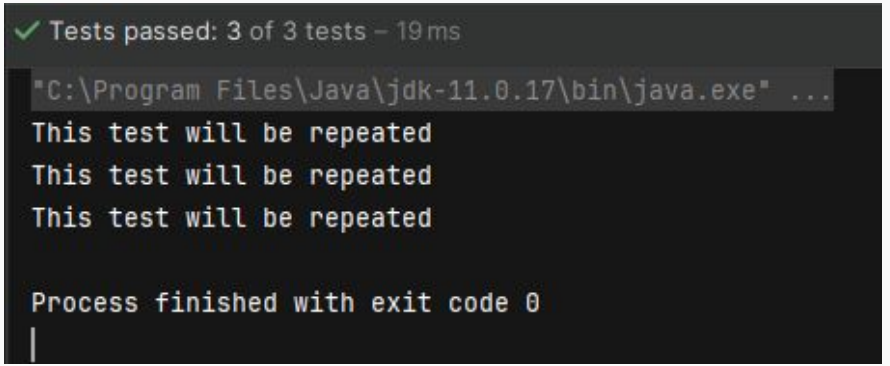
Basic Annotations

Annotation	Description
@Test	Denotes that a method is a test method
@RepeatedTest	Denotes that a method is a test template for a repeated test
@TestMethodOrder	Used to configure the test method execution order for the annotated test class
@DisplayName	Declares a custom display name for the test class or test method
@Tag	Used to declare tags for filtering tests, either at the class or method level
@Disabled	Used to disable a test class or test method
@Timeout	Used to fail a test

@RepeatedTest

```java
@RepeatedTest(3)
public void repeatedTest() {
    System.out.println("This test will be repeated");
}
```



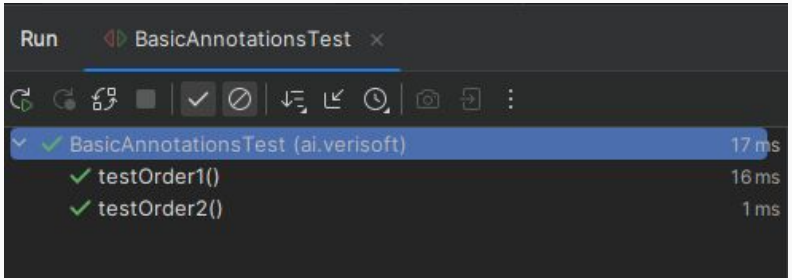


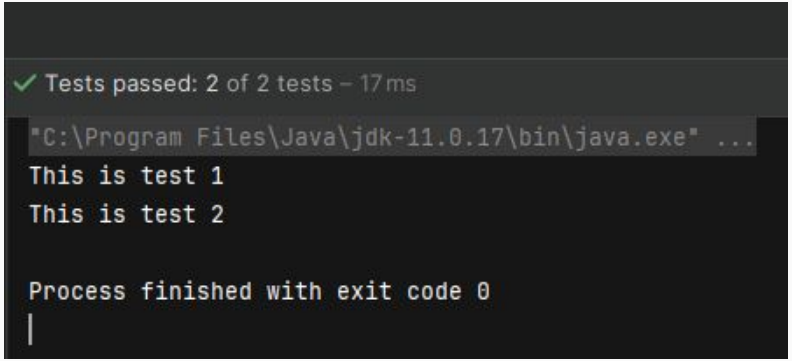
@MethodOrder

```java
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class BasicAnnotationsTest {


    @Test
    @Order(1)
    public void testOrder1() {
        System.out.println("This is test 1");
    }

    @Test
    @Order(2)
    public void testOrder2() {
        System.out.println("This is test 2");
    }
}
```



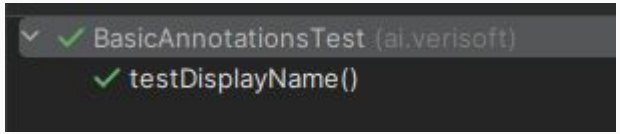


Check out the class order as well….

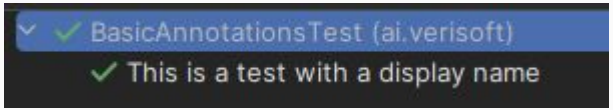
@DisplayName

```java
@Test
@DisplayName("This is a test with a display name")
public void testDisplayName() {
    System.out.println("This is a test with a display name");
}
```

Without @DisplayName



With @DisplayName



@Tag

```java
@Test
@Tag("Sanity")
public void testWithTag() {
    System.out.println("This is a test with a tag");
}
```

```
mvn clean test -Dgroups="Sanity"
```

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.039 s -- in ai.verisoft.BasicAnnotationsTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

@Timeout

```java
@Test
@Timeout(1)
public void testTimeout() throws InterruptedException {
    Thread.sleep(1200);
}
```

```
java.util.concurrent.TimeoutException: testTimeout() timed out after 1 second
```

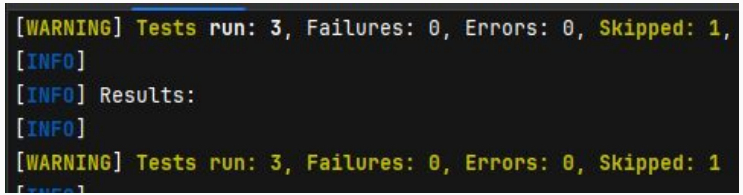
@Disabled

```java
@Test
@Tag("Sanity")
@Disabled
public void testWithTag() {
    System.out.println("This is a test with a tag");
}

@Test
@Tag("Sanity")
@Tag("Regression")
public void testWithMultipleTags() {
    System.out.println("This is a test with multiple tags");
}

@Test
@Tag("Sanity")
public void testWithRegressionTags() {
    System.out.println("This is a test with regression tags");
}
```

```
mvn clean test -Dgroups="Sanity, Regression"
```



Assertions

What is an assertion?

- How does Junit 5 knows if a test failed?
- A failure is an unhandled exception in Junit 5 (and any other unit test framework)
  - Assertions is a type of exception
  - Failed on assertion - status Failed
  - Failed on unhandled exception - status Error
- Junit 5 has its own assertions
- Other assertions packages may also be used (AssertJ for example)

AssertEquals & AssertTrue

```
@Test
void standardAssertions() {
    assertEquals(2, calculator.add(1, 1));
    assertEquals(4, calculator.multiply(2, 2),
        "The optional failure message is now the last parameter");
    assertTrue('a' < 'b', () -> "Assertion messages can be lazily evaluated -- "
        + "to avoid constructing complex messages unnecessarily.");

}
```

General Assertions

- `assertEquals(expected, actual) / assertEquals(expected, actual, message)`
- `assertNotEquals(unexpected, actual) / assertNotEquals(unexpected, actual, message)`
- `assertTrue(condition) / assertTrue(condition, message)`
- `assertFalse(condition) / assertFalse(condition, message)`
- `assertNull(actual) / assertNull(actual, message)`
- `assertNotNull(actual) / assertNotNull(actual, message)`
- `assertSame(expected, actual) / assertSame(expected, actual, message)`
- `assertNotSame(unexpected, actual) / assertNotSame(unexpected, actual, message)`
- `assertArrayEquals(expected, actual) / assertArrayEquals(expected, actual, message)`
- `assertIterableEquals(expected, actual) / assertIterableEquals(expected, actual, message)`
- `assertLinesMatch(expected, actual) / assertLinesMatch(expected, actual, message)`

Group Assertions

```
@Test
void groupedAssertions() {

    // In a grouped assertion all assertions are executed, and all
    // failures will be reported together.
    assertAll("person",
        () -> assertEquals("Jane", person.getFirstName()),
        () -> assertEquals("Doe", person.getLastName())
    );
}
```

Soft Asserts

- Not a part of Junit 5 out of the box assertions
- Can be found in VeriSoft framework
- Exists in other frameworks as well (AssertJ)

Soft Asserts

```
@Test
    public void softAsserTest(){
        SoftAsserts softAsserts = new SoftAsserts();
        boolean condition1 = false;
        boolean condition2 = false;

        // Some test code

        softAsserts.assertTrue(condition1, "Condition 1 should be true");

        // Some more test code
        System.out.println("This is a ");


        softAsserts.assertTrue(condition2, "Condition 2 should be true");

        softAsserts.assertAll();
    }
}
```

Assert Exception

```
@Test
void exceptionTesting() {
    Exception exception = assertThrows(ArithmeticException.class, () ->
        calculator.divide(1, 0));
    assertEquals("/ by zero", exception.getMessage());
}
```

Assert Exception

- `assertThrows(expectedType, executable)`
- `assertThrows(expectedType, executable, message)`
- `assertThrowsExactly(expectedType, executable)`
- `assertThrowsExactly(expectedType, executable, message)`
- `assertDoesNotThrow(executable)`
- `assertDoesNotThrow(executable, message)`

Timeout Assertion

```
@Test
void timeoutNotExceeded() {

    // The following assertion succeeds.
    assertTimeout(ofMinutes(2), () -> {
    // Perform task that takes less than 2 minutes.
    });
}
```

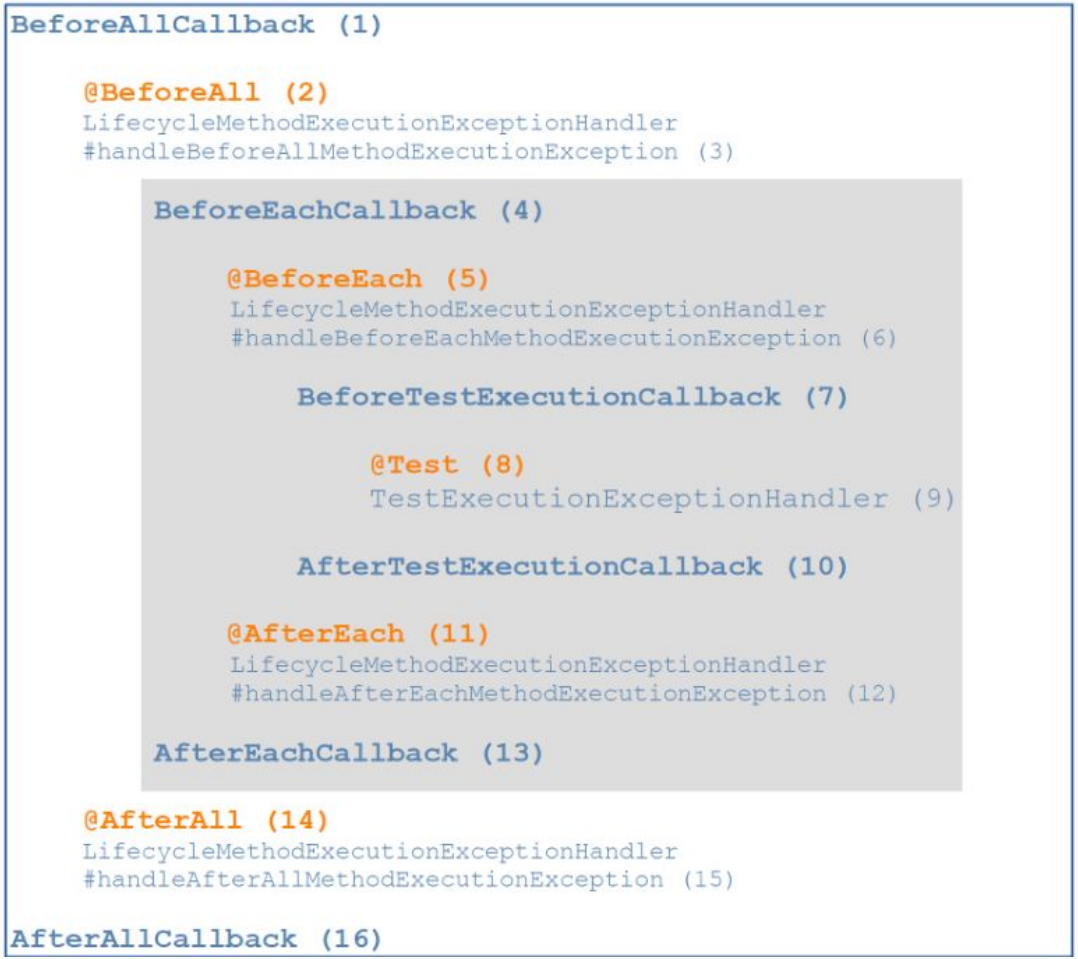
Timeout Assert

- `assertTimeout(duration, executable)`
- `assertTimeout(duration, executable, message)`
- `assertTimeoutPreemptively(duration, executable)`
- `assertTimeoutPreemptively(duration, executable, message)`

Assertions - Last notes

- `fail() / fail(message)`
- `AssertJ`
- `Assume`
- `Soft Assert`

Life Cycle Annotations



Test Life Cycle

LifeCycle Annotations

Annotation	Description
@BeforeAll	Denotes that the annotated method should be executed before all tests in a given class
@AfterAll	Denotes that the annotated method should be executed after all tests in a given class
@BeforeEach	Denotes that the annotated method should be executed before each test in a given class
@AfterEach	Denotes that the annotated method should be executed after each test in a given class

@BeforeAll, @AfterAll, @BeforeEach, @AfterEach

```
public class LifeCycleAnnotationTest {

    @BeforeAll
    public static void beforeAll() {
        System.out.println("In the before all tests");
    }

    @AfterAll
    public static void afterAll() {
        System.out.println("In the after all tests");
    }

    @BeforeEach
    public void beforeEach() {
        System.out.println("In the before each test");
    }

    @AfterEach
    public void afterEach() {
        System.out.println("In the after each test");
    }

    @Test
    public void test1() {
        System.out.println("test #1");
    }

    @Test
    public void test2() {
        System.out.println("test #2");
    }

}
```

@BeforeAll, @AfterAll, @BeforeEach, @AfterEach

```
In the before all tests
In the before each test
test #1
In the after each test
In the before each test
test #2
In the after each test
In the after all tests
```

@BeforeAll, @AfterAll, @BeforeEach, @AfterEach

```
@BeforeAll
public static void beforeAll() {
    System.out.println("In the before all tests");
}

@AfterAll
public static void afterAll() {
    System.out.println("In the after all tests");
}
```

To be continued…