JUNIT 5

The Next Evolution of Unit Testing

John Pendexter

About me

- Consultant at Manifest Solutions
- Writer of Unit Tests
- Aspiring mandolin player

What will we cover?

- Overview of JUnit 5/Jupiter
- New testing features of JUnit 5
- Better test structuring with JUnit 5 features
- JUnit 5 extensions
- Migration from legacy test suites

Why should I upgrade?

JUnit 4 works just fine!

JUnit 5 is basically an entire rewrite, isn't that risky?

- Takes advantage of Java 8 (lambdas!)
- Features to allow for better test structuring
- Some of the annotations are named more clearly
- We're developers, we like using new technologies
- Its only test code!

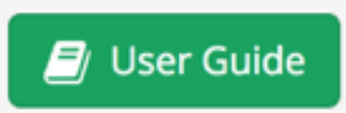
JUnit

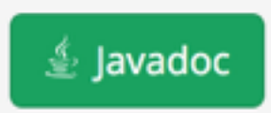
Requirements

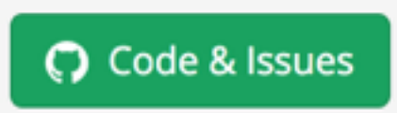
- Java 8 or greater
- JUnit 5
- Desire to write better tests
- *IDE with support for JUnit 5

JUnit 5

The new major version of the programmer-friendly testing framework for Java 8







http://junit.org/junit5/

JUnit 5 includes several sub-projects

JUnit Jupiter (**`org.junit.jupiter`**)
JUnit Platform (**`org.junit.platform`**)
JUnit Vintage (**`org.junit.vintage`**)

A note on philosophy…

JUnit

Annotations

`@Test`

`@RepeatedTest`

`@TestFactory`

`@DisplayName`

`@BeforeEach`

`@AfterEach`

`@BeforeAll`

`@AfterAll`

`@Nested`

`@Tag`

`@Disabled`

`@ExtendWith`

`@ParameterizedTest`

`@TestInstance`

`@TestTemplate`

Stops

Used

An

Meta-annotations

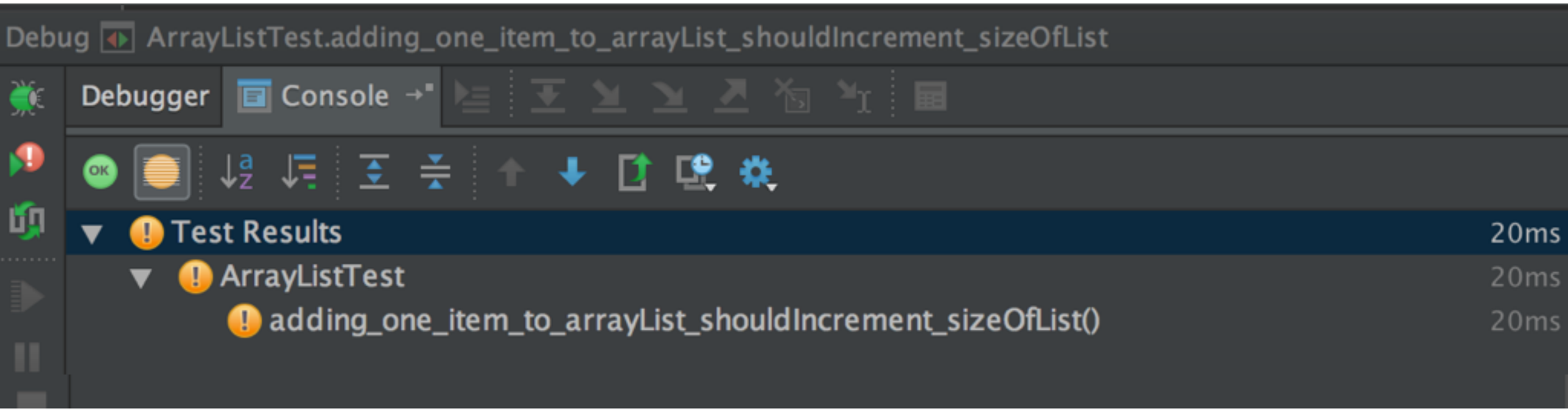
Meta-annotations

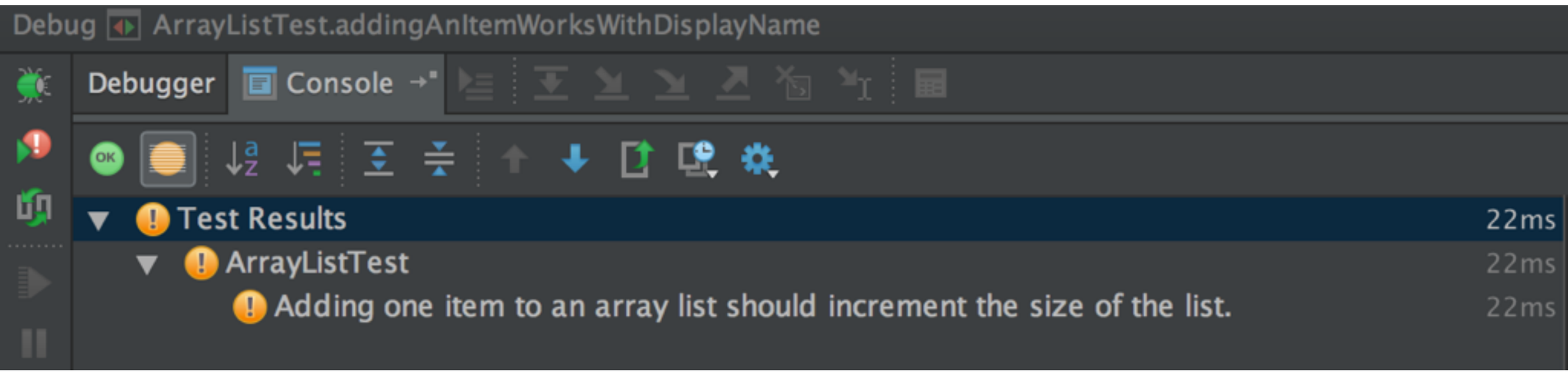
```
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@
@
pu
}
```

```
@NetworkIntegrationTest
public class SpotifyServiceIntegrationTest {
    ...
}
```

JUnit 5

@DisplayName





@TestInstance

```
@TestInstance(Lifecycle.PER_METHOD)
```

```
@TestInstance(Lifecycle.PER_CLASS)
```

JUnit 5

Assertions

```
@Test
@DisplayName("Spotify service returns valid top tracks list.")
public void spotifyServiceCanReturnTopTracksList() throws IOException {
    TopTracksList list = spotifyService.getTopTracksList(ARTIST_ID);
    assertNotNull(list);
    assertEquals(10, list.getTracks().size(), "The number of actual top
tracks did not match the expected.");
}
```

```
    assertNotNull(list);
    assertEquals(10, list.getTracks().size(),
"The number of actual top tracks did not match the expected."
);
```

JUnit 5

Assertions

```java
@Test
@DisplayName("…")
public void multipleAsserts() throws IOException {
    TopTracksList topTracksList = spotifyService.getTopTracksList(ARTIST_ID);
    List<Track> tracks = topTracksList.getTracks();

    assertEquals("Uncle Pen", tracks.get(0).getName());
    assertEquals("Southern Flavor", tracks.get(1).getName());
    assertEquals("Man of Constant Sorrow", tracks.get(2).getName());
    assertEquals("Pancho and Lefty", tracks.get(3).getName());
}
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java ...
Picked up JAVA_TOOL_OPTIONS: -Djava.awt.headless=true
Connected to the target VM, address: '127.0.0.1:61878', transport: 'socket'

org.opentest4j.AssertionFailedError: The track name was not what was expected. ==>
Expected :Man of Constant Sorrow
Actual   :Foggy Mountain Breakdown
 <Click to see difference>

 <5 internal calls>
    at com.pendext.junit.spotify.SpotifyServiceIntegrationTest.spotifyServiceReturnsExpec

Disconnected from the target VM, address: '127.0.0.1:61878', transport: 'socket'

Process finished with exit code 255
```

JUnit 5

Assertions

```
@Test
@DisplayName("…")
public void assertAllLambda() throws IOException {
    TopTracksList topTracksList = spotifyService.getTopTracksList(ARTIST_ID);
    List<Track> tracks = topTracksList.getTracks();
    assertAll("Top tracks returned from Spotify are exactly the tracks expected.",
            () -> assertEquals("Uncle Pen", tracks.get(0).getName()),
            () -> assertEquals("Southern Flavor", tracks.get(1).getName()),
            () -> assertEquals("Man of Constant Sorrow", tracks.get(2).getName()),
            () -> assertEquals("Pancho and Lefty", tracks.get(3).getName()));
}
```

JUnit 5

Assertions

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java ...
Picked up JAVA_TOOL_OPTIONS: -Djava.awt.headless=true
Connected to the target VM, address: '127.0.0.1:61956', transport: 'socket'


Expected :Man of Constant Sorrow
Actual   :Foggy Mountain Breakdown
 <Click to see difference>


Expected :Pancho and Lefty
Actual   :I Saw The Light
 <Click to see difference>


org.opentest4j.MultipleFailuresError: Top tracks returned from Spotify are exactly the tracks expected. (2 failures)
    The track name was not what was expected. ==> expected: <Man of Constant Sorrow> but was: <Foggy Mountain Breakdown>
    The track name was not what was expected. ==> expected: <Pancho and Lefty> but was: <I Saw The Light>
 <3 internal calls>
    at com.pendext.junit.spotify.SpotifyServiceIntegrationTest.spotifyServiceReturnsExpectedTopTracks(SpotifyServiceInteg

Disconnected from the target VM, address: '127.0.0.1:61956', transport: 'socket'

Process finished with exit code 255
```

Assertions

JUnit 4 exception testing

```
@Test(expected = IOException.class)
public void exceptionTestingStrategy1() throws IOException {
    exceptionsExample.basicExceptionExample("message");
    fail("This code is unreachable!”); // test passes
}
```

JUnit 5

Assertions

JUnit 4 exception testing

```java
@Test
public void exceptionTestingStrategy2() {
    String expectedMessage = RandomStringUtils.randomAlphanumeric(10);
    try {
        exceptionsExample.basicExceptionExample(expectedMessage);
    } catch (IOException e) {
        assertEquals(expectedMessage, e.getMessage());
    }
}
```

Assertions

JUnit 4 exception testing

```
@Rule
public ExpectedException expectedException =
ExpectedException.none();

@Test
public void exceptionTestingStrategy3() throws IOException {
    expectedException.expect(IOException.class);
    expectedException.expectMessage("message");
    exceptionsExample.basicExceptionExample("not a message");
}
```

Assertions

Testing Exceptions

```java
@Test
@DisplayName("This test should throw an IO exception. ")
public void basicExceptionExample() {
    String expectedMessage = RandomStringUtils.randomAlphabetic(10);
    Throwable actualException = assertThrows(IOException.class, () ->
            exceptionExample.basicExceptionExample(expectedMessage)
    );
    assertEquals(expectedMessage, actualException.getMessage());
}
```

JUnit 5

Assertions

Testing Exceptions

```
@Test
@DisplayName("This test shows the assertThrows assertion within an assertAll")
public void variousExceptionsExampleWithLambdas() {
    assertAll("Test against various exceptions being throw from a single method",
            () -> {
                Throwable actualException = assertThrows(IOException.class, () ->
                        exceptionExample.variousExceptionsExample(1)
                );
                assertEquals("expected message", actualException.getMessage());
            },
            () -> assertThrows(RuntimeException.class, () ->
                    exceptionExample.variousExceptionsExample(1)
            ),
            () -> assertThrows(ClassCastException.class, () ->
                    exceptionExample.variousExceptionsExample(2)
            ),
            () -> assertThrows(CompilerException.class, () ->
                    exceptionExample.variousExceptionsExample(4)
            )
    );
}
```

Assertions

Third party assertion libraries

JUnit 5 does not have the equivalent of JUnit 4's `assertThat()` that takes a Hamcrest `Matcher`.

Developers are encouraged to use third party assertion libraries in conjunction with JUnit 5.

AssertJ

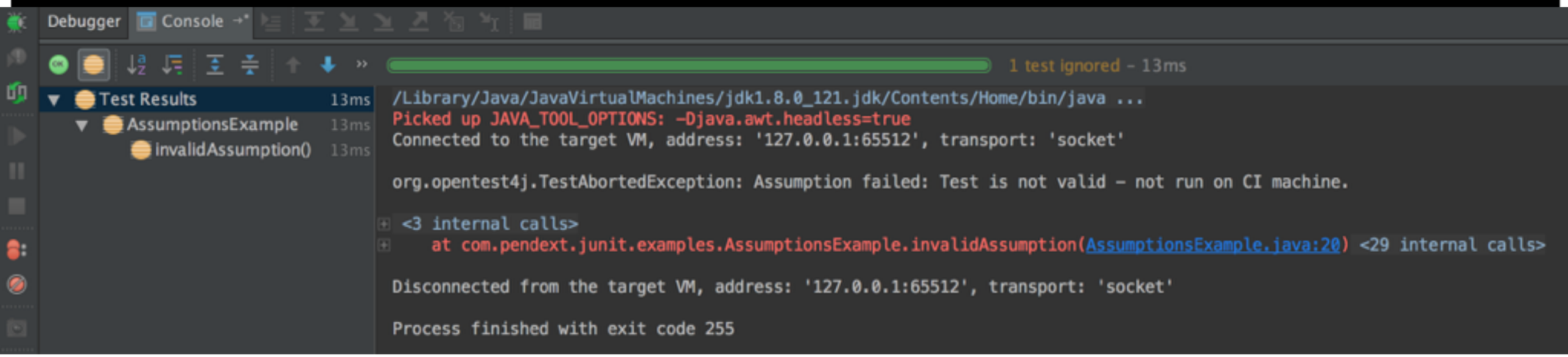
Hamcrest

JUnit 5

Assumptions

```java
@Test
public void validAssumption() {
    assumeTrue(System.getProperty("user.country").equals("US")); // allows the test to continue
    assertEquals(2, 3);
}
```

```java
@Test
public void invalidAssumption() {
    assumeTrue("CI".equals(System.getenv("ENV")),
            () -> "Test is not valid - not run on CI machine."); // does not allow the test to continue
    assertEquals(2, 2);
}
```



Assumptions

```
@Test
public void assumingExampleWithAssertAll() {
    assertAll("Show usage of assumptions within an assertAll",
            () -> assumingThat(System.getProperty("user.country").equals("CZ"),
                  () -> assertEquals(3, 3)),
            () -> assumingThat(System.getProperty("user.country").equals("US"),
```

1 test failed - 24ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java ...
Picked up JAVA_TOOL_OPTIONS: -Djava.awt.headless=true
Connected to the target VM, address: '127.0.0.1:49212', transport: 'socket'


Expected :2
Actual   :3
 <Click to see difference>



Expected :expected
Actual   :actual
 <Click to see difference>



org.opentest4j.MultipleFailuresError: Show usage of assumptions within an assertAll (2 failures)
    expected: <2> but was: <3>
    expected: <expected> but was: <actual>
 <3 internal calls>
    at com.pendext.junit.examples.AssumptionsExample.assumingExampleWithAssertAll(AssumptionsExample.java:34) <29 internal calls>

Disconnected from the target VM, address: '127.0.0.1:49212', transport: 'socket'
```

Tagging and Filtering

```java
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Tag("integration")
public @interface IntegrationTest {
}
```

```java
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@IntegrationTest
@Tag("requiresNetwork")
public @interface NetworkIntegrationTest {
}
```

```xml
<artifactId>maven-surefire-plugin</artifactId>
<version>2.19</version>
<configuration>
    <goal>
        <properties>
            <includeTags>*</includeTags>
            <excludeTags>integration</excludeTags>
        </properties>
    </goal>
</configuration>
```

Tag Syntax Rules

- A tag must not be null or blank.
- A tag must not contain whitespace.
- A tag must not contain ISO control characters.

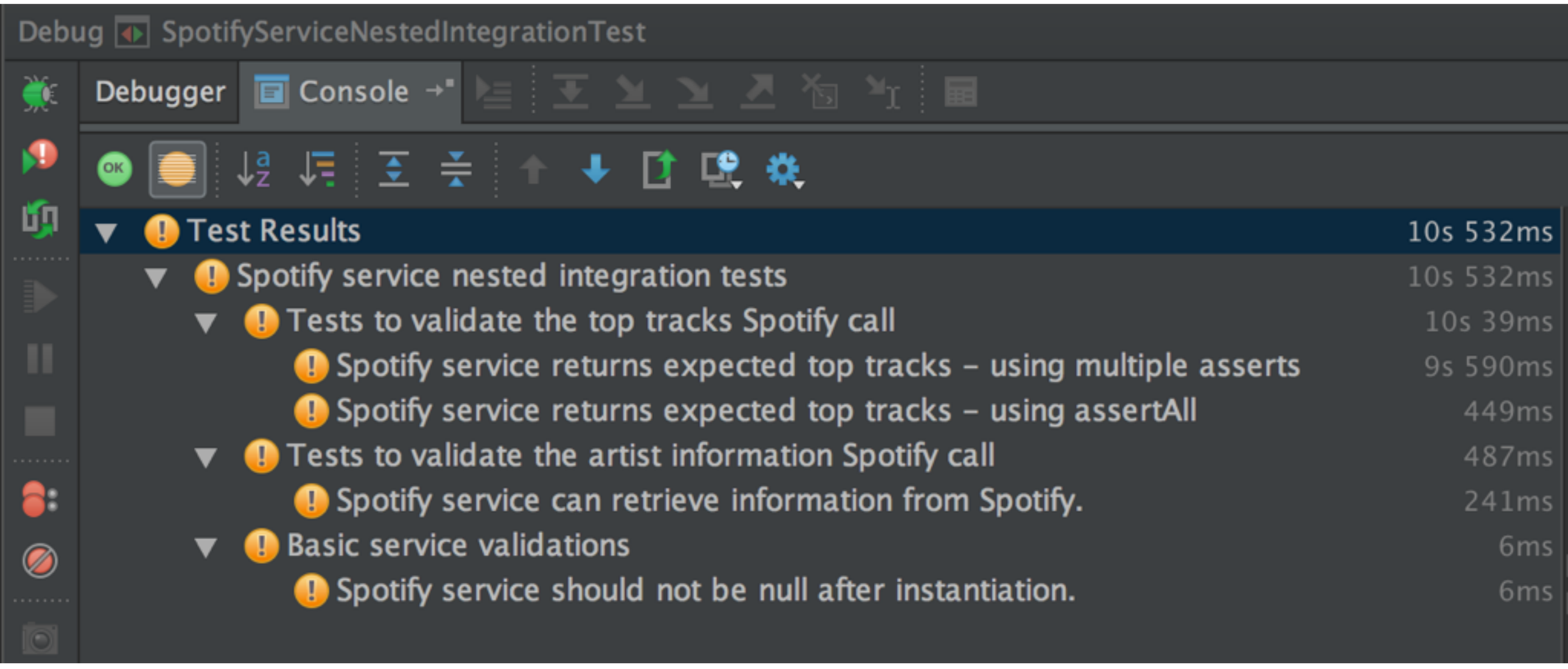
JUnit 5

Nested Tests

```
class SpotifyServiceNestedIntegrationTest {
    private static SpotifyService spotifyService;
    private final String ARTIST_ID = “5CWbfANRpZbnxdstzcNg5H";

    @BeforeEach
    public void beforeEach() {
        spotifyService = new SpotifyService();
    }

    @Nested
    @DisplayName("Basic service validations")
    class BasicTest {
        @Test
        @DisplayName("Spotify service should not be null after instantiation.")
        public void spotifyServiceCanBeInstantiated() {
            assertTrue(spotifyService != null);
        }
    }

    …
}
```

JUnit 5

Nested Tests

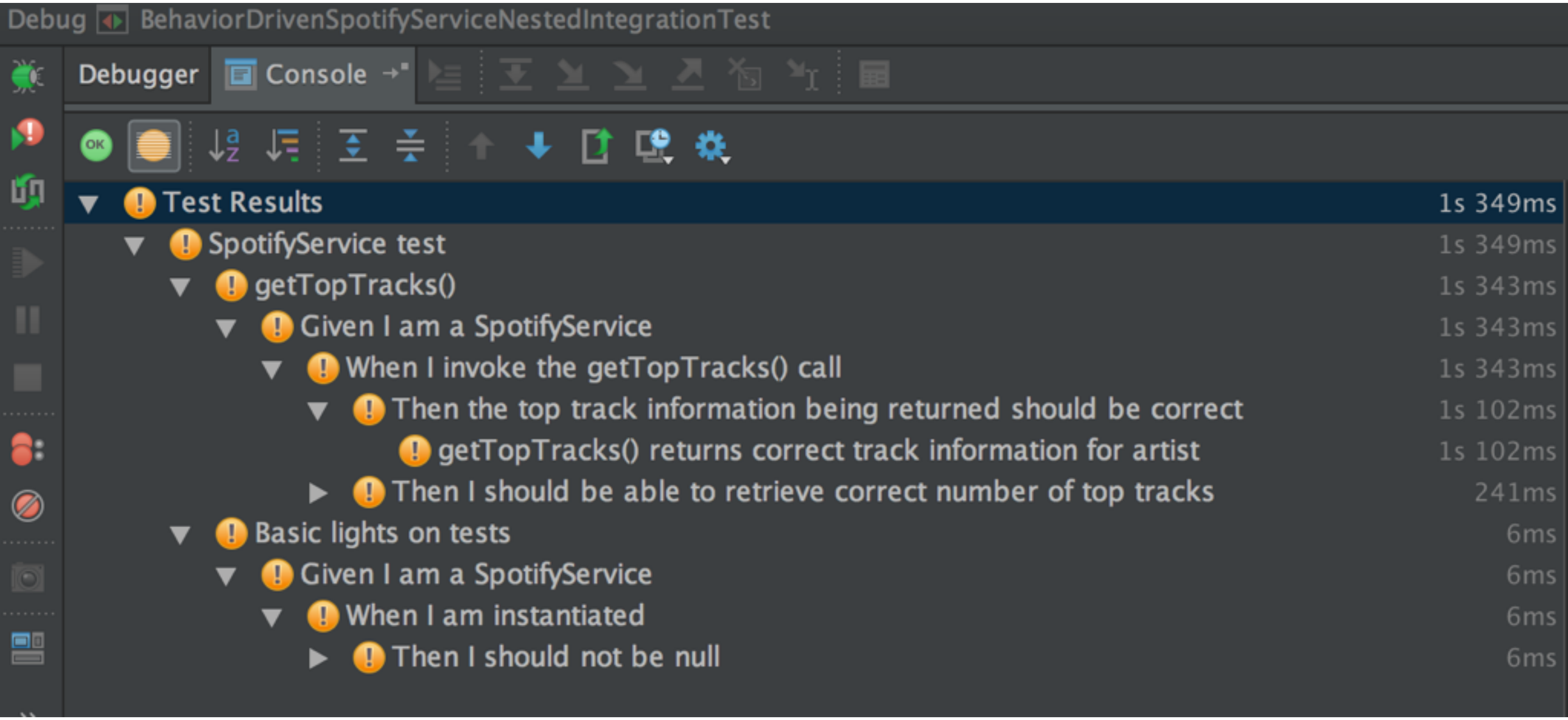


BDD Style Tests

```java
@Nested
@DisplayName("Basic lights on tests")
class BasicTests {
    @Nested
    @DisplayName("Given I am a SpotifyService")
    class Given {
        @Nested
        @DisplayName("When I am instantiated")
        class When {
            @Nested
            @DisplayName("Then I should not be null")
            class Then {
                @Test
                @DisplayName("Spotify service should not be null after instantiation.")
                public void spotifyServiceCanBeInstantiated() {
                    assertTrue(spotifyService != null);
                }
            }
        }
    }
}
```

JUnit 5

BDD Style Tests



JUnit 5

Constructor and Method Injection

Prior to JUnit 5 test methods and constructors were unable to have parameters (using the standard **`Runner`** implementation).

In JUnit 5…

```
org.junit.jupiter.api.extension.ParameterResolver
```

Defines the API for injecting parameters dynamically at runtime.

JUnit 5

Constructor and Method Injection

```
org.junit.jupiter.api.extension.ParameterResolver
```

Applies to

```
public TestConstructor() { … }, @Test, @TestFactory,
@BeforeEach, @AfterEach, @BeforeAll, @AfterAll
```

As long as the parameter can be resolved at runtime with a registered `ParameterResolver`

JUnit 5

Constructor and Method Injection

Built in **`ParameterResolver`**s

TestInfoParameterResolver

RepetitionInfoParameterResolver

TestReporterParameterResolver

JUnit 5

Constructor and Method Injection

TestInfoParameterResolver ,
RepetitionInfoParameterResolver

```
@Test
@RepeatedTest(value = 10, name = "{currentRepetition} / {totalRepetitions}")
@DisplayName("Repeat!")
public void repeatedTestExample(TestInfo testInfo,
RepetitionInfo repetitionInfo) {
    assertEquals(testInfo.getDisplayName(),
            "Repeat! " + repetitionInfo.getCurrentRepetition() + " / " +
repetitionInfo.getTotalRepetitions());

}
```

JUnit 5

Constructor and Method Injection

TestReporterParameterResolver

```
@Test
public void testReporterExample(TestReporter testReporter) throws IOException {

    testReporter.publishEntry("start time", String.valueOf(LocalDateTime.now()));

    // test goes here!!

    testReporter.publishEntry("end time", String.valueOf(LocalDateTime.now()));
}
```

Default Methods/Test Interfaces

```
public interface DefaultMethodInterface {

    default void defaultMethod() {
        // default code goes here
    }

}
```

JUnit 5

Default Methods/Test Interfaces

```
public interface TestDecorator {

    Logger logger = LoggerFactory.getLogger(“test-logger");

    @BeforeAll
    static void beforeAll() {
        // logging or other work here
    }
    @AfterAll
    static void afterAll() {
        // logging or other work here
    }
    @BeforeEach
    default void beforeEach(TestInfo testInfo) {
        // logging or other work here
    }
    @AfterEach
    default void afterEach(TestInfo testInfo) {
        // logging or other work here
    }
}
```

JUnit 5

Default Methods/Test Interfaces

Creating tests against interface contracts

```
class ArtistInfoTest implements EqualsTestable<ArtistInfo> {

    private String billMonroeJson = "{ ... }";

    private String otherArtistJson = "{ ... }";

    @Override
    public ArtistInfo createObject() throws IOException {
        return new ArtistInfo(billMonroeJson);
    }
    @Override
    public ArtistInfo createUnequalObject() throws IOException {
        return new ArtistInfo(otherArtistJson);
    }
}
```

Extension Model

JUnit 4 had `Runner`, `@Rule`, and `@ClassRule` for extending the behavior of test classes

JUnit 5 has the annotation `@ExtendWith(ExtensionClass.class)`

```
@ExtendWith({ FooExtension.class, BarExtension.class })
class ArtistInfoTest {
    // ...
}
```

Extension Model

`@ExtendWith` defines a set of APIs for extending the behavior of JUnit 5 test classes

- `ContainerExecutionCondition`
  `TestExecutionCondition`
- `TestInstancePostProcessor`
- `ParameterResolver`

Extension Model

JUnit 5 also includes container/test level execution callbacks as part of the extension model

```
BeforeAllCallback
    BeforeEachCallback
        BeforeTestExecutionCallback
        AfterTestExecutionCallback
    AfterEachCallback
AfterAllCallback
```

JUnit 5

Parameterized Tests

```
@ParameterizedTest
@ValueSource(strings = { "This", "is", "a", "parameterized",
"test"})
public void basicParameterizedTestExample(String argument) {
    assertNotNull(argument);
}
```

`@ValueSource` allows for `int`, `long`, `double`, and `String` types

Parameterized Tests

```
@ParameterizedTest
@MethodSource("provider")
void testWithMethodSource(String argument) {
    assertNotNull(argument);
}

static Stream<String> provider() {
    return Stream.of("This", "is", "a", "parameterized",
"test");
}
```

JUnit 5

Parameterized Tests

Also available…

```
@ParameterizedTest
@CsvFileSource(resources = "/twoColumnFile.csv")
void testWithCsvFileSource(String first, int second) {
    // ... assertions go here
}
```

```
@ArgumentsSource(NewArgumentsProvider.class) // … test method goes here
static class NewArgumentsProvider implements ArgumentsProvider {
/* ... */
}
```

What else is new?

Test templates

Dynamic tests

Combinations of all of these new features!

Upgrading from JUnit 4

Although the JUnit Jupiter programming model and extension model will not support JUnit 4 features such as `Rules` and `Runners` natively, it is not expected that source code maintainers will need to update all of their existing tests, test extensions, and custom build test infrastructure to migrate to JUnit Jupiter.

Source: http://junit.org/junit5/docs/current/user-guide/#migrating-from-junit4

Upgrading from JUnit 4

But what about my JUnit 4 (or even, *gasp* JUnit 3) tests?

Just make sure you have the junit-vintage-engine artifact included in your project and the existing tests will be picked up by the JUnit Platform Launcher.

Source: http://junit.org/junit5/docs/current/user-guide/#migrating-from-junit4

Upgrading from JUnit 4

Limited JUnit 4 `Rule` Support

`org.junit.rules.ExternalResource` (including **`org.junit.rules.TemporaryFolder`**)

`org.junit.rules.Verifier` (including **`org.junit.rules.ErrorCollector`**)

`org.junit.rules.ExpectedException`

Source: http://junit.org/junit5/docs/current/user-guide/#migrating-from-junit4

Upgrading from JUnit 4

Limited JUnit 4 **`Rule`** Support

These **`Rule`**s will work unchanged in legacy test suites.

Source: http://junit.org/junit5/docs/current/user-guide/#migrating-from-junit4

What is missing or still in development?

Spring Framework Integration (in progress)

Mocking Framework Integration (in process)

~~An initial release candidate!~~

Resources

http://junit.org/junit5/

https://github.com/junit-team/junit5-samples/tree/master/junit5-mockito-extension

https://github.com/sbrannen/spring-test-junit5

https://github.com/pendext

JUnit

Questions?