# *selenium-jupiter*

## *A JUnit 5 extension for Selenium WebDriver*

Boni García

Version 1.1.0

# Table of Contents

JUnit 5 is the next generation of the well-known testing framework JUnit. *Jupiter* is the name given to the new programming and extension model provided by JUnit 5. Regarding the extension model of JUnit 5, it allows to incorporate extra capabilities for Jupiter tests. On the other hand, Selenium WebDriver is a testing framework which allows to control browsers (e.g. Chrome, Firefox, and so on) programmatically to carry out automated testing of web applications. This documentation presents ***selenium-jupiter***, a JUnit 5 extension aimed to provide seamless integration of Selenium WebDriver within Jupiter tests. *selenium-jupiter* is open source (Apache 2.0 license) and is hosted on GitHub.

# Quick reference

***selenium-jupiter*** has been built using the dependency injection capability provided by the extension model of JUnit 5. Thank to this feature, different types objects can be injected in JUnit 5 in `@Test` methods as parameters. Concretely, *selenium-jupiter* allows to inject subtypes of the **WebDriver** interface (e.g. *ChromeDriver*, *FirefoxDriver*, and so on).

Using *selenium-jupiter* it's easy as pie. First, you need to import the dependency in your project (typically as *test* dependency). In Maven, it is done as follows:

```
<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>selenium-jupiter</artifactId>
    <version>1.1.0</version>
    <scope>test</scope>
</dependency>
```

**NOTE** *selenium-jupiter* 1.1.0 depends on ***selenium-java*** **3.4.0**, ***webdrivermanager*** **1.7.1**, and **appium java-client 5.0.0-BETA9**. Therefore, by using the *selenium-jupiter* dependency, those libraries (*selenium-java*, *webdrivermanager*, and *appium java-client*) will be added as transitive dependencies to your project.

Then, you need to declare *selenium-jupiter* extension in your JUnit 5 test, simply annotating your test with `@ExtendWith(SeleniumExtension.class)`. Finally, you need to include one or more parameters in your `@Test` methods whose types implements the `WebDriver` interface (e.g. `ChromeDriver` to use Chrome, `FirefoxDriver` for Firefox, and so for). That's it. *selenium-jupiter* control the lifecycle of the `WebDriver` object internally, and you just need to use the `WebDriver` object in your test to drive the browser(s) you want. For example:

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeAndFirefoxJupiterTest {

    @Test
    public void testWithOneChrome(ChromeDriver chrome) {
        // using Chrome in this test
    }

    @Test
    public void testWithFirefox(FirefoxDriver firefox) {
        // using Firefox in this test
    }

    @Test
    public void testWithChromeAndFirefox(ChromeDriver chrome,
            FirefoxDriver firefox) {
        // using Chrome and Firefox in this test
    }

}
```

The `WebDriver` subtypes supported by *selenium-jupiter* are the following:

- `ChromeDriver`: Used to control Google Chrome browser.
- `FirefoxDriver`: Used to control Firefox browser.
- `EdgeDriver`: Used to control Microsoft Edge browser.
- `OperaDriver`: Used to control Opera browser.
- `SafariDriver`: Used to control Apple Safari browser (only possible in OSX El Capitan or greater).
- `HtmlUnitDriver`: Used to control HtmlUnit (headless browser).
- `PhantomJSDriver`: Used to control PhantomJS (headless browser).
- `InternetExplorerDriver`: Used to control Microsoft Internet Explorer. Although this browser is supported, Internet Explorer is deprecated (in favor of Edge) and its use is highly discouraged.
- `RemoteWebDriver`: Used to control remote browsers (*Selenium Grid*).
- `AppiumDriver`: Used to control mobile devices (Android, iOS).

**WARNING** The browser to be used must be installed in the machine running the test beforehand (except in the case of `RemoteWebDriver`, in which the requirement is to known a Selenium Server URL). In the case of mobile devices (`AppiumDriver`), the emulator should be up and running in local or available in a Appium Server identified by an URL.

## Motivation

Selenium WebDriver allows to control different types of browsers (such as Chrome, Firefox, Edge, and so on) programmatically using different programming languages. This is very useful to implement automated tests for web applications. Nevertheless, in order to use WebDriver, we need to pay a prize. For security reasons, the automated manipulation of a browser can only be done using native features of the browser. In practical terms, it means that a binary file must be placed in between the test using the WebDriver API and the actual browser. One the one hand, the communication between the WebDriver object and that binary is done using the (W3C WebDriver specification, formerly called *JSON Wire Protocol*. It consists basically on a REST service using JSON for requests and responses. On the other hand, the communication between the binary and the browser is done using native capabilities of the browser. Therefore, the general schema of Selenium WebDriver can be illustrated as follows:

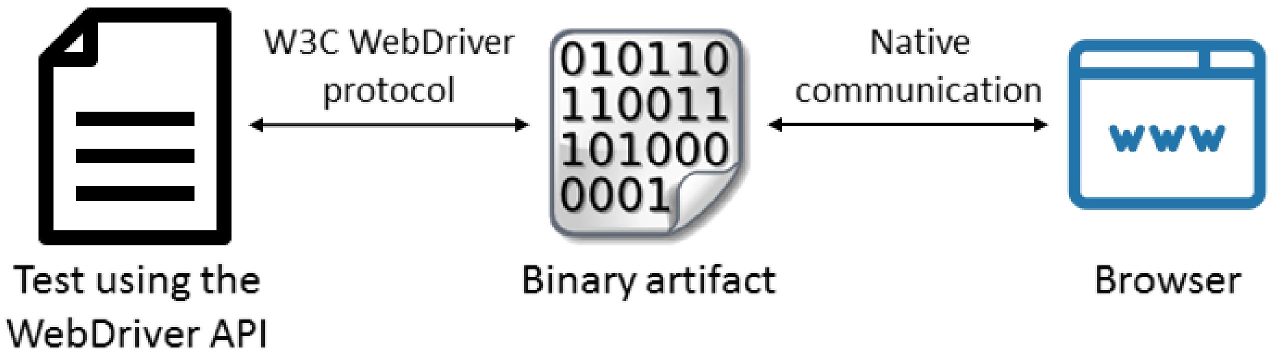

*Figure 1. WebDriver general scenario*

From a tester point of view, the need of this binary component is a pain in the neck, since it should be downloaded manually for the proper platform running the test (i.e. Windows, Linux, Mac). Moreover, the binary version should be constantly updated. The majority of browsers evolve quite fast, and the corresponding binary file required by WebDriver needs to be also updated. The following picture shows a fine-grained diagram of the different flavor of WebDriver binaries and browsers:

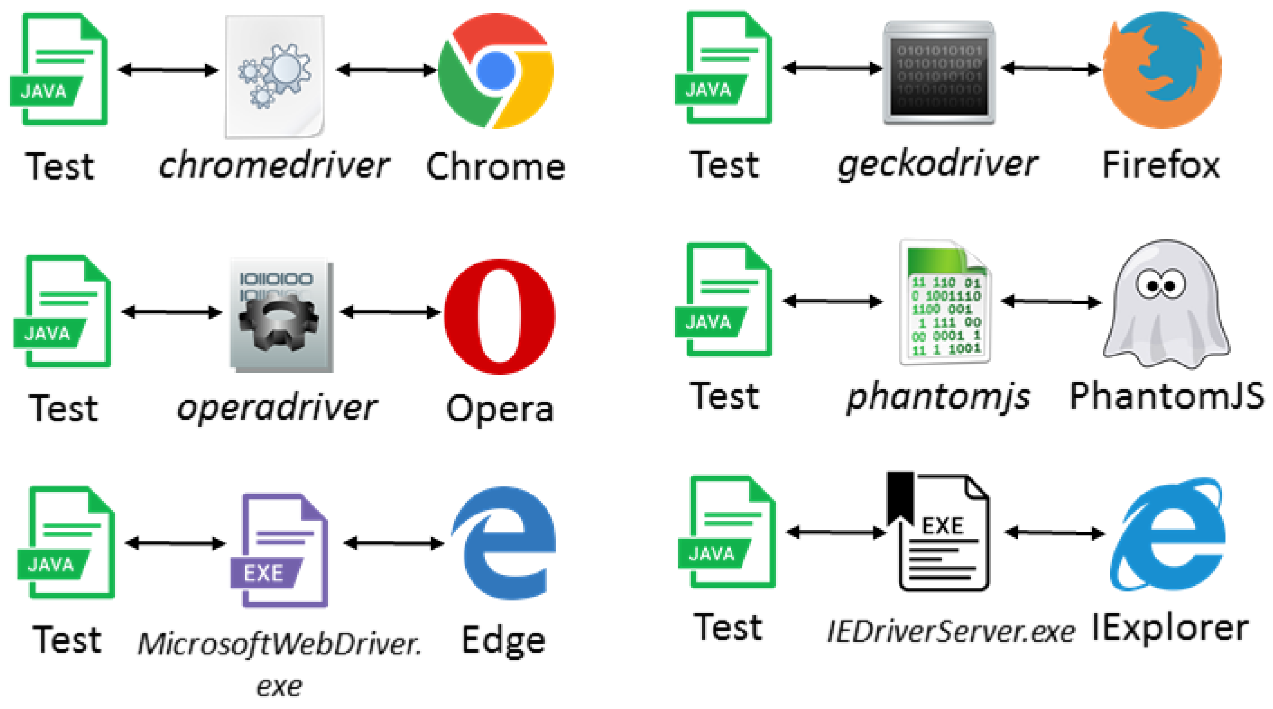

*Figure 2. WebDriver scenario for Chrome, Firefox, Opera, PhantomJS, Edge, and Internet Explorer*

Concerning Java, in order to locate these drivers, the absolute path of the binary controlling the browser should be exported in a given environment variable before creating a WebDriver instance, as follows:

```
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
System.setProperty("webdriver.opera.driver", "/path/to/operadriver");
System.setProperty("webdriver.ie.driver", "C:/path/to/IEDriverServer.exe");
System.setProperty("webdriver.edge.driver", "C:/path/to/MicrosoftWebDriver.exe");
System.setProperty("phantomjs.binary.path", "/path/to/phantomjs");
System.setProperty("webdriver.gecko.driver", "/path/to/geckodriver");
```

In order to simplify the life of Java WebDriver users, in March 2015 the utility WebDriverManager was first released. WebDriverManager is a library which automates all this process (download the proper binary and export the proper variable) for Java in runtime. The WebDriverManager API is quite simple, providing a singleton object for each of the above mentioned browsers:

```
ChromeDriverManager.getInstance().setup();
FirefoxDriverManager.getInstance().setup();
OperaDriverManager.getInstance().setup();
PhantomJsDriverManager.getInstance().setup();
EdgeDriverManager.getInstance().setup();
InternetExplorerDriverManager.getInstance().setup();
```

The solution implemented by WebDriverManager is today supported by similar tools for other languages, such as webdriver-manager for **Node.js** or WebDriverManager.Net for **.NET**.

On August 2017, a new major version of the well-know testing JUnit framework was released. This leads to ***selenium-jupiter***, which can be seen as the natural evolution of *WebDriverManager* for **JUnit 5** tests. Internally, *selenium-jupiter* is built using two foundations:

1. It uses *WebDriverManager* to manage the binaries requires by WebDriver.
2. It uses the *dependency injection* feature of the extension model of JUnit 5 to inject WebDriver objects within `@Test` methods.

All in all, using Selenium WebDriver to control browsers using Java was never that easy. Using JUnit 5 and *selenium-jupiter*, you simply need to declare the flavor of browser you want to use in your test method and use it.

# Examples

This section contains a comprehensive collection of examples demonstrating the basic usage of *seleniun-jupiter* in JUnit 5 tests using different types of browsers. All these examples are part of the test suite of *selenium-jupiter* and are executed on Travis CI.

## Chrome

The following example contains a simple usage of Chrome in JUnit 5. The complete source code of this test is hosted on GitHub. Notice that this class contains two tests (methods annotated with `@Test`). The first one (`testWithOneChrome`) declares just one `ChromeDriver` parameter, and therefore this test controls a single Chrome browser. On the other hand, the second `@Test` ( `testWithTwoChromes`) declares two different `ChromeDriver` parameters, and so, it controls two Chrome browsers.

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

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeJupiterTest {

    @Test
    public void testWithOneChrome(ChromeDriver chrome) {
        chrome.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(chrome.getTitle().startsWith("selenium-jupiter"));
    }

    @Test
    public void testWithTwoChromes(ChromeDriver chrome1, ChromeDriver chrome2) {
        chrome1.get("http://www.seleniumhq.org/");
        chrome2.get("http://junit.org/junit5/");

        assertTrue(chrome1.getTitle().startsWith("Selenium"));
        assertTrue(chrome2.getTitle().equals("JUnit 5"));
    }

}
```

## Firefox

The following test uses Firefox as browser(s). To that aim, `@Test` methods simply need to include `FirefoxDriver` parameters.

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

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class FirefoxJupiterTest {

    @Test
    public void testWithOneFirefox(FirefoxDriver firefox) {
        firefox.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(firefox.getTitle().startsWith("selenium-jupiter"));
    }

    @Test
    public void testWithTwoFirefoxs(FirefoxDriver firefox1,
            FirefoxDriver firefox2) {
        firefox1.get("http://www.seleniumhq.org/");
        firefox2.get("http://junit.org/junit5/");

        assertTrue(firefox1.getTitle().startsWith("Selenium"));
        assertTrue(firefox2.getTitle().equals("JUnit 5"));
    }

}
```

## Edge

The following example uses one Edge browser. Notice that this test is disabled (with JUnit 5's `@Disabled` annotation), due to the fact that Edge is not available on Travis CI. However, this test could be executed on a Windows machine with Edge without problems.

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

import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.edge.EdgeDriver;

import io.github.bonigarcia.SeleniumExtension;

@Disabled("Edge is not available on Travis CI")
@ExtendWith(SeleniumExtension.class)
public class EdgeJupiterTest {

    @Test
    void webrtcTest(EdgeDriver edge) {
        edge.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(edge.getCurrentUrl().contains("selenium-jupiter"));
    }

}
```

**TIP** The required version of *MicrosoftWebDriver.exe* depends on the version on Edge to be used (more info here). By default, WebDriverManager downloads and uses the latest version of the binaries. Nevertheless, a concrete version can be fixed. Take a look to the advance examples section to find out how to setup the different options of WebDriverManager.

## Opera

Are you one of the few using Opera? No problem, you can still make automated tests with JUnit 5, WebDriver, and *selenium-jupiter*, as follows:

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

import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.opera.OperaDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class OperaJupiterTest {

    @Disabled("Opera not available on Travis CI")
    @Test
    public void test(OperaDriver opera) {
        opera.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(opera.getCurrentUrl().contains("github"));
    }

}
```

## Safari

You can also use Safari in conjunction with *selenium-jupiter*. Take into account that `SafariDriver` requires Safari 10 running on OSX El Capitan or greater.

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

import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.safari.SafariDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class SafariJupiterTest {

    @Disabled("SafariDriver requires Safari 10 running on OSX El Capitan or greater.")
    @Test
    public void test(SafariDriver safari) {
        safari.get("http://www.seleniumhq.org/");

        assertTrue(
                safari.getTitle().equals("Selenium - Web Browser Automation"));
    }

}
```

## PhamtomJS

PhamtomJS is a headless browser (i.e. a browser without GUI), and it can be convenient for different types of tests. The following example demonstrates how to use PhamtomJS with *selenium-jupiter*.

```java
import static org.junit.jupiter.api.Assertions.assertNotNull;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.phantomjs.PhantomJSDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class PhantomjsJupiterTest {

    @Test
    public void test(PhantomJSDriver phantomjs) {
        phantomjs.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertNotNull(phantomjs.getPageSource());
    }

}
```

## HtmlUnit

HtmlUnit is another headless browser that can be used easily in a Jupiter test, for example like this:

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

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class HtmlUnitJupiterTest {

    @Test
    public void test(HtmlUnitDriver htmlUnit) {
        htmlUnit.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(htmlUnit.getTitle().contains("JUnit 5 extension"));
    }

}
```

# Advanced

So far, we have discovered how to use different browsers (Chrome, Firefox, Edge, Opera, Safari, PhamtomJS, HtmlUnit) with its default options and supposing that the browser to be used is installed on the machine running the test. Nevertheless, if you have used intensively Selenium WebDriver, different questions might come to your mind:

- What if I need to specify options (e.g. `ChromeOptions`, `FirefoxOptions`, etc) to my WebDriver object?
- What if need to specify desired capabilities (e.g. browser type, version, platform)?
- And what about remote browsers (Selenium Grid)? How is `RemoteWebDriver` supported by *selenium-jupiter*?

In order to support the advance features of Selenium WebDriver, *selenium-jupiter* provides several annotations aimed to allow a fine-grained control of the WebDriver object instantiation. These annotations are:

- `DriverOptions`: Annotation to configure *options* (e.g. `ChromeOptions` for Chrome, `FirefoOptions` for Firefox, `EdgeOptions` for Edge, `OperaOptions` for Opera, and `SafariOptions` for Safari).
- `DriverCapabilities`: Annotation to configure the desired *capabilities* (WebDriver's object `DesiredCapabilities`).
- `DriverUrl`: Annotation used to identify the URL value needed to instantiate a `RemoteWebDriver` object.

All these 3 annotation can be used both at *parameter-level* (applied to a single WebDriver parameter), and also at *field-level* (applied globally in a test class). Keep reading to find out several examples about that.

## Using options

The following example shows how to specify options for Chrome. To that aim, the annotation `@DriverOptions` is used to set a collection of options. This array of options is configured using the annotation `@Option` declaring the parameters `name` and `value`). In the example, the annotation `@DriverOptions` is used at *parameter-level*. In the first test (called `headlessTest`), we are setting the argument `--headless`, used in Chrome to work as a headless browser. In the second test ( `webrtcTest`), we are using two different arguments: `--use-fake-device-for-media-stream` and `--use-fake-ui-for-media-stream`, used to fake user media (i.e. camera and microphone) in WebRTC applications.

```java
import static io.github.bonigarcia.SeleniumJupiter.ARGS;
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.DriverOptions;
import io.github.bonigarcia.Option;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeWithOptionsJupiterTest {

    @Test
    void headlessTest(@DriverOptions(options = {
            @Option(name = ARGS, value = "--headless") }) ChromeDriver chrome) {
        chrome.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(chrome.getTitle().startsWith("selenium-jupiter"));
    }

    @Test
    void webrtcTest(
            @DriverOptions(options = {
                    @Option(name = ARGS, value = "--use-fake-device-for-media-stream"
),
                    @Option(name = ARGS, value = "--use-fake-ui-for-media-stream") })
ChromeDriver chrome)
            throws InterruptedException {

        chrome.get(
                "https://webrtc.github.io/samples/src/content/devices/input-output/");

        Thread.sleep(3000); // Wait 3 seconds to see the video
    }

}
```

As introduced before, this annotation `@DriverOptions` can be used also at *field-level*, as shown in this other example. This test is setting to `true` the Firefox preferences `media.navigator.streams.fake` and `media.navigator.permission.disabled`, used also for WebRTC.

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;

import io.github.bonigarcia.DriverOptions;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class FirefoxWithGlobalOptionsJupiterTest {

    @DriverOptions
    FirefoxOptions firefoxOptions = new FirefoxOptions();
    {
        // Flag to use fake media for WebRTC user media
        firefoxOptions.addPreference("media.navigator.streams.fake", true);

        // Flag to avoid granting access to user media
        firefoxOptions.addPreference("media.navigator.permission.disabled",
                true);
    }

    @Test
    void webrtcTest(FirefoxDriver firefox) throws InterruptedException {
        firefox.get(
                "https://webrtc.github.io/samples/src/content/devices/input-output/");

        Thread.sleep(3000); // Wait 3 seconds to see the video
    }

}
```

## Using capabilities

The annotation `@DriverCapabilities` is used to specify WebDriver capabilities (i.e. type browser, version, platform, etc.). These capabilities are typically used for Selenium Grid tets (i.e. tests using remote browsers). To that aim, an Selenium Hub (also known as *Selenium Server*) should be up an running, and its URL should known. This URL will be specified using the *selenium-jupiter* annotation `@DriverUrl`.

The following example provides a complete example about this. As you can see, in the test setup (`@BeforeAll`) a Selenium Grid is implemented, first starting a Hub (a.k.a. *Selenium Server*), and then a couple of nodes (Chrome a Firefox) are registered in the Hub. Therefore, remote test using `RemoteWebDriver` can be executed, simply pointing to the Hub (whose URL in this case is `http://localhost:4444/wd/hub` in this example) and selecting the browser to be used using the `Capabilities`.

```java
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.grid.selenium.GridLauncherV3;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.RemoteWebDriver;

import io.github.bonigarcia.Capability;
import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.DriverUrl;
import io.github.bonigarcia.SeleniumExtension;
import io.github.bonigarcia.wdm.ChromeDriverManager;
import io.github.bonigarcia.wdm.FirefoxDriverManager;

@ExtendWith(SeleniumExtension.class)
public class RemoteWebDriverWithCapabilitiesJupiterTest {

    @BeforeAll
    static void setup() throws Exception {
        // Start hub
        GridLauncherV3.main(new String[] { "-role", "hub", "-port", "4444" });

        // Register Chrome in hub
        ChromeDriverManager.getInstance().setup();
        GridLauncherV3.main(new String[] { "-role", "node", "-hub",
                "http://localhost:4444/grid/register", "-browser",
                "browserName=chrome,version=59", "-port", "5555" });

        // Register Firefox in hub
        FirefoxDriverManager.getInstance().setup();
        GridLauncherV3.main(new String[] { "-role", "node", "-hub",
                "http://localhost:4444/grid/register", "-browser",
                "browserName=firefox", "-port", "5556" });

        // Wait to finish browser registration
        Thread.sleep(2000);
    }

    @Test
    void testWithoutUrl(RemoteWebDriver remoteWebDriver) {
        assertThrows(NullPointerException.class, () -> {
            exercise(remoteWebDriver);
        });
    }

    @Test
    void testWithRemoteChrome(
            @DriverUrl("http://localhost:4444/wd/hub") @DriverCapabilities(capability
= {
```

```
                      @Capability(name = "browserName", value = "chrome"),
                      @Capability(name = "version", value = "59") }) RemoteWebDriver
remoteChrome)
            throws InterruptedException {
        exercise(remoteChrome);
    }

    @Test
    void testWithRemoteFirefox(
            @DriverUrl("http://localhost:4444/wd/hub") @DriverCapabilities(capability
= {
                      @Capability(name = "browserName", value = "firefox") })
RemoteWebDriver remoteChrome)
            throws InterruptedException {
        exercise(remoteChrome);
    }

    void exercise(WebDriver webdriver) {
        webdriver.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(webdriver.getTitle().contains("JUnit 5 extension"));
    }

}
```

The equivalent example, this time using global capabilities and URL (i.e. using `@DriverCapabilities` and `@DriverUrl` at *field-level* is also available on GitHub.

## AppiumDriver

The annotation `@DriverCapabilities` can be also used to specify the desired capabilities to create an instances of AppiumDriver to drive mobile devices (Android or iOS). If not `@DriverUrl` is specified, *selenium-jupiter* will start automatically an instance of Appium Server (by default in port 4723) in the localhost after each test execution (this server is shutdown before each test). For example:

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

import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;

import io.appium.java_client.AppiumDriver;
import io.github.bonigarcia.Capability;
import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class AppiumChromeJupiterTest {

    @Disabled("Android emulator not available on Travis CI")
    @Test
    void testWithAndroid(
            @DriverCapabilities(capability = {
                    @Capability(name = "browserName", value = "chrome"),
                    @Capability(name = "deviceName", value = "Android") })
AppiumDriver<WebElement> android)
            throws InterruptedException {

        String context = android.getContext();
        android.context("NATIVE_APP");
        android.findElement(By.id("com.android.chrome:id/terms_accept"))
                .click();
        android.findElement(By.id("com.android.chrome:id/negative_button"))
                .click();
        android.context(context);

        android.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertTrue(android.getTitle().contains("JUnit 5 extension"));
    }

}
```

We can also specify a custom Appium Server URL changing the value of `@DriverUrl`, at field-level or parameter-level:

```java
import java.io.File;
import java.net.URISyntaxException;

import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;

import io.appium.java_client.AppiumDriver;
import io.appium.java_client.MobileElement;
import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class AppiumApkJupiterTest {

    @DriverCapabilities
    DesiredCapabilities capabilities = new DesiredCapabilities();
    {
        try {
            File apk = new File(this.getClass()
                    .getResource("/selendroid-test-app.apk").toURI());
            capabilities.setCapability("app", apk.getAbsolutePath());
            capabilities.setCapability("deviceName", "Android");

        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    @Disabled("Android emulator not available on Travis CI")
    @Test
    void testWithAndroid(AppiumDriver<MobileElement> android)
            throws InterruptedException {
        WebElement button = android.findElement(By.id("buttonStartWebview"));
        button.click();

        WebElement inputField = android.findElement(By.id("name_input"));
        inputField.clear();
        inputField.sendKeys("Custom name");
    }

}
```

## Tuning WebDriverManager

As introduced before, *selenium-jupiter* internally uses WebDriverManager to manage the required binary to control browsers. This tool can be configured in several ways, for example to force using a given version of the binary (by default it tries to use the latest version), or force to use the cache (instead of connecting to the online repository to download the binary artifact). For further information about this configuration capabilities, please take a look to the WebDriverManager documentation.

In this section we are going to present a couple of simple examples tuning somehow WebDriverManger. The following example shows how to force a version number for a binary, concretely for Edge:

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

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.edge.EdgeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class EdgeSettingVersionJupiterTest {

    @BeforeAll
    static void setup() {
        System.setProperty("wdm.edgeVersion", "3.14393");
    }

    @Disabled("Edge not available on Travis CI")
    @Test
    void webrtcTest(EdgeDriver edge) {
        edge.get("http://www.seleniumhq.org/");

        assertTrue(edge.getTitle().equals("Selenium - Web Browser Automation"));
    }

}
```

This other example shows how to force cache (i.e. binaries previously downloaded by WebDriverManager) to avoid the connection with online repository to check the latest version:

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

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ForceCacheJupiterTest {

    @BeforeAll
    static void setup() {
        System.setProperty("wdm.forceCache", "true");
    }

    @Test
    public void test(ChromeDriver chrome) {
        chrome.get("https://bonigarcia.github.io/selenium-jupiter/");

        assertTrue(chrome.getTitle().contains("JUnit 5 extension"));
    }

}
```

## Support

There are several ways to get in touch with *selenium-jupiter*:

- Questions about *selenium-jupiter* are supposed to be discussed in StackOverflow, using the tag *selenium-jupiter*.
- Comments, suggestions and bug-reporting should be done using the GitHub issues.
- If you think *selenium-jupiter* can be enhanced, consider contribute to the project by means of a pull request.

## About