Injecting DesiredCapabilities object using Spring dependency injection mechanism

On this page, we will describe how to inject desired capabilities into our test class. We assume you are familiar with the page object concept, and read at least one of the basic UI testing pages - either the web UI basic test or the mobile UI basic test.

Selenium was designed in a way that tests should be agnostic to the type of browser or mobile device they are controlling. WebDriver and WebElement were built as interfaces for this specific reason.

The Verisoft framework supports this concept and the VerisoftDriver / VerisoftMobileDriver object is

a. Injected

b. Agnostic to types of browser / mobile device

However, as demonstrated in the web UI or mobile UI examples, the DesiredCapabilities object is located at the top of the class and annotated with the @DriverCapabilities annotation. For example:

```
@DriverCapabilities
private DesiredCapabilities capabilities = new DesiredCapabilities();
{
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--no-sandbox");
    options.addArguments("--headless");

    capabilities.setBrowserName("chrome");
    capabilities.setCapability("browserVersion", "113");
    capabilities.setCapability("driverVersion", "113");
    options.merge(capabilities);
}
```

And here lies the issue - we are specifying the use of Chrome in an agnostic class.

In order to solve this issue, we will use the Spring framework (more accurately, use the IoC feature of the Spring framework). To sum up our solution we will:

1. Add Spring dependencies to the pom.xml file
2. Write a Bean that will inject the capabilities
3. Annotate our test class with SpringExtension.class and add @ContextConfiguration
4. Inject the capabilities and populate the DesiredCapabilities code

Add Spring dependencies to the pom.xml file

We will need to add the following dependencies:

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.27</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-test -->
<dependency>
   <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-test</artifactId>
    <version>5.3.27</version>
 </dependency>

 <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
 <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.27</version>
 </dependency>
```

Note that versions may change. 5.3.27 is currently the latest stable version of Spring.

Write a Bean that will inject the capabilities

A Bean is a Spring concept. A bean is an object that is injected into another object. We will create a new class:

```
@Configuration
@ComponentScan("co.verisoft")
public class CapabiitiesInjection {


    @Bean
    public DesiredCapabilities getCapabilities() {
        DesiredCapabilities capabilities = new DesiredCapabilities();

        ChromeOptions options = new ChromeOptions();
        options.addArguments("--no-sandbox");
        options.addArguments("--headless");

        capabilities.setBrowserName("chrome");
        capabilities.setCapability("browserVersion", "113");
        capabilities.setCapability("driverVersion", "113");
        options.merge(capabilities);

        return capabilities;
    }
}
```

A few interesting things to take into account:

1. The `@Configuration` annotation indicates that this class is the “gateway” towards all the other beans. There is one class in each project which is marked as @Configuration. All the other Bean classes will be marked as @Component.
2. The `@ComponentScan("co.verisoft")` tells Spring to look for all the classes in the co.verisoft packages which are marked with @Component, and save them as possible injectors.
3. The `@Bean` annotation tells Spring that if someone is looking for an object of the type `DesiredCapabilities`, this method should be executed.
4. The rest of the code is a simple creation of DesiredCapabilities object.

There is a lot more to say about this process with Spring, and there are many options you might consider adopting during development. Spring is not in the scope of this document.

Annotate our test class with SpringExtension.class

Spring ships with an extension to be able to handle IoC in JUnit 5. Our test class should look like this:

```
@ExtendWith(SpringExtension.class)
@ContextConfiguration(classes = {CapabiitiesInjection.class})
public class DependencyInjectionExampleTest extends BaseTest{

   ...

}
```

The `@ContextConfiguration` annotation tells Spring where to find the beans. Specifically, if you have on the class you marked as @Configuration a @ComponentScan annotation, you may only mark the @Configuration class, it will search for the rest of the @Component classes.

Inject the capabilities and populate the DesiredCapabilities code

Finally, we can inject the value:

```
@DriverCapabilities
@Autowired
private DesiredCapabilities capabilities;
```

Specifically, `@Autowired` is the one we need. The `@DriverCapabilities` is not part of the IoC injection and serves a different purpose (see basic web UI tests / basic mobile UI tests).

That's it. the capabilities object searched, with the Spring mechanism for a Bean which returns a DesiredCapabilities object. This Bean is found at the method `getCapabilities` in the class `CapabiitiesInjection` and populates the object capabilities with the result of the method