Implementing Digital Zoom API using Cucumber JVM hooks

by Prasant Sutaria

Pre-requisite: Before we get into details of implementing the Smart reporting API in your framework. We will need to add the Perfecto Smart reporting dependency to Project's **pom.xml** file (If maven is used as the build tool). Please follow the instruction provided in - Download the Reporting SDK#Java

Cucumber JVM has supported hooks since earlier version. Hooks in Cucumber JVM are similar to TestNG's Listeners, which provide interface to implement code that will be executed at certain events in test execution life cycle. For more details on what is hooks - https://docs.cucumber.io/cucumber/api/#hooks

When we implement hooks to report scenario results and take screenshots, we may need to share state (web driver, reportiumClient and other related objects) between glue code (step definition and hooks classes). There are several ways to share state between glue code:

Implementation Design options	
1. Simple solution would be to create static objects and access between glue code.	**Pros:** Simple to implement. **Cons:** It will not be suitable for parallel execution of Scenarios.
2. Use Dependency Injection (DI) to inject state to glue code. We can implement the concept of World object used by other frameworks like Protractor, ruby-cucumber etc. by implementing dependency injection frameworks like pico-container, guice etc. For more details on different DI frameworks supported by cucumber - https://docs.cucumber.io/cucumber/state/	**Pros:** a. Clean implementation of code b. It creates local global object with scope of test scenario execution, which can be shared between classes without giving away the power of parallel execution. **Cons:** Complex implementation while setting up framework

In this article, we will look into implementation of **pico-container DI framework** to share state between glue code.

Steps

1. We will start with creating a Java Maven project and add following dependencies to pom.xml file:

pom.xml

```
        <!-- cucumber-picocontainer dependency -->
        <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-picocontainer</artifactId>
        <version>${cucumber.version}</version>
        <scope>test</scope>
    </dependency>


        <!-- Junit dependency -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>


        <!-- Cucumber Junit dependency -->
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>${cucumber.version}</version>
        <scope>test</scope>
    </dependency>
```

Latest Cucumber version at the time of writing this article was **4.2.2**. For latest version of cucumber jvm visit - https://mvnrepository.com/artifact/io.cucumber/cucumber-jvm

2. Create a class called World, it will represent the object to be injected into glue code.

World class must extend PicoFactory class. World class can have any number of member variables and the variable be accessed by getter and setter methods. For our purpose, World class has 2-member variables: web driver and reportium instance.

World.java

```
package io.perfecto.stepdefs;

import org.openqa.selenium.remote.RemoteWebDriver;

import cucumber.runtime.java.picocontainer.PicoFactory;
import io.perfecto.reporting.Reports;

// Class to hold webdriver and reportium client
public class World extends PicoFactory{

        private RemoteWebDriver webDriver;
        private Reports report;

        // Get Webdriver
        public RemoteWebDriver getWebDriver() {
                return webDriver;
        }

        // Set Webdriver
        public void setWebDriver(RemoteWebDriver webDriver) {
                this.webDriver = webDriver;
        }

        // Get Reportium client
        public Reports getReport() {
                return report;
        }

        // Set Reportium client
        public void setReport(Reports report) {
                this.report = report;
        }
}
```

3. Implement Constructor in Step definition classes and hooks class. The constructor will accept a World type parameter.

StepDefinition.java

```
package io.perfecto.stepdefs;
import org.junit.Assert;
import org.openqa.selenium.remote.RemoteWebDriver;
import cucumber.api.java.en.Given;

public class StepDef{
        // World Object instance to inject driver and reportium objects
        private World world;

        public StepDef(World world) {
                this.world = world;
        }

        @Given("^I have (\\d+) cukes in my belly$")
        public void i_have_cukes_in_my_belly(int arg1) throws Throwable {
                // Getting web driver instance from world object
                RemoteWebDriver driver = this.world.getWebDriver();
                driver.get("http://perfecto.io");
                Assert.assertEquals("Wrong title", driver.getTitle());
        }
}
```

4. Implement Cli runner for cuke runner instead of Junit runner. For more details about cli runner - https://docs.cucumber.io/cucumber/api/#from-the-command-line

CukeRunner.java

```
package io.perfeco.cucumber;
import cucumber.api.cli.Main;
public class CukeRunner {

        public static void main(String[] args) throws Throwable {
        final String[] cucumberArgs = {
                "-g",
                "io.perfecto.stepdefs",
                "classpath:features"
        };
        Main.main(cucumberArgs);
    }
}
```

Known Limitation of the Approach	Work Around

<table>
<tr>
<td>1. Before and After Step hooks are not available in Cucumber JVM version 1.X.X and 2.X.X.
Cucumber official change logs - https://github.com/cucumber/cucumber-jvm/blob/master/CHANGELOG.md</td>
<td>The reporting steps can be started within Step Definition. It should be the very first sentence in the step definition.

Work Around

<pre>@Given("^I have (\\d+) cukes in my belly$")

        public void i_have_cukes_in_my_belly(int arg1)
throws Throwable {
                // Start reporting step
                world.getReport().startStep("I have a
lot of cukes in my belly");

                // Getting web driver instance from
world object
                RemoteWebDriver driver = this.world.
getWebDriver();
                driver.get("http://perfecto.io");
                Assert.assertEquals("Wrong title",
driver.getTitle());

        }</pre></td>
</tr>
<tr>
<td>2. For Cucumber 3.X.X and 4.X.X, no API is exposed to get Step text in @beforeStep and @afterStep hooks.</td>
<td>1. We can implement above work around but some effort will be required to add start step call to each step definition.
2. We can start Steps by number and map them with steps in scenario logically. We can use the world object and steps hooks.This approach will help reducing the effort to add start step calls.
Example Step - 1 will represent first step in scenario, Step - 2 to second step etc.
Attach project implements the above idea.</td>
</tr>
<tr>
<td>3. The hooks are not capable to determining the reason of failure. They have api exposed to determine whether the scenario is pass or fail but no reason is attach to failure.</td>
<td>We can use World object to provide failure reason. But this will be a big effort in existing project. New project can design custom assertion method to update world object with failure reason.</td>
</tr>
</table>

Download the example project from here - https://github.com/PerfectoCode/Reporting-Sample-Cucumber_JVM/tree/master/cucumber_sample/src/test/java/io