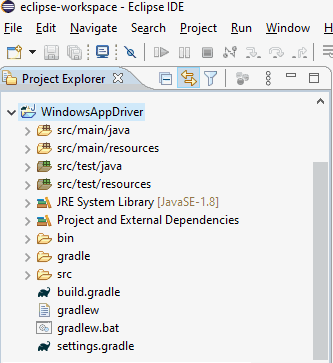
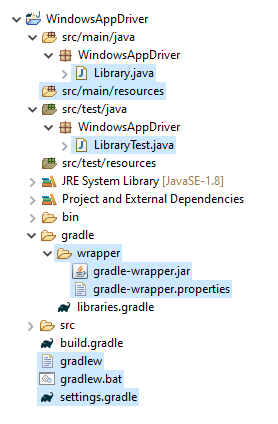
Para el siguiente ejemplo se realizó la automatización de la calculadora de Windows y para esto se utilizó el patrón de diseño POM, y también se implementó serenity para la generación de reportes.

Lo primero que debemos hacer es crear un proyecto **Gradle**. Para ello seguiremos los pasos que se muestran en el gif de abajo (gradlepProject.gif)

Creado el proyecto, debería quedar con la siguiente estructura.

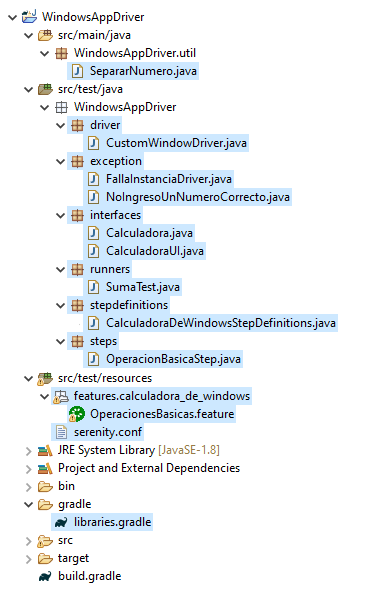


De esta estructura borraremos la carpeta **src**/**main**/**resources**, la carpeta **wrapper** que **está dentro** de la carpeta **gradle** con todo su contenido, y los archivos **Library.java**, **LibraryTest.java**, **gradlew**, **gradlew.bat**, **settings.gradle**, pues estos no serán útiles para el proceso de automatización.



Hecho esto, procedemos a renombrar el paquete que está en **src**/**main**/**java** de llamarse **WindowsAppDriver** en este caso, le cambiaremos el nombre a **WindowsAppDriver.util**, ya que más adelante implementaremos ahí una función (renameUtils.gif).

Hecho lo anterior, se deben crear los directorios y documentos que se pueden apreciar en la siguiente imagen.



**Al terminar** de crear los **paquetes** y **archivos**, lo siguiente que se hará es importar las librerías que se utilizarán para realizar la prueba de automatización, lo siguiente es **entrar** a la carpeta **gradle** y al documento llamado **libraries**.**gradle** donde se escribirán los nombres de las dependencias y sus versiones a importar:

ext{

slf4jVersion = '1.7.7'

serenityCoreVersion = '2.0.91'

serenityCucumberVersion = '1.0.30'

junitVersion='4.12'

assertJVersion='3.8.0'

logbackVersion='1.2.3'

cucumberVersion = '4.8.0'

appiumWebDriverVersion= '7.3.0'

seleniumWebDriverVersion='3.141.59'

libs = [

slf4jApi: "org.slf4j:slf4j-api:$slf4jVersion",

logback: "ch.qos.logback:logback-classic:${logbackVersion}",

appiumWebDriver: "io.appium:java-client:${appiumWebDriverVersion}",

seleniumWebDriver: "org.seleniumhq.selenium:selenium-java:${seleniumWebDriverVersion}",

test: [

serenity: [

core: "net.serenity-bdd:serenity-core:${serenityCoreVersion}",

junit: "net.serenity-bdd:serenity-junit:${serenityCoreVersion}",

screenplay: "net.serenity-bdd:serenity-screenplay:${serenityCoreVersion}",

screenplayWebdriver: "net.serenity-bdd:serenity-screenplay-webdriver:${serenityCoreVersion}",

cucumber: "net.serenity-bdd:serenity-cucumber4:${serenityCucumberVersion}"

],

cucumber: [

java: "io.cucumber:cucumber-core:${cucumberVersion}",

java: "io.cucumber:cucumber-java:${cucumberVersion}",

junit: "io.cucumber:cucumber-junit:${cucumberVersion}"

],

junit: "junit:junit:${junitVersion}",

assertj: "org.assertj:assertj-core:${assertJVersion}"

]

]

}

Lo siguiente a este paso es abrir el archivo llamado build.gradle donde se invocarán las dependencias escritas anteriormente y se configurarán algunas características para el proyecto.

defaultTasks 'clean','test','aggregate'

repositories {

mavenLocal()

jcenter()

}

buildscript {

repositories {

mavenLocal()

jcenter()

maven {

url = 'http://repo.maven.apache.org/maven2'

}

}

dependencies {

classpath("net.serenity-bdd:serenity-gradle-plugin:2.1.6")

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

apply plugin: 'net.serenity-bdd.aggregator'

apply from: "$rootDir/gradle/libraries.gradle"

sourceCompatibility = 1.8

targetCompatibility = 1.8

/\*\*

\* This is needed to make sure there are no Cucumber 2 dependencies in the classpath.

\*/

Configurations.all {

resolutionStrategy {

force "io.cucumber:cucumber-core:${cucumberVersion}"

}

}

dependencies {

compile libs.logback

compile libs.appiumWebDriver

compile libs.seleniumWebDriver

testCompile libs.test.cucumber.java,

libs.test.cucumber.junit,

libs.test.serenity.core,

libs.test.serenity.screenplay,

libs.test.serenity.junit,

libs.test.serenity.screenplayWebdriver,

libs.test.serenity.cucumber,

libs.test.junit,

libs.test.assertj

}

test {

testLogging.showStandardStreams = true

systemProperties System.getProperties()

}

tasks.withType(JavaCompile) {

options.encoding = 'UTF-8'

}

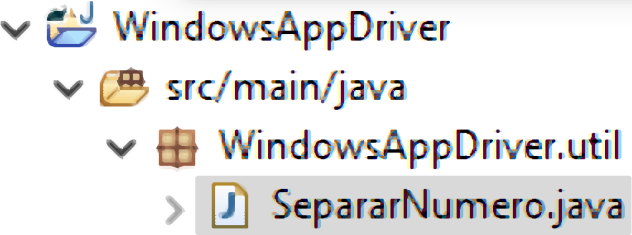
test.finalizedBy(aggregate)

gradle.startParameter.continueOnFailure = true

test.finalizedBy(aggregate)

Ahora el siguiente paso es actualizar las librerías que acabamos de configurar como se muestra en el siguiente gif (gradleRefresh.gif)

A continuación, ingresamos a la clase SepararNumero.java, donde escribiremos dos funciones que nos ayudaran a separar los números enteros que serán ingresados a la calculadora, estos métodos serán utilizados más adelante:



package WindowsAppDriver.util;

import java.util.ArrayList;

import java.util.List;

public class SepararNumero {

private SepararNumero() {}

public static List<Integer> entero(int entero) {

int punto = cantidadDeDigitosEnteros(entero);

List<Integer> listaDeNumeros = new ArrayList<>();

listaDeNumeros.add(entero % 10);

do {

entero = entero / 10;

if((entero % 10) != 0 || punto != 0) {

listaDeNumeros.add(entero % 10);

}

punto--;

} while (entero != 0);

return listaDeNumeros;

}

public static int cantidadDeDigitosEnteros(int numero) {

int contador = 0;

do {

contador++;

numero = numero / 10;

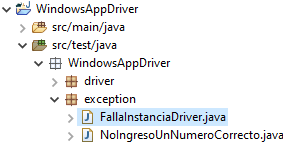
} while (numero != 0);

return contador;

}

}

Lo que sigue ahora, es crear excepciones personalizadas para los drivers de la calculadora, para empezar escribiremos la excepción en caso de que el driver no se pueda instanciar correctamente y la otra excepción, se lanzará en caso de que el número que sea ingresado a la calculadora no sea válido (que no sea válido significa que no esté especificado entre los números disponibles para hacer click dentro de la calculadora).



package WindowsAppDriver.exception;

public class FallaInstanciaDriver extends Error{

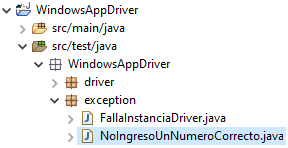
private static final long serialVersionUID = 1L;

public FallaInstanciaDriver(String mensaje, Exception e) {

super(mensaje, e);

}

}



package WindowsAppDriver.exception;

public class NoIngresoUnNumeroCorrecto extends Error {

private static final long serialVersionUID = 2L;

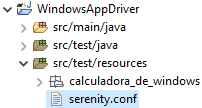
public NoIngresoUnNumeroCorrecto(String mensaje) {

super(mensaje);

}

}

Lo que sigue, es configurar las variables de entorno para serenity, dónde se indicará en **provided.mydriver** el nombre del proyecto junto al paquete y el nombre de la clase del driver donde se va a utilizar esta variable, para **windows.program.path** se pondrá el nombre de la aplicación.



webdriver {

driver = provided

provided.type = mydriver

provided.mydriver = " WindowsAppDriver.driver.CustomWindowDriver"

environments {

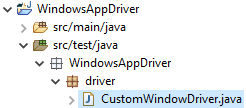
default {

windows.program.path = "Microsoft.WindowsCalculator\_8wekyb3d8bbwe!App"

}

Con las variables de entorno configuradas, lo que sigue es es configurar el driver para la calculadora de Windows, aquí obtendremos la variable de entorno que escribimos en el archivo anterior , en este caso, será la variable **windows.program.path**, y quedará guardada en el String **app**, luego instanciaremos un objeto del tipo **DesiredCapabilities** para configurar algunas características del driver.

Ahora en capabilitie introducimos el string “**app**” en el primer argumento del método **setCapabilitie** para indicar que en el siguiente segundo argumento se va a introducir el nombre o directorio de la aplicación, donde seguidamente pondremos la variable de tipo String **app** hecho esto, lo que queda es inicializar el driver. El driver tiene que ser de tipo **WindowsDriver**, además tendremos que introducir la URL junto con el puerto donde va a ser ejecutado WinAppDriver (el puerto predeterminado de WinAppDriver es el 4723), finalmente le indicamos al driver que espere por lo menos 2 segundos para realizar el mapeo de la aplicación y que luego retorne el driver.



package serenity.cucumber.winappdriver.driver;

import java.io.IOException;

import java.net.URL;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import serenity.cucumber.winappdriver.exception.FallaInstanciaDriver;

import net.serenitybdd.core.environment.EnvironmentSpecificConfiguration;

import net.thucydides.core.guice.Injectors;

import net.thucydides.core.util.EnvironmentVariables;

import net.thucydides.core.webdriver.DriverSource;

import org.openqa.selenium.remote.DesiredCapabilities;

import io.appium.java\_client.windows.WindowsDriver;

public class CustomWADDriver implements DriverSource {

public CustomWADDriver() {

super();

}

@Override

public WebDriver newDriver() {

try {

EnvironmentVariables vA = Injectors.getInjector().getInstance(EnvironmentVariables.class);

String app = EnvironmentSpecificConfiguration.from(vA).getProperty("windows.program.path");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("app", app);

WindowsDriver driver = new WindowsDriver(new URL("http://127.0.0.1:4723"), capabilities);

driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

return driver;

} catch (IOException e) {

throw new FallaInstanciaDriver("No se pudo instanciar el driver $", e);

}

}

@Override

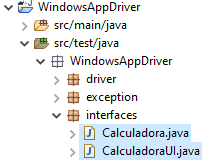
public boolean takesScreenshots() {

return true;

}

}

Después de configurar el driver, lo que sigue es configurar las clases que van a obtener los elementos u objetos de la aplicación, para este paso, lo que se hace es obtener todos los objetos como tipo **WebDriver**, acá no utilizaremos los métodos propios de mapeo de objetos de WinAppDriver pues se necesitaría de inicializar una clase que retorne un driver, para evitar esto, se usará en su lugar la anotación @FindBy y se mapearán los objetos por medio del **xpath** (para el caso de todos los elementos menos el frame principal que se mapea con el nombre).



Código de CalculadoraUI.java

package WindowsAppDriver.interfaces;

import org.openqa.selenium.WebElement;

import net.serenitybdd.core.annotations.findby.FindBy;

import net.serenitybdd.core.pages.PageObject;

public class CalculadoraUI extends PageObject {

@FindBy(xpath = "//Button[@AutomationId='num1Button']") public WebElement botonNumeroUno;

@FindBy(xpath = "//Button[@AutomationId='num2Button']") public WebElement botonNumeroDos;

@FindBy(xpath = "//Button[@AutomationId='num3Button']") public WebElement botonNumeroTres;

@FindBy(xpath = "//Button[@AutomationId='num4Button']") public WebElement botonNumeroCuatro;

@FindBy(xpath = "//Button[@AutomationId='num5Button']") public WebElement botonNumeroCinco;

@FindBy(xpath = "//Button[@AutomationId='num6Button']") public WebElement botonNumeroSeis;

@FindBy(xpath = "//Button[@AutomationId='num7Button']") public WebElement botonNumeroSiete;

@FindBy(xpath = "//Button[@AutomationId='num8Button']") public WebElement botonNumeroOcho;

@FindBy(xpath = "//Button[@AutomationId='num9Button']") public WebElement botonNumeroNueve;

@FindBy(xpath = "//Button[@AutomationId='num0Button']") public WebElement botonNumeroCero;

@FindBy(name = "Calculadora") public WebElement framePrincipal;

@FindBy(xpath = "//Button[@AutomationId='TogglePaneButton']") public WebElement menuPrincipal;

@FindBy(xpath = "//ListItem[@AutomationId='Scientific']") public WebElement menuCalculadoraCientifica;

@FindBy(xpath = "//Button[@AutomationId='plusButton']") public WebElement botonSuma;

@FindBy(xpath = "//Button[@AutomationId='equalButton']") public WebElement botonIgual;

@FindBy(xpath = "//Text[@AutomationId='CalculatorResults']") public WebElement resultado;

@FindBy(xpath = "//Text[@AutomationId='Header']") public WebElement header;

}

En esta clase, lo que se hace, es definir los métodos para poder interactuar con los elementos que son mapeados en la ventana de la calculadora científica, **observar** que también se están tomando los screenshots (Serenity.takeScreenshot();) para usar estas capturas en los reportes.

Código de Calculadora.java

package WindowsAppDriver.interfaces;

import net.serenitybdd.core.Serenity;

import WindowsAppDriver.exception.NoIngresoUnNumeroCorrecto;;

public class Calculadora extends CalculadoraUI {

private void ingresarMenuPrincipal() {

menuPrincipal.click();

Serenity.takeScreenshot();

}

public void cambiarCalculadoraCientifica() {

if(!isScientMode()) {

ingresarMenuPrincipal();

menuCalculadoraCientifica.click();

Serenity.takeScreenshot();

}

}

public void ingresarDigito(int digito) {

switch (digito) {

case 1:

botonNumeroUno.click();

break;

case 2:

botonNumeroDos.click();

break;

case 3:

botonNumeroTres.click();

break;

case 4:

botonNumeroCuatro.click();

break;

case 5:

botonNumeroCinco.click();

break;

case 6:

botonNumeroSeis.click();

break;

case 7:

botonNumeroSiete.click();

break;

case 8:

botonNumeroOcho.click();

break;

case 9:

botonNumeroNueve.click();

break;

case 0:

botonNumeroCero.click();

Serenity.takeScreenshot();

break;

default:

throw new NoIngresoUnNumeroCorrecto("No ingreso un digito correcto");

}

}

public void operacionSuma() {

botonSuma.click();

Serenity.takeScreenshot();

}

public void realizarOperacion() {

botonIgual.click();

}

public String obtenerResultado() {

return resultado.getText();

}

public void cerrarCalculadora() {

getDriver().quit();

}

public String obtenerSuma() {

return resultado.getText();

}

public boolean isScientMode() {

if( header.getText().equals("Científica"))return true;

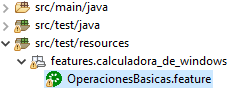
return false;

}

}

Terminada esta sección de mapeo de la calculadora, lo que sigue es hacer la parte de la feature con el lenguaje Gherkin. Para continuar accederemos al archivo llamado OperacionesBasicas.feature y pondremos el siguiente código que describe que prueba se desea hacer en la calculadora.

Avertencia: Como trabajaremos el lenguaje Gherkin en español, debemos asegurarnos de que el formato de texto para todo el proyecto esté en UTF-8 (utf8Format.gif).



# language: es

Característica: Calculadora de Windows

Yo como usuario del sistema operativo windows quiero utilizar la calculadora cientifica

para realizar operaciones basicas.

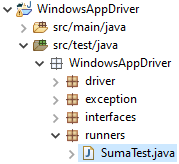
Escenario: Operacion basica de suma de dos enteros.

Dado la calculadora esta en modo Cientifico

Cuando se realiza la suma de 1001 mas 1000

Entonces se verifica que el resultado de la suma es 2001

Para generar las funciones del archivo stepdefinitions, tendremos que editar primero el archivo SumaTest.java, es importante tener en cuenta que la opción features es la ruta donde está el archivo con extensión **.features** que contiene las características de la prueba. Para continuar este archivo deberá llevar el siguiente código.



package WindowsAppDriver.runners;

import org.junit.runner.RunWith;

import io.cucumber.junit.CucumberOptions;

import static io.cucumber.junit.CucumberOptions.SnippetType.CAMELCASE;

import net.serenitybdd.cucumber.CucumberWithSerenity;

@RunWith(CucumberWithSerenity.class)

@CucumberOptions(

plugin = { "pretty" },

features = "src/test/resources/features/calculadora\_de\_windows/OperacionesBasicas.feature",

glue = "",

snippets = CAMELCASE,

monochrome = true,

dryRun = false

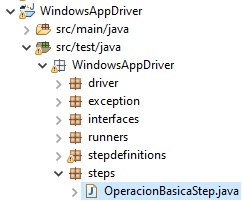
)

public class SumaTest {

}

Hecho esto, ya podemos ejecutar el archivo SumaTest.java para obtener las funciones de los stepdefinitions (featureSteps.gif) y ponelas en los stepsdefinitions de la calculadora.

Lo que sigue ahora es crear los steps que vamos a poner en el StepDefitions, esta clase tendrá los métodos que van a inicializar la calculadora, los que va a interactuar con los botones, y el método que verificará que la suma sea correcta.



package WindowsAppDriver.steps;

import org.fluentlenium.core.annotation.Page;

import net.thucydides.core.annotations.Step;

import WindowsAppDriver.interfaces.Calculadora;

import static org.junit.Assert.assertEquals;

import static WindowsAppDriver.util.SepararNumero.cantidadDeDigitosEnteros;

import static WindowsAppDriver.util.SepararNumero.entero;

public class OperacionBasicaStep {

@Page

Calculadora calculadora;

@Step

public void modoCalculadoraCientifica() {

calculadora.cambiarCalculadoraCientifica();

}

@Step("Se realiza la suma")

public void realizarSuma(Integer sumandoUno, Integer sumandoDos) {

ingresarNumero(sumandoUno);

calculadora.operacionSuma();

ingresarNumero(sumandoDos);

calculadora.realizarOperacion();

}

@Step

public void ingresarNumero(Integer numero) {

for (int i = cantidadDeDigitosEnteros(numero) - 1; i >= 0; i--) {

calculadora.ingresarDigito(entero(numero).get(i));

}

}

@Step

public void verificarLaSumaTotal(int totalEsperado) {

int sumaTotal = Integer.parseInt(calculadora.obtenerResultado().replace("La pantalla muestra", "").replace(".", "").trim());

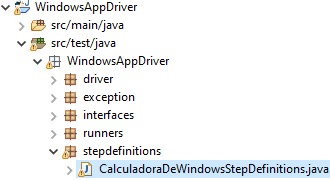
calculadora.cerrarCalculadora();

assertEquals(sumaTotal, totalEsperado);

}

}

Teniendo lista esta clase, ingresamos al documento de los stepdefinitions de la calculadora de Windows, donde copiaremos los métodos que se retornaron de haber ejecutado la parte de features e inicializaremos la clase **OperacionBasicaStep**, también implementaremos los métodos que creamos en la clase anterior, de modo que el código quede así.



package WindowsAppDriver.stepdefinitions;

import net.thucydides.core.annotations.Steps;

import WindowsAppDriver.steps.OperacionBasicaStep;

import io.cucumber.java.es.Dado;

import io.cucumber.java.es.Cuando;

import io.cucumber.java.es.Entonces;

public class CalculadoraDeWindowsStepDefinitions {

@Steps

OperacionBasicaStep calculadoraStep;

@Dado("la calculadora esta en modo Científico")

public void laCalculadoraEstaEnModoCientífico() {

calculadoraStep.modoCalculadoraCientifica();

}

@Cuando("se realiza la suma de {int} mas {int}")

public void seRealizaLaSumaDeMas(Integer sumandoUno, Integer sumandoDos) {

calculadoraStep.realizarSuma(sumandoUno, sumandoDos);

}

@Entonces("se verifica que el resultado de la suma es {int}")

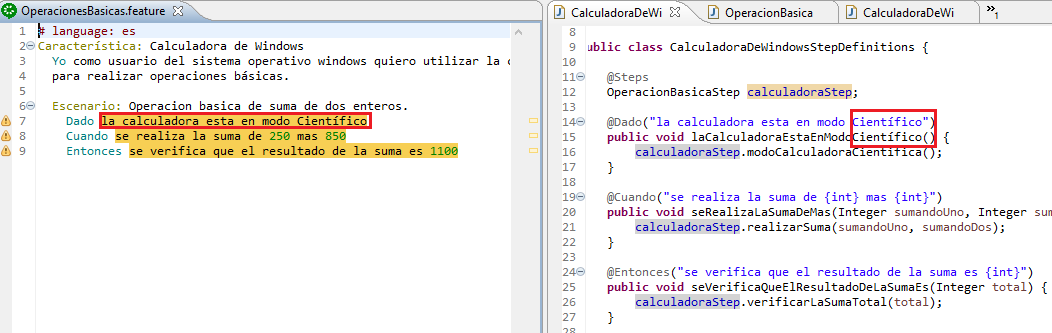
public void seVerificaQueElResultadoDeLaSumaEs(Integer total) {

calculadoraStep.verificarLaSumaTotal(total);

}

}

Es importante resaltar que si en el lenguaje Gherkin hemos escrito estos métodos con tilde, en la implementación de los stepfinitions también lo hagamos, porque si no se hace, habrían problemas a la hora de compilar.



Hecho todo esto, lo que queda es iniciar el servidor y compilar el proyecto con gradle. Para iniciar el servidor, iremos al directorio donde tengamos instalado el WinAppDriver, y en esta carpeta, haremos click en su ruta y escribiremos cmd, luego de que iniciemos la consola de comandos dentro del directorio de WinAppDriver, lo que queda es escribir WinAppDriver.exe en la terminal y oprimir **ENTER**, cuando se oprima **ENTER**, el servidor de WinAppDriver quedará escuchando solicitudes en el puerto 4723, si se quiere un puerto distinto, se tiene que escribir un argumento extra después de escribir WinAppDriver.exe (Ejemplo: WinAppDriver.exe 9999) y después de ejecutar el comando, debajo de este, aparecerá el puerto en el que se está ejecutando WinAppDriver (startWinAppDriv.gif), para terminar de ejecutar WinAppDriver hay que oprimier **ENTER** sobre la terminal.

Lo que sigue es ingresar al directorio donde está el proyecto de WindowsAppDriver y escribir sobre el directorio **CMD** para poder iniciar la terminal de comandos ahí. Una vez que la terminal de comandos esté ubicada en ese directorio escribimos **gradle build** y oprimimos **ENTER**, hecho esto, esperamos a que la ejecución de la prueba para la calculadora termine (gradleBuild.gif).

Cuando se termine la ejecución, podremos entrar a la carpeta **WindowsAppDriver**/**target**/**site**/**serenity** y ahí veremos generado el reporte del proyecto (reporte.gif).