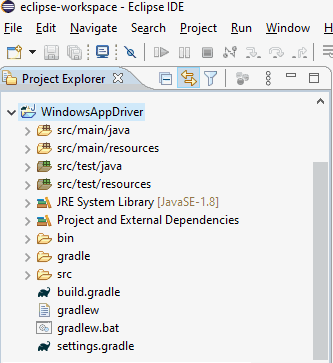
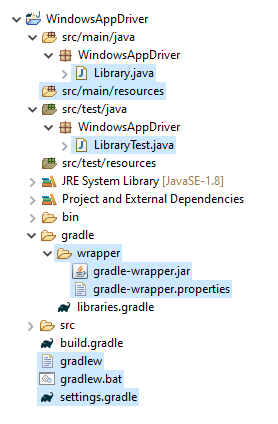
Para el siguiente ejemplo se realizó la automatización de la calculadora de Windows y para esto se utilizó el patrón de diseño POM, y también se implementó serenity para la generación de reportes.

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.

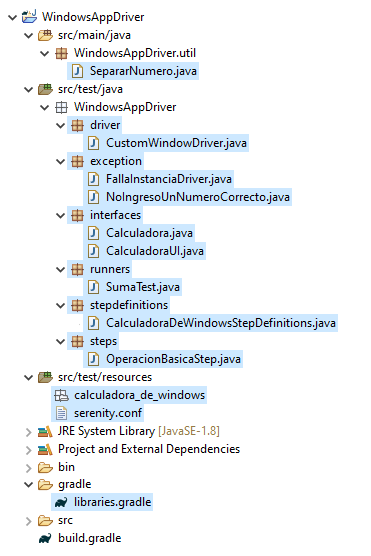


De esta estructura borraremos la carpeta **src**/**main**/**resources**, la carpeta **wrapper** que **está dentro** de la carpeta **gradle** con todo su contenido, y los archivos **Library.java**, **LibraryTest.java**, **gradlew**, **gradlew.bat**, **settings.gradle**, pues estos no serán útiles para 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:2.0.91")

}

}

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 y escribimos lo siguiente:

