

INTEGRACIÓN CONTINUA

RESOLUCIÓN DE EJERCICIOS

A continuación, encontrará el desarrollo de los ejercicios que resolvió en la semana. Contraste las respuestas entregadas por el docente con las desarrolladas por usted. En caso que no coincidan, y persistan dudas, le invitamos a repasar los contenidos y/o consultar con su profesor.

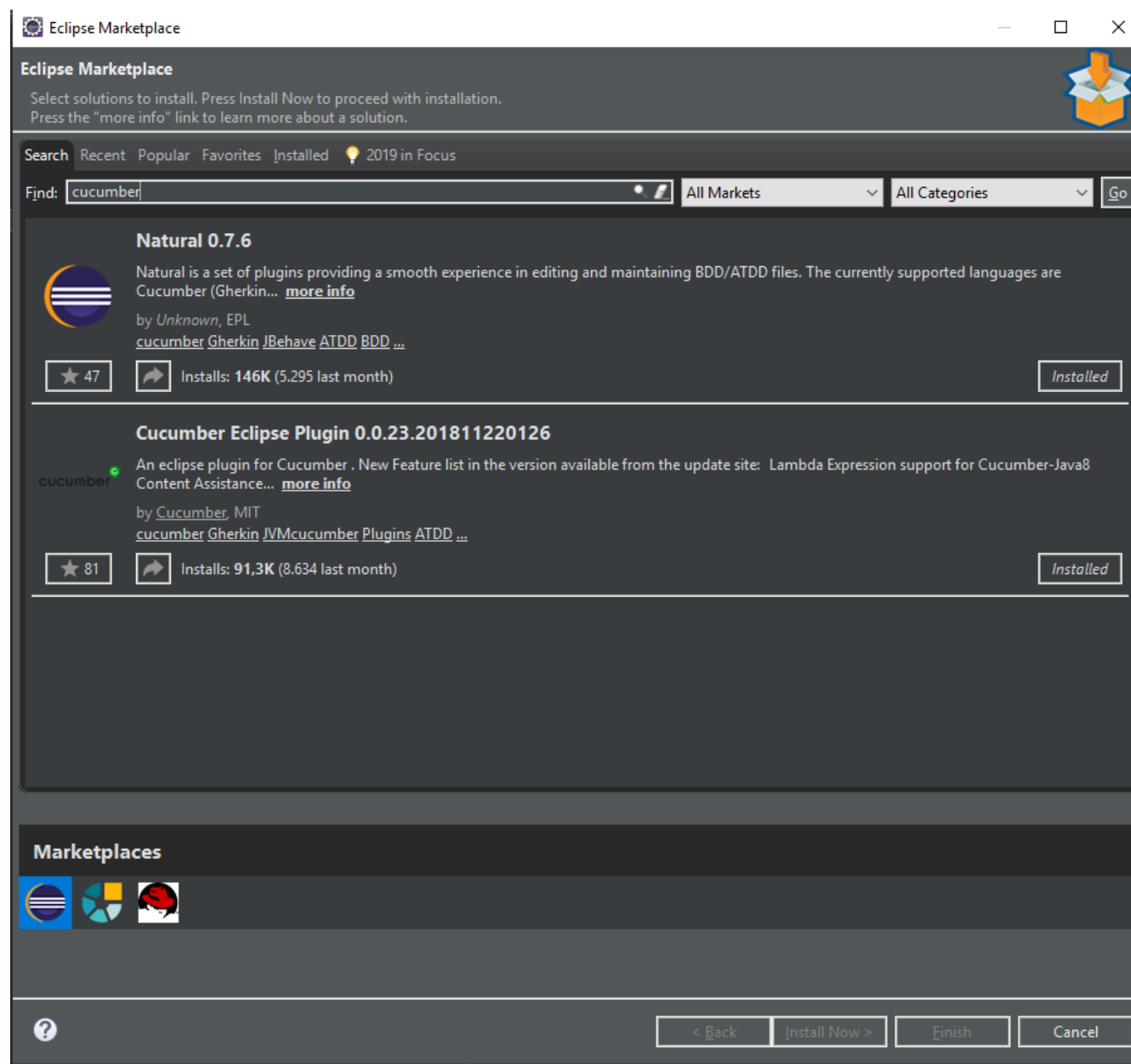
1.1 Realizar pruebas de automatización utilizando Cucumber con selenium.

Paso 1 abrimos nuestro entorno de desarrollo IDE Eclipse, creamos un nuevo proyecto.

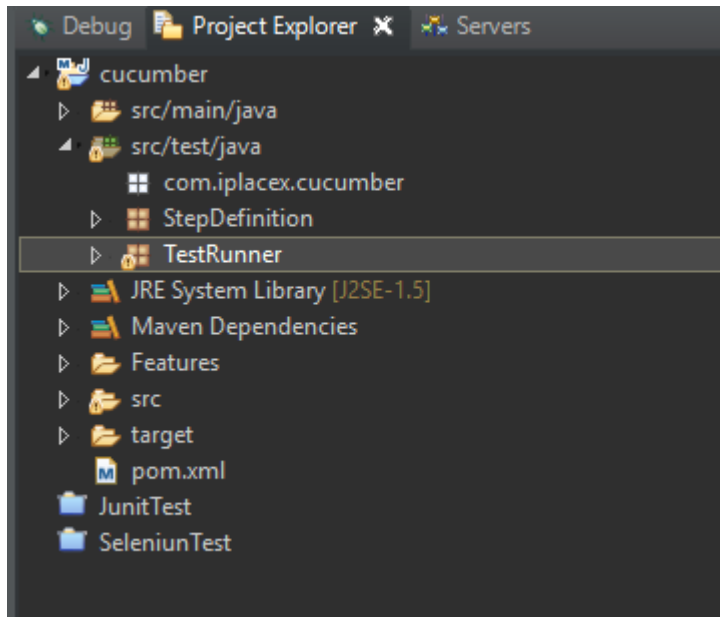
Pas 2 Agregamos las siguientes librerías a nuestro archivo pom.xml

```
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-junit</artifactId>
  <version>4.5.4</version>
  <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/io.cucumber/gherkin -->
<!-- https://mvnrepository.com/artifact/info.cukes/gherkin -->
<dependency>
  <groupId>info.cukes</groupId>
  <artifactId>gherkin</artifactId>
  <version>2.12.2</version>
  <scope>provided</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-server -->
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-server</artifactId>
  <version>3.141.59</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-core -->
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-core</artifactId>
  <version>4.5.4</version>
</dependency>
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-java</artifactId>
  <version>4.3.1</version>
  <scope>test</scope>
</dependency>
</dependencies>
</project>
```

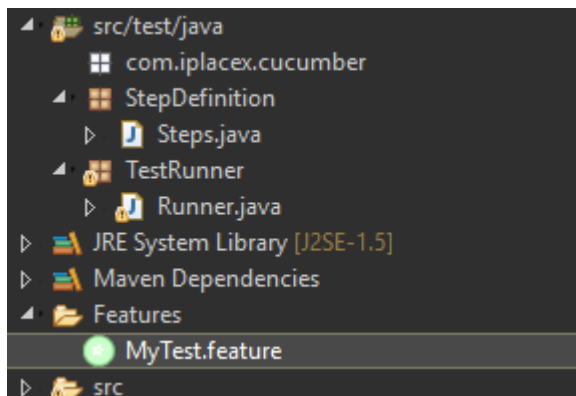
Paso 3 Nos dirigimos al Marketplace de eclipse y realizamos la instalación de los complementos relacionados a cucumber e instalamos las dos herramientas que nos muestra en pantalla Cucumber Eclipse plugin y natural para la interpretación de las instrucciones del archivo features que veremos más adelante.



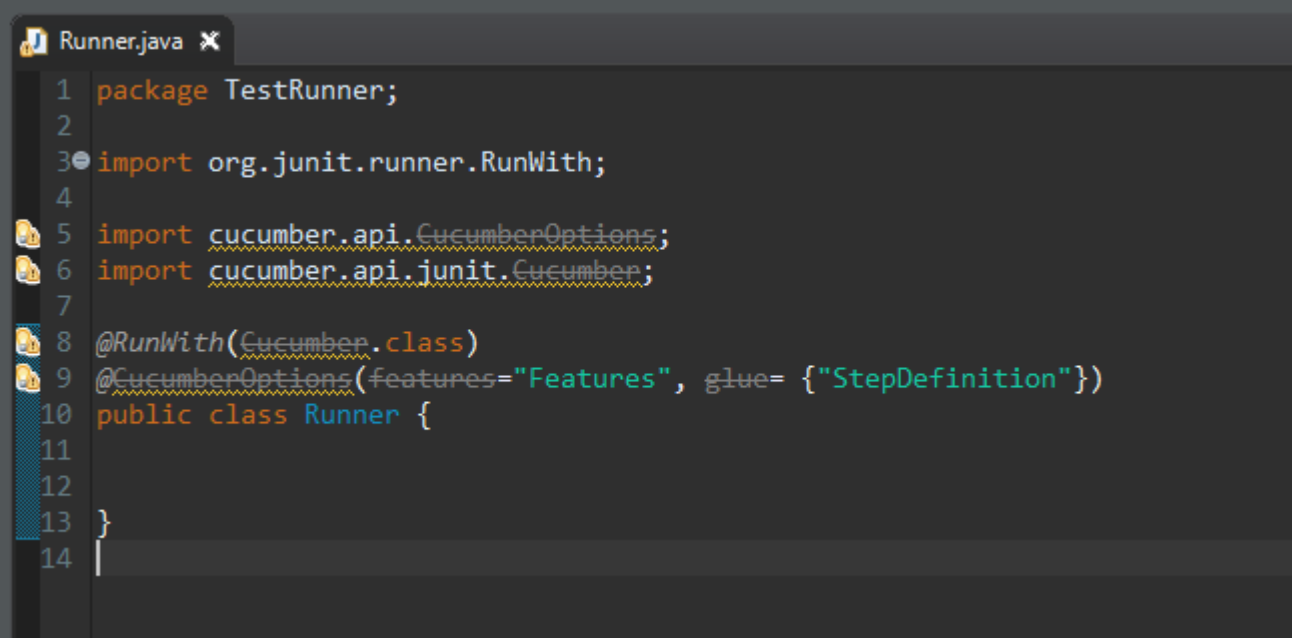
Paso 4 dentro de la carpeta `src/test/java` creamos dos package `StepDefinitio` y `TestRunner` y en el directorio principal creamos una carpeta llamada `Feature`,



Paso 5 Creamos la siguiente estructura de clases dentro de los package y un archivo dentro de la carpeta `Feature` con extension `.feature`



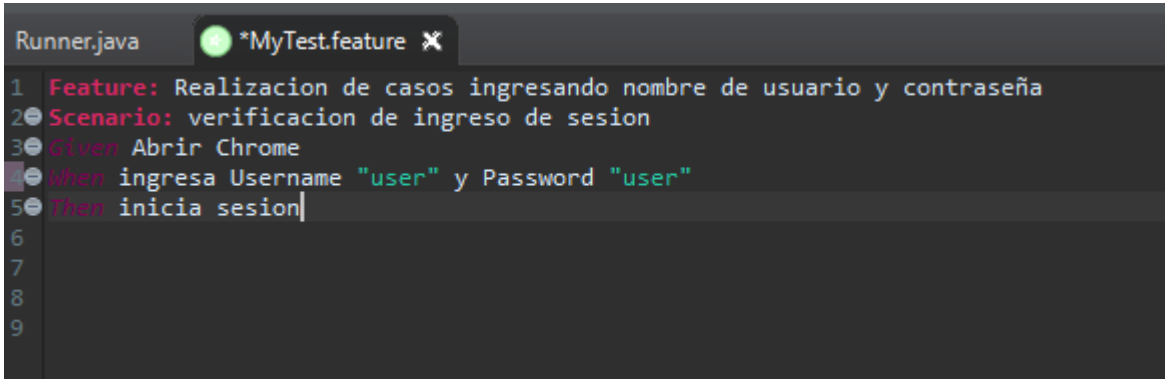
Paso 6: En la clase Runner realizamos la configuración de los parámetros que unen la clase dentro de package StepDefinition y la carpeta Features.



```
1 package TestRunner;
2
3 import org.junit.runner.RunWith;
4
5 import cucumber.api.CucumberOptions;
6 import cucumber.api.junit.Cucumber;
7
8 @RunWith(Cucumber.class)
9 @CucumberOptions(features="Features", glue= {"StepDefinition"})
10 public class Runner {
11
12
13 }
14
```

Paso 7: Nos dirigimos al archivo dentro de la carpeta features e el cual realizaremos las instrucciones mediante el lenguaje Gherkin el cual está basado en la metodología BDD para dar instrucciones en lenguaje entendible tanto para profesionales de Ti o de negocio.

En la siguiente imagen ingresamos las instrucciones de nuestro caso de prueba el cual realizaremos la impresión con consola, al ejecutarlo por primera vez nos pedirá implementar ciertos métodos en la clase Step(pasos).



En la imagen realizamos la ejecución de las pruebas sin especificar los métodos en la clase Steps.

```
You can implement missing steps with the snippets below:

@Given("Abrir Chrome")
public void abrir_Chrome() {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}

@When("ingresa Username {string} y Password {string}")
public void ingresa_Username_y_Password(String string, String string2) {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}

@Then("inicia sesion")
public void inicia_sesion() {
    // Write code here that turns the phrase above into concrete actions
    throw new cucumber.api.PendingException();
}
```

1.2 Caso 1 imprimir texto en consola:

Haciendo referencia al paso 1 implementaremos los métodos que nos requiere el paso 7, para esto nosotros crearemos dentro del package StepDefintion y crearemos la clase Steps, copiando los métodos que nos piden en la ejecución anterior pero que solo muestre datos por consola.

Paso 1: Para este paso lo primero que debemos hacer es realizar la importación de las siguientes librerías de java mediante el comando import.

```
1 package StepDefinition;
2
3 import org.openqa.selenium.By;
4 import org.openqa.selenium.WebDriver;
5 import org.openqa.selenium.chrome.ChromeDriver;
6
7 import cucumber.api.java.en.Given;
8 import cucumber.api.java.en.Then;
9 import cucumber.api.java.en.When;
```

Luego pegamos el texto que aparece en consola en nuestra clase steps, si nos damos cuenta los métodos que nos pide cucumber son los mismos que especificamos en el archivo MyTest.feature esto es porque cada paso que ingresamos los tenemos que programar mediante el lenguaje de java.

Como primera parte en cada método realizaremos una impresión con consola mostrando que es lo que hace en cada paso.

```
11
12
13 public class Steps {
14     @Given("Abrir Chrome")
15     public void abrir_Chrome() {
16         // Write code here that turns the phrase above into concrete actions
17         System.out.println("En este paso abre el navegador google chrome");
18     }
19
20     @When("ingresa Username {string} y Password {string}")
21     public void ingresa_Username_y_Password(String string, String string2) {
22         // Write code here that turns the phrase above into concrete actions
23         System.out.println("En ete paso ingresa las credenciales de usuario");
24     }
25
26     @Then("inicia sesion")
27     public void inicia_sesion() {
28         // Write code here that turns the phrase above into concrete actions
29         System.out.println("En ete paso inicia sesion con los datos ingresados desde nuestro archivo Mytest.feature");
30     }
31 }
32
33
34
35
```

Luego lo ejecutamos y nos mostrara lo que ingresamos en cada método

```
Feature: Realizacion de casos ingresando nombre de usuario y contraseña

  Scenario: verificacion de ingreso de sesion      # /C:/Users/Iplacex-PC/eclipse-workspace/cucumber/Features/MyTest.feature:2
  En este paso abre el navegador google chrome
    Given Abrir Chrome                            # Steps.abrir_Chrome()
  En ete paso ingresa las credenciales de usuario
    When ingresa Username "user" y Password "user" # Steps.ingresa_Username_y_Password(String,String)
  En ete paso inicia sesion con los datos ingresados desde nuestro archivo Mytest.feature
    Then inicia sesion                            # Steps.inicia_sesion()

1 Scenarios (1 passed)
3 Steps (3 passed)
```

Paso 2: una vez que nuestros métodos realicen la impresión por consola procedemos a darle funcionalidad a cada uno según la especificación ingresada en los casos de prueba.

```
WebDriver driver;

@Given("Abrir Chrome")
public void abrir_Chrome() {
    // Write code here that turns the phrase above into concrete actions
    System.out.println("En este paso abre el navegador google chrome");
    System.setProperty("webdriver.chrome.driver", "C:\\Users\\Iplacex-Pc\\Downloads\\chromedriver.exe");
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.get("https://ced.iplacex.cl/login/index.php");
}

@When("ingresa Username {string} y Password {string}")
public void ingresa_Username_y_Password(String string, String string2) {
    // Write code here that turns the phrase above into concrete actions
    System.out.println("En ete paso ingresa las credenciales de usuario");
    System.out.println("Este paso ingresa el nombre de usuario y contraseña en la pagina de login");
    driver.findElement(By.id("username")).sendKeys(string);
    driver.findElement(By.id("password")).sendKeys(string2);
}

@Then("inicia sesion")
public void inicia_sesion() {
    // Write code here that turns the phrase above into concrete actions
    System.out.println("En ete paso inicia sesion con los datos ingresados desde nuestro archivo Mytest.feature");
    driver.findElement(By.id("loginbtn")).click();
}
```

1.3 Ingresamos las credenciales de nuestro portal de alumnos de esta forma validamos nuestro entorno de pruebas modificando la línea de comando: `when ingresa Username "user" y Password "user"` en nuestro archivo MyTest.Feature. Ejemplo de muestra en gif adjunto en Google drive:
<https://drive.google.com/file/d/1aitiThP2Bbb0rZbaVDIZbSUdDFNxBI0H/view?usp=sharing>

Nos aparecerá un mensaje de que las credenciales no son correctas, pero si ingresa su nombre de usuario y contraseña dentro del archivo features ingresara sin problema.

1.4 Ingresaremos distintas credenciales y antes de presionar acceder borraremos los campos, para esto tendremos que hacer una modificación a nuestro archivo feature agregando un paso antes de que inicie la sesión, lo que realizara la prueba es ingresar datos de en el recuadro de inicio de sesión, luego los borrara y por último presionara click en aceptar.

Luego restaría realizar pruebas modificando el archivo feature la siguiente línea de comandos los campos: `when ingresa Username "user" y Password "user"`

Como muestra en el siguiente GIF que se encuentra almacenado en el siguiente recurso en Google Drive :
<https://drive.google.com/file/d/1nduWThyXpuLhAN9qKnvi0g6i8sOdM0n7/view?usp=sharing>

RESPUESTAS ESPERADAS

1. Respecto a Cucumber con Selenium herramientas para realizar pruebas de automatización, facilitando la lectura y la comprensión del flujo de aplicaciones. ¿Por qué integrar estas dos herramientas?

Cucumber es una infraestructura de pruebas que ayuda a crear puentes entre los desarrolladores de software y los gestores de negocio. Las pruebas están escritas en un lenguaje sencillo que se basa en el estilo de desarrollo basado en el comportamiento (BDD) de Given, When, Then, que cualquier lego puede entender. Los casos de pruebas son posteriormente colocados en archivos de características que cubren uno o más escenarios de pruebas. Cucumber interpreta las pruebas en el lenguaje de programación especificado, y utiliza Selenium para dirigir los casos de pruebas en un navegador. Nuestras pruebas se traducen a código de Java.

2. ¿Cuál es la importancia de crear una estructura automatizada para realizar pruebas funcionales con Selenium y Cucumber?

Crear una infraestructura automatizada para probar aplicaciones web es un gran desafío, ya que, al utilizar Selenium y Cucumber facilita mucho esa tarea y entrega como resultado suites de pruebas más sostenibles. Crear una suite de pruebas automática para una aplicación de Internet enriquecida. Por otra parte, podemos escribir y utilizar archivos de características y consejos para crear fácilmente el código de la aplicación en diferentes navegadores. Planificar con antelación las siguientes áreas adicionales como:

- **Mantener los datos de las pruebas**
- **Utilizar el interpretador de archivos de características**
- **Ejecutar las pruebas**

