

## ***ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)***

### **Implementación de Given, When, Then**

En la presente guía, aprenderemos cómo interactúa el actor con las tareas (tasks) sobre la interfaz de usuario (userinterfaces) usaremos la capa (models) para manipular los datos, también aprenderemos cómo identificar los objetos de nuestra desktop application, y cómo implementar las verificaciones en el desarrollo de Questions. Para el desarrollo de cada paso descrito en la historia de usuario.

#### Tabla de contenido

Implementar @Given .....	2
Identificar los objetos .....	9
Creación de user interface.....	13
Implementar @When .....	16
Implementar @Then.....	22



## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

### Implementar @Given

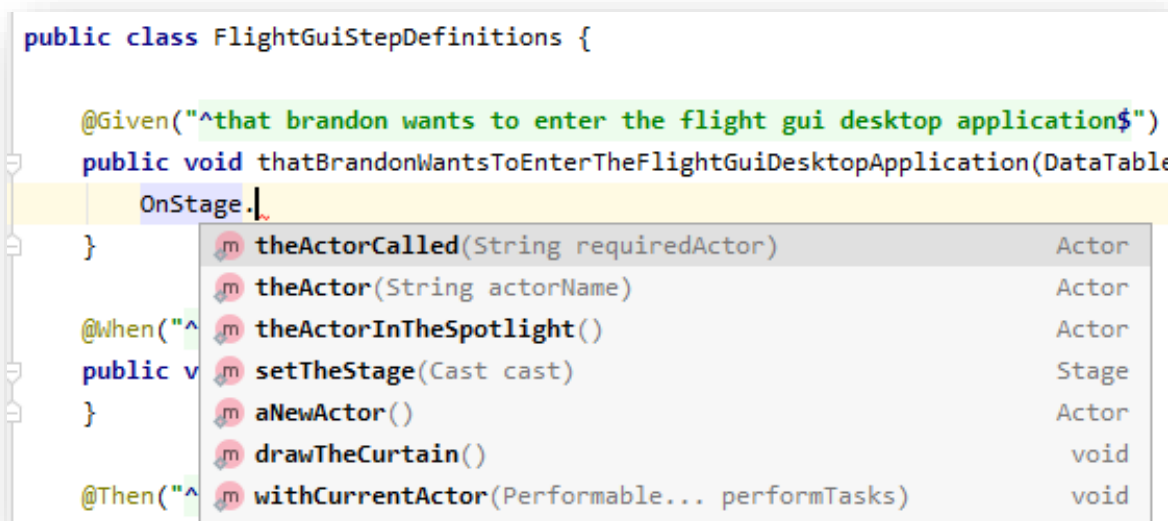
Retomando donde terminamos la guía pasada, vamos a nuestra clase “FlightGuiStepDefinitions” y comencemos con la implementación del paso:

```
@Given("^that brandon wants to enter the flight gui desktop application$")
```

Dicha implementación la haremos en el método llamado:

```
public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(DataTable  
arg1) {  
}
```

Lo primero que haremos será escribir el nombre de nuestro actor. En esta guía, nuestro actor se llama “brandon”, Usaremos la clase “OnStage” y el método “theActorCalled()”.



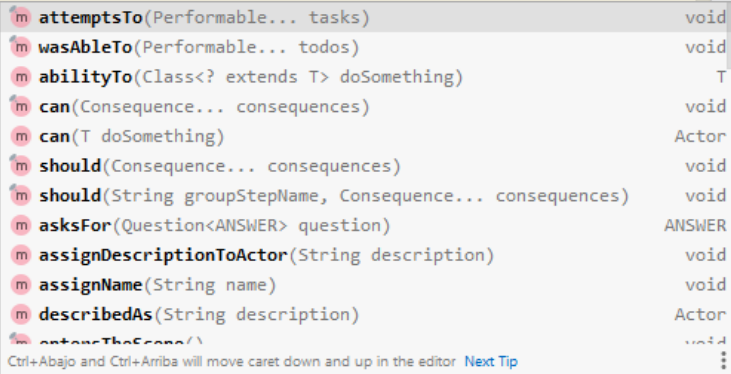
```
public class FlightGuiStepDefinitions {  
  
    @Given("^that brandon wants to enter the flight gui desktop application$")  
    public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(DataTable  
        arg1) {  
        OnStage.  
    }  
  
    @When("^")  
    public void  
    }  
  
    @Then("^")  
    }  
}
```

m	theActorCalled(String requiredActor)	Actor
m	theActor(String actorName)	Actor
m	theActorInTheSpotlight()	Actor
m	setTheStage(Cast cast)	Stage
m	aNewActor()	Actor
m	drawTheCurtain()	void
m	withCurrentActor(Performable... performTasks)	void

y escribamos el nombre del actor dentro del método, escribiremos el carácter punto, “.”. Observaremos que nos mostrará todos los métodos que nuestro actor brandon puede ejecutar.

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

```
public class FlightGuiStepDefinitions {  
  
    @Given("^that brandon wants to enter the flight gui desktop application$")  
    public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(DataTable arg1) {  
        OnStage.theActorCalled( requiredActor: "brandon").  
    }  
  
    @When("^he enters the data to buy the")  
    public void heEntersTheDataToBuyTheTicket() {  
    }  
  
    @Then("^he verifies the purchase with")  
    public void heVerifiesThePurchaseWithTheTicket() {  
    }  
}
```



Sin embargo, haremos uso del método `wasAbleTo()` sólo cuando estemos implementando el paso del Given (Precondiciones).

```
@Given("^that brandon wants to enter the flight gui desktop application$")  
public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(DataTable arg1) {  
    OnStage.theActorCalled( requiredActor: "brandon").wasAbleTo();  
}
```

Dentro de los paréntesis escribiremos la tarea que deseamos ejecutar. Y usaremos la siguiente estructura. Una acción (la clase), seguida de un punto "." Y una frase de complemento (el método de la clase) seguida de un par de paréntesis.

```
OnStage.theActorCalled( requiredActor: "brandon").wasAbleTo(OpenThe.flightAppDesktop());
```

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Antes de continuar vamos a hacer modificaciones en el método que estamos implementado vamos a cambiar

```
@Given("^that brandon wants to enter the flight gui desktop application$")
public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(DataTable arg1) {
    ...
}
```

Por

```
@Given("^that (.*) wants to enter the flight gui desktop application$")
public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(String
brandon,List <LoginModel> dataSet) {...}
```

Y nuestro Given debe verse así:

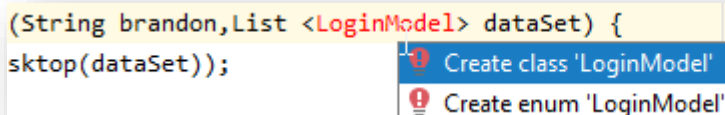
```
public class FlightGuiStepDefinitions {

    @Given("^that (.*) wants to enter the flight gui desktop application$")
    public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(String brandon,List <LoginModel> dataSet) {
        OnStage.theActorCalled(brandon).wasAbleTo(OpenThe.flightAppDesktop(dataSet));
    }
}
```

En la variable *String* encapsulamos la información que se encuentra dentro *(.\*)* y le asignamos la variable al método *theActorCalled(brandon)* y además cambiamos en el método *thatBrandonWantsToEnterTheFlightGuiDesktopApplication( DataTable arg1)* por *(List<LoginModel> dataSet)*, con el fin de manipular la data del given como una lista enviándosela a la clase *LoginModel* que crearemos a continuación.

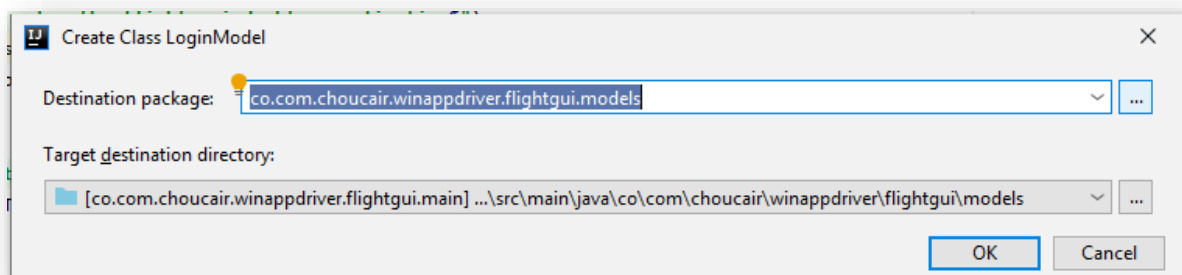
## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Como se indico anteriormente vamos a crear la clase *LoginModel* dentro del paquete *models*, para hacerlo ponemos el puntero en la clase *LoginModel* y presionamos las teclas *Alt + Enter*

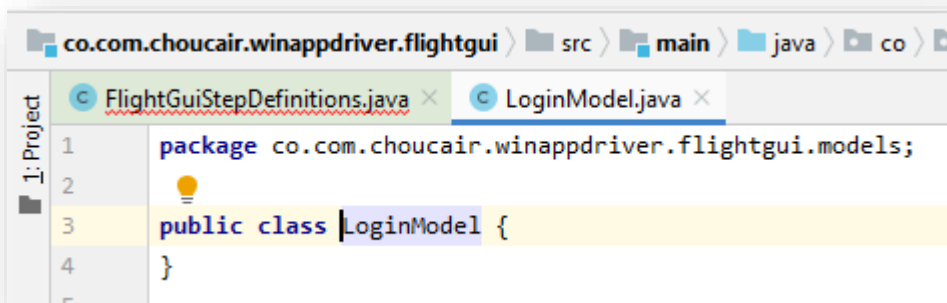


```
(String brandon, List <LoginModel> dataSet) {  
sktop(dataSet));
```

Dar clic en Create class 'LoginModel', seguido presionamos las teclas *Mayús + Enter* y escogemos el paquete *Models* y damos clic en el botón *OK*.



y una vez más damos clic en el botón *OK* y nos creara la clase vacía



```
package co.com.choucair.winappdriver.flightgui.models;  
  
public class LoginModel {  
}  

```

Volvemos a la clase *stepDefinition* y agregamos `import java.util.List;`

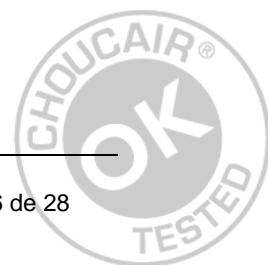
## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

En la Clase *LoginModel* creamos los atributos de tipo privado que están en la feature en el Given con el mismo nombre(user, password).

```
Given that brandon wants to enter the flight gui desktop application
| user | password |
| john | HP      |
```

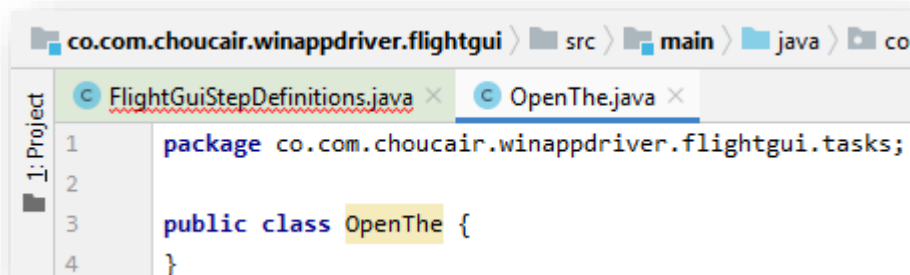
Y creamos los llamados getters y setters que son métodos para obtener o modificar propiedades de una clase. Para nuestro caso solo crearemos los getters ya que solo queremos obtener la información que tenga el método.

```
LoginModel.java x
1  package co.com.choucair.winappdriver.flightgui.models;
2
3  public class LoginModel {
4      private String user;
5      private String password;
6
7      public String getUser() {
8          return user;
9      }
10
11     public String getPassword() {
12         return password;
13     }
14 }
15
```



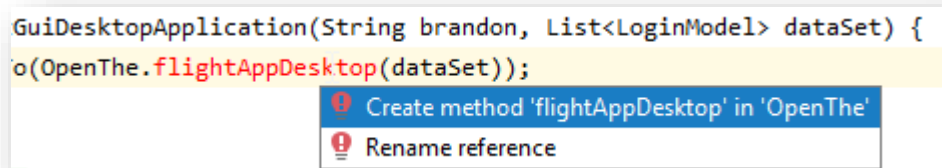
## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Paso a seguir será crear la clase *OpenThe* y el método *flightAppDesktop()*, repetimos los pasos que hicimos para crear la clase *LoginModel*, con la diferencia que la Clase *OpenThe* se alojará en el paquete *Tasks*



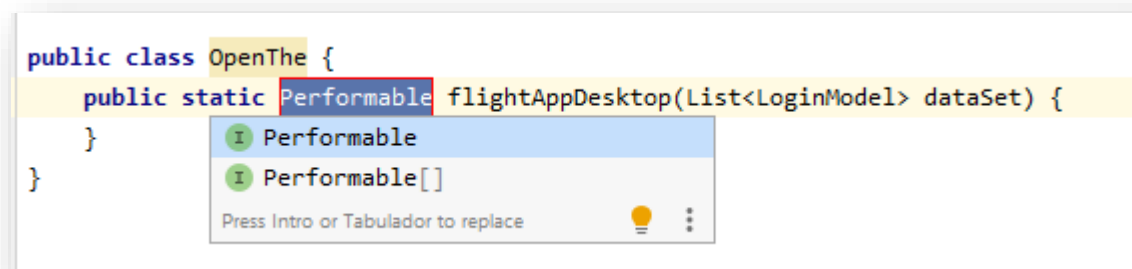
```
co.com.choucair.winappdriver.flightgui > src > main > java > co
FlightGuiStepDefinitions.java x OpenThe.java x
1 package co.com.choucair.winappdriver.flightgui.tasks;
2
3 public class OpenThe {
4 }
```

Volvemos a la Clase ***FlightGuiStepDefinitions*** y vamos a crear el método ***flightAppDesktop()***, para esto nos posicionamos sobre el método y presiona la tecla *Alt + Enter*



```
GuiDesktopApplication(String brandon, List<LoginModel> dataSet) {
o(OpenThe.flightAppDesktop(dataSet));
}
Create method 'flightAppDesktop' in 'OpenThe'
Rename reference
```

y crea el método en la clase *OpenThe* .



```
public class OpenThe {
    public static Performable flightAppDesktop(List<LoginModel> dataSet) {
    }
}
Performable
Performable[]
Press Intro or Tabulador to replace
```

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Nos creara el método estático, el cual vamos a modificar *Performable* por el nombre de la clase que creamos *OpenThe*, crearemos el return y a la clase le implementaremos de Task para que nos genere el @Override, en esta parte es donde el actor realizara la acciones. Además declararemos una variable publica de tipo *List<LoginModel>* y le crearemos su constructor.

```
1 package co.com.choucair.winappdriver.flightgui.tasks;
2
3 import co.com.choucair.winappdriver.flightgui.models.LoginModel;
4 import net.serenitybdd.screenplay.Actor;
5 import net.serenitybdd.screenplay.Task;
6 import net.serenitybdd.screenplay.Tasks;
7
8 import java.util.List;
9
10 public class OpenThe implements Task {
11     private List<LoginModel> dataSet;
12
13     public OpenThe(List<LoginModel> dataSet) { this.dataSet = dataSet; }
14
15
16
17     public static OpenThe flightAppDesktop(List<LoginModel> dataSet) {
18         return Tasks.instrumented(OpenThe.class, dataSet);
19     }
20
21     @Override
22     public <T extends Actor> void performAs(T actor) {
23
24     }
25 }
```

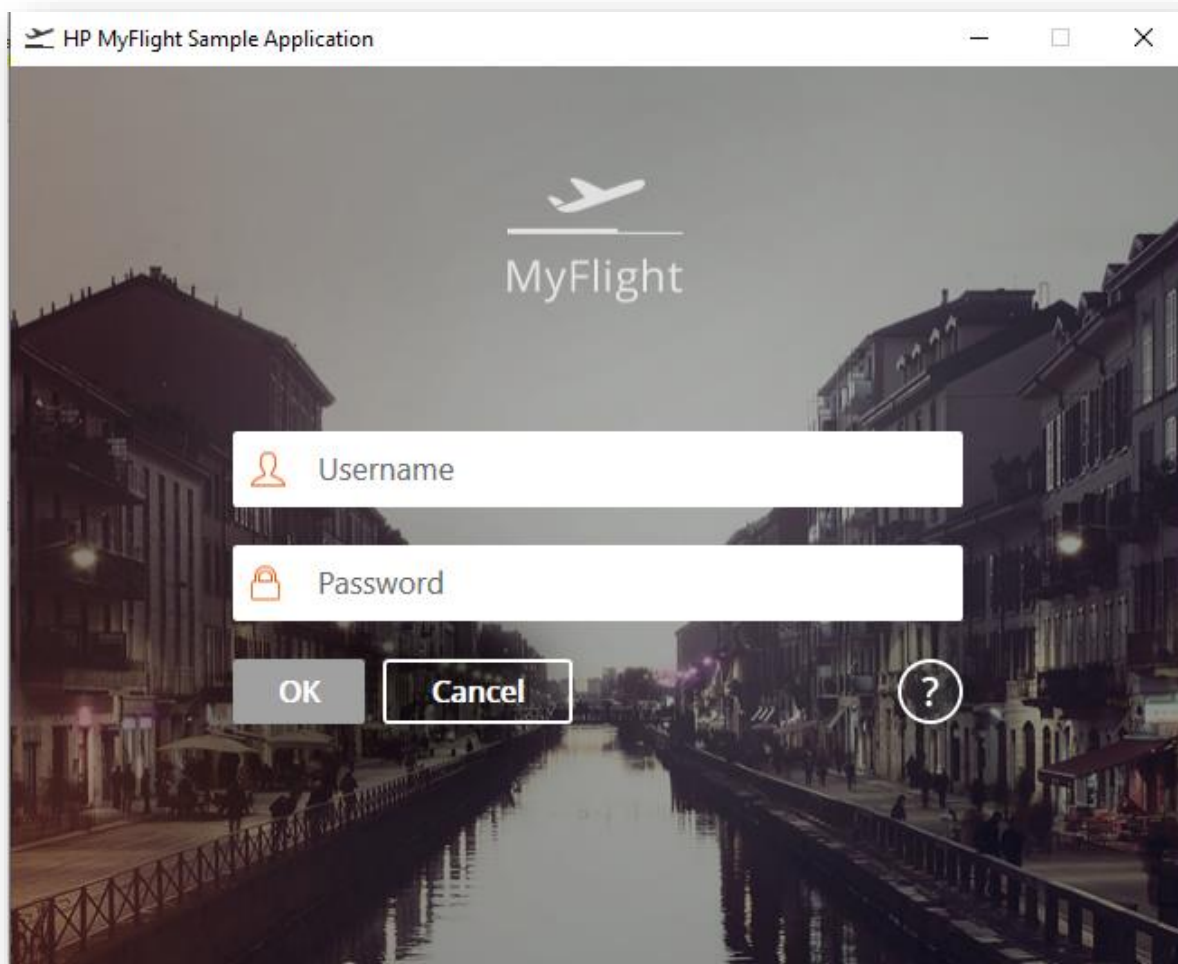
Antes de continuar es necesario mapear los objetos, mas adelante volveremos a esta clase.



## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

### Identificar los objetos

Como lo mencionamos en la guía anterior la aplicación de escritorio que vamos a inspeccionar se llama Flight GUI, a continuación, se aprenderá a inspeccionar los objetos con la aplicación llamada “*inspect.exe*”

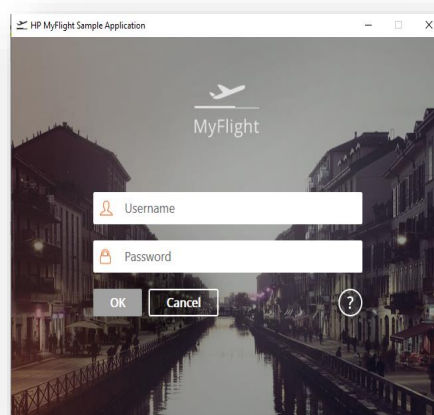
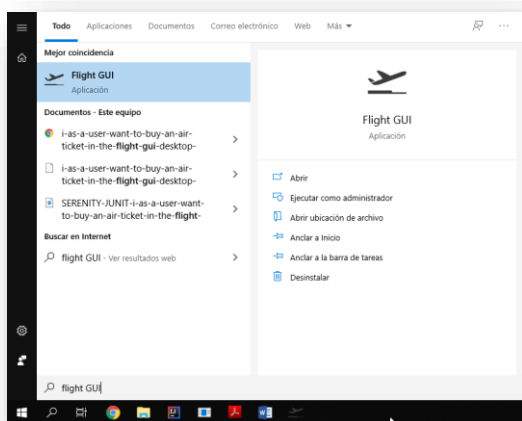


Para este caso se mapearán los objetos para el ingreso del usuario la contraseña y el botón ok.

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

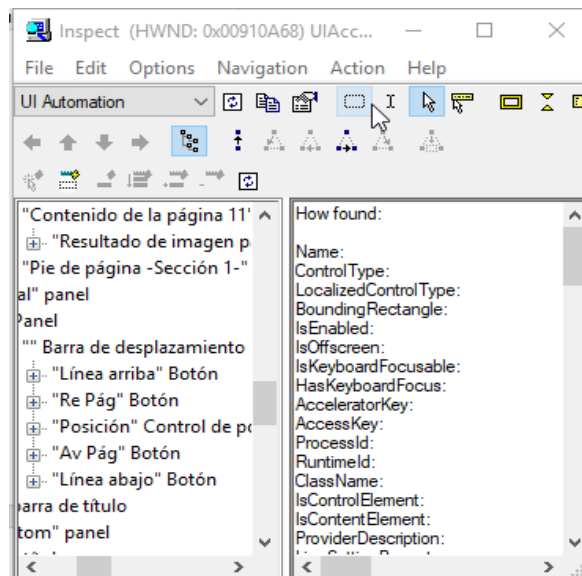
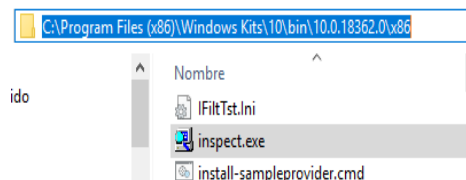
Es necesario tener la aplicación Flight GUI abierta , la cual la podemos buscar en la lupa de Windows 10.

**Nota:** Esta aplicación está contenida en el paquete de HP UFT, si no lo tienes instalado puedes usar la calculadora de Windows para hacer el ejercicio y deberías reestructurar lo que llevamos del proyecto.



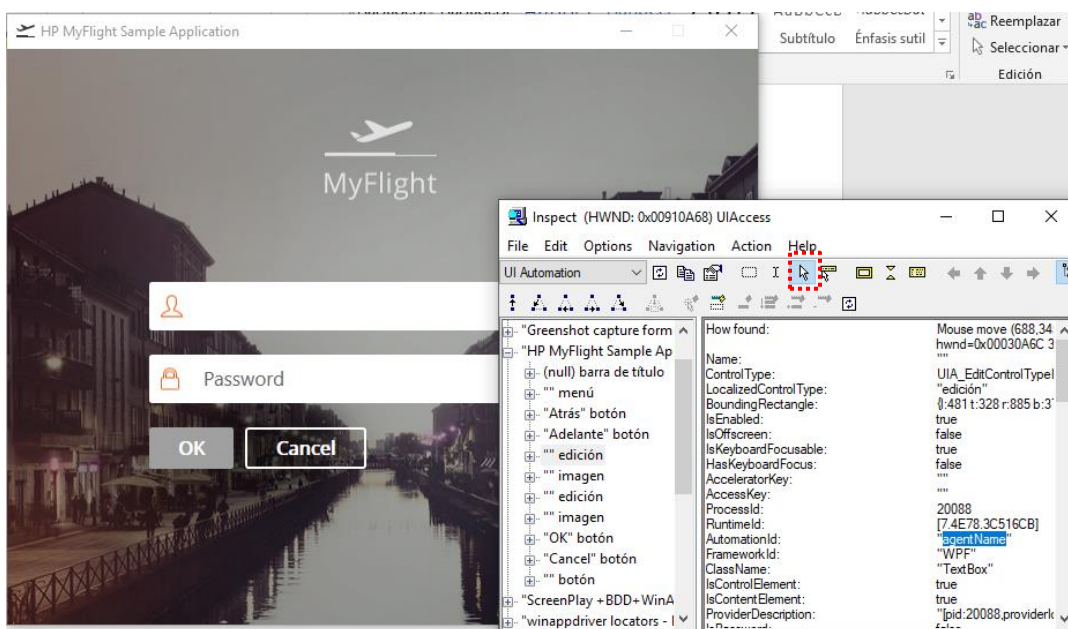
También es necesario abrir **inspect.exe** que hace parte del SDK de Windows que ya se instalo en la primera guía, éste se encuentra ubicado en la ruta **C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x86**

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When, @Then)



## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Una vez tenemos las dos aplicaciones abiertas, activamos el puntero en la aplicación **inspect** y movemos el mouse al elemento que deseamos inspeccionar, para el ejemplo de la siguiente imagen se inspecciona el campo Username el cual vamos a utilizar el atributo AutomationId para ubicarlo en este caso llamado "agentName".



Se adjunta una tabla del orden de los atributos para localizar los elementos.

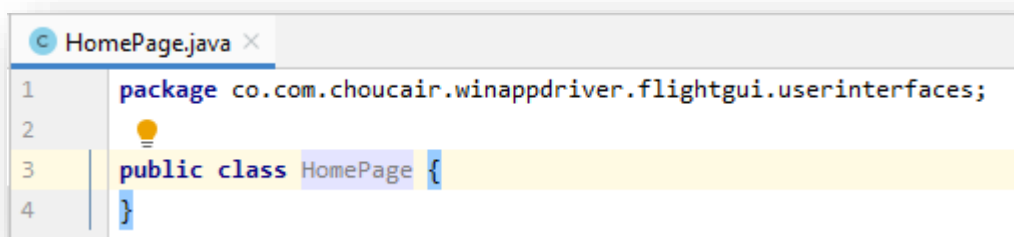
Client API	Locator Strategy	Matched Attribute in inspect.exe	Example
FindElementByAccessibilityId	accessibility id	AutomationId	AppNameTitle
FindElementByClassName	class name	ClassName	TextBlock
FindElementById	id	RuntimeId (decimal)	42.333896.3.1
FindElementByName	name	Name	Calculator
FindElementByTagName	tag name	LocalizedControlType (upper camel case)	Text
FindElementByXPath	xpath	Any	//Button[0]

**Ahora ya puedes ensayar y localizar los objetos faltantes.**

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

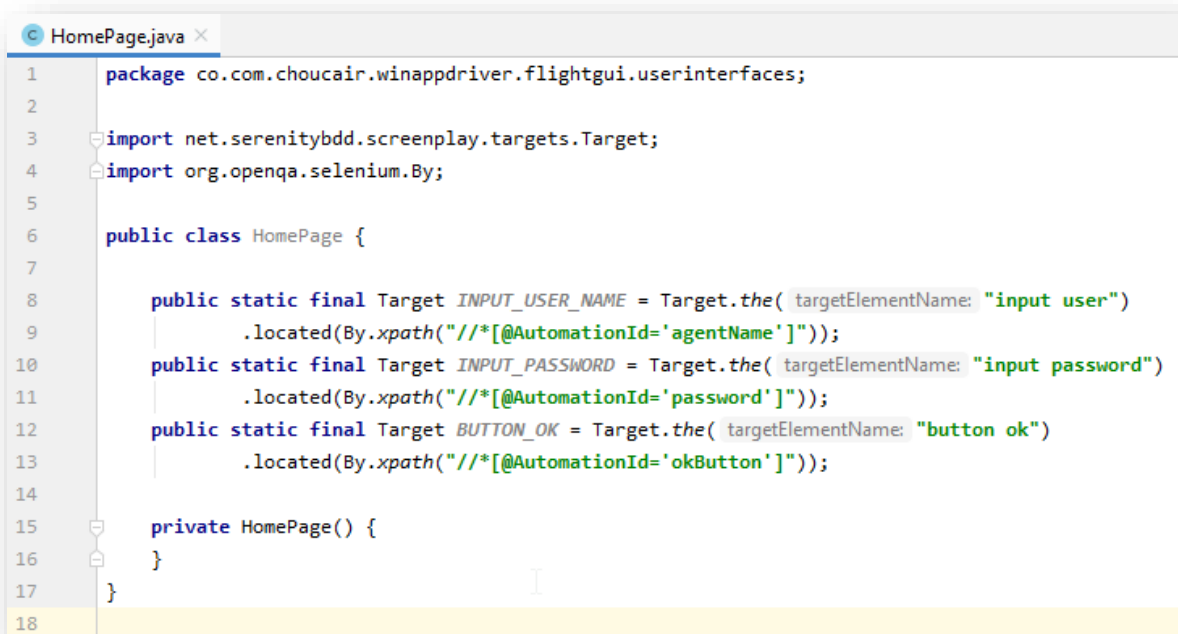
### Creación de user interface

Iremos a la ruta **src > main > co.com.choucair.winappdriver.flightgui > userinterfaces** da clic derecho > **New > Java Class** y ponemos el nombre de la clase que desee para el ejemplo le llamare "HomePage"



```
1 package co.com.choucair.winappdriver.flightgui.userinterfaces;
2
3 public class HomePage {
4 }
```

En esta clase declararemos los objetos estáticos y crearemos un constructor privado vacío.



```
1 package co.com.choucair.winappdriver.flightgui.userinterfaces;
2
3 import net.serenitybdd.screenplay.targets.Target;
4 import org.openqa.selenium.By;
5
6 public class HomePage {
7
8     public static final Target INPUT_USER_NAME = Target.the( targetElementName: "input user")
9         .located(By.xpath("//*[@AutomationId='agentName']"));
10    public static final Target INPUT_PASSWORD = Target.the( targetElementName: "input password")
11        .located(By.xpath("//*[@AutomationId='password']"));
12    public static final Target BUTTON_OK = Target.the( targetElementName: "button ok")
13        .located(By.xpath("//*[@AutomationId='okButton']"));
14
15    private HomePage() {
16    }
17 }
18
```

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Ahora regresamos a la clase **OpenThe** donde el actor realizara las acciones para poder realizar el inicio de sesión a la aplicación.

```
public class OpenThe implements Task {
    private List<LoginModel> dataSet;

    public OpenThe(List<LoginModel> dataSet) { this.dataSet = dataSet; }

    public static OpenThe flightAppDesktop(List<LoginModel> dataSet) {
        return Tasks.instrumented(OpenThe.class, dataSet);
    }

    @Override
    public <T extends Actor> void performAs(T actor) {
        actor.attemptsTo(
            Enter.theValue(dataSet.get(0).getUser()).into(INPUT_USER_NAME),
            Enter.theValue(dataSet.get(0).getPassword()).into(INPUT_PASSWORD),
            Click.on(BUTTON_OK));
    }
}
```

Sin embargo, debemos tener en cuenta las buenas prácticas de programación por lo que será necesario crear una clase en el paquete *utils* a la cual llamaremos *Constants* y en ella declararemos todas las constantes que sean necesarias en nuestra automatización.

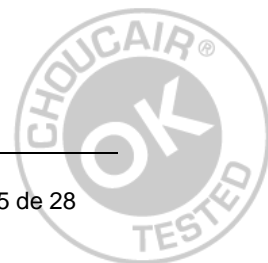
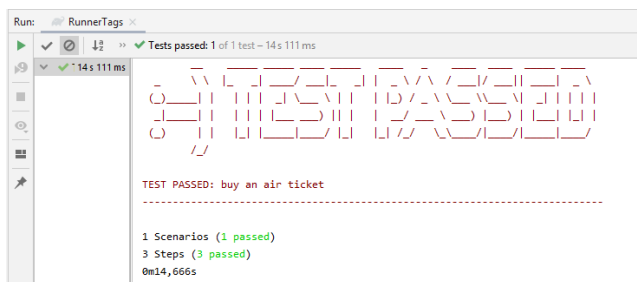
```
Constants.java x
1 package co.com.choucair.winappdriver.flightgui.utils;
2
3 public class Constants {
4     public static final int ZERO = 0;
5 }
```

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Nuestra clase finalmente quedara así:

```
OpenThe.java
1 package co.com.choucair.winappdriver.flightgui.tasks;
2 import co.com.choucair.winappdriver.flightgui.models.LoginModel;
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5 import net.serenitybdd.screenplay.Tasks;
6 import net.serenitybdd.screenplay.actions.Click;
7 import net.serenitybdd.screenplay.actions.Enter;
8 import java.util.List;
9 import static co.com.choucair.winappdriver.flightgui.userinterfaces.HomePage.*;
10 import static co.com.choucair.winappdriver.flightgui.utils.Constants.ZERO;
11
12 public class OpenThe implements Task {
13     private List<LoginModel> dataSet;
14     public OpenThe(List<LoginModel> dataSet) { this.dataSet = dataSet; }
15
16     public static OpenThe flightAppDesktop(List<LoginModel> dataSet) {
17         return Tasks.instrumented(OpenThe.class, dataSet);
18     }
19
20     @Override
21     public <T extends Actor> void performAs(T actor) {
22         actor.attemptsTo(
23             Enter.theValue(dataSet.get(ZERO).getUser()).into(INPUT_USER_NAME),
24             Enter.theValue(dataSet.get(ZERO).getPassword()).into(INPUT_PASSWORD),
25             Click.on(BUTTON_OK));
26     }
27 }
```

En este punto puedes ejecutar el proyecto y verificar que si este realizando el inicio de sesión





## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

### Implementar @When

Vayamos a nuestra clase “**FlightGuiStepDefinitions**” y empecemos la implementación del paso:

```
@When("^he enters the data to buy the ticket$")
```

Dicha implementación la haremos en el método llamado:

```
public void heEntersTheDataToBuyTheTicket(DataTable arg1) {...}
```

Sin embargo, modificaremos el método así:

```
public void heEntersTheDataToBuyTheTicket(List<BookFlightModel> dataSet) {...}
```

Como lo hicimos anterior mente vamos a crear la clase *BookFlightModel* en el paquete *models* y declaramos todos los atributos y sus getters.



```
BookFlightModel.java x
1 package co.com.choucair.winappdriver.flightgui.models;
2
3 public class BookFlightModel {
4     private String fromCity;
5     private String toCity;
6     private String date;
7     private String classFlight;
8     private String tickets;
9     private String passengerName;
10
11     public String getFromCity() { return fromCity; }
12
13
14     public String getToCity() { return toCity; }
15
16
17     public String getDate() { return date; }
18
19
20     public String getClassFlight() { return classFlight; }
21
22
23     public String getTickets() { return tickets; }
24
25
26     public String getPassengerName() { return passengerName; }
27
28 }
29
```



## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Ahora continuamos con la implementación del paso @When.

```
@When("^he enters the data to buy the ticket$")
public void heEntersTheDataToBuyTheTicket(List<BookFlightModel> dataSet) {
    OnStage.theActorInTheSpotlight().attemptsTo(BookFlight.with(dataSet));
}
```

Para este paso vas a llamar al actor con el método *theActorInTheSpotlight()* y ejecutaremos la tarea con el método **attemptsTo()**, dentro de este método crearemos la clase **BookFlight** en el paquete **tasks** y el método complemento de la clase lo llamaremos *with*.

```
BookFlight.java
1 package co.com.choucair.winappdriver.flightgui.tasks;
2
3 import co.com.choucair.winappdriver.flightgui.models.BookFlightModel;
4 import net.serenitybdd.screenplay.Actor;
5 import net.serenitybdd.screenplay.Task;
6 import net.serenitybdd.screenplay.Tasks;
7
8 import java.util.List;
9
10 public class BookFlight implements Task {
11     private List<BookFlightModel> dataSet;
12
13     public BookFlight(List<BookFlightModel> dataSet) {
14         this.dataSet = dataSet;
15     }
16
17     public static BookFlight with(List<BookFlightModel> dataSet) {
18         return Tasks.instrumented(BookFlight.class, dataSet);
19     }
20
21     @Override
22     public <T extends Actor> void performAs(T actor) {
23
24     }
25 }
```

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

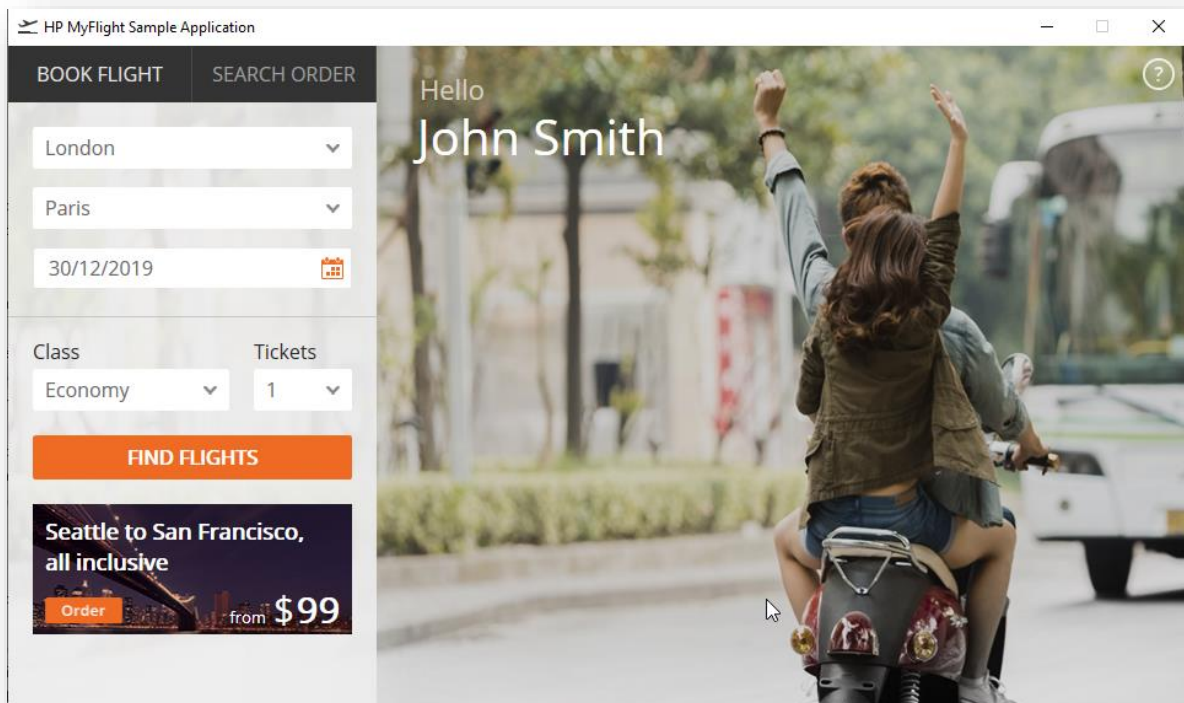
Crea otra clase en el paquete userinterface la cual tendrá el resto del mapeo de los objetos de aplicativo Flight GUI por ejemplo *BookFlightPage*

```
BookFlightPage.java x
6 public class BookFlightPage {
7     public static final Target SELECT_FROM_CITY = Target.the( targetElementName: "select from city")
8         .locatedBy("//*[@AutomationId='fromCity']");
9     public static final Target SELECT_TO_CITY = Target.the( targetElementName: "select to city")
10        .locatedBy("//*[@AutomationId='toCity']");
11     public static final Target SELECT_DATE = Target.the( targetElementName: "select to date")
12        .locatedBy("//*[@AutomationId='PART_TextBox']");
13     public static final Target SELECT_CLASS = Target.the( targetElementName: "select to class")
14        .locatedBy("//*[@AutomationId='Class']");
15     public static final Target SELECT_TICKETS = Target.the( targetElementName: "select to tickets")
16        .located(By.xpath("//*[@AutomationId='numOfTickets']"));
17     public static final Target BUTTON = Target.the( targetElementName: "button find flight")
18        .locatedBy("//*[@Name='FIND FLIGHTS']");
19     public static final Target SELECT_FLIGHT = Target.the( targetElementName: "select to flight")
20        .locatedBy("//*[@Name='FlightsGUI.Logic.GuiFlight']");
21     public static final Target BUTTON_FLIGHT = Target.the( targetElementName: "button flight")
22        .locatedBy("//*[@AutomationId='selectFlightBtn']");
23     public static final Target PASSENGER_NAME = Target.the( targetElementName: "input passenger name")
24        .locatedBy("//*[@AutomationId='passengerName']");
25     public static final Target BUTTON_ORDER = Target.the( targetElementName: "button order")
26        .locatedBy("//*[@AutomationId='orderBtn']");
27     public static final Target VALIDATION_TEXT = Target.the( targetElementName: "text of validation")
28        .locatedBy("//*[@AutomationId='orderCompleted']");
29
30     private BookFlightPage() {
31     }
32 }
```

BookFlightPage > BookFlightPage()

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Luego que la automatización realiza el inicio de sesión nos muestra una nueva ventana, por ende, es necesario cambiar de ventana y para lograr esto es necesario crear una interacción debido a que serenity no nos proporciona una clase para cambiar de window. Y si no cambiamos de ventana no podemos encontrar los objetos que se encuentran en esta.



## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Para crear la interacción vamos a crear una clase llamada *SwitchTo* en el paquete *interactions* como se logra apreciar en la siguiente imagen la cual usara el actor en la clase *BookFlight*.

```
SwitchTo.java x
1 package co.com.choucair.winappdriver.flightgui.interactions;
2 import ...
8
9 /**
10  * <h1>Switch Window(NavTab)</h1>
11  * <p>Esta clase se diseño para cambiar de ventana (switch NavTab)</p>
12  *
13  * @version: 1.0
14  * @author: bquevedof (Brandon Quevedo Funez)
15  * @version: 1.0
16  * @since: 10/12/2019
17  */
18 public class SwitchTo {
19     private SwitchTo() {
20     }
21     public static NavTab window() { return Tasks.instrumented(NavTab.class); }
24 }
25
26 class NavTab implements Interaction {
27     @Override
28     public <T extends Actor> void performAs(T actor) {
29         Set<String> allWindows = Serenity.getWebdriverManager().getCurrentDriver().getWindowHandles();
30         for (String handle : allWindows) {
31             Serenity.getWebdriverManager().getCurrentDriver().switchTo().window(handle);
32         }
33     }
34 }
35 }
```

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Vuelve a la clase *BookFlight* y crea las acciones que utilizara el actor incluyendo la interacción que acabamos de realizar, tu clase se debe ver asi:

```
BookFlight.java
1  package co.com.choucair.winappdriver.flightgui.tasks;
2  import ...
15
16  public class BookFlight implements Task {
17      private List<BookFlightModel> dataSet;
18
19      public BookFlight(List<BookFlightModel> dataSet) { this.dataSet = dataSet; }
22
23      public static BookFlight with(List<BookFlightModel> dataSet) {
24          return Tasks.instrumented(BookFlight.class, dataSet);
25      }
26
27      @Override
28      public <T extends Actor> void performAs(T actor) {
29          actor.attemptsTo(
30              SwitchTo.window(),
31              Enter.theValue(dataSet.get(ZERO).getFromCity()).into(SELECT_FROM_CITY),
32              Enter.theValue(dataSet.get(ZERO).getToCity()).into(SELECT_TO_CITY),
33              Enter.theValue(dataSet.get(ZERO).getDate()).into(SELECT_DATE),
34              Enter.theValue(dataSet.get(ZERO).getClassFlight()).into(SELECT_CLASS),
35              Enter.theValue(dataSet.get(ZERO).getTickets()).into(SELECT_TICKETS),
36              Click.on(BUTTON),
37              Click.on(SELECT_FLIGHT),
38              Click.on(BUTTON_FLIGHT),
39              Enter.theValue(dataSet.get(ZERO).getPassengerName()).into(PASSENGER_NAME),
40              Click.on(BUTTON_ORDER));
41      }
42  }
43
```

BookFlight > with()

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

### Implementar @Then

Una vez hayamos realizado nuestras tareas anteriores, terminaremos realizando la verificación de que toda nuestra historia de usuario salió conforme a lo esperado. En esta guía buscaremos darle implementación a la siguiente instrucción en nuestra historia de usuario.

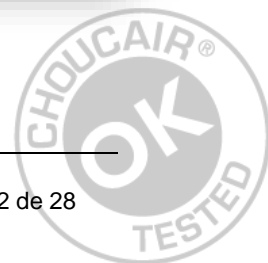
```
@Then("^he verifies the purchase with the message completed$")  
public void heVerifiesThePurchaseWithTheMessageCompleted() {...}
```

En el método, es donde estaremos trabajando la implementación para nuestra verificación en el "Question". Para convertirlo en algo más dinámico, entonces usaremos como en la guía pasada las expresiones regulares pertinentes. Reemplazando la palabra "completed" por "(.\*)" y declarando un parámetro String, tendremos lo siguiente:

```
@Then("^he verifies the purchase with the message (.*)$")  
public void heVerifiesThePurchaseWithTheMessageCompleted(String question) {  
}
```

Para escribir la pregunta, usaremos el método "should" de nuestro Actor. Dentro escribiremos entonces *GivenWhenThen* seguido de un punto y escogeremos el método *seeThat()*. Y dentro del método crearemos la clase *VerifyWith* dicha clase la crearemos dentro del paquete "questions". Y el método "the", será un método estático de dicha clase.

```
@Then("^he verifies the purchase with the message (.*)$")  
public void heVerifiesThePurchaseWithTheMessageCompleted(String question) {  
    OnStage.theActorInTheSpotLight().should(GivenWhenThen.seeThat(VerifyWith.the(question)));  
}
```



## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Ahora procedemos a crear nuestra pregunta en el paquete que corresponde, verificamos que se haya creado la clase **VerifyWith** y que esté completamente vacía, usaremos una interface llamada Question y como si fuese del tipo List, le definiremos el tipo que deseamos tener como respuesta, int, String, double, etc... En este caso, será del tipo Boolean. Luego volvemos al stepdefinicion y se debe crear el método estático *the*, que debe ser del tipo de la clase. Además, creamos un atributo de tipo private con su respectivo constructor.

```
VerifyWith.java x
1 package co.com.choucair.winappdriver.flightgui.questions;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Question;
5
6 public class VerifyWith implements Question<Boolean> {
7     private String question;
8
9     public VerifyWith(String question) {
10         this.question = question;
11     }
12
13     @ public static VerifyWith the(String question) {
14         return new VerifyWith(question);
15     }
16
17     @Override
18     public Boolean answeredBy(Actor actor) {
19         return null;
20     }
21 }
```





## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Ahora procedemos a escribir la línea que tomará el texto para la validación, desde el aplicativo Flight GUI, inspeccionado en las guías anteriores y se lo asignaremos a una variable de tipo String. En cuanto a la validación, toda “la magia” la haremos en el método llamado “answeredBy”. Para obtener el String usaremos la sentencia “Text.of”, le pasaremos nuestro objeto tipo Target, y le diremos que es visto por nuestro actor como un String.

```
@Override
public Boolean answeredBy(Actor actor) {
    String answer = Text.of(VALIDATION_TEXT).viewedBy(actor).asString();
```

Y por último validaremos que el texto obtenido contenga la palabra *completed*.

```
VerifyWith.java
1 package co.com.choucair.winappdriver.flightgui.questions;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Question;
5 import net.serenitybdd.screenplay.questions.Text;
6
7 import static co.com.choucair.winappdriver.flightgui.userinterfaces.BookFlightPage.VALIDATION_TEXT;
8
9 public class VerifyWith implements Question<Boolean> {
10     private String question;
11
12     public VerifyWith(String question) {
13         this.question = question;
14     }
15
16     @ public static VerifyWith the(String question) {
17         return new VerifyWith(question);
18     }
19
20     @Override
21     public Boolean answeredBy(Actor actor) {
22         String answer = Text.of(VALIDATION_TEXT).viewedBy(actor).asString();
23         return question.contains(answer);
24     }
25 }
```

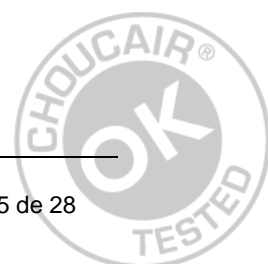




## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Ahora volvemos a la clase *FlightGuiStepDefinitions* y debe verse así:

```
FlightGuiStepDefinitions.java
9  import cucumber.api.java.en.Then;
10 import cucumber.api.java.en.When;
11 import net.serenitybdd.screenplay.GivenWhenThen;
12 import net.serenitybdd.screenplay.actors.OnStage;
13
14 import java.util.List;
15
16 public class FlightGuiStepDefinitions {
17
18     @Given("^that (.*) wants to enter the flight gui desktop application$")
19     public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(String brandon, List<LoginModel> dataSet) {
20         OnStage.theActorCalled(brandon).wasAbleTo(OpenThe.flightAppDesktop(dataSet));
21     }
22
23     @When("^he enters the data to buy the ticket$")
24     public void heEntersTheDataToBuyTheTicket(List<BookFlightModel> dataSet) {
25         OnStage.theActorInTheSpotLight().attemptsTo(BookFlight.with(dataSet));
26     }
27
28     @Then("^he verifies the purchase with the message (.*)$")
29     public void heVerifiesThePurchaseWithTheMessageCompleted(String question) {
30         OnStage.theActorInTheSpotLight().should(GivenWhenThen.seeThat(VerifyWith.the(question)));
31     }
32 }
```



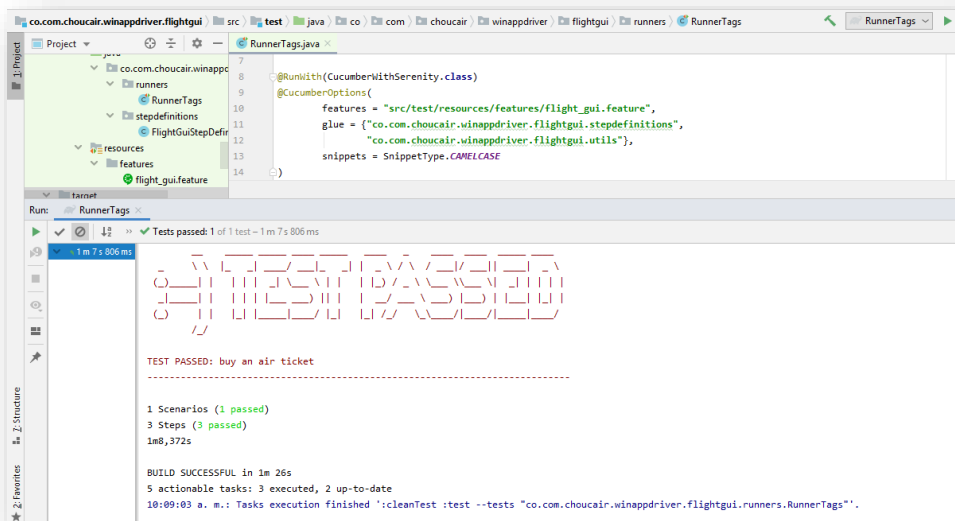
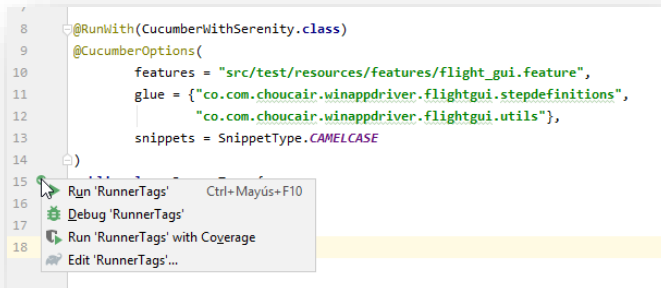
## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Sin embargo, vamos a declarar unos import static con el fin de tener un stepdefinitions más limpio.

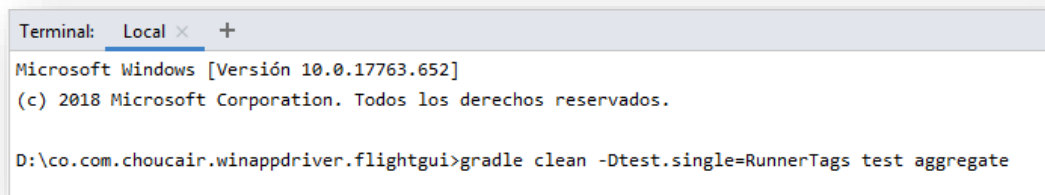
```
FlightGuiStepDefinitions.java
3  import co.com.choucair.winappdriver.flightgui.models.BookFlightModel;
4  import co.com.choucair.winappdriver.flightgui.models.LoginModel;
5  import co.com.choucair.winappdriver.flightgui.questions.VerifyWith;
6  import co.com.choucair.winappdriver.flightgui.tasks.BookFlight;
7  import co.com.choucair.winappdriver.flightgui.tasks.OpenThe;
8  import cucumber.api.java.en.Given;
9  import cucumber.api.java.en.Then;
10 import cucumber.api.java.en.When;
11 import java.util.List;
12
13 import static net.serenitybdd.screenplay.GivenWhenThen.seeThat;
14 import static net.serenitybdd.screenplay.actors.OnStage.theActorCalled;
15 import static net.serenitybdd.screenplay.actors.OnStage.theActorInTheSpotLight;
16
17 public class FlightGuiStepDefinitions {
18
19     @Given("^that (.*) wants to enter the flight gui desktop application$")
20     public void thatBrandonWantsToEnterTheFlightGuiDesktopApplication(String brandon, List<LoginModel> dataSet) {
21         theActorCalled(brandon).wasAbleTo(OpenThe.flightAppDesktop(dataSet));
22     }
23
24     @When("^he enters the data to buy the ticket$")
25     public void heEntersTheDataToBuyTheTicket(List<BookFlightModel> dataSet) {
26         theActorInTheSpotLight().attemptsTo(BookFlight.with(dataSet));
27     }
28
29     @Then("^he verifies the purchase with the message (.*)$")
30     public void heVerifiesThePurchaseWithTheMessageCompleted(String question) {
31         theActorInTheSpotLight().should(seeThat(VerifyWith.the(question)));
32     }
33 }
```

## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Volvemos a la clase RunnerTags y ejecutamos nuestra automatización

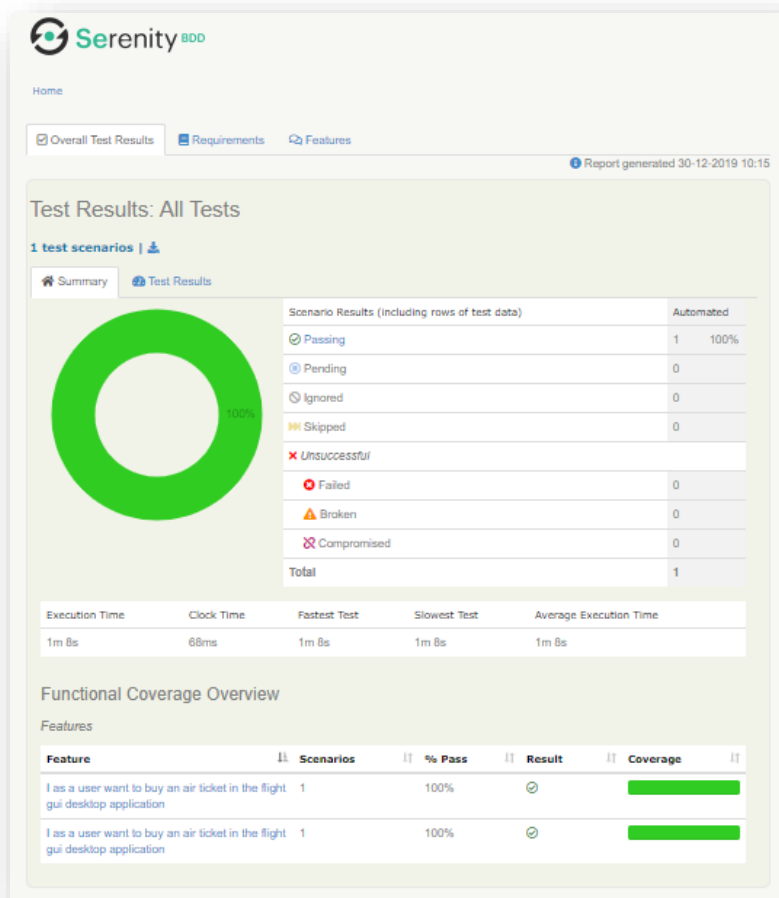


Ahora para ejecutarlo por terminal de gradle hacemos uso de comando en la consola



## ScreenPlay +BDD+WinAppDriver 3 - Implementación de (@Given, @When,@Then)

Cuando termine dirígete a la ruta `co.com.choucair.winappdriver.flightgui/target/site/serenity/index.html` y abre el archivo con un navegador por ejemplo Chrome, para ver el reporte de serenity



**¡Felicidades ya conoces lo básico para automatizar con WinAppDriver usando el patrón de ScreenPlay!**

**¡Ahora solo te falta ampliar tus conocimientos con estudio y práctica!**

