[Solución]
Práctico 13
Waits y Factories

Factory

Ejercicio
Crear una clase llamada **DocusignFactory** y una clase llamada **DocusignFactoryTest**.
DocusignFactoryTest debe acceder al sitio "https://go.docusign.com/o/trial/" y debe tener un método de test que valide que el título de Docusign sea "DocuSign Free Trial"

DocusignFactory debe invocar 3 veces a la clase DocusignFactoryTest creando nuevas instancias de la misma.

```
public class DocusignFactory {

    @Factory
    public Object[] factoryMethod() {
        return new Object[] { new DocusignFactoryTest(),
                new DocusignFactoryTest(),
                new DocusignFactoryTest() };
    }
}
```

Ejercicio
Crear una clase llamada **ShopifyTest** y una clase llamada **ShopifyFactory**.
ShopifyTest debe tener una variable de clase privada de tipo int
También, debe contar con un constructor que reciba un valor por parámetro, y setee esa variable con ese valor.
La clase **ShopifyTest** debe tener un constructor que reciba y setee ese parámetro. El constructor sin parámetros, debe setear el parámetro en 0.
El @beforeMethod debe acceder a **"https://es.shopify.com/"**
Agregar un método de test llamado testButtons que obtenga todos los botones de la página, e imprima en pantalla el texto del botón que coincide con el parámetro.

```
public ShopifyTest() {
    this.param = 0;

}
public ShopifyTest(int param) {
    this.param = param;
}

@Test
public void testButtons() {
    List<WebElement> webElementList = driver.findElements(By.tagName("button"));

    ArrayList<WebElement> noEmptyButtons = new ArrayList<WebElement>();

    for (WebElement we: webElementList){
        if (we.getText().equals("") == false){
            noEmptyButtons.add(we);
        }
    }

    WebElement btnParam = noEmptyButtons.get(param);

    System.out.println(btnParam.getText());
}
```

La clase **ShopifyFactory**, debe tener un método **@Factory** que instancie la clase **ShopifyTest** de forma tal que cada instancia imprima un botón diferente.

```
@Factory
public Object[] factoryMethod() {

    return new Object[] {
            new ShopifyTest( param: 0),
            new ShopifyTest( param: 1),
            new ShopifyTest( param: 2),
            new ShopifyTest( param: 3)
    };
};
```

Thread Sleep

Ejercicio
Crear un método que acceda a Salesforce y espere 5 segundos antes de hacer click en “Forgot password”.

```
@Test
public void threadTest() throws InterruptedException {
    WebDriver driver = getDriver( url: "salesforce");
    Thread.sleep( millis: 5000);
    String forgotPassword = driver.findElement(By.partialLinkText("Forgot")).getText();
    driver.findElement(By.partialLinkText("Forgot")).click();
}
```

Waits Implícitos

Ejercicio
Crear un método llamado **implicitWaitTest** que acceda a Salesforce y espera hasta 10 segundos antes de tirar una excepción.
Hacer click en Forgot Account

```
@Test
public void implicitWait() {
    WebDriver driver = getDriver( url: "salesforce");
    driver.manage().timeouts().implicitlyWait( l: 100, TimeUnit.SECONDS) ;

    String forgotPassword = driver.findElement(By.partialLinkText("Forgot")).getText();
    driver.findElement(By.partialLinkText("Forgot")).click();
}
```

Waits Explicitos

Ejercicio
Acceder a **https://www.spotify.com/uy/signup/**
Completar el email y la confirmación del email con test@test.com
Esperar que aparezca el mensaje de error utilizando un explicit wait

Validar que el mensaje de error sea “**Este correo electrónico ya está conectado a una cuenta.”**

```
private final String REGISTERED_EMAIL_ERROR_MSG = "Este correo electrónico ya está conectado a una cuenta. Inicia sesión.";

@Test
public void validateExistingEmailErrorTest() {
    driver.findElement(By.xpath("//input[@placeholder='Introduce tu correo electrónico.']")).sendKeys( ...charSequences: "test@test.com");
    driver.findElement(By.xpath("//input[@placeholder='Vuelve a introducir tu correo electrónico.']")).sendKeys( ...charSequences: "test@test.com");

    WebDriverWait wait = new WebDriverWait(driver, timeOutInSeconds: 30);
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//*[contains(text(), 'Este correo electrónico ya está conectado a una cuenta.')]")));

    WebElement emailErrorMessage = driver.findElement(By.xpath("//*[contains(text(), 'Este correo electrónico ya está conectado a una cuenta.')]"));

    Assert.assertEquals(emailErrorMessage.getText(), REGISTERED_EMAIL_ERROR_MSG );
}
```

Parametrización y Runners

Ejercicio

Dentro de la carpeta clase13, crear un archivo testng.xml.

Realizar una clase llamada testParametrizables y un método llamado pruebaConParametros que reciba por parámetro un tipo de tagName. En base a si el parámetro es **h1, h2 y h3** mostrar en pantalla lo que se va a imprimir, y obtener por su tagname todos los elementos. Luego imprimirlos en pantalla. En caso de que no se encuentren elementos, se debe mostrar un mensaje indicando que no se encontraron seleccionados.

```
<parameter name ="tag_name" value="h1"></parameter>
```

```java
@Test
@Parameters ({"tag_name"})
public void printHs (@Optional("a") String tag){
    List<WebElement> elements = driver.findElements(By.tagName(tag));

    System.out.print("Se mostran los elementos: ");
    if (tag.equals("h1")){
        System.out.println("H1");
    } else if (tag.equals("h2")) {
        System.out.println("H2");
    } else if (tag.equals("a")) {
        System.out.println("Hiperlinks");
    } else {
        System.out.println(tag);
    }
```

Dependent tests

Ejercicio
Crear una clase llamada **dependentMethods** que importe utilice la notación *dependsOnMethods* y en el cual el método **testOne**, **testTwo** y un método **testThree.** El método **testOne** depende de que los otros dos métodos finalicen.

```java
@Test(dependsOnMethods = { "testTwo", "testThree" })
public void testOne() {
    System.out.println("Test method one");
}

@Test
public void testTwo() {
    System.out.println("Test method two");
}

@Test
public void testThree() {
    System.out.println("Test method three");
}
```

Ejercicio

Crear una clase llamada **DependentTest**.

Dividir el test en métodos dependientes uno de otros, de forma tal que el último test, valide los mensajes de error desplegado en pantalla al clickear el botón de registrarse. Se debe completar solo los dos primeros campos del formulario de registro y usar la notación: **dependentOnMethods**. Agregar asserts y waits

```
public String URL = "https://www.spotify.com/";
public WebDriver driver;

@BeforeTest
public void setBaseURL(){
   System.setProperty("webdriver.chrome.driver", "drivers/chromedriver");
   driver = new ChromeDriver();
   driver.get(URL);
}

@Test
public void welcomePageTest(){
   driver.get(URL);
   System.out.println(driver.getTitle());
   Assert.assertEquals(driver.getTitle(), "Escuchar es todo - Spotify");
   driver.findElement(By.xpath("//a[@href='https://www.spotify.com/uy/signup/']")).click();
}

@Test (dependsOnMethods = { "welcomePageTest"})
public void titleTest() {
   Assert.assertEquals(driver.getTitle(), "Registrarte - Spotify");
}

@Test (dependsOnMethods = { "titleTest"})
public void fillingFormFieldsTest(){
   driver.findElement(By.xpath("//input[@name='email']")).sendKeys("testing@gmail.com");
   driver.findElement(By.xpath("//input[@name='confirm']")).sendKeys("testing@gmail.com");
   driver.findElement(By.xpath("//input[@name='password']")).sendKeys("testing@gmail.com");
}

@Test (dependsOnMethods = { "fillingFormFieldsTest"})
public void registerClickTest(){

   driver.findElement(By.xpath("//button[@type='submit']")).click();
}
```