



# QA & Testing

## Junior Academy

160 horas

# 2.2 Introducción automatizar pruebas para aplicaciones web

Jonatan Villén / Rubén García

## Introducción a Selenium

- Arquitectura y comandos Selenium WebDriver
- Introducción a XPath y CSS Selector
- Comandos WebDriver
- Switches Alerts and Windows
- Action Class
- JUnit en Selenium.

## Buenas Prácticas

- Patrón Page Object Model
- Reusabilidad
- BDD - Cucumber
- Reporting



# Buenas Prácticas

## Patrones de diseño

### ¿Qué es un patrón?

- Un patrón es una técnica para resolver problemas comunes en el desarrollo del software.
- Solución a un problema de diseño, ¿Beneficios?

# Buenas Prácticas

## Patrones de diseño

### ¿Qué es un patrón?

- Un patrón es una técnica para resolver problemas comunes en el desarrollo del software.
- Solución a un problema de diseño, ¿Beneficios?
  - Solución común
  - Problemas resueltos anteriormente
  - Reutilizable
  - Formalizar un “vocabulario común”
  - Facilitar aprendizaje
  - Estandarizar
- Asimismo, no pretenden:
  - Imponer ciertas alternativas de diseño frente a otras.
  - Eliminar la creatividad inherente al proceso de diseño.

# Buenas Prácticas

## Patrones de diseño

### Ejemplos:

- Patrones estructurales:
  - Adapter o Wrapper
  - Bridge
  - Composite
  - Facade
  - ...
- Patrones de comportamiento
  - Observer
  - State
  - Strategy
  - ...





# Buenas Prácticas

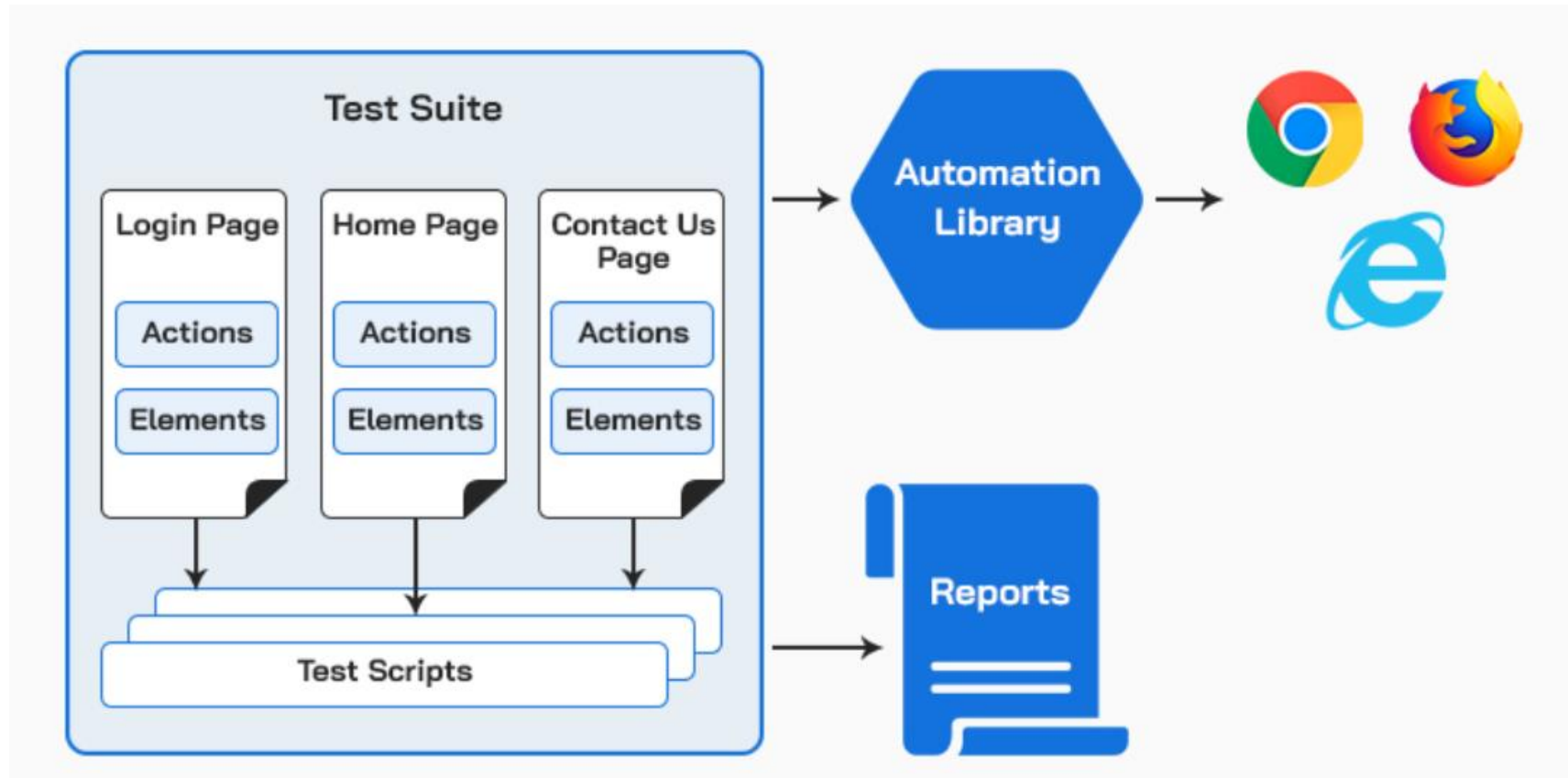
## Patrón Page Object Model

### **Patrón Page Object Model (POM)**

- Patrón de diseño usado en la automatización de pruebas
- Genera un repositorio de objetos con los elementos y acciones de cada página web
- Reduce la duplicación de código (código reusable)
- Mejora el mantenimiento de las pruebas
- Robustez en las pruebas

# Buenas Prácticas

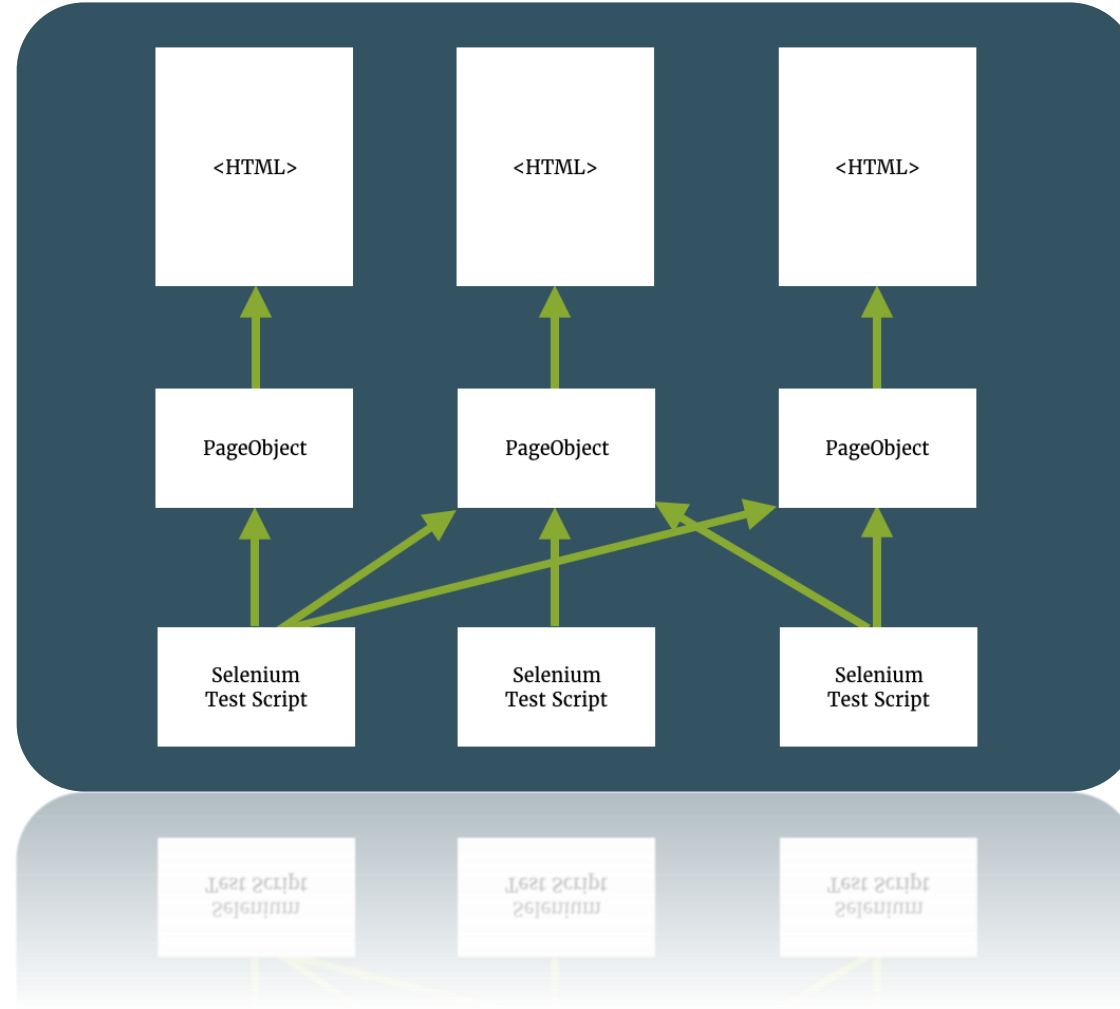
## Patrón Page Object Model





# Buenas Prácticas

## Patrón Page Object Model



# Buenas Prácticas

## Patrón Page Object Model

### Patrón Page Factory

- Se utiliza con el patrón Page Object Model
- Mayor mantenibilidad de código
- Código uniforme y estandarizado
- Gestión del repositorio de objetos
- Permite gestionar todas la páginas del patrón page object model
  - Inicializar las páginas y sus elementos de forma única
  - Instanciar las páginas
  - Acceder a componentes
  - Utilizar métodos de las páginas
  - Uso de la anotación **@FindBy** (lo veremos más adelante)

## ¿Comenzamos?

<https://demoqa.com/books>

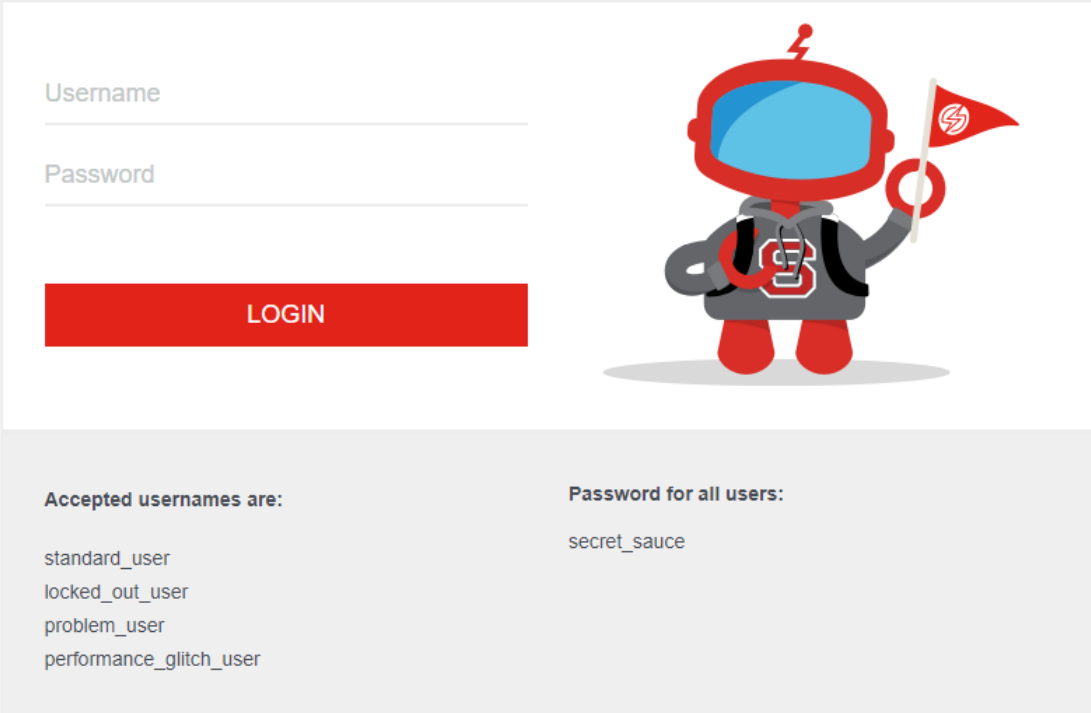
<https://www.saucedemo.com/>

# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 1

### Ejemplo “Login”

- Elementos
  - 
  - 
  -
- Acciones
  -



A login form for SWAGLABS. It features a red robot character on the right holding a flag. The form has two input fields: "Username" and "Password". Below the fields is a red "LOGIN" button. At the bottom, there is a grey box containing the accepted usernames and the password for all users.

Username

Password

LOGIN

Accepted usernames are:

- standard\_user
- locked\_out\_user
- problem\_user
- performance\_glitch\_user

Password for all users:

secret\_sauce

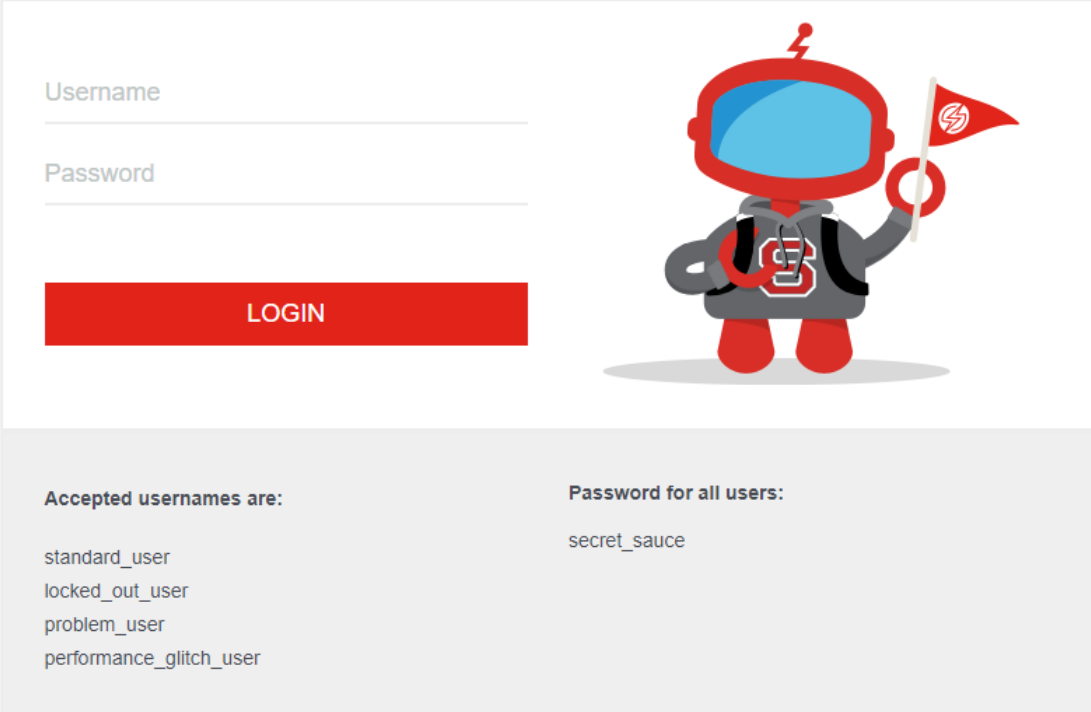
# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 1



### Ejemplo “Login”

- Elementos
  - Introducir usuario: **Username**
  - Introducir contraseña: **Password**
  - Botón de acceso: “**Login**”
- Acciones
  - Hacer login



A login form for SWAGLABS. It features a red robot character on the right holding a flag. The form has two input fields: "Username" and "Password", both with red borders. Below the fields is a red "LOGIN" button. At the bottom, there is a grey section with two columns of text: "Accepted usernames are:" followed by a list of usernames, and "Password for all users:" followed by the password "secret\_sauce".

Accepted usernames are:	Password for all users:
standard_user	secret_sauce
locked_out_user	
problem_user	
performance_glitch_user	

# Buenas Prácticas

## Patrón Page Object Model

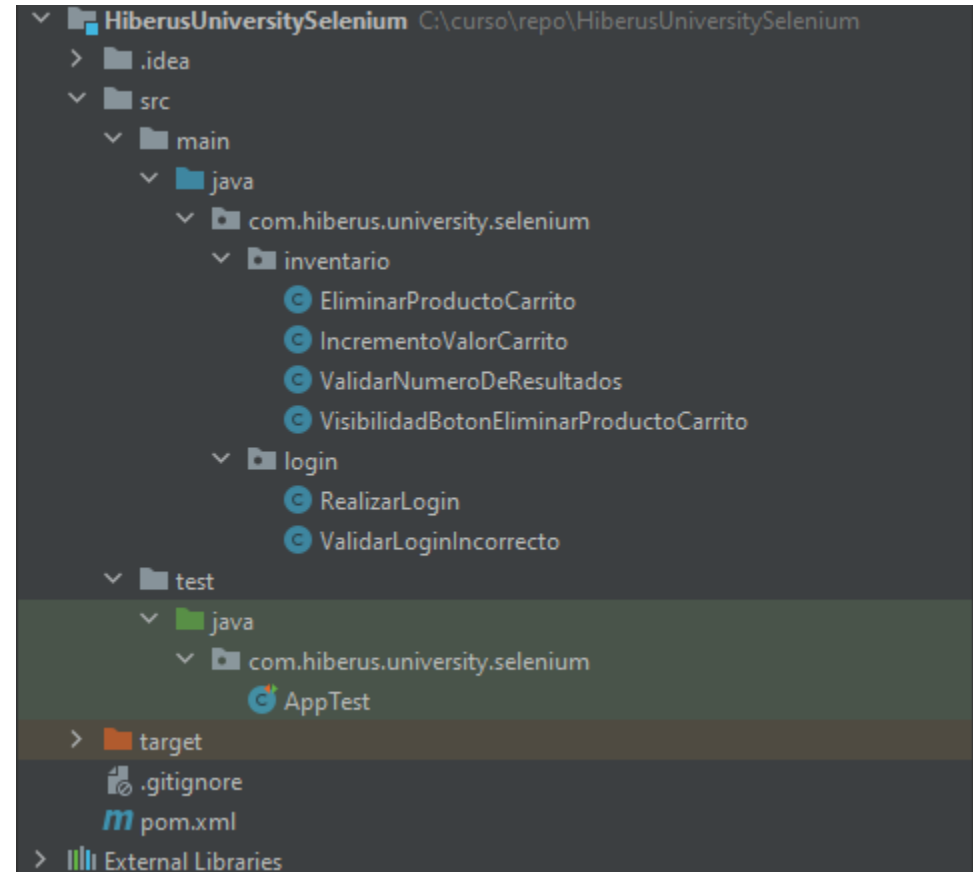
Del Proyecto actual

- **Paso 1: Definir las páginas y sus elementos**
- Paso 2: Implementar una estructura común
- Paso 3: Implementar cada página con sus elementos y métodos

### Ejercicio 1 (1 hora)

Definir todas las páginas de  
<https://www.saucedemo.com/>

con sus elementos y acciones posibles



## Patrón Page Object Model – Ejercicio 1

- **CartPage**

- Elementos:
  - checkoutButton
  - removeButton
  - continueShoppingButton
  - openMenu
  - itemsList
- Métodos
  - clickCheckout()
  - getItemCount()
  - clickContinueShopping()

- **CheckOutStepOnePage**

- Elementos:
  - firstNameInput
  - lastNameInput
  - postalCodeInput
  - ContinueButton
- Métodos
  - enterFirstName()
  - enterLastName()
  - enterPostalCode()
  - clickContinue()

- **CheckOutStepSecondPage**

- Elementos:
  - itemTotalElement
  - taxElement
  - totalElement
- Métodos
  - getItemTotal()
  - getTax()
  - getTotal()



# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 1

- **InventoryPage**

- Elementos:
  - openMenu
  - shoppingCartElement
  - inventoryContainerElement
  - productSortContainerSelect
- Métodos
  - addItemToCartByName()
  - removeItemByName()
  - clickOnShoppingCart()
  - sortInventoryByNameAsc()
  - sortInventoryByNameDesc()
  - sortInventoryByPriceAsc()
  - sortInventoryByPriceDesc()

- **InventoryItemPage**

- Elementos:
  - openMenu
  - itemName
  - itemDescription
  - addToCartButton
  - removeButton
  - itemPrice
  - shoppingCartElement
  - backToProductsButton
- Métodos:
  - addToCart()
  - removeToCart()
  - checkImage()

- **LoginPage**

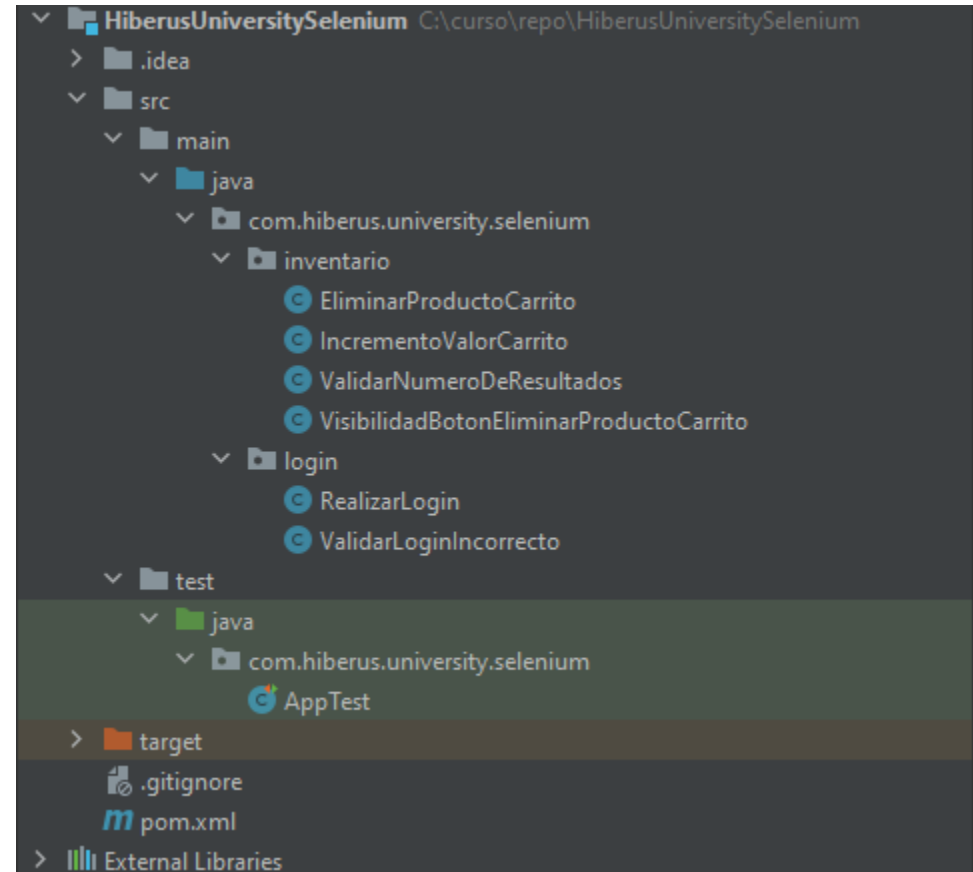
- Elementos:
  - usernameInput
  - passwordInput
  - loginButton
- Métodos:
  - clickLogin()
  - enterUsername()
  - enterPassword()
  - hasLockedOutError()
  - hasErrorLogin()

# Buenas Prácticas

## Patrón Page Object Model

Del Proyecto actual

- Paso 1: Definir las páginas y sus elementos
- **Paso 2: Implementar una estructura común**
- Paso 3: Implementar cada página con sus elementos y métodos

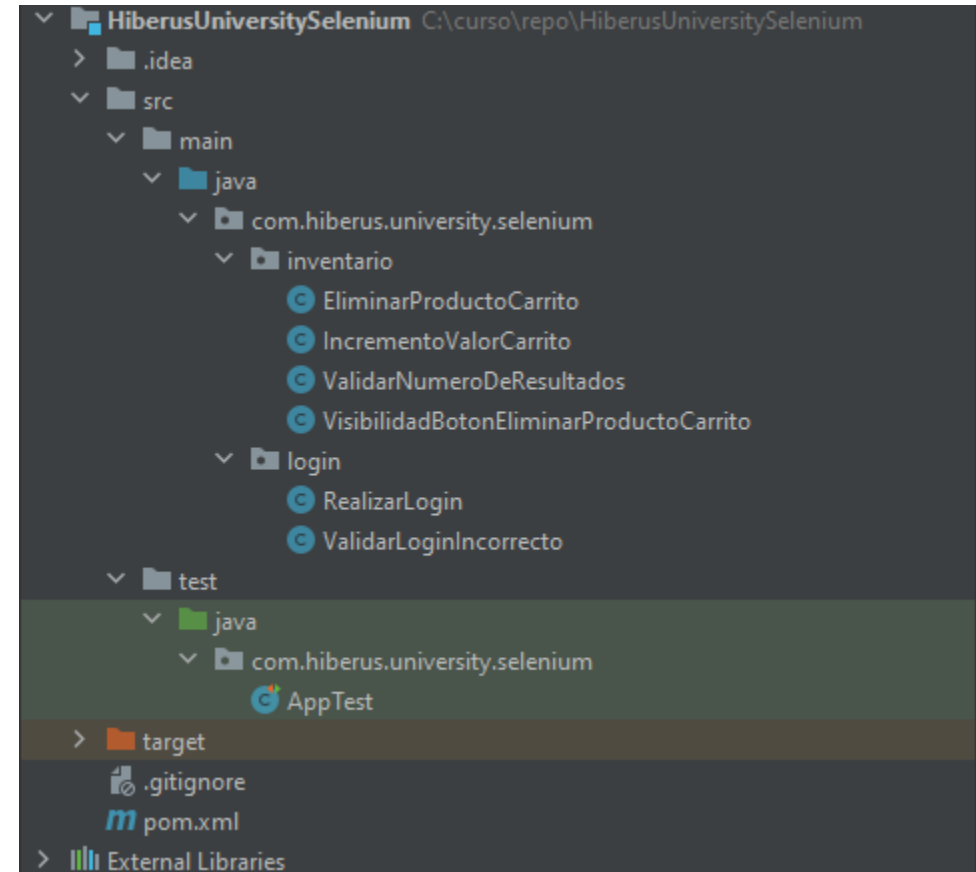


# Buenas Prácticas

## Patrón Page Object Model – Paso 2

Paso 2:

- Refactorizar proyecto actual
- Crear nuevos paquetes
- Crear nuevas clases



# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 2

- Creación de los paquetes necesarios:
  - model
  - pages
  - stepdefs
  - utils
- Creación clase InventoryItem:
  - Nombre del item
  - Descripción del item
  - Precio del item

The image shows a screenshot of an IDE with two windows. The top window displays the project structure of 'HiberusUniversitySelenium'. The 'src/main/java' directory is expanded, showing the 'com.hiberus.university.selenium' package. Inside this package, the 'model' directory is highlighted, and the 'InventoryItem' class is selected. The bottom window shows the code for the 'InventoryItem' class. The code is organized into three sections: 'Atributos' (Attributes), 'Constructores' (Constructors), and 'Getters y Setters' (Getters and Setters). The 'Atributos' section contains three private String fields: 'name', 'description', and 'price'. The 'Constructores' section contains two public constructors: one that takes all three fields as parameters and another that takes only the 'name' parameter. The 'Getters y Setters' section contains six public methods: 'getName()', 'setName()', 'getDescription()', 'setDescription()', 'getPrice()', and 'setPrice()'. The code is as follows:

```
package com.hiberus.university.selenium.model;

public class InventoryItem {

    private String name;
    private String description;
    private String price;

    public InventoryItem(String name, String description, String price) {
        this.name = name;
        this.description = description;
        this.price = price;
    }

    public InventoryItem(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

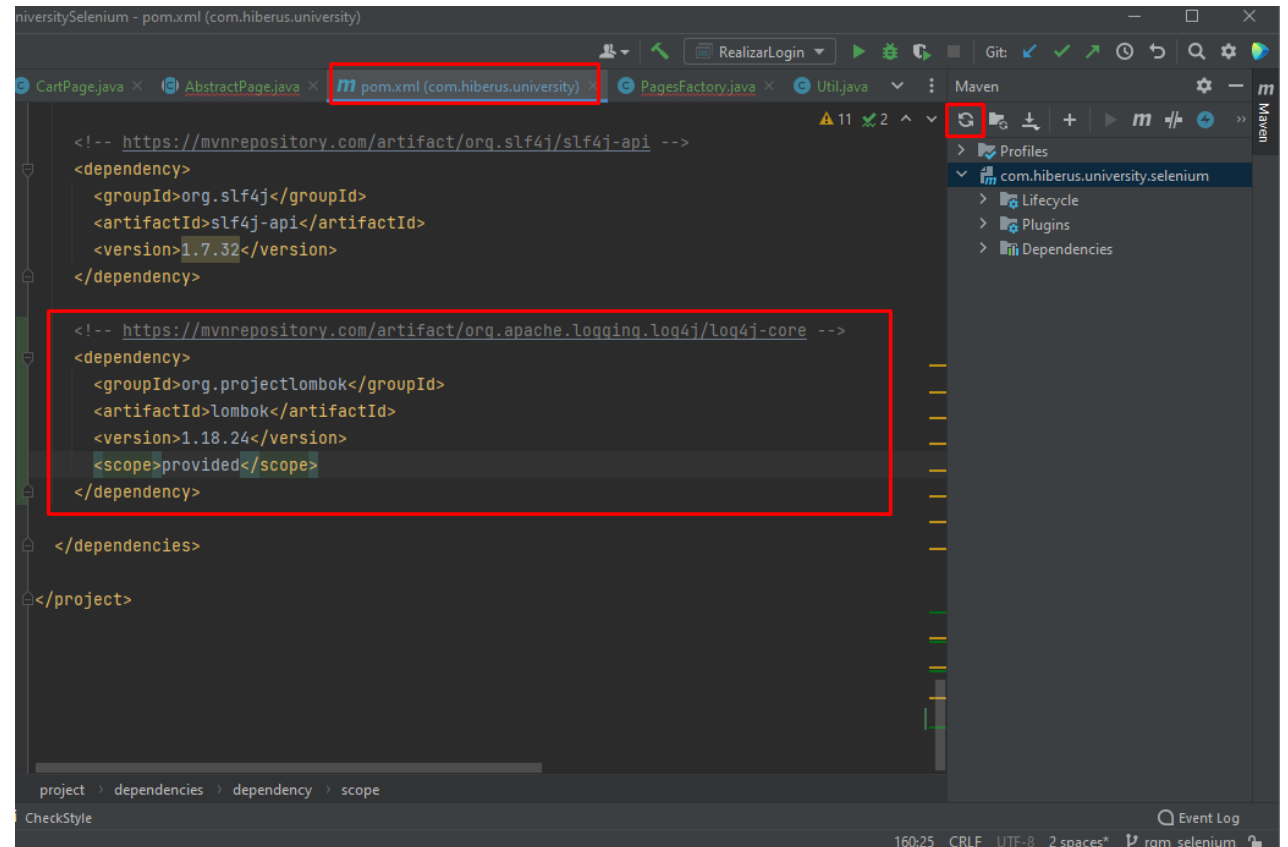
    public String getPrice() {
        return price;
    }

    public void setPrice(String price) {
        this.price = price;
    }
}
```

# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 2

- Añadir dependencia Lombok:
  - <https://projectlombok.org/setup/maven>
  - Facilita el código por anotación
    - Getters
    - Setters
    - Constructores
    - Logs
    - ...
  - Copiar la dependencia en el pom.xml del proyecto
  - Actualizar dependencias



# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 2

- Obtenemos la clase **MyFluentWait.java**:
  - Dentro del paquete util
  - Implements de `Wait<T>`
  - Añadir atributos:
    - timeout
    - interval
  - Crear constructores
  - Métodos:
    - `withTimeout()`
    - `pollingEvery()`
    - `until()`

<https://www.codepile.net/>

```
1 package com.hiberus.university.selenium.util;
2
3 // Licensed to the Software Freedom Conservancy (SFC) under one
4 // or more contributor license agreements. See the NOTICE file
5 // distributed with this work for additional information
6 // regarding copyright ownership. The SFC licenses this file
7 // to you under the Apache License, Version 2.0 (the
8 // "License"); you may not use this file except in compliance
9 // with the License. You may obtain a copy of the License at
10 //
11 // http://www.apache.org/licenses/LICENSE-2.0
12 //
13 // Unless required by applicable law or agreed to in writing,
14 // software distributed under the License is distributed on an
15 // "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
16 // KIND, either express or implied. See the License for the
17 // specific language governing permissions and limitations
18 // under the License.
19
20 import static com.google.common.base.Preconditions.checkNotNull;
21
22 import com.google.common.base.Throwables;
23 import com.google.common.collect.ImmutableList;
24 import com.google.common.collect.Lists;
25 import java.time.Clock;
26 import java.time.Duration;
27 import java.time.temporal.ChronoUnit;
28 import java.time.temporal.TemporalUnit;
29 import java.util.Collection;
30 import java.util.List;
31 import java.util.function.Function;
32 import java.util.function.Supplier;
33 import org.openqa.selenium.TimeoutException;
34 import org.openqa.selenium.WebDriverException;
35 import org.openqa.selenium.support.ui.Sleeper;
36 import org.openqa.selenium.support.ui.Wait;
37
38 /** An implementation of the {@link Wait} interface that may have its timeout and polling interval ... */
39 public class MyFluentWait<T> implements Wait<T>
40 {
41
42     public static final Duration FIVE_HUNDRED_MILLIS = Duration.of( amount: 500, ChronoUnit.MILLIS);
43
44     private final T input;
45     private final Clock clock;
46     private final Sleeper sleeper;
47
48     private Duration timeout = FIVE_HUNDRED_MILLIS;
49     private Duration interval = FIVE_HUNDRED_MILLIS;
50     private Supplier<String> messageSupplier = () -> null;
51
52     private List<Class<? extends Throwable>> ignoredExceptions = Lists.newLinkedList();
53 }
```

# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 2

- Creación clase **AbstractPage.java**:

- Dentro del paquete pages
- Crear clase abstracta
- Añadir atributos:
  - WebDriver
  - Wait<WebDriver
- Crear constructores
- Añadir anotación @Slf4j
- Métodos:
  - moveTo()
  - isPageLoaded()
  - navigateTo()

```
@Slf4j
abstract class AbstractPage {
    private WebDriver driver;
    protected Wait<WebDriver> wait;

    AbstractPage(WebDriver driver) {
        this.driver = driver;
    }

    public abstract WebElement getPageLoadedTestElement();

    protected WebDriver getDriver() {
        return driver;
    }

    protected Wait<WebDriver> getWait() { return wait; }

    protected void setWait(Wait<WebDriver> wait) { this.wait = wait; }

    public void waitForPageLoad() {
        WebElement testElement = getPageLoadedTestElement();
        wait.until(ExpectedConditions.visibilityOf(testElement));
    }

    protected void moveTo(WebElement elem) {
        if (((RemoteWebDriver) driver).getCapabilities().getBrowserName().equals("firefox")) {
            ((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", elem);
        } else {
            Actions actions = new Actions(driver);
            actions.moveToElement(elem).build().perform();
        }
    }

    protected boolean isPageLoaded(WebElement elem) {
        boolean isLoading = false;

        try {
            isLoading = elem.isDisplayed();
        } catch (org.openqa.selenium.NoSuchElementException e) {
            e.printStackTrace();
        }

        return isLoading;
    }

    public void navigateTo(String url) {
        WebDriver driver = getDriver();

        try {
            driver.navigate().to(url);
        } catch (java.lang.Exception e) {
            if (e instanceof TimeoutException) {
                log.info("Timeout loading home page");
            } else if (e instanceof ScriptTimeoutException) {
                log.info("Script Timeout loading home page");
            } else {
                log.error(e.getMessage());
            }
        }
    }
}
```



# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 2

- Creación clase **PagesFactory.java**:

- Dentro del paquete pages
- Crear clase pública
- Añadir atributos:
  - WebDriver
  - List<PagesFactory>
- Crear constructores
- Añadir anotación @Slf4j
- Métodos:
  - start()
  - getInstance()
  - getDriver()

```
@Slf4j
abstract class AbstractPage {
    private WebDriver driver;
    protected Wait<WebDriver> wait;

    AbstractPage(WebDriver driver) {
        this.driver = driver;
    }

    public abstract WebElement getPageLoadedTestElement();

    protected WebDriver getDriver() {
        return driver;
    }

    protected Wait<WebDriver> getWait() { return wait; }

    protected void setWait(Wait<WebDriver> wait) { this.wait = wait; }

    public void waitForPageLoad() {
        WebElement testElement = getPageLoadedTestElement();
        wait.until(ExpectedConditions.visibilityOf(testElement));
    }

    protected void moveTo(WebElement elem) {
        if (((RemoteWebDriver) driver).getCapabilities().getBrowserName().equals("firefox")) {
            ((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", elem);
        } else {
            Actions actions = new Actions(driver);
            actions.moveToElement(elem).build().perform();
        }
    }

    protected boolean isPageLoaded(WebElement elem) {
        boolean isLoading = false;

        try {
            isLoading = elem.isDisplayed();
        } catch (org.openqa.selenium.NoSuchElementException e) {
            e.printStackTrace();
        }

        return isLoading;
    }

    public void navigateTo(String url) {
        WebDriver driver = getDriver();

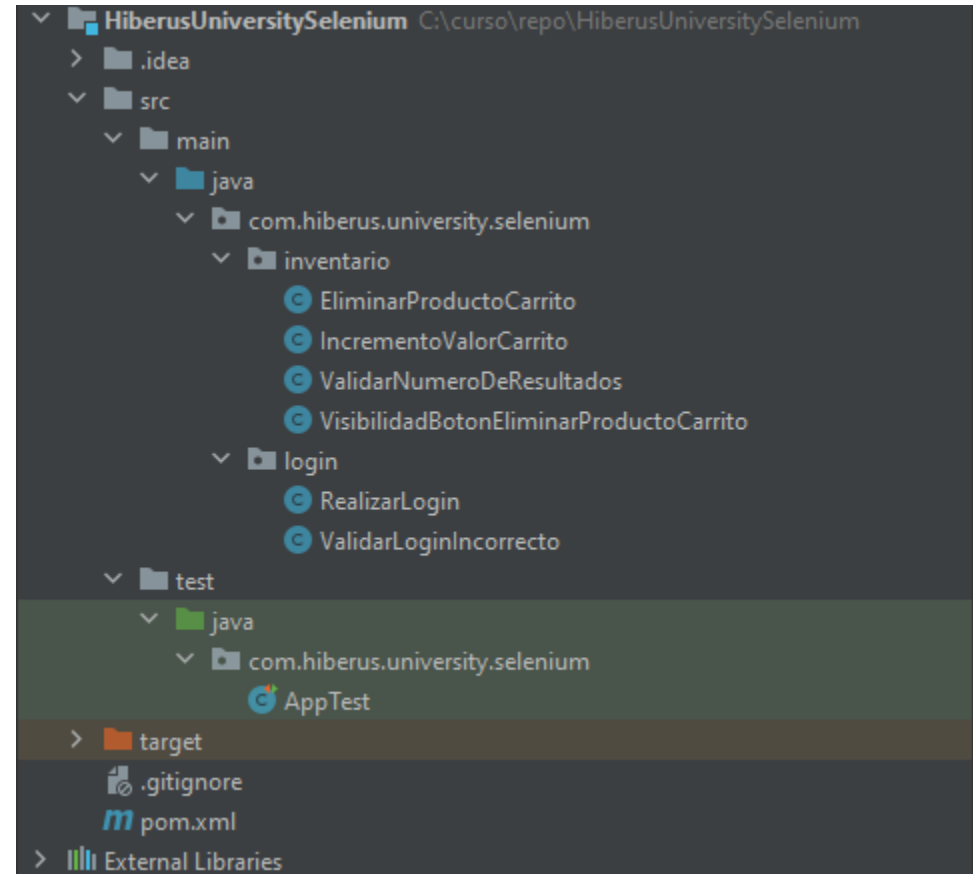
        try {
            driver.navigate().to(url);
        } catch (java.lang.Exception e) {
            if (e instanceof TimeoutException) {
                log.info("Timeout loading home page");
            } else if (e instanceof ScriptTimeoutException) {
                log.info("Script Timeout loading home page");
            } else {
                log.error(e.getMessage());
            }
        }
    }
}
```

# Buenas Prácticas

## Patrón Page Object Model

Del Proyecto actual

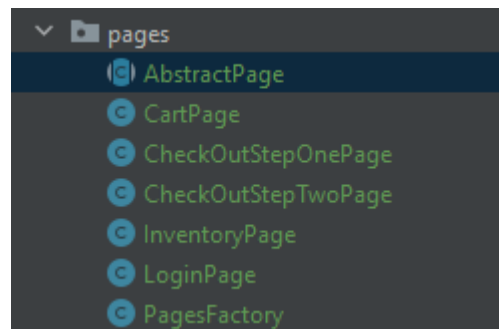
- Paso 1: Definir las páginas y sus elementos
- Paso 2: Implementar una estructura común
- **Paso 3: Implementar cada página con sus elementos y métodos**



# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 2

- Ejemplo LoginPage.java
  - Extiende de AbstractPage.java
  - Constante con la URL de la página
  - Elementos definidos anteriormente
  - Métodos definidos anteriormente



```
@Slf4j
public class LoginPage extends AbstractPage {
    public static final String PAGE_URL = "https://www.saucedemo.com";

    @FindBy(xpath = "//input[@data-test='username']")
    private WebElement usernameElem;

    @FindBy(xpath = "//input[@data-test='password']")
    private WebElement passwordElem;

    @FindBy(xpath = "//input[@value='LOGIN']")
    private WebElement loginElem;

    public LoginPage(WebDriver driver) {
        super(driver);
        PageFactory.initElements(driver, page: this);
    }

    @Override
    public WebElement getPageLoadedTestElement() { return loginElem; }

    public void login() {
        log.info("Logging in...");
        try {
            loginElem.click();
        } catch (org.openqa.selenium.TimeoutException e) {
            log.info("Timeout clicking login: " + e.getClass().getSimpleName());
        } catch (Exception e) {
            log.info("Clicking login, caught exception, type=" + e.getClass().getSimpleName());
        }
    }

    public void clickLogin() { loginElem.click(); }

    public void enterPassword(String password) {
        passwordElem.click();
        passwordElem.sendKeys(password);
    }

    public void enterUsername(String username) {
        usernameElem.click();
        usernameElem.sendKeys(username);
    }

    public boolean hasLockedOutError() {
        WebElement elem = getDriver().findElement(By.xpath("//button[@class='error-button']"));
        return elem.isDisplayed();
    }

    public boolean hasUsernamePasswordError() {
        WebElement elem = getDriver().findElement(By.xpath("//button[@class='error-button']"));
        return elem.isDisplayed();
    }
}
```

# Buenas Prácticas

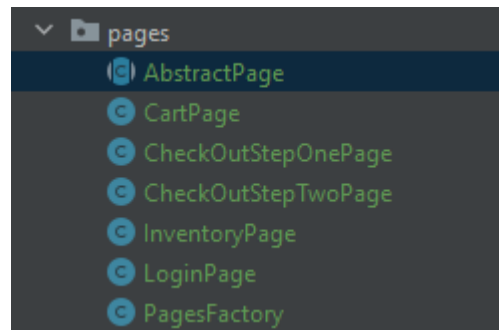
## Patrón Page Object Model – Ejercicio 2

- Ejemplo LoginPage.java
  - Extiende de AbstractPage.java
  - Constante con la URL de la página
  - Elementos definidos anteriormente
  - Métodos definidos anteriormente

### Ejercicio 2 (2 horas)

Creación del resto de clases \*Page.java de la página sauceDemo con los elementos y métodos definidos en el ejercicio anterior:

- CartPage
- CheckoutStepOnePage
- CheckoutStepSecondPage
- InventoryPage
- LoginPage



```
@Slf4j
public class LoginPage extends AbstractPage {
    public static final String PAGE_URL = "https://www.saucedemo.com";

    @FindBy(xpath = "//input[@data-test='username']")
    private WebElement usernameElem;

    @FindBy(xpath = "//input[@data-test='password']")
    private WebElement passwordElem;

    @FindBy(xpath = "//input[@value='LOGIN']")
    private WebElement loginElem;

    public LoginPage(WebDriver driver) {
        super(driver);
        PageFactory.initElements(driver, page: this);
    }

    @Override
    public WebElement getPageLoadedTestElement() { return loginElem; }

    public void login() {
        log.info("Logging in...");
        try {
            loginElem.click();
        } catch (org.openqa.selenium.TimeoutException e) {
            log.info("Timeout clicking login: " + e.getClass().getSimpleName());
        } catch (Exception e) {
            log.info("Clicking login, caught exception, type=" + e.getClass().getSimpleName());
        }
    }

    public void clickLogin() { loginElem.click(); }

    public void enterPassword(String password) {
        passwordElem.click();
        passwordElem.sendKeys(password);
    }

    public void enterUsername(String username) {
        usernameElem.click();
        usernameElem.sendKeys(username);
    }

    public boolean hasLockedOutError() {
        WebElement elem = getDriver().findElement(By.xpath("//button[@class='error-button']"));
        return elem.isDisplayed();
    }

    public boolean hasUsernamePasswordError() {
        WebElement elem = getDriver().findElement(By.xpath("//button[@class='error-button']"));
        return elem.isDisplayed();
    }
}
```

# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 2

- Resolución
  - Rama exercise2

# Buenas Prácticas

## Patrón Page Object Model

- Es el momento de realizar una refactorización del proyecto para su ejecución con el nuevo modelo planteado.
- Para ello:

- Utilización de la factoría de páginas creada:

```
PagesFactory.start(driver);
```

- Esto hace que estén disponibles todas las páginas con sus elementos y métodos en la ejecución de las pruebas
  - Para la utilización de cada una de ellas es necesario obtener la instancia de la factoría y la página que se quiera utilizar, ejemplo:

```
PagesFactory pf = PagesFactory.getInstance();  
LoginPage loginPage = pf.getLoginPage();  
loginPage.enterUsername("standard_user");  
loginPage.enterPassword("secret_sauce");  
loginPage.clickLogin();
```

```
package com.hiberus.university.selenium.login;  
  
import io.github.bonigarcia.wdm.WebDriverManager;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.chrome.ChromeDriver;  
import org.openqa.selenium.chrome.ChromeOptions;  
  
import java.util.concurrent.TimeUnit;  
  
public class RealizarLogin {  
  
    public static WebDriver driver;  
  
    public static void main( String[] args ) throws InterruptedException {  
  
        //Paso0  
        String userProfile= "C:\\\\Users\\pue\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\";  
        WebDriverManager.chromedriver().setup(); // cargar Chromedriver  
        ChromeOptions options = new ChromeOptions(); // Crear instancia para opciones de chrome  
        options.addArguments("user-data-dir=" + userProfile); //  
  
        driver= new ChromeDriver(options);  
        driver.manage().timeouts().implicitlyWait( 30, TimeUnit.SECONDS);  
        driver.manage().window().maximize();  
  
        // Paso1  
  
    }  
}
```

# Buenas Prácticas

## Patrón Page Object Model

- Ejemplo de ejecución del caso de prueba “Realizar Login” del plan de pruebas:
  - El primer paso es inicializar el WebDriverManager
  - Segundo paso, configurar WebDriver
  - Tercer paso, cargar las páginas de la factoría
  - Cuarto paso, incluir los pasos del test
  - Quinto paso, validación
  - Sexto paso, cerrar navegador.

```
package com.hiberus.university.selenium.login;

import com.hiberus.university.selenium.pages.InventoryPage;
import com.hiberus.university.selenium.pages.LoginPage;
import com.hiberus.university.selenium.pages.PagesFactory;
import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;

import java.util.concurrent.TimeUnit;

public class RealizarLogin {

    public static WebDriver driver;

    public static void main(String[] args) {

        //Paso0 1
        WebDriverManager.chromedriver().setup(); 2
        ChromeOptions options = new ChromeOptions();
        driver = new ChromeDriver(options);
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();

        // Paso1 3
        PagesFactory.start(driver);
        driver.get(LoginPage.PAGE_URL);
        PagesFactory pf = PagesFactory.getInstance();
        LoginPage loginPage = pf.getLoginPage();
        loginPage.enterUsername("standard_user");
        loginPage.enterPassword("secret_sauce");
        loginPage.clickLogin();
        if (InventoryPage.PAGE_URL.equals(driver.getCurrentUrl())) {
            System.out.println("Test OK");
        } else {
            System.out.println("Test KO");
        }

        driver.close(); 6
    }
}
```



# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 3

- Ejemplo

### Validar mensaje de Login incorrecto:

1. Ir a la página <https://www.saucedemo.com>
2. Escribir el username [standard](#) (Introducir mal el usuario para forzar el error)
3. Escribir el password [secret\\_sauce](#)
4. Pulsar en el botón del Login
5. Validar que aparece el WebElement del mensaje de error

```
public static void main( String[] args ) {  
    // Step 0  
    WebDriverManager.chromedriver().setup();  
    ChromeOptions options = new ChromeOptions();  
    driver = new ChromeDriver(options);  
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);  
    driver.manage().window().maximize();  
  
    // Step 1  
    PagesFactory.start(driver);  
    driver.get(LoginPage.PAGE_URL);  
    PagesFactory pf = PagesFactory.getInstance();  
    LoginPage loginPage = pf.getLoginPage();  
    // Step 2  
    loginPage.enterUsername("bad_standard_user");  
    // Step 3  
    loginPage.enterPassword("secret_sauce");  
    // Step 4  
    loginPage.clickLogin();  
    // Step 5  
    if (loginPage.hasUsernamePasswordError()) {  
        System.out.println("Test OK");  
    } else {  
        System.out.println("Test KO");  
    }  
    driver.close();  
}
```

# Buenas Prácticas

## Patrón Page Object Model – Ejercicio 3

- Ejemplo de ejecución del caso de prueba “Realizar Login” del plan de pruebas:
  - El primer paso es inicializar el WebDriverManager
  - Segundo paso, configurar WebDriver
  - Tercer paso, cargar las páginas de la factoría
  - Cuarto paso, incluir los pasos del test
  - Quinto paso, validación
  - Sexto paso, cerrar navegador.

### Ejercicio 3 (2 horas)

Refactorización del plan de pruebas con el patrón Page Object Model para su ejecución con el nuevo modelo.

```
package com.hiberus.university.selenium.login;

import com.hiberus.university.selenium.pages.InventoryPage;
import com.hiberus.university.selenium.pages.LoginPage;
import com.hiberus.university.selenium.pages.PagesFactory;
import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;

import java.util.concurrent.TimeUnit;

public class RealizarLogin {

    public static WebDriver driver;

    public static void main(String[] args) {

        //Paso0 1
        WebDriverManager.chromedriver().setup(); 2
        ChromeOptions options = new ChromeOptions();
        driver = new ChromeDriver(options);
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();

        // Paso1 3
        PagesFactory.start(driver);
        driver.get(LoginPage.PAGE_URL);
        PagesFactory pf = PagesFactory.getInstance();
        LoginPage loginPage = pf.getLoginPage();
        loginPage.enterUsername("standard_user");
        loginPage.enterPassword("secret_sauce");
        loginPage.clickLogin();

        if (InventoryPage.PAGE_URL.equals(driver.getCurrentUrl())) {
            System.out.println("Test OK");
        } else {
            System.out.println("Test KO");
        }

        driver.close(); 6
    }
}
```

# Buenas Prácticas

## Patrón Page Object Model

- Resolución
  - Rama exercise3

# Buenas Prácticas

## Patrón Page Object Model y Page Factory

Los patrones de diseño en desarrollo ofrecen una solución previamente utilizada y de efectividad probada, así como la correspondiente documentación, con la que enfrentarse a una serie de problemas comunes dentro del mundo del desarrollo y la automatización de pruebas. Y en concreto, el **Page Object Pattern** y **Page Factory** nos ofrece las ventajas de:

- **Reducir la duplicidad** de código.
- Conseguir pruebas más **robustas** y fácilmente entendibles
- **Mejorar la mantenibilidad** de nuestras pruebas funcionales automatizadas, punto que se pone de relevancia cuando tenemos una web o aplicación que evoluciona rápidamente.
- **Gestión** mas sencilla del código

