# Hoja informativa: Modelo de objetos de página

Para que el código de tus pruebas automatizadas sea más fácil de usar y mantener, puedes utilizar pautas de programación especiales. Estas se conocen como **patrones de diseño**.

Uno de los patrones de diseño es el **Modelo de Objetos de Página** o **POM**.

En este patrón, cada página de una aplicación corresponde a una clase. Los elementos de la página web se convierten en los atributos de esta clase y las interacciones con ellos se convierten en los métodos.

Las clases que contienen localizadores y métodos se llaman **objetos de página (page objects)**.

## ¿Por qué utilizar POM?

El modelo de objetos de página se utiliza para facilitar el mantenimiento del código de prueba, ya que ayuda a separar el lado técnico del código de su lógica. Los beneficios son los siguientes:

- el código se vuelve más legible;
- se facilita más la edición del código;
- las definiciones de localizadores se pueden reutilizar.

## Cómo crear objetos de página

Sigue estos pasos:

1. Crea una clase para la página de inicio de sesión.
2. Define los localizadores de los elementos requeridos como atributos de clase.
3. Agrega el constructor de la clase, es decir, el método `__init__`.
4. Define las interacciones con elementos de página como métodos de clase.

**Ejemplo.** La prueba automatizada debe rellenar los campos Correo electrónico y Contraseña en la página de inicio de sesión. El objeto de página de la página de inicio de sesión se verá así:

```
# Importa webdriver y la clase By para los localizadores
from selenium import webdriver
from selenium.webdriver.common.by import By

# Inicializar el controlador
driver = webdriver.Chrome()

# Paso 1. Declara la clase de objeto de la página
class LoginPageAround:
    # Paso 2. Define el localizador del campo Correo electrónico
    email_field = [By.ID, 'email']
    # Define el localizador del campo Contraseña
    password_field = [By.ID, 'password']

    # Paso 3. Agrega el constructor de clase
    def __init__(self, driver):
        self.driver = driver # Inicializa los atributos

    # Paso 4. Agrega interacciones como métodos
    # El método rellena el campo Correo electrónico
    def set_email(email):
        driver.find_element(*email_field).send_keys(email)

    # El método rellena el campo Contraseña
    def set_password(password):
        driver.find_element(*password_field).send_keys(password)
```

El método `__init__` generalmente se utiliza en objetos de página al principio, antes de las definiciones de métodos. Pasa `self` como primer argumento y el controlador como segundo argumento.

## Localizadores en POM

📌 En POM debes definir solo aquellos elementos que son necesarios para la prueba.

**Para definir un localizador como un atributo de clase, haz lo siguiente:**

1. Declara un atributo de clase, nómbralo igual que el elemento.
2. Asigna una lista o una tupla a la variable.
3. El primer valor es la forma de encontrar el localizador en la página, el segundo valor es el localizador mismo.

Asegúrate de mantener este orden exacto de valores.

**Ejemplo.** Un objeto de página con tres localizadores.

```
from selenium import webdriver
from selenium.webdriver.common.by import By


class LoginPageAround:
    # El localizador del campo Correo electrónico
    email_field = [By.ID, 'email']
    # El localizador del campo Contraseña
    password_field = [By.ID, 'password']
    # El localizador del botón Iniciar sesión
    sign_in_button = [By.CLASS_NAME, 'auth-form__button']
```

**No tienes que almacenar una página completa en el objeto de página.** Puedes definir solo algunos de los elementos de la clase, por ejemplo, el encabezado, el pie de página y el menú de la barra lateral, si son iguales en cada página. Esto se hace para combinar localizadores similares en la misma clase.

## Métodos en POM

En POM, cada elemento se convierte en un atributo de clase, por lo que puedes reutilizarlo en el código. Para hacerlo, debes **desempaquetar** la variable antes de utilizarla en un método.

Los valores de atributo se desempaquetan mediante el operador `*`.

**Ejemplo.** Este es un método para hacer clic en el botón Iniciar sesión.

```
    # El localizador del botón Iniciar sesión
    sign_in_button = [By.CLASS_NAME, 'auth-form__button']

    # El método hace clic en el botón Iniciar sesión
    def click_sign_in_button(self):
        self.driver.find_element(*self.sign_in_button).click()
```

**Esto es lo que sucede en el código:** cuando el método se refiere a la variable `sign_in_button`, el operador toma sus valores y los pasa al método. El método `find_element()` comprende que debe buscar por nombre de clase el localizador `sign_in_button`. Esto es desempaquetar.

## Tipos de métodos

Todos los métodos en POM se dividen en dos grupos:

- acciones,
- pruebas.

Las acciones son aquellos métodos que emulan las acciones del usuario o usuaria en la página, tales como hacer clic en un botón o seleccionar desde una lista.

Las pruebas son aquellos métodos que comprueban las propiedades de los elementos, por ejemplo, si el elemento es visible o si el texto del elemento coincide con el texto esperado.

## Combinación de métodos en pasos

Puedes agrupar métodos y combinarlos en pasos dentro de un objeto de página.

Un **paso** es un método que contiene una cadena de acciones o pruebas. Los métodos combinados en un paso conducen a un resultado específico.

## Cómo crear un paso

Para combinar métodos en un paso, realiza lo siguiente:

1. Crea un nuevo método. Asígnale un nombre que refleje el resultado del paso.
2. Llama a los métodos dentro del paso en el orden requerido.

Observa este ejemplo que combina los métodos para iniciar la sesión del usuario o usuaria en la aplicación en el paso `login`:

```
from selenium.webdriver.common.by import By


class LoginPageAround:
    # El localizador del campo Correo electrónico
    email_field = [By.ID, 'email']
    # El localizador del campo Contraseña
    password_field = [By.ID, 'password']
    # El localizador del botón Iniciar sesión
    sign_in_button = [By.CLASS_NAME, 'auth-form__button']

    def __init__(self, driver):
        self.driver = driver

    # El método rellena el campo Correo electrónico
    def set_email(self, email):
        self.driver.find_element(*self.email_field).send_keys(email)

    # El método rellena el campo Contraseña
    def set_password(self, password):
        self.driver.find_element(*self.password_field).send_keys(password)

    # El método hace clic en el botón Iniciar sesión
    def click_sign_in_button(self):
        self.driver.find_element(*self.sign_in_button).click()

    # El método de inicio de sesión combina el correo electrónico, la contraseña y el clic del botón
    # Este es el paso
    def login(self, email, password):
        self.set_email(email)
        self.set_password(password)
        self.click_sign_in_button()
```

📎 **Recuerda estos puntos:**

1. Asigna nombres significativos a los pasos.
2. Es importante lograr un equilibrio al combinar métodos, no combines todos los métodos en un solo paso.
3. Es mejor combinar aquellos métodos que a menudo se suceden entre sí, de esta forma, podrás reutilizarlos en pruebas distintas.

## Un ejemplo de autotest de POM

El programa prueba la página de inicio de sesión de Around. Introduce el correo electrónico y la contraseña y hace clic en el botón Iniciar sesión.

```
# Importa el controlador
from selenium import webdriver
# Importa la clase de la página de inicio de sesión
from page_object import LoginPageAround


class TestAround:

    driver = None

    @classmethod
    def setup_class(cls):
        # Crea un controlador para Chrome
        cls.driver = webdriver.Chrome()

    def test_login(self):
        self.driver.get('https://around-v1.nm.tripleten-services.com/signin?lng=es')

        # Crea un objeto de página para la página de inicio de sesión
        login_page = LoginPageAround(self.driver)
        # Iniciar sesión
        login_page.login('email', 'password')

    @classmethod
    def teardown_class(cls):
        # Cerrar el navegador
        cls.driver.quit()
```

## Método `setup_class()`

En el método `setup_class()`, inicializamos `driver` para que funcione con `webdriver.Chrome()`.

```
    @classmethod
    def setup_class(cls):
        # Crea un controlador para Chrome
        cls.driver = webdriver.Chrome()
```

Este método se utiliza una sola vez en la clase y antes de todas las pruebas; ayuda a evitar que se inicialice el controlador para cada prueba.

Cuando utilices este método, haz referencia a `driver` como `self.driver` en tus pruebas. A continuación te mostramos un ejemplo:

```
# Crea un objeto de página para la página de inicio de sesión
login_page = LoginPageAround(self.driver)
```

Puedes ver a `@classmethod` antes de la definición del método, este es un **decorador**.

Los decoradores se utilizan para ampliar la funcionalidad de los métodos y funciones. El decorador `@classmethod` ayuda a diferenciar entre métodos de clase y métodos regulares.

Es por eso que `setup_class` es un método de clase, y su primer parámetro es `cls`, no `self`.

## Método `teardown_class()`

El método `teardown_class()` cierra el navegador y sale de `driver`. Esto se hace mediante la sentencia `cls.driver.quit()`.

```
    @classmethod
    def teardown_class(cls):
        # Cerrar el navegador
        cls.driver.quit()
```

Al igual que `setup_class()`, el método `teardown_class()` se ejecuta solo una vez para toda la clase, pero al final de todas las pruebas.