Selenium Page object model

Il Page Object Model (POM) è un design pattern ampiamente utilizzato nell'automazione dei test con Selenium per migliorare la manutenibilità, la leggibilità e la riusabilità del codice. L'obiettivo principale del POM è separare la logica di automazione dallo sviluppo dell'interfaccia utente, consentendo una gestione più efficiente degli elementi della pagina e delle azioni ad essi associate.

A cosa serve il Page Object Model (POM)?

1. **Separazione dei Concetti:**
    - Il POM separa la logica di automazione (codice di test) dalla struttura dell'interfaccia utente (elementi della pagina). Questa separazione facilita la manutenzione e la modifica del codice.

2. **Riusabilità del Codice:**
    - I Page Object possono essere riutilizzati in diversi test. Se un elemento della pagina cambia, è possibile apportare la modifica in un unico punto, nel Page Object corrispondente, senza dover modificare tutti i test.

3. **Leggibilità del Codice:**
    - Il POM migliora la leggibilità del codice, poiché le interazioni con gli elementi della pagina sono encapsulate all'interno di metodi nei Page Object. I test diventano più chiari e autoesplicativi.

4. **Manutenibilità:**
    - Quando ci sono modifiche nell'interfaccia utente, è possibile apportare le modifiche necessarie nei Page Object senza dover rivedere l'intero codice di automazione.

Come si usa il Page Object Model con Selenium?

1. **Creazione dei Page Object:**
    - Per ogni pagina o componente dell'applicazione, crea una classe Page Object. Questa classe conterrà metodi che rappresentano le azioni che è possibile eseguire sulla pagina e variabili che rappresentano gli elementi della pagina.

```java
public class LoginPage {
    private WebDriver driver;
    private By usernameInput = By.id("username");
    private By passwordInput = By.id("password");
    private By loginButton = By.id("loginButton");

    public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

    public void enterUsername(String username) {
        driver.findElement(usernameInput).sendKeys(username);
    }

    public void enterPassword(String password) {
        driver.findElement(passwordInput).sendKeys(password);
    }

    public void clickLoginButton() {
        driver.findElement(loginButton).click();
    }
}
```

2. **Utilizzo dei Page Object nei Test:**
    - Nei test, istanzia gli oggetti Page Object e utilizzali per eseguire le azioni e le verifiche sulla pagina.

```
public class LoginTest {
    WebDriver driver = new ChromeDriver();
    LoginPage loginPage = new LoginPage(driver);

    @Test
    public void testLogin() {
        driver.get("https://example.com");
        loginPage.enterUsername("user123");
        loginPage.enterPassword("pass123");
        loginPage.clickLoginButton();

        // Aggiungi asserzioni o verifiche necessarie
    }

    @After
    public void tearDown() {
        driver.quit();
    }
}
```

3. **Gestione delle Aspettative:**
    - Nei Page Object, è possibile aggiungere metodi per gestire aspettative, come l'attesa della presenza di un elemento o l'attesa di un certo stato della pagina.

```
public class HomePage {
    private WebDriver driver;
    private By welcomeMessage = By.id("welcomeMessage");

    public HomePage(WebDriver driver) {
        this.driver = driver;
    }

    public boolean isWelcomeMessageDisplayed() {
        return driver.findElement(welcomeMessage).isDisplayed();
    }
}
```

```
public class LoginTest {
    WebDriver driver = new ChromeDriver();
    LoginPage loginPage = new LoginPage(driver);
    HomePage homePage = new HomePage(driver);

    @Test
    public void testLogin() {
        driver.get("https://example.com");
        loginPage.enterUsername("user123");
        loginPage.enterPassword("pass123");
        loginPage.clickLoginButton();

        assertTrue(homePage.isWelcomeMessageDisplayed());
    }

    @After
    public void tearDown() {
        driver.quit();
    }
}
```

4. **Struttura del Progetto:**
    - Organizza il progetto in modo che i Page Object siano in una directory separata. Ad esempio:

```
src
├── main
│   └── java
│       └── pages
│           ├── LoginPage.java
│           └── HomePage.java
└── test
    └── java
        └── tests
            └── LoginTest.java
```

Il Page Object Model è una pratica comune nel campo dell'automazione dei test web, e la sua adozione può migliorare notevolmente la manutenibilità e l'efficienza dei tuoi test.