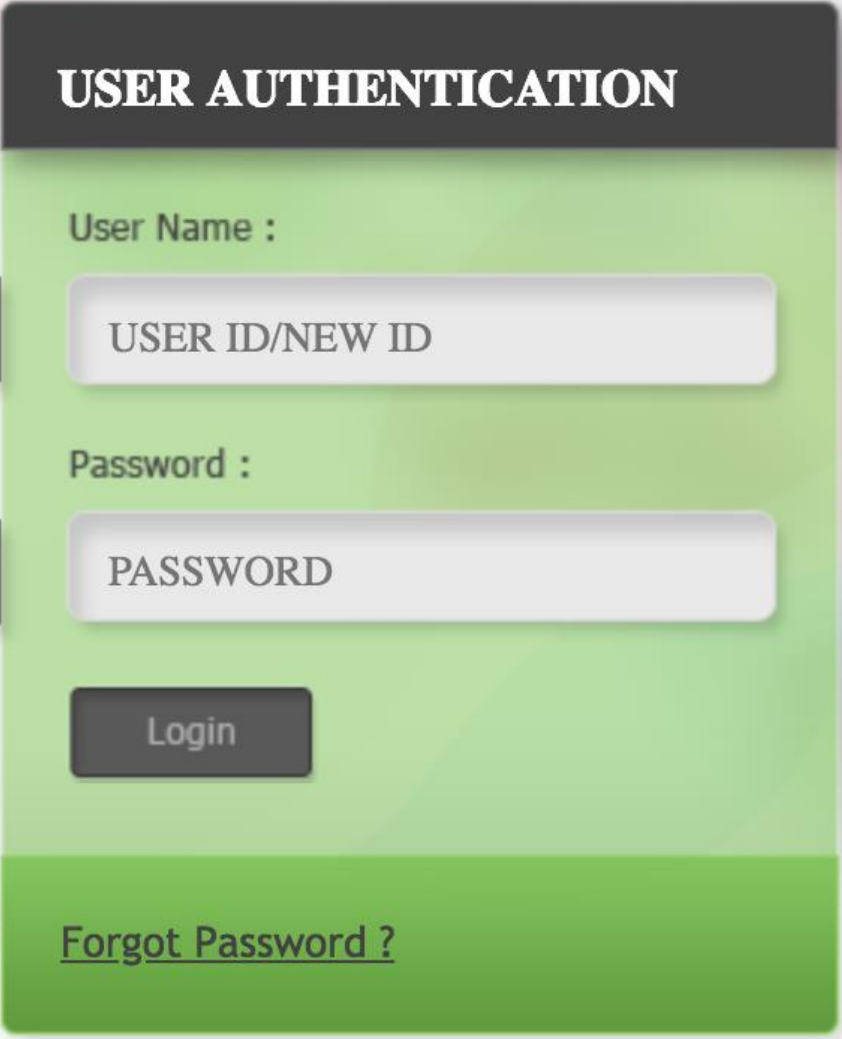


Page Object Model in Selenium

Page Object model is an object design pattern in Selenium, where web pages are represented as classes, and the various elements on the page are defined as variables on the class. All possible user interactions can then be implemented as methods on the class:



The image shows a user authentication form with a green background and a dark grey header. The header contains the text "USER AUTHENTICATION" in white, bold, uppercase letters. Below the header, there are two input fields. The first input field is labeled "User Name :" and contains the text "USER ID/NEW ID". To the left of this input field is a dark grey icon of a person. The second input field is labeled "Password :" and contains the text "PASSWORD". To the left of this input field is a dark grey icon of a padlock. Below the input fields is a dark grey button with the text "Login" in white. At the bottom of the form is a green bar with the text "Forgot Password ?" in white.

```

package com.selenium;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class PageFactory {

    @FindBy(xpath="//a[@id='userName']")
    WebElement userName;

    @FindBy(xpath="//a[@id='userName']")
    WebElement login;

    public void enterUserName(String userNam){
        userName.sendKeys(userNam);
    }

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

```

well-named methods in classes are easy to read. this works as better way to implement test routines that are both readable and easier to maintain or update in the future.

In order to support Page Object model, we use Page Factory. Page Factory in Selenium is an extension to Page Object and can be used in various ways. In this case we will use Page Factory to initialize web elements that are defined in web page classes or Page Objects.

Web page classes or Page Objects containing web elements need to be initialized using Page Factory before the web element variables can be used. This can be done simply through the use of *initElements* function on PageFactory:

PageFactory.initElements(driver, this);

OR

```
public LoginPage(WebDriver driver) {  
    this.driver = driver;  
    PageFactory.initElements(driver, this);  
}
```

Page Factory will initialize every *WebElement* variable with a reference to a corresponding element on the actual web page based on configured “locators”. This is done through the use of *@FindBy* annotations. With this annotation, we can define a strategy for looking up the element, along with the necessary information for identifying it:

```
@FindBy(xpath = "//a[@id='userName']")  
WebElement login;
```

Advantages of POM

1. Page Object Pattern says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. The Second benefit is the **object repository is independent of test cases**, so we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG/JUnit for functional Testing and at the same time with JBehave/Cucumber for acceptance testing.
3. Code becomes less and optimized because of the reusable page methods in the POM classes.
4. **Methods** get **more realistic names** which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

```
package com.selenium;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {

    WebDriver driver;

    @FindBy(xpath = "//a[@id='userName']")
    WebElement userName;
    @FindBy(xpath = "//a[@id='useName']")
    WebElement login;

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

    public void enterUserName(String userNam) {
        userName.sendKeys(userNam);
    }

    public HomePage clickLogin() {
        login.click();
        return new HomePage(driver);
    }
}
```