Page Factory is **a class provided by Selenium WebDriver to support Page Object Design patterns**. In Page Factory, testers use @FindBy annotation. The initElements method is used to initialize web elements. @FindBy: An annotation used in Page Factory to locate and declare web elements using different locators.

Selenium Page Factory Pattern is like an extension to [Page Object Model](http://www.seleniumeasy.com/selenium-tutorials/page-object-model-framework-introduction), but Page Factory is much enhanced model. To start with, we just need to import package ‘org.openqa.selenium.support.PageFactory’

"Factory class can be used to make using Page Objects simpler and easier".

We use [Page Factory pattern](https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/support/PageFactory.html) to initialize web elements which are defined in Page Objects.

We should initialize page objects using initElements() method from PageFactory Class as below, Once we call initElements() method, all elements will get initialized. PageFactory.initElements() static method takes the driver instance of the given class and the class type, and returns a Page Object with its fields fully initialized.

Home homePage = **new** HomePage(driver);

PageFactory.initElements(driver, homePage);

Or,

*// To initialize elements.*

HomePage homePage = PageFactory.initElements(driver, HomePage.**class**);

Or, **as a constructor for page class as below:**

**public** **HompePage**(WebDriver driver) {

**this**.driver = driver;

PageFactory.initElements(driver, **this**);

}

We should preferably use a constructor which takes a WebDriver instance as its only argument or falling back on a no-arg constructor. An exception will be thrown if the class cannot be instantiated.

Page Factory will initialize every WebElement variable with a reference to a corresponding element on the actual web page based on “locators” defined. This is done by using **@FindBy annotations**.

Annotations?

In Page Factory, Annotations are used to give descriptive names for WebElements to improve code readability. And annotation **@FindBy** is used to identify Web Elements in the page.

By default, PageFactory will search for elements on the page with a matching id attribute, If that fails, then it will search by the name attribute. But as we need more control over identifying elements in the HTML page and mapping them to our Page Object fields. One way to do this is to use the @FindBy annotation, as shown in the following code:

The [@FindBy](https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/support/FindBy.html) annotation supports all the other locators strategies that we use:  
**id, name, className, css, xpath, tagName, linkText and partialLinkText**

Sample framework selenium java

<https://medium.com/software-testing-break-and-improve/intellij-idea-selenium-webdriver-automated-web-tests-with-page-objects-in-15-minutes-50aff32b7492>

Page**Factory** allows us to create test pages with the same mechanism behind it. This way, we don’t need to write WebDriver initialization code for each of our pages.

<https://www.geeksforgeeks.org/difference-between-pom-and-pagefactory/>

functional tests

POM- Page Factory is class used to initialize elements and write findby methods

Run a sample test

Check the report

Run only failed tests

Common methods like file upload in page.java

Test class – setup and tear down methods

Parameterization

Assertions

Implicit wait

This is always applied globally for all the elements wherever ‘driver’ keyword is used. So in general, it is used in setup method.

Dynamic in nature. Waits until the element found or timeout

Time can be changed anywhere at any point in your code

Explicity wait

Graphical user interface, text, application

Description automatically generated

What if both waits used at a time

## **webdriver first will follow the implicit wait and then follow the explicit wait since browser behavior will be sequential like other programing languages due single thread usage.**

If you have ajax elements , then explicit wait is best option

Below methods are useful to avoid extra lines of code whenever you use explicit wait for an element

Graphical user interface, text, application, email

Description automatically generated

<https://www.edureka.co/blog/uploading-file-usiing-selenium/>

<https://www.perfecto.io/blog/responsive-web-design-testing-strategy-leveraging-selenium-automation>

3.6.3

<https://www.browserstack.com/guide/run-visual-tests-with-selenium>

tell about yourself-

previous experience

projects

English knowledge – brief talk in English on – my experience in SF, US