



SYNTAX
TECHNOLOGIES

Selenium

Class 11

Agenda

Selenium Components

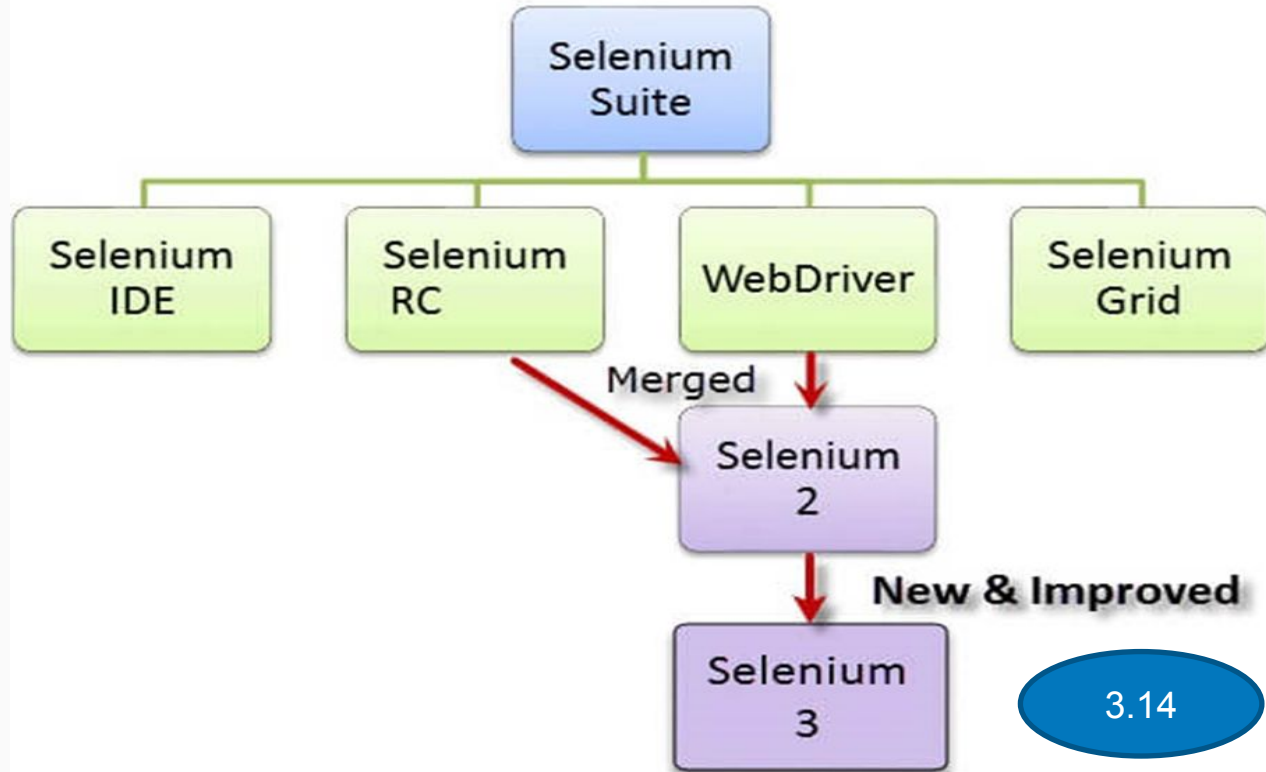
Page Object Model

Selenium

Selenium has 4 components:

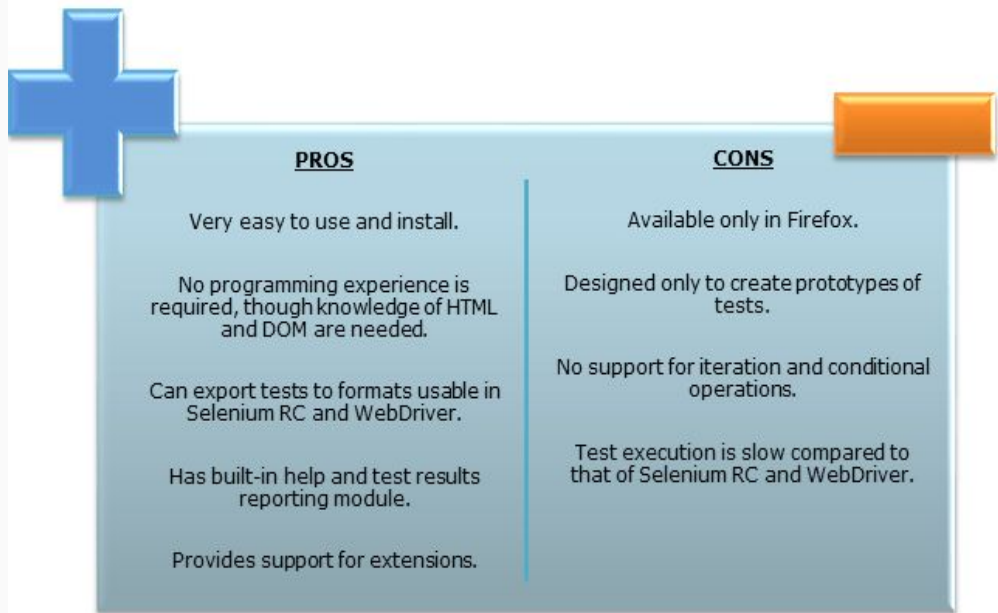
1. Selenium Integrated Development Environment (IDE) [outdated]
2. Selenium Remote Control (RC) [outdated]
3. Selenium WebDriver
4. Selenium Grid

Selenium Component upto selenium 3.X



Selenium IDE

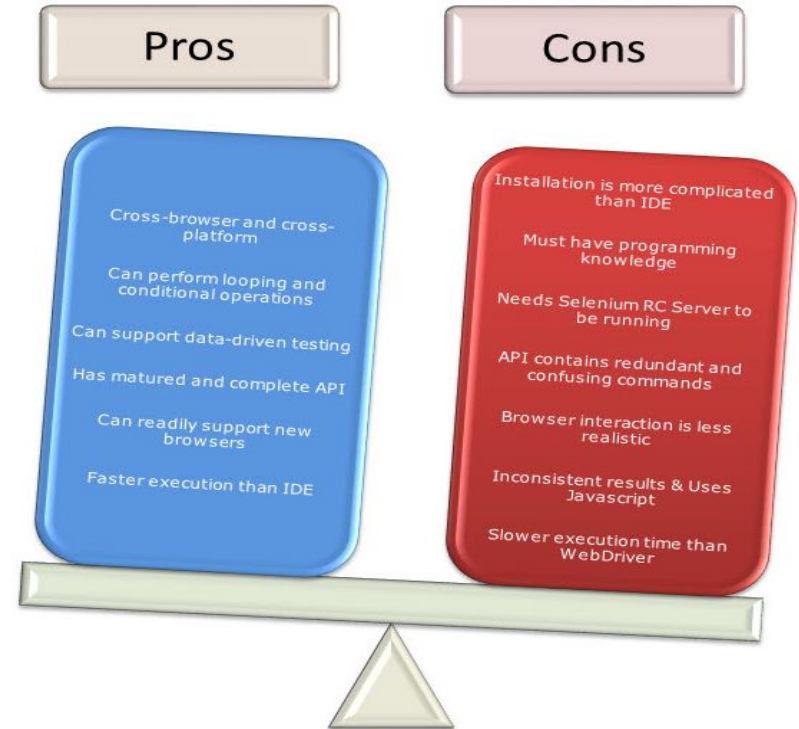
- Selenium Integrated Development Environment (IDE) is the simplest tool in the Selenium suite and is the easiest to learn.
- It is a Firefox plugin that you can install as easily as you can with other plugins. However, because of its simplicity, Selenium IDE should only be used as a prototyping tool.
- If we want to create more advanced test cases, you will need to use Selenium WebDriver.



| <u>PROS</u> | <u>CONS</u> |
|---|---|
| Very easy to use and install. | Available only in Firefox. |
| No programming experience is required, though knowledge of HTML and DOM are needed. | Designed only to create prototypes of tests. |
| Can export tests to formats usable in Selenium RC and WebDriver. | No support for iteration and conditional operations. |
| Has built-in help and test results reporting module. | Test execution is slow compared to that of Selenium RC and WebDriver. |
| Provides support for extensions. | |

Selenium RC

- Selenium RC is the first automated web testing tool that allowed users to use a programming language they prefer.
- As of version 2.25.0, RC can support the following programming languages:
 - Java
 - C#
 - PHP
 - Python
 - Perl
 - Ruby



Selenium Grid

Selenium Grid

Selenium Grid is a tool **used together with Selenium RC to run parallel tests** across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once.

Features:

- Enables **simultaneous running of tests** in **multiple browsers and environments**.
- **Saves time** enormously.
- Utilizes the **hub-and-nodes** concept. The hub acts as a central source of Selenium commands to each node connected to it.

Selenium WebDriver

Selenium WebDriver proves itself to be better than both Selenium IDE and Selenium RC in many aspects.

It implements a more modern and stable approach in automating the browser's actions. WebDriver, unlike Selenium RC, does not rely on JavaScript for Automation.

It controls the browser by directly communicating with it. The supported languages are the same as those in Selenium RC.

What is POM?

Page Object model is a design pattern for enhancing test maintenance and reducing code duplication.

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.

Page Object Model is a design pattern to create Object Repository for web UI elements. For each web page in the application there should be corresponding page class. This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.

Advantages of Page Object Model

- Code reusability – We could achieve code reusability by writing the code once and use it in different tests.
- Code maintainability – There is a clean separation between test code and page specific code such as locators and layout which becomes very easy to maintain code. Code changes only on Page Object Classes when a UI change occurs. It enhances test maintenance and reduces code duplication.
- Object Repository – Each page will be defined as a java class. All the fields in the page will be defined in an interface as members. The class will then implement the interface.
- Readability – Improves readability due to clean separation between test code and page specific code

Design and Implementation of POM using Selenium

How to Design:

- For each Page in the application we will be creating a separate Java Class.
- Each class is referred to as a PageObjects and returns other PageObjects to facilitate the flow between pages.
- Page Object class is responsible to find the WebElements of that page and also hold methods which perform operations on those WebElements.
- We will divide our Framework structure into 4 parts.
- Test Base classes, Page classes, Test classes and Utility classes.

How to implement :

1. Page Object model without PageFactory
2. Page Object Model with PageFactory

POM without PageFactory

```
package com.hrms.pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;

import com.hrms.testbase.BaseClass;

public class LoginPageWithoutFindBy extends BaseClass {

    public WebElement username = driver.findElement(By.id("txtUsername"));
    public WebElement password = driver.findElement(By.id("txtPassword"));
    public WebElement btnLogin = driver.findElement(By.id("btnLogin"));

}
```

- In LoginPage class, we declared the Locators for the WebElement Username, Password, Login button and Logo and also we can define the actions that we can perform on these elements like input text and click the button.

POM without PageFactory

```
package com.hrms.testcases;

import org.testng.annotations.Test;

import com.hrms.pages.LoginPageWithoutFindBy;
import com.hrms.utils.CommonMethods;

public class LoginPageTest extends CommonMethods{

    @Test
    public void validLogin() {
        LoginPageWithoutFindBy login=new LoginPageWithoutFindBy();
        sendKeys(login.username, "Admin");
        sendKeys(login.password, "Hum@nhrm123");
        click(login.btnLogin);
    }
}
```

- The second thing we do is create the Separate Test Class for each page class or each functionality.
- In our example, we are creating the LoginPageTest Class
- In this Test Class, we create the Test scripts.
- In our example, we are creating the Login test. We created the object of **LoginPage class** and called elements of LoginPage class in our Test Class to create the test script.

Page Factory in Selenium POM Framework

Page Factory is an inbuilt page object model concept for Selenium WebDriver but it is very optimized.

Here as well we follow the concept of separation of Page Object repository and Test methods. Additionally with the help of **PageFactory** class we use annotations **@FindBy** to find WebElement.

We use **initElements** method to initialize web elements.

The **PageFactory** Class is an extension of the Page Object design pattern. It is used to **initialize** the elements of the Page Object or **instantiate** the Page Objects itself.

PageFactory is used to initialize elements of a Page class without having to use 'FindElement' or 'FindElements'.

Annotations can be used to supply descriptive names of target objects to improve code readability.

@FindBy

In PageFactory, we use the **@FindBy** annotations to store the WebElements.

The **@FindBy** annotation supports all the other locators strategies that we use:
id, name, className, css, xpath, tagName, linkText
and partialLinkText

```
@FindBy(name="txtUsername")  
private WebElement user_name;
```

PageFactory.init Elements

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

PageFactory.initElements(driver, this);

or

PageFactory.InitElements(WebDriver, PageObject);

InitElements

This Instantiate an Instance /Elements of the given class.

This method will attempt to instantiate the class given to it, preferably using 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.

WebDriver – The driver that will be used to look up the elements

PageObjects – A class which will be initialised

Returns: An instantiated instance of the class with WebElement and List<WebElement> fields

Page Factory in Selenium POM Framework

```
package com.hrms.pageobjects;

import org.openqa.selenium.WebElement;

public class LoginPageElements {

    @FindBy(id = "txtUsername")
    public WebElement username;

    @FindBy(id = "txtPassword")
    public WebElement password;

    @FindBy(id = "btnLogin")
    public WebElement btnLogin;

    public LoginPageElements() {
        PageFactory.initElements(BaseClass.driver, this);
    }
}

package com.hrms.testcases;

import org.testng.annotations.Test;

public class LoginPageTest extends CommonMethods {

    @Test
    public void validLogin() {
        LoginPageElements login = new LoginPageElements();
        sendKeys(login.username, "Admin");
        sendKeys(login.password, "Hum@nhrm123");
        click(login.btnLogin);
    }
}
```