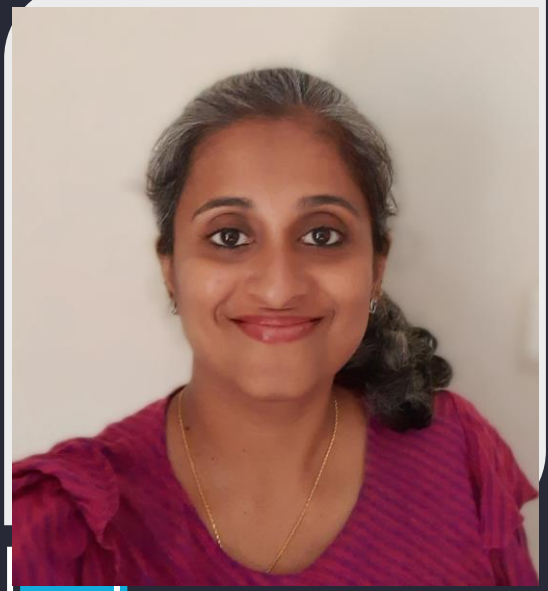
SELENIUM WITH JAVA

POORNIMA KRISHNAN



- 15+ Years Experience
- 12+ Years in Capgemini
- Certified in ISTQB certified tester(Foundation Level)
- Worksoft SAP Automation
- PSM 1
- Azure Fundamentals

Contact Me On:

Email: poornima.krishnan@capgemini.com

AGENDA

- Introduction to Selenium
- Record and Playback using Selenium IDE
- Webdriver
- Selenium locators
- Complex locators – CSS , Xpath
- Synchronisation

- TestNG Framework
- Annotations used in TestNG
- Selenium Automation Framework
- Page Object Model and Page Factory in Selenium
- Extent Reports
- Parametrization in Selenium
- Data Driven Testing – From Excel file
- Parallel Execution in Selenium

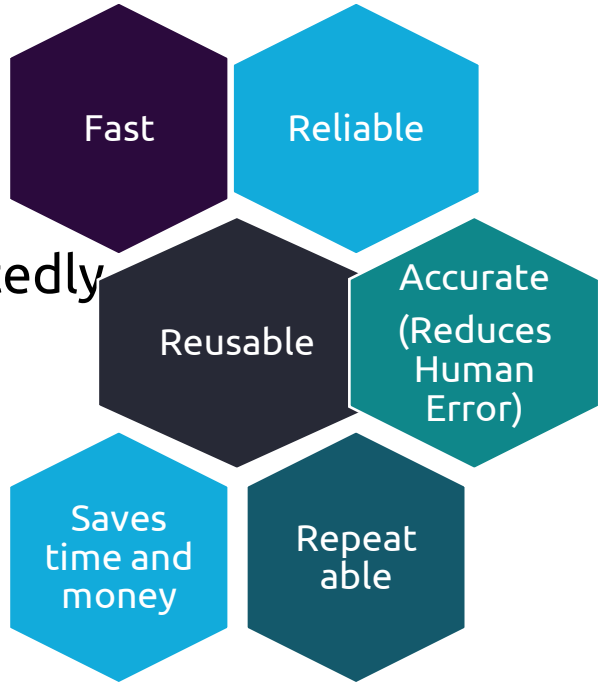
- Selenium Grid
- Hub and Node configuration
- Running Selenium Tests using Grid

PATHWAY

https://degreed.com/pathway/e9kmjz4w8o

- *Mandatory to complete the assessments given at the end of pathways*
- *Mandatory to complete all the assignments/lab guide provided at the end of pathways*
- *Mandatory to provide the feedback*

AUTOMATED TESTING



Automate the manual process to test an application or software
Separate testing tool in which the test script is created and executed repeatedly
Check expected results and actual results by running automation scripts.

- Selenium is an open-source project for a range of tools and libraries aimed at supporting browser automation

OS	Android, iOS, Windows, Linux, Mac, Solaris
Programming languages	C#, Java, Python, PHP, Ruby, Perl, JavaScript
Browsers	Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera, Safari

SELENIUM INTEGRATION

- Selenium can be used to automate functional tests and can be integrated with automation test tools such as Maven, Jenkins, & Docker
- Selenium can also be integrated with tools such as TestNG, & JUnit for managing test cases and generating reports.

COMPONENTS OF SELENIUM

Selenium IDE

1. Chrome, Firefox and Edge add-on that will do simple record-and-playback of interactions with the browser
2. Scripts are recorded in Selenese, a special test scripting language for Selenium

Selenium WebDriver

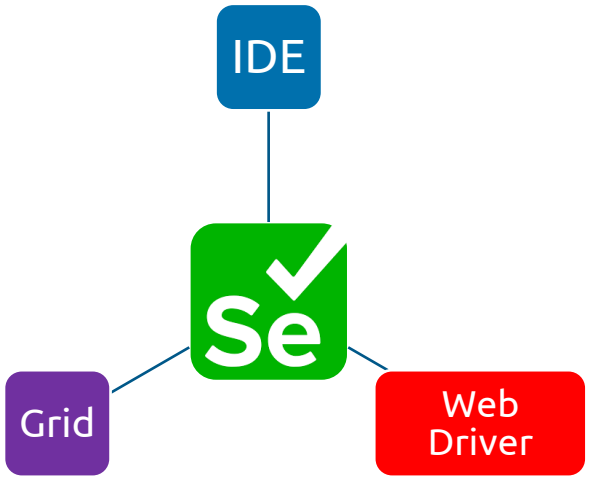
Selenium WebDriver provides a programming interface to create and execute test cases. Test scripts are written to identify web elements on web pages and then desired actions are performed on those elements.

WebDriver directly calls the methods of different browsers hence we have separate driver for each browser

- Mozilla Firefox Driver (Gecko Driver)
- Google Chrome Driver
- Internet Explorer Driver
- Opera Driver
- Safari Driver
- HTML Unit Driver

Selenium Grid

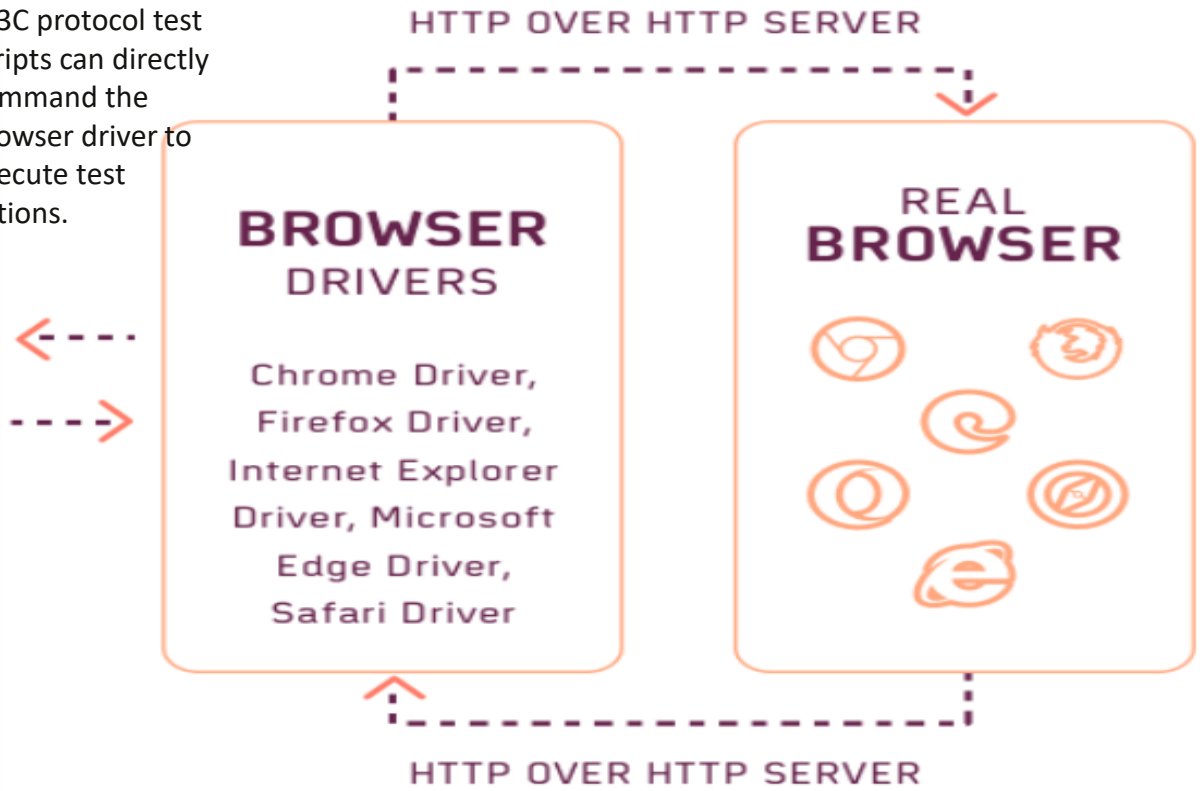
Tool that allows parallel execution of test scripts across different browsers on different platforms using Hub and node



SELENIUM WEBDRIVER ARCHITECTURE

It contains methods and classes of Selenium WebDriver that are required to create test automation scripts.
It provides an application programming interface (API), i.e., a set of functions that performs the Selenium commands from the test script

With WebDriver W3C protocol test scripts can directly command the browser driver to execute test actions.



Act as a bridge between the Selenium client libraries and the real browsers.
They help in running Selenium commands on the browser
Responsible for executing user actions, like mouse clicks, page navigation, button clicks, etc., on the browser.
For every supported browser we have a unique browser driver.
These browser drivers take commands from the Selenium test scripts and pass them to the respective browsers

The browser receives the command and calls the respective functions or methods to perform the desired automation task.

SELENIUM WEBDRIVER INSTALLATION

1.Download and Install Java 8 or higher version.

2.Download and configure Eclipse or any Java IDE of your choice.

3.Download Selenium WebDriver Java Client

1. Open URL: https://docs.seleniumhq.org/download/
2. Scroll down through the web page and locate Selenium Client & WebDriver Language Binding
3. Click on the "Download" link of Java Client Driver
4. The downloaded file would be in zipped format. Unpack the contents in a convenient director

4.Configure Selenium WebDriver

1. Create a new Java Project in Eclipse .
2. Load all the essential jar files in order to create Selenium Test Scripts in Properties ->Java Build Path->Libraries->Add External JARs

SELENIUM WEBDRIVER USING MAVEN

Maven is a powerful *project management tool* that is based on POM (project object model). It is used for projects build, dependency and documentation.

There are many problems that we face during the project development.

- 1) Adding set of Jars in each project. It must include all the dependencies of jars also.
- 2) Creating the right project structure: We must create the right project structure in servlet, struts etc, otherwise it will not be executed.
- 3) Building and Deploying the project: We must have to build and deploy the project so that it may work.

Maven simplifies the above-mentioned problems.

- It makes a project easy to build
- It provides uniform build process (maven project can be shared by all the maven projects)
- It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)

- *Steps to install Maven Eclipse Plug-in is provided in the ToC shared earlier.

MAVEN POM.XML FILE

POM is an acronym for Project Object Model.

- The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.
- Maven reads the pom.xml file, then executes the goal.

MAVEN ELEMENTS

Element	Description
project	It is the root element of pom.xml file.
groupId	It uniquely identifies your project across all projects. A group ID should follow Java's package name rules. This means it starts with a reversed domain name you control. For example, org.apache.maven
artifactId	artifactId is the name of the jar without version. If you created it, then you can choose whatever name you want with lowercase letters and no strange symbols. If it's a third party jar, you have to take the name of the jar as it's distributed. eg. maven, commons-math
dependencies	defines dependencies for this project.
dependency	defines a dependency. It is used inside dependencies.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>com.javatpoint.application1</groupId>
 <artifactId>my-app</artifactId>
 <version>1</version>


</project>
```

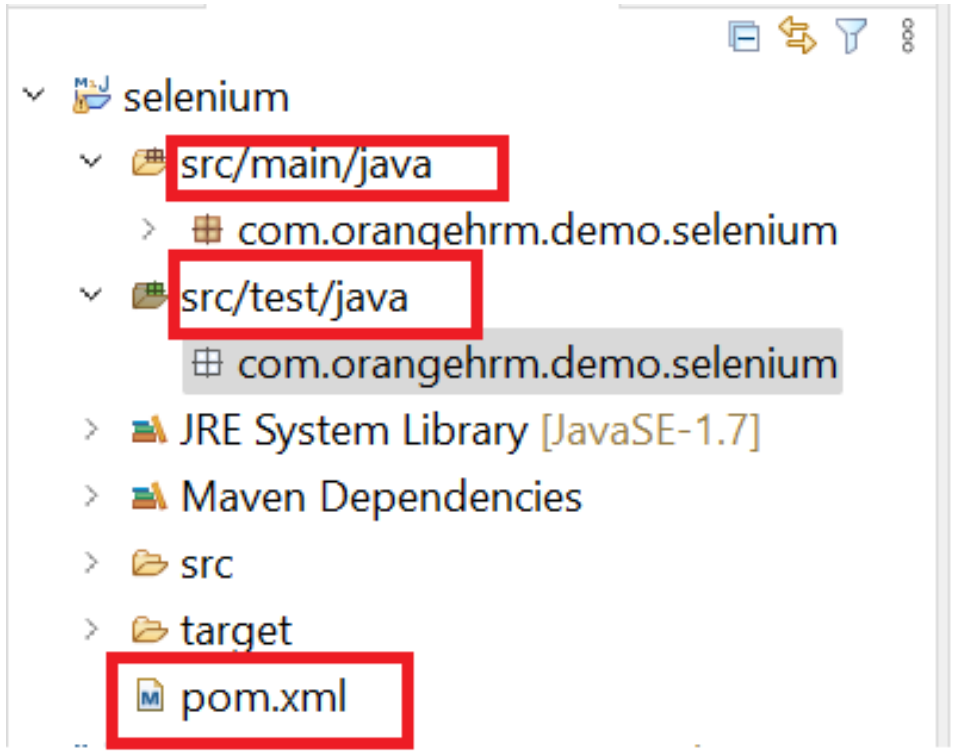
```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.sel
     <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.9.1</version>
      </dependency>
!-- https://mvnrepository.com/artifact/org.testng/testng -->
dependency>
   <groupId>org.testng</groupId>
   <artifactId>testng</artifactId>
   <version>7.8.0</version>
   <scope>test</scope>
/dependency>

</dependencies>
```

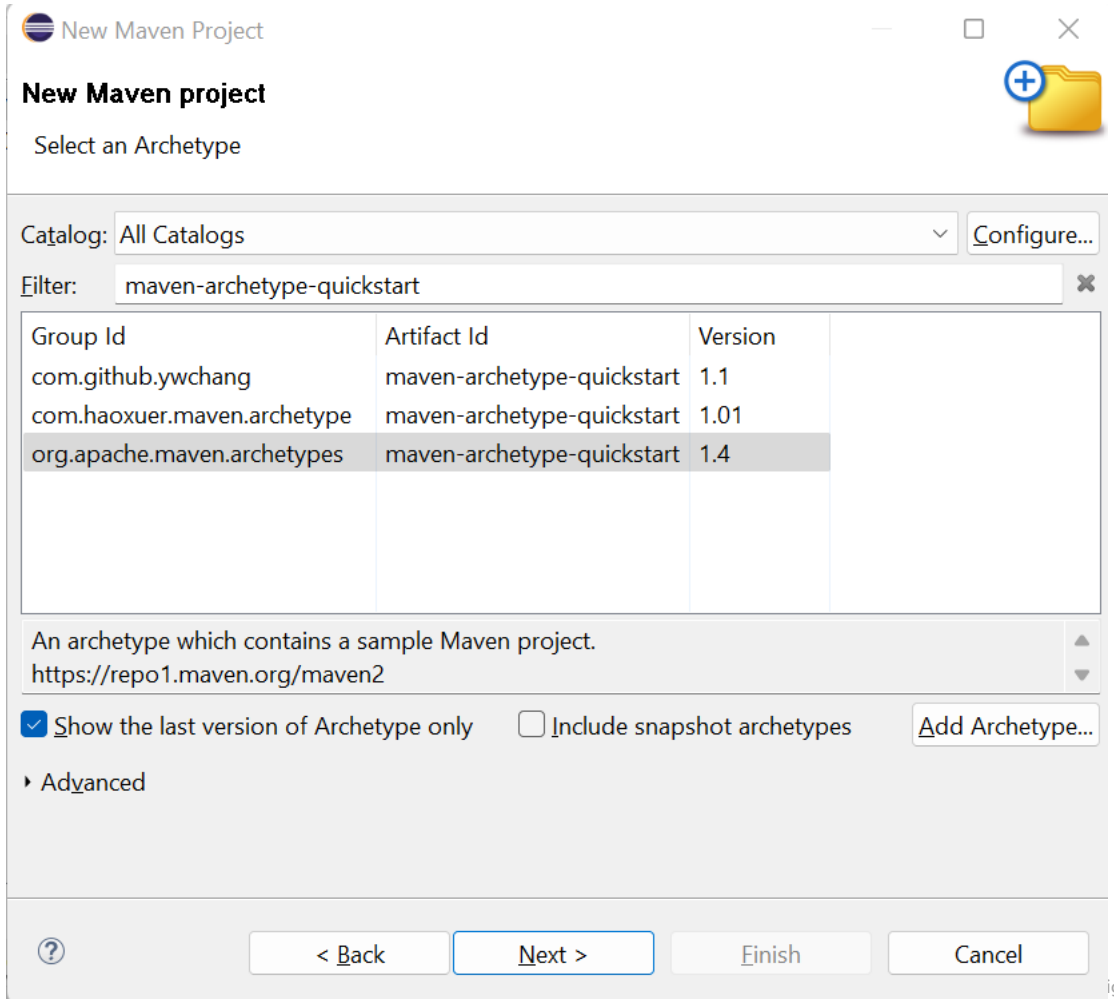
CREATE NEW MAVEN PROJECT IN ECLIPSE

When creating your New Maven Project please choose the below archetype

Group id – org.apache.maven.archetype

Artifact id – maven-archetype-quickstart





HOW TO ADD MAVEN DEPENDENCIES IN POM.XML

1. Navigate to https://mvnrepository.com/
2. Search for the respective dependencies
3. Navigate to the latest version. Click on it
4. Copy the code snippet from Maven tab
5. Paste the code snippet within the dependencies tag in pom.xml.
6. Ensure Build automatically option is checked in Eclipse.

7. Dependencies to be added
    1. **Selenium Java**
    2. **WebDriverManager**
    3. **TestNG**

WEBDRIVERMANAGER

- WebDriverManager is an open-source Java library that carries out the management (i.e., download, setup, and maintenance) of the drivers required by Selenium WebDriver (e.g., chromedriver, geckodriver, msedgedriver, etc.) in a fully automated manner.
- WebDriverManager is primarily used as a Java dependency.
- In addition, WebDriverManager provides other relevant features, such as the capability to discover browsers installed in the local system, building WebDriver objects.

INVOKING BROWSERS

Import Packages/Statements

In java, import statements are used to import the classes present in another packages. In simple words, import keyword is used to import built-in and user-defined packages into your java source file.

- References the WebDriver interface which is required to instantiate a new web browser
  - `import org.openqa.selenium.WebDriver;`

- References the Driver class that is required to instantiate a specific driver onto the browser instantiated by the WebDriver class.
  - `import org.openqa.selenium.chrome.ChromeDriver;`
  - `import org.openqa.selenium.edge.EdgeDriver;`
  - `import org.openqa.selenium.firefox.FirefoxDriver;`

Instantiating objects and variables

- A driver object is instantiated through:
  - WebDriver driver=new ChromeDriver();
  - WebDriver driver = new FirefoxDriver();
  - WebDriver driver = new EdgeDriver();

SELENIUM COMMANDS SYNTAX

Method name

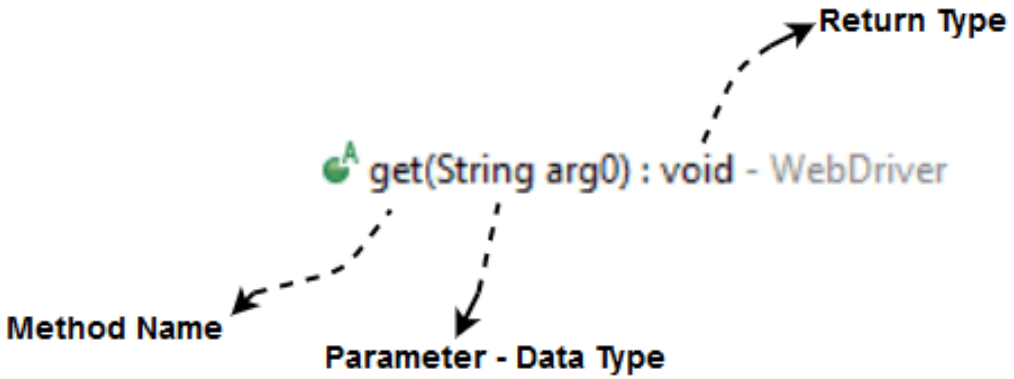
- To access any method of any class, we need to create an object of class and then all the public methods will appear for the object.

Parameter

- A parameter is an argument which is passed to a method to perform some specific operation.

Return type

- Methods can return a value or returning nothing (void). If the void is mentioned after the method, it means, the method is returning no value. If it is returning a value, then it must display the type of the value for e.g. **getTitle(): String**



SELENIUM COMMANDS

The commands provided by Selenium WebDriver can be broadly classified in following categories:

- Browser Commands
  - get, getTitle, getCurrentUrl
- Navigation Commands
  - navigate().to(String arg0)
- WebElement Commands
  - Clear , sendkeys, click

FINDING ELEMENTS - LOCATORS

- Selenium Performs actions (such as click, typing) on the Page HTML Elements.
- Locators are the way to identify an *HTML* element on a web page.
- Selenium WebDriver uses any of the below locators to identify the element on the page and performs the Action

ClassName	ID	Name	TagName
CssSelector	LinkText	PartialLinkText	XPath

FINDING ELEMENTS - LOCATORS

Selenium/pom.xml | invokeBrowser.java | invokeDifferentBrowsers.java | Locator1.java | Locator2.java

```
1  package seleniumTraining;
2
3  import org.openqa.selenium.By;
4  import org.openqa.selenium.WebDriver;
5  import org.openqa.selenium.chrome.ChromeDriver;
6
7  public class Locator1 {
8
9      public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         System.setProperty("webdriver.chrome.driver", "C:/Users/pokrishn/Documents/PVK/chromedriver.exe");
12         WebDriver driver = new ChromeDriver() ;
13         driver.get("https://demo.guru99.com/test/login.html");
14         driver.findElement(By.name("email")).sendKeys("test123@gmail.com");
15         driver.findElement(By.id("passwd")).sendKeys("password");
16         System.out.println(driver.findElement(By.linkText("Forgot your password?")).getText());
17         driver.findElement(By.className("button")).click();
18
19     }
20
21 }
22
```

LOCATORS – CSS AND XPATH

HTML Structure

<input type="text" placeholder="Username" id= “inputUsername” value=" ">

LOCATORS - CSS

- Tagname#id
- Tagname[attribute='value']
- Parent_tagname child_tagname
- Tagname[@attribute='value']:nth-child(index). - Child items
- regular expression - tagname[attribute*='string']
- Tagname
- Tagname.classname
- .classname

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.setProperty("webdriver.chrome.driver", "C:/Users/pokrishn/Documents/PVK/chromedriver.exe");
    WebDriver driver = new ChromeDriver() ;
    driver.get("https://demo.guru99.com/test/login.html");
    // Locators using CSS Selector

    // ID --- tagname#id
    driver.findElement(By.cssSelector("input#email")).sendKeys("CSS1");

    //Tagname[attribute='value']
    driver.findElement(By.cssSelector("input[id='email']")).sendKeys("CSS2");

    //Parenttagname childtagname
    System.out.println(driver.findElement(By.cssSelector("div form div p a")).getText());

    //Tagname[attribute='value']:nth-child(index). -  Child items
    driver.findElement(By.cssSelector("div[class='form_content clearfix'] div:nth-child(1) input")).sendK


    //regular expression tagname[attribute*='string'] - CSS
    driver.findElement(By.cssSelector("input[name*=\"passwd\"]")).sendKeys("password");

    //tagname
    System.out.println(driver.findElement(By.cssSelector("h3")).getText());

    // Class name  ---- tagname.classname
    driver.findElement(By.cssSelector("p.submit")).click();
```

LOCATORS - XPATH

- Use //
- //tagname
- //Tagname[@attribute='value']
- //Tagname[@attribute='value'][index]
- //parentTagname/childTagname
- //Using Regular expression
- //Using Sibling xpath /following-sibling::
- ///Using parent Xpath parent::div

Sibling and Parent not available for CSS Locator

```
15        //tagname
16        System.out.println(driver.findElement(By.xpath("//h3")).getText());
17
18        //Tagname[@attribute='value']
19        driver.findElement(By.xpath("//input[@name='email']")).sendKeys("XP1");
20        driver.findElement(By.xpath("//input[@name='email']")).clear();
21
22
23
24        //Tagname[@attribute='value'][index]
25        driver.findElement(By.xpath("//input[@class='is_required validate account_input form-control'][1]")).
26        driver.findElement(By.xpath("//input[@class='is_required validate account_input form-control'][1]")).
27
28        //parentTagname/childTagname
29        driver.findElement(By.xpath("//div /div /input")).sendKeys("XP3");
30
31
32        //Using Regular expression
33        driver.findElement(By.xpath("//form/div/div[2]/span/input[contains(@id,'pass')]")).sendKeys("pwd");
34
35
36        //Using Sibling xpath /following-sibling::
37        driver.findElement(By.xpath("//form/div/p/input/following-sibling::button")).click();
38
39        ///Using parent Xpath parent::div
40        System.out.println(driver.findElement(By.xpath("//div/div/h3/parent::div")).getText());
41
42
43
```

HANDLING DROPDOWNS

Static Dropdowns

- Contains select tag
- WebElement select
- Select class
  - Selectbyindex
  - SelectbyVisibleText
  - SelectbyValue
  - getFirstSelectionOption().getText()

```
public class Dropdowns {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.setProperty("webdriver.chrome.driver", "C:/Users/pokrishn/Documents/PVK/chromedriver.exe");
        WebDriver driver = new ChromeDriver() ;
        driver.get("file:///C:/Users/pokrishn/Documents/PVK/L&D/Selenium/Checkbox.html");

        //Static Dropdown
        WebElement staticdropdown = driver.findElement(By.xpath("(//select[@id='cars'])"));
        Select dropdown =new Select(staticdropdown);
        dropdown.selectByIndex(1);
        dropdown.selectByValue("audi");
        dropdown.selectByVisibleText("Volvo");
```

HANDLING DROPDOWNS

Dynamic Dropdown

- Use index
- Use Parent Child relationship xpath to uniquely identify

```
        //driver.close();

        //Dynamic Dropdown
        driver.get("https://www.goindigo.in/");
        driver.findElement(By.linkText("Book")).click();
        driver.findElement(By.xpath("//div/ul/li/div/div/div/div/a[contains(@href,'flight-booking_Header')]")).click();
        driver.close();

    }
```

SYNCHRONISATION IN SELENIUM

Implicit Wait

- Global setting for driver
- Waits every single time when selecting an element on that page

```
8         driver.get("file:///C:/Users/pokrishn/Documents/PVK/L&D/Selenium/Checkbox.html");
9         //Implicit wait
0         driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
1         //Static Dropdown
2         WebElement staticdropdown = driver.findElement(By.xpath("(//select[@id='cars'])"));
3         Select dropdown =new Select(staticdropdown);
4         dropdown.selectByIndex(1);
5         dropdown.selectByValue("audi");
```

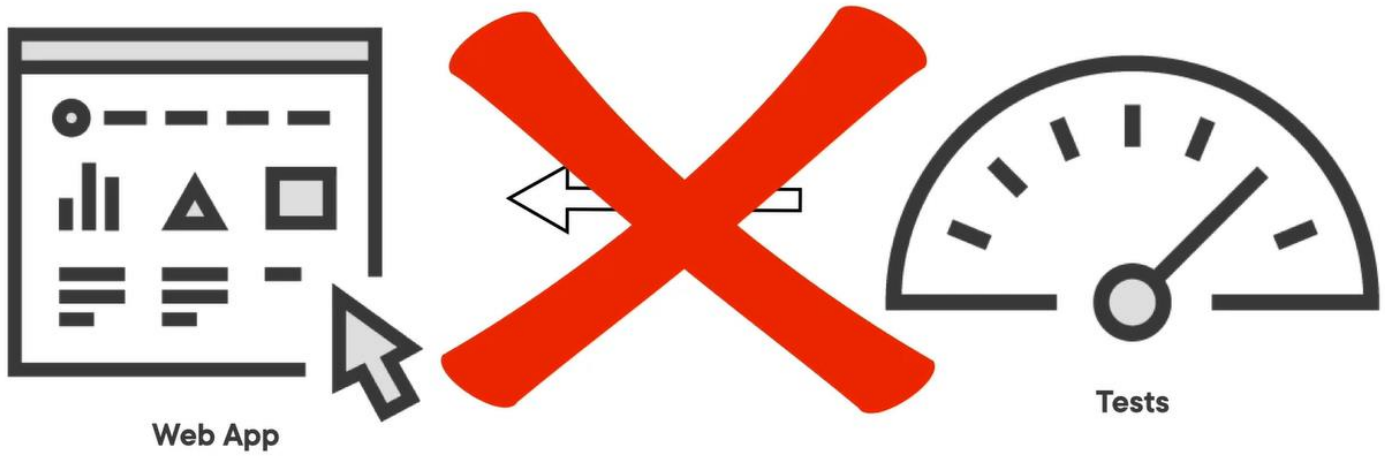
Explicit Wait

Web Driver waits for expected conditions or maximum time exceeded

```
        // TODO Auto-generated method stub
        System.setProperty("webdriver.chrome.driver", "C:/Users/pokrishn/Documents/PVK/chromedriver.e
        WebDriver driver = new ChromeDriver() ;
        WebDriverWait wait=new WebDriverWait(driver, Duration.ofSeconds(30));
        driver.get("file:///C:/Users/pokrishn/Documents/PVK/L&D/Selenium/Checkbox.html");
        //Explicit wait
        WebElement staticdropdown = driver.findElement(By.xpath("(//select[@id='cars'])"));
        wait.until(ExpectedConditions.visibilityOf(staticdropdown));
        Select dropdown =new Select(staticdropdown);
        dropdown.selectByValue("audi");


    }
```

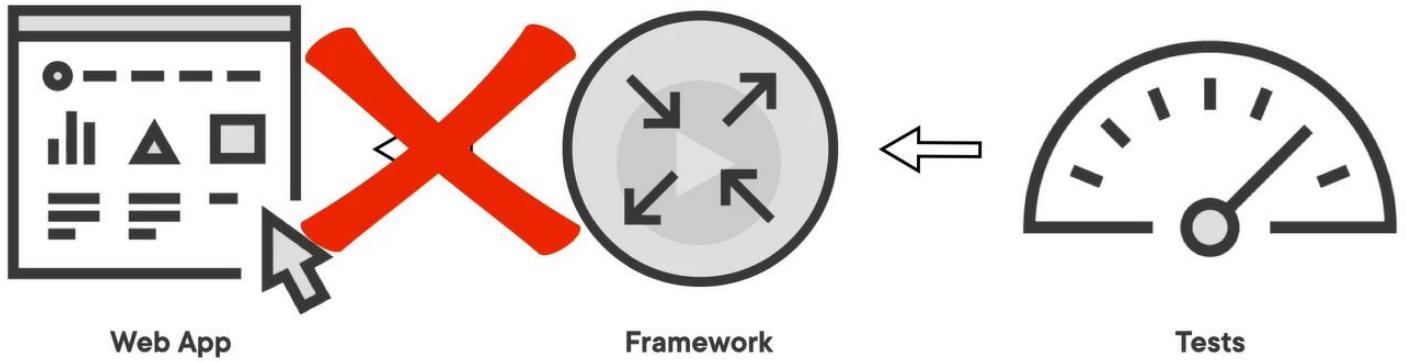
SELENIUM AUTOMATION FRAMEWORK

Record and playback – not recommended
Depends on application itself
Brittle Tests – If any changes are introduced , the tests break.
Increases modification efforts
Less reusability



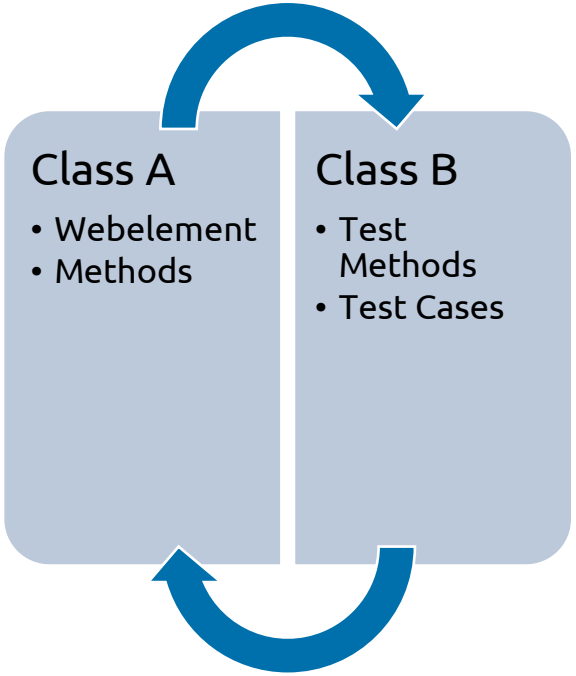
SELENIUM AUTOMATION FRAMEWORK

Tests depend on automation framework
Automation framework is almost like internal language for writing test
Page Object Model
Any changes in application – update automation framework
Changes only at one place



Page Object Model (POM) in Selenium

- Design pattern, popularly used in test automation that creates Object Repository for web UI elements
- Each web page in the application, has a corresponding Page Class.
- Page Class will identify the WebElements of that web page and contains Page methods which perform
- operations on those WebElements
- Name of these methods should be given as per the task they are performing

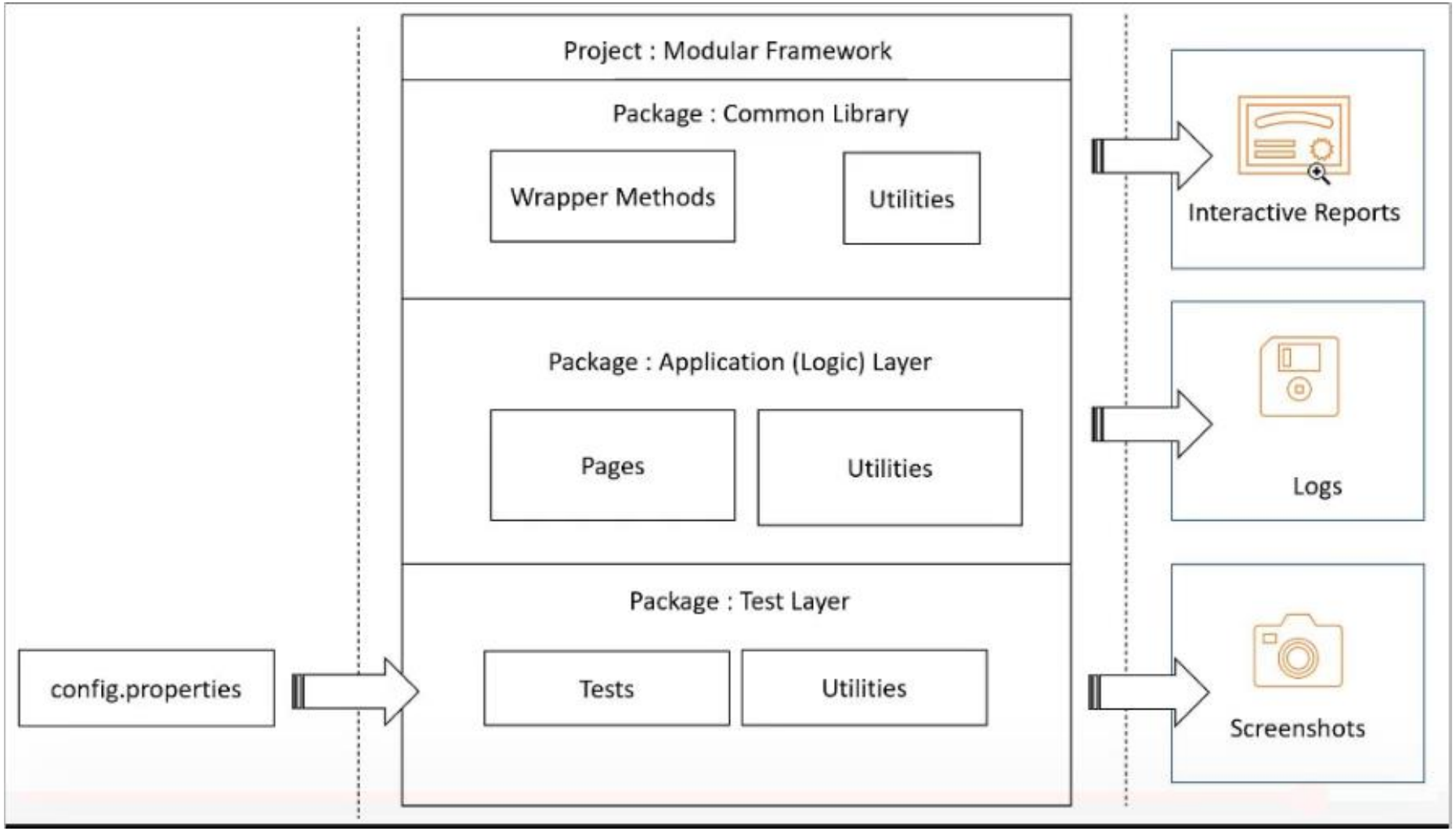


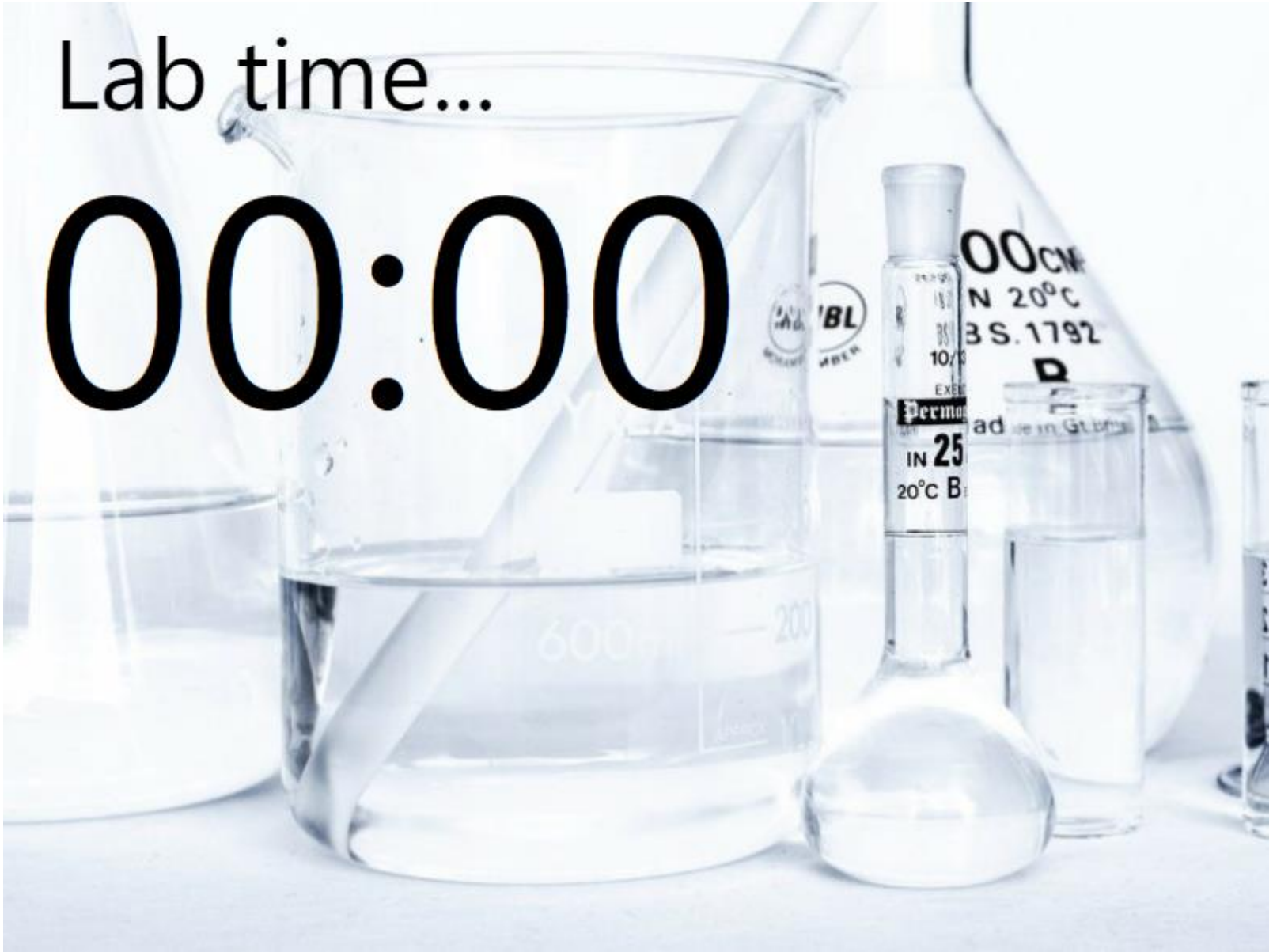
- Advantages
  - Operations and flows in the UI should be separated from verification.
  - This concept makes code cleaner and easy to understand
  - Object repository is independent of test cases. So changes to objects need to be made only at 1 location.
  - Reduces code duplication and improves test maintenance due to resuable methods.
  - Methods get more realistic names which can be easily mapped with the operation happening

Page Factory in Selenium

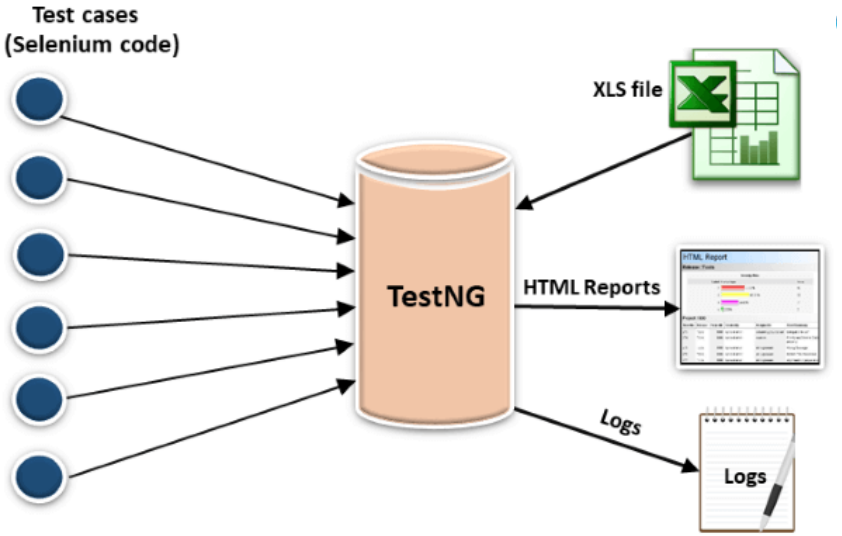
- Optimized way to create object repository in Page Object Model framework concept
- Initialization of Page objects or to instantiate the Page object itself.
- Initialize Page class elements without using "FindElement/s by using annotations **@FindBy** to find WebElement and initElements method to initialize web elements

GENERAL STRUCTURE





TESTNG FRAMEWORK



Automation testing framework in which NG stands for “Next Generation”inspired by JUnit which uses the annotations (@)

Advantages of using TestNG Framework

- Simplifies the way the tests are coded.
- WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format
- Generate the report in a proper format including number of test cases run, the number of test cases passed, the number of test cases failed, and the number of test cases skipped
- Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities which test case should be executed first.The same test case can be executed multiple times
- Cross Browser testing i.e Execute multiple test cases on multiple browsers
- Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest
- The TestNG framework can be easily integrated with tools like TestNG Maven, Jenkins, etc.

- *Steps to install TestNG Plug-in is provided in the ToC shared earlier.

ANNOTATIONS IN TESTNG

- Lines of code that can control how the method below them will be executed
- Always preceded by the @ symbol
- TestNG does not require you to have a main() method.
- Methods need not be static
- @Test annotation.Import the package org.testng.annotations.*.
- Use the Assert class to conduct verification operations in TestNG. Import the org.testng.Assert package
  - way to verify that the expected result and the actual result matched
  - generic syntax of TestNG Assertions:
    - Assert.Method(actual, expected)

ANNOTATIONS USED IN TESTNG

- @Test – Each method under @Test is treated as a separate Test case. It is executed alphabetically.

- @BeforeTest -methods under this annotation will be executed prior to the first test case
- @AfterTest -methods under this annotation will be executed after all test cases in the TestNG file are executed.

- @BeforeMethod - methods under this annotation will be executed prior to each method in each test case.
- @AfterMethod - methods under this annotation will be executed after each method in each test case.

- @BeforeSuite - The annotated method will be run before all tests in this suite have run
- @AfterSuite - The annotated method will be run after all tests in this suite have run

- @BeforeGroups - The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.
- @AfterGroups: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

- @BeforeClass: The annotated method will be run before the first test method in the current class is invoked.
- @AfterClass: The annotated method will be run after all the test methods in the current class have been run.

TESTNG XML FILE

1. Select the project, right click and select TestNG-> Convert to TestNG.
2. Select the required class files to be added
3. Click on Finish

You can trigger all the test cases from the single xml file. To trigger the execution right click on the xml file and select Run as -> TestNG Suite

test1.java | testng.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3  <suite name="Suite">
4    <test name="Test">
5      <classes>
6        <class name=".test1"/>
7      </classes>
8    </test> <!-- Test -->
9  </suite> <!-- Suite -->
10
```

Design | Source

TESTNG REPORTS

TestNG Reports are the default HTML reports which are generated once the test cases are executed using TestNG. These reports help you to identify the information about test cases and the status of a project

Emailable-report.html

Select emailable-report.html from test-output.

Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite							
Place Order	2	0	0	0	7,520		

Class	Method	Start	Time (ms)
Suite			
Place Order — passed			
com.orangehrm.demo.pageobjectmodel.task1	VerifyInvalidLogin	1698655502563	322
	placeOrder	1698655505293	1341

Place Order

com.orangehrm.demo.pageobjectmodel.task1#VerifyInvalidLogin

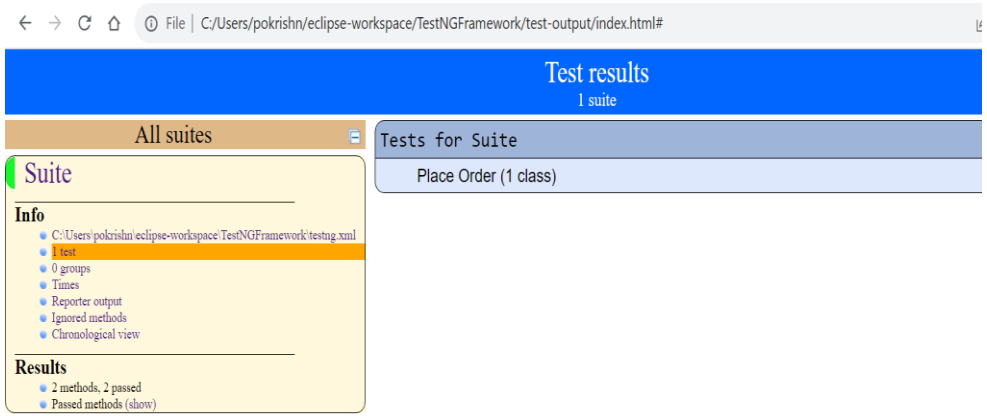
back to summary

com.orangehrm.demo.pageobjectmodel.task1#placeOrder

back to summary

Index.html

Select index.html from test-output.



EXCLUDE/INCLUDE TEST CASES

1. Disable or exclude the test cases by using the enable attribute to the @Test annotation and assign False value to the enable attribute

```
@Test(enabled=false)
```

2. To exclude From TestNG xml file

add the <method> tag in xml file which has access to all the methods of a class

```
<methods>
 <exclude name = "testmethodname"/>
 </methods>
```

3. To include in TestNG xml file

add the <method> tag in xml file which has access to all the methods of a class

```
<methods>
 <include name = "testmethodname"/>
 </methods>
```

TESTNG GROUPS

TestNG Groups allow you to perform groupings of different test methods

Groups are specified in the testng.xml file with <groups> tag.

Groups can be specified either in the <suite> tag or <test> tag.

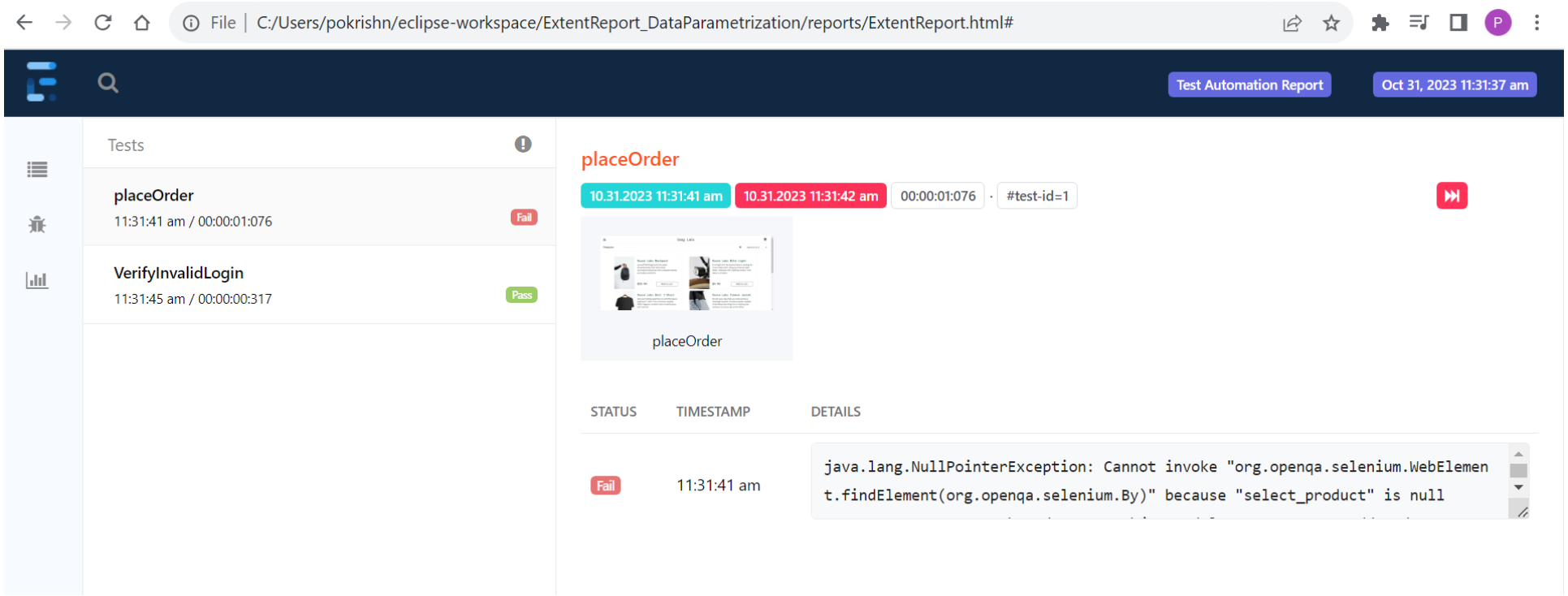
- If the <groups> tag is specified inside the <suite> tag, then it is applied to all the <test> tags of XML file.
- If the <groups> tag is specified within a particular <test> folder, then it is applied to that particular <test> tag only.

PARAMETERS IN TESTNG

- Priority
  - Usually executed in alphabetical order.
  - To change the order of execution , use priority parameter
  - Test(priority=0)
  - TestNG will execute the @Test annotation with the lowest priority value up to the largest
- alwaysRun=true/false
  - The test will always run
- To provide multiple parameters, separate with comma.
  - @Test(priority = 0, alwaysRun = true)

EXTENT REPORTS

- open-source reporting library useful for test automation
- HTML documents that depict results as pie charts
- Add events, screenshots, tags, devices, authors or any other relevant information



TESTNG LISTENERS

TestNG provides the @Listeners annotation which listens to every event that occurs in a selenium code. Listeners are activated either before the test or after the test case. It is an interface that modifies the TestNG behavior

Listeners are implemented by the **ITestListener** interface

onTestStart(): An onTestStart() is invoked only when any test method gets started.

onTestSuccess(): An onTestSuccess() method is executed on the success of a test method.

onTestFailure(): An onTestFailure() method is invoked when test method fails.

onTestSkipped(): An onTestSkipped() run only when any test method has been skipped.

onTestFailedButWithinSuccessPercentage(): This method is invoked each time when the test method fails but within success percentage.

onStart(): An onStart() method is executed on the start of any test method.

onFinish(): An onFinish() is invoked when any test case finishes its execution.

ADDING LISTENERS INTO SELENIUM CODE

1. Create a listerner class file which implements ITestListener
2. Add the code as required onTestStart, onTestFailure
3. OnTestFailure – call the method to capture the screenshot
4. Add the location of the listener class into TestNG xml file

PARAMETRIZATION

1. Config Files
2. Parameters
3. Data Provider

PARAMETRIZATION USING CONFIG FILE

1. Create a file with parameters in key-value format in .properties extension
2. Use Properties of Java.util and load the file .
   - Note – the file needs to be converted into file input stream.
3. Use .getProperty to get the value into a string and use in code.

PARAMETRIZATION USING TEST NG XML FILE

Suppose we want to set the global variables such url settings, username, password or API Keys, there are some values which are constant in all the test cases, in such case we use the TestNG Parameters.

TestNG Parameters are present in the xml file. They can be applied either inside the tag or tag. If we want to apply the parameters to all the test cases, then the parameters are applied inside the tag. If the parameter is specific to a particular folder, then the parameter is applied within a tag.

```
  https://testng.org/testng-1.0.dtd (doctype)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
3⊖<suite name="Suite">
4⊖<listeners>
5     <listener class-name="com.orangehrm.demo.pageobjectmodel.Listeners"></listener>
6 </listeners>
7 <parameter name="url" value="https://www.saucedemo.com/"></parameter>
8⊖  <test thread-count="5" name="Test">
```

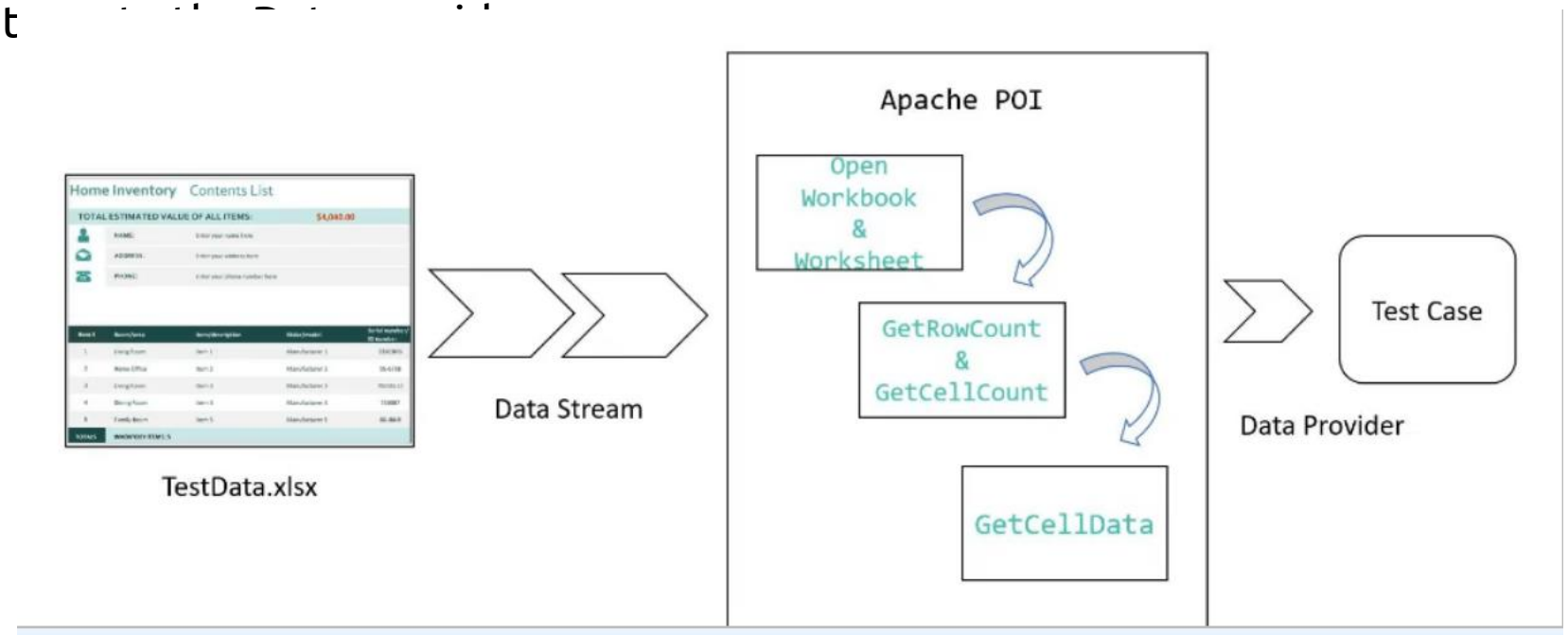
PARAMETRIZATION USING DATA PROVIDER

1. Used to Drive the data and pass multiple data sets
2. Add annotation @DataProvider
3. Create a method to return
4. a 2D array of an object where columns are the arguments needed in one test execution and rows are the number of data passed in each execution
5. Provide the name of the data provider to the @Test method and provide the data in the same order

```
@Test(dataProvider="getData")
Run | Debug
public void placeOrder(String username,String password,String productname,String firstname,String l

        //String productname = "Sauce Labs Backpack";

        InventoryPage inventorypage = loginpage.LogintoApp(username, password);
        inventorypage.getinventorylist();
        inventorypage.addproducttocart(productname);
        CartPage cartpage = inventorypage.gotocart();
        Boolean match = cartpage.verifycart(productname);
        //System.out.print(match);
        assertEquals(match, true);
        CheckOutPage checkoutpage = cartpage.gotocheckout();
        checkoutpage.enterdetailsincheckoutpage(firstname, lastname, pincode);



}
@DataProvider
public Object[][] getData() {
    return new Object[][] {{"standard_user","secret_sauce","Sauce Labs Backpack","Firstname", "Last
}
```

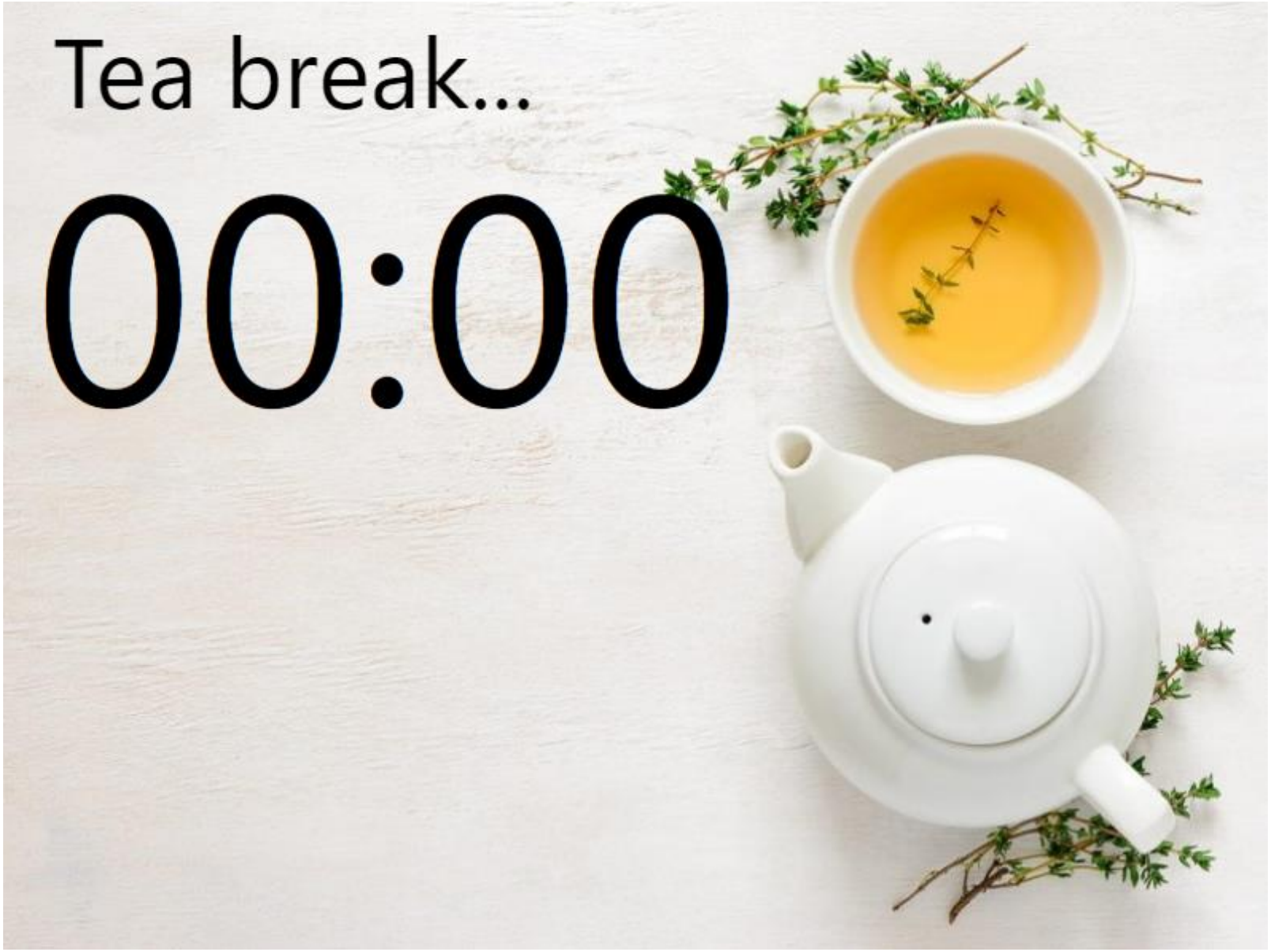
DATA DRIVEN TESTING USING EXCEL

- Add dependency for POI APACHE from Maven
- Create utility to read data from excel
  - Pass the file as an input stream
  - Get the rowcount, column count and the cell data
  - Read data to get



PARALLEL EXECUTION IN SELENIUM

- Reduces execution time
- In order to run the tests in parallel just add these two key-value pairs in testNG xml file in suite-
  - parallel="{methods/tests/classes}"
  - thread-count="{number of thread you want to run simultaneously}".
    - **methods** –run all the tests independently on separate threads(maximum available threads)
    - **tests** –run the methods specified in the test tag in the same thread but independent of each other.
    - **class** –run the all the test methods stated in a class in same thread and with each class's method running in different thread.

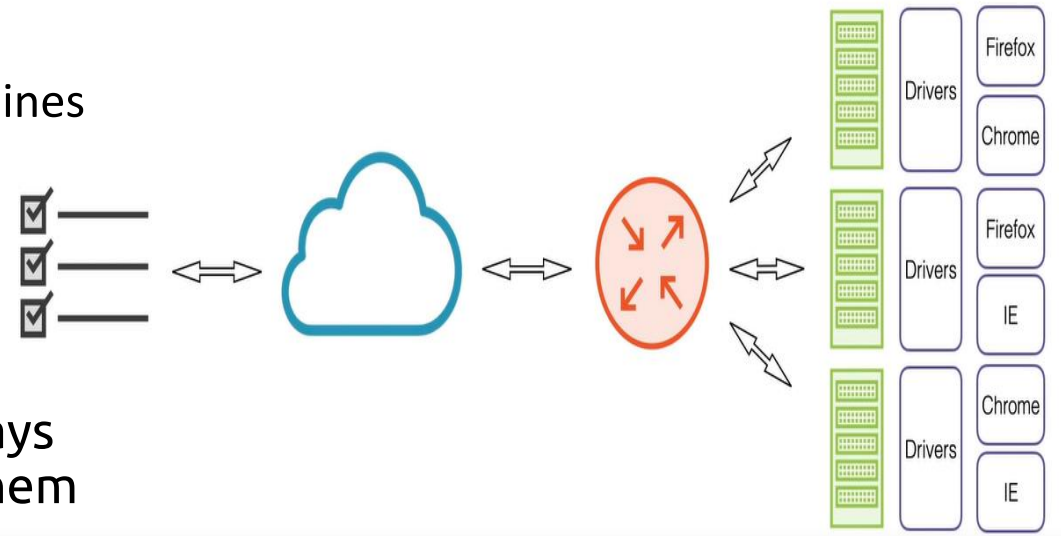


SELENIUM GRID

Selenium Grid allows the execution of WebDriver scripts on remote machines by routing commands sent by the client to remote browser instances.

Grid aims to:

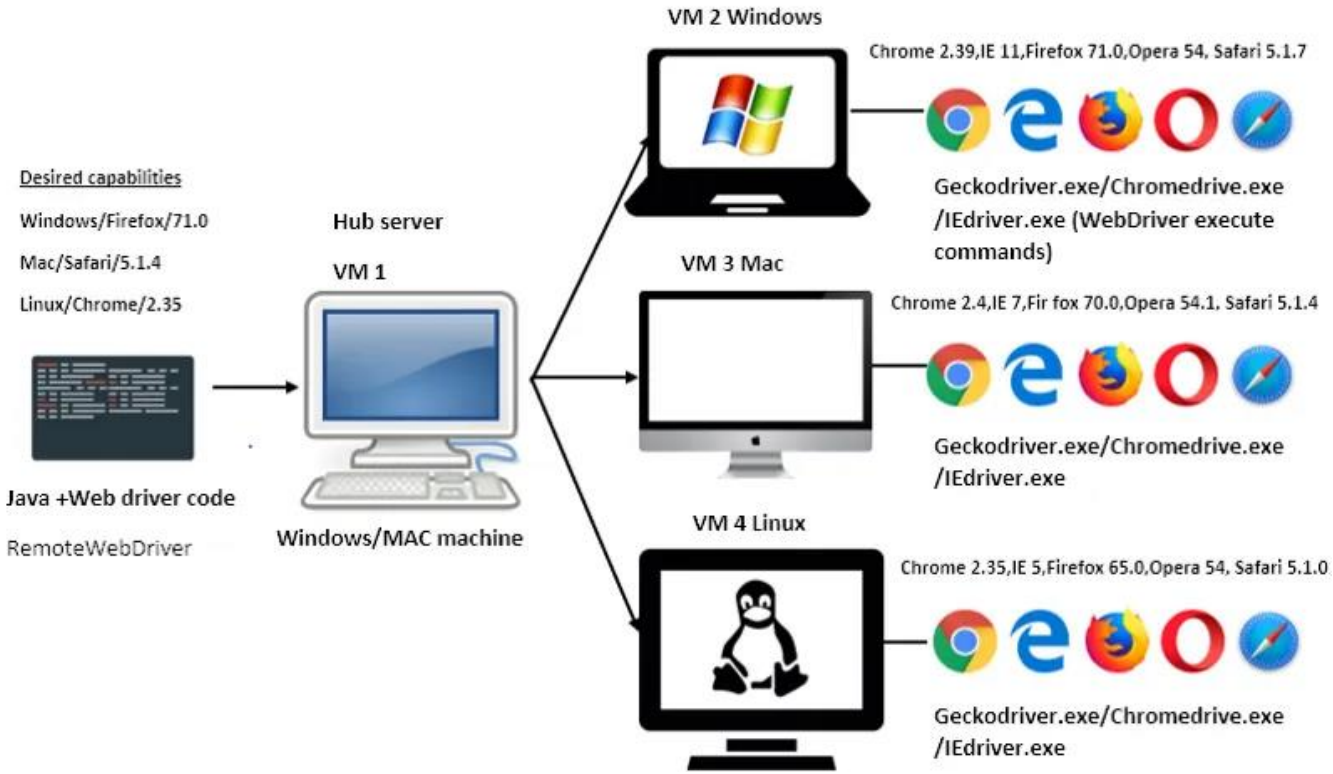
- Provide an easy way to run tests in parallel on multiple machines
- Allow testing on different browser versions
- Enable cross platform testing



- Grid is composed by six different components
- which gives you the option to deploy it in different ways
- Depending on your needs, you can start each one of them
  - on its own (Distributed),
  - group them in Hub & Node,
  - or all in one on a single machine (Standalone).

HUB AND NODE

- Combine different machines in a single Grid
- Machines with different operating systems and/or browser versions, for example
- Have a single entry point to run WebDriver tests in different environments
- Scaling capacity up or down without tearing down the Grid.
- A Hub is composed by the following components: Router, Distributor, Session Map, New Session Queue, and Event Bus.



STEPS TO CONFIGURE HUB AND NODE

- Download latest version of Selenium Server standalone jar
- Download the required versions of the drivers in the same folder
- In cmd of the folder type the below commands
  - java -jar selenium-server-4.15.0.jar hub
    - This will start the hub.
  - java -jar selenium-server-4.15.0.jar node --detect-drivers true --selenium-manager true
    - This will add the node to the hub above.
  - To add another physical machine as a node
  - java -jar selenium-server-4.10.0.jar node --detect-drivers true --publish-events tcp://<hub-ip>:4442 --subscribe-events tcp://<hub-ip>:4443
- Open the URL of the hub as mentioned in the cmd
- You should see the nodes added and the respective configuration

RUNNING SELENIUM TESTS USING GRID

1. Configure Hub and node
2. To design test scripts that will run on the grid, we need to use DesiredCapabilites and the RemoteWebDriver objects.
   - DesiredCapabilites is used to set the type of browser and OS that we will automate.
   - `import org.openqa.selenium.remote.DesiredCapabilities;`
   - RemoteWebDriver is used to set which node (or machine) that our test will run against.
   - `import org.openqa.selenium.remote.RemoteWebDriver;`

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setBrowserName("chrome");
driver = new RemoteWebDriver(new URL("http://192.168.0.161:4444"),caps );
```

Good work!

You've completed the concepts of Selenium+Java.