**Day 6 :**

**TestNG: Data provider.**

## **What are DataProviders in TestNG?**

The *DataProviders* in TestNG are another way to pass the parameters in the test function, the other one being TestNG parameters. *DataProviders* pass different values to the *TestNG Test Case* in a single execution and in the form of *TestNG Annotations*.

It is a part of the inbuilt TestNG data-driven testing for which TestNG is quite popular. *DataProviders* help in passing the parameters in different ways.

***DataProvider Syntax:***

The *TestNG DataProvider* is used in the following manner:

@DataProvider (name = "name\_of\_dataprovider")

public Object[][] dpMethod() {

return new Object [][] { values}

}

**How DataProvider works?**

Data provider is an annotation available in TestNg.

It allows testers to execute a single test method with multiple sets of data.

We have to write the test method only once and mention the Data provider .

This will execute the test method multiple times with multiple sets of data provided.

Steps to Implement Data provider :

**Step 1 : Create a 2 Dimensional Object array and mention the size of it wrt rows and column size of your Excel sheet.**

**Step 2 : Create 2 for loops (1 for Rows and 1 for Columns) and get the cell value and store it in a 2d array.**

**Step 3 : Return the Object array**

**Note** : For TestNg executing Data provider, always use @BeforeMethod and @AfterMethod for before and after test Implementation.

If we use @BeforeTest and @AfterTest then the Data provider won’t work properly, the data entry will happen before closing the browser.

**Groups**

## **TestNG Groups with Example**

We use groups in **Testng** when,

* We don’t want to define test methods separately in different classes (depending upon functionality) and
* At the same time I want to ignore (not execute) some test cases as if they do not exist in the code.
* So to carry out this we have to Group them. This is done by using the “include” and “exclude” mechanisms supported in testNG.

In the example below, we have shown the syntax of how to use groups in the XML file.

@Test (groups = { "smoke"})

Customize your XML to pick the mentioned group from the test classes. Below mentioned is the syntax of how to declare groups in an XML file e.g.

<groups>

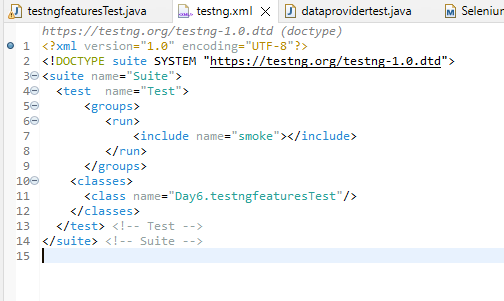
<run>

<include name="bonding" />

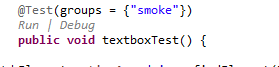
</run>

</groups>

**This is how our testng.xml should look like for using groups**

\

And this is how we should add groups parameter to out testNg tests.



**Priority**

## **What Is Prioritisation In TestNG?**

Prioritisation in TestNG is a way to provide a sequence to the methods so that they do not run out of order. Since alphabetically running test cases in TestNG have no logical sequence (*concerning the tests and code*), providing priority to these test cases helps us managing our tests' execution.

*Priority in TestNG test cases is a parameter with attribute value as "priority".*

### ***How to give Priority in TestNG test?***

*The following is the syntax for allocating a priority to a test case method.*

*@Test (priority = 1)*

*public void func(){*

*//test code*

*}*

**

*In the OpenBrowser method, I am trying to open the browser and enter the URL "www.demoqa.com." The "CloseBrowser" method, however, is used to close the driver. The priorities set are 0 for OpenBrowser and 1 for CloseBrowser, so I expect the OpenBrowser method to run first.*

*As expected, the OpenBrowser method ran first because of a lower priority. Had I not declared the priority here, it would have run alphabetically, i.e., CloseBrowser first and then OpenBrowser.*

**Parallel Testing**

**What is Parallel Execution in TestNG?**

Parallel testing is a process where multiple tests are executed simultaneously/in parallel in different thread processes.

With respect to Selenium and TestNG, it allows you to execute multiple tests on different browsers, devices, environments in parallel and at the same time, instead of running it sequentially.

The main purpose of running tests in parallel mode is to **reduce execution time** and do **maximum environment coverage** (browsers/devices/environment) in less time.

Suppose, for an application you need to execute a sanity automation suite of 50 test cases in **Chrome** and **Firefox** browser. If you go with the traditional sequential flow, you need to execute the suite for Chrome browser first which would take 1 hr and then you need to execute for Firefox browser which takes another 1 hr.

Total time taken sequentially - 2 hours.

**Time is Money.**

So, in total you would need 2 hrs to test in both the browsers.

But By using a parallel mechanism, you can run simultaneously for both the browsers in just 1 hr thereby reducing the execution time by 50%.

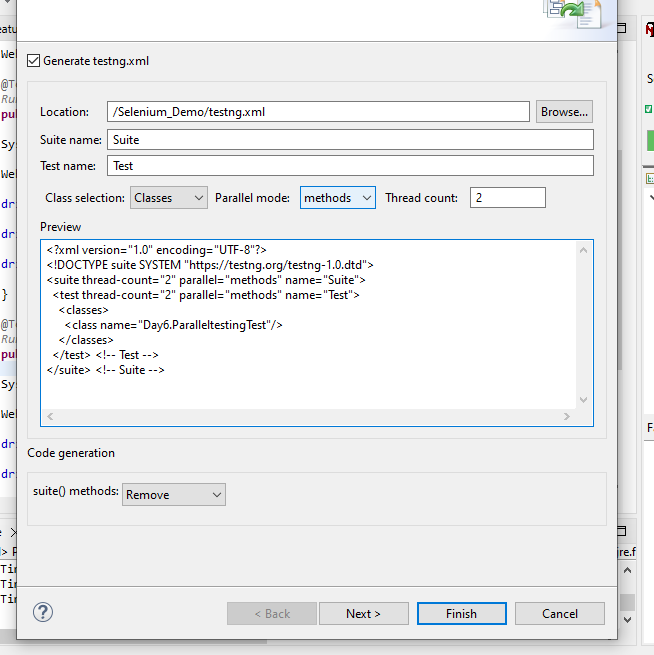
**Steps to implement Parallel testing in TestNG :**

To do so you need to first create a **testing.xml** file and add a parallel attribute for the test suite with value as **methods**.

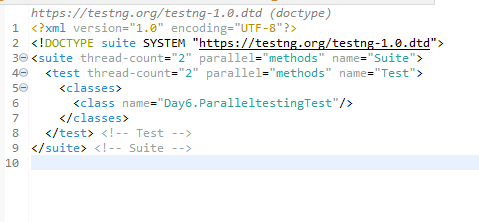
#### Executing Parallel Test Methods in TestNG

**Step 1** To create a **testing.xml** file, right click on the **ParallelTest** class and select **TestNG** >> **Convert To TestNG**.

**Step 2** You may select **Parallel mode** and **ThreadCount** value of your choice while creating the **testing.xml** file or you may update it later as per the requirement change. I have selected **Parallel mode** as **methods** and **ThreadCount** as **2**.

****

**This is how our testng.xml file should look like :**

****

**Step 3** Right click on the testing.xml file and select “Run As” -> “TestNG Suite”. Observe the time taken to execute both the methods in parallel mode.

**Day 7 :**

**Selenium Grid : Grid Architecture and Demo**

Selenium Grid is a tool in the Selenium suite that facilitates parallel test execution by allowing tests to run on multiple machines or browsers concurrently, providing a scalable and efficient solution for distributed test automation.

Selenium suite comprises four components:

1. Selenium Grid
2. Selenium IDE
3. Selenium RC
4. Selenium Webdriver

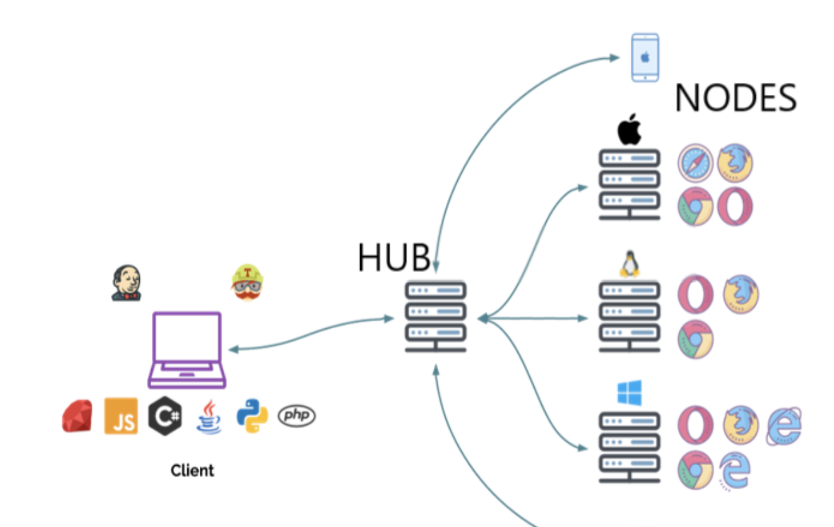
**Selenium Grid**

* Selenium Grid is a feature in Selenium that allows you to run test cases in different machines across different platforms.
* The control of triggering the test cases is on the local machine, and when the test cases are triggered, they are automatically executed by the remote machine. Suppose you have 5 test cases.
* Your local machine is running multiple applications, so you want to run your test cases in a remote machine. You need to configure the remote server so that the test cases can be executed there.

## 

## 

## **Architecture of Selenium Grid**



### **Hub**

* A Hub is a central point or a local machine that receives all the test requests and distributes them to the right nodes. The machine which actually triggers the test case known as Hub.
* There can be only one hub in a selenium grid.
* The machine which is containing the hub triggers the test case, but you will see the browser being automated on other machines.

### 

### **Node**

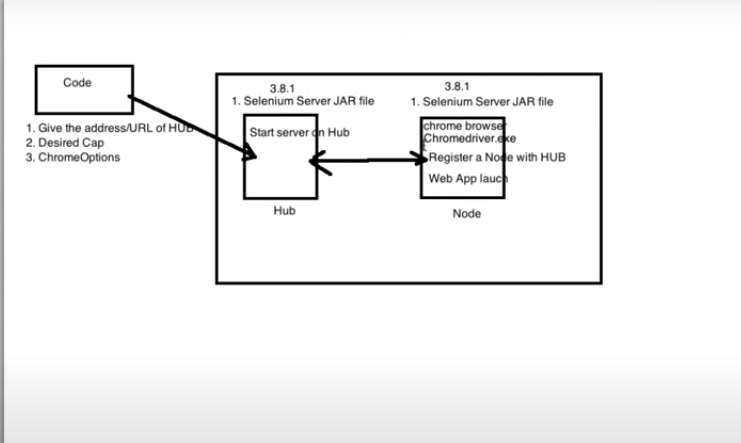
Nodes are the selenium instances which **will execute the test cases that you loaded on the hub.**

Nodes can be launched on multiple machines with different platforms and browsers.

Simple Selenium Grid architecture to setup both Hub and node in the same localhost.

Name of the Jar file is –**Selenium standalone server.jar**

With the help of this Jar file we can setup both Hub and Nodes.



**Brief Working :**

**Before writing the code Process :**

1. **Download Selenium0standloneserver.**
2. **We have to install Hub server in our Local.**
3. **We have to install Node server in our Local.**

**In the code/ Eclipse :**

1. In code, we should give the address/URL of Hub.
2. We should specify the DesiredCapabilities of the platform. Browser, version for Node.
3. We should use ChromeOptions in our code.

**For the Hub :**

1.For Hub , we need a Selenium standalone server jar file to be specified.

2. We need to start the Hub server on Hub.

**For Node to execute in chrome browser :**

1. For Hub , we need a Selenium standalone server jar file to be specified.
2. Chrome browser should be installed.
3. Chromedriver.exe should be present in node.
4. We need to Register a node with Hub.(Connecting Hub with node is called Registering.)

**Implementation :**

**Step 1** : Download Selenium standalone server jar file and keep it in a folder.

**Step 2** : Open command prompt at the location of your Selenium server jar file/ change path after opening cmd.

**Step 3 : Hub Configuration**

type the command to install Selenium Hub :

Java -jar selenium-server-standalone-3.141.59.jar -role hub (include .jar in the command)

**The following message will be displayed in your cmd.**

20:12:29.036 INFO [Hub.start] - Selenium Grid hub is up and running

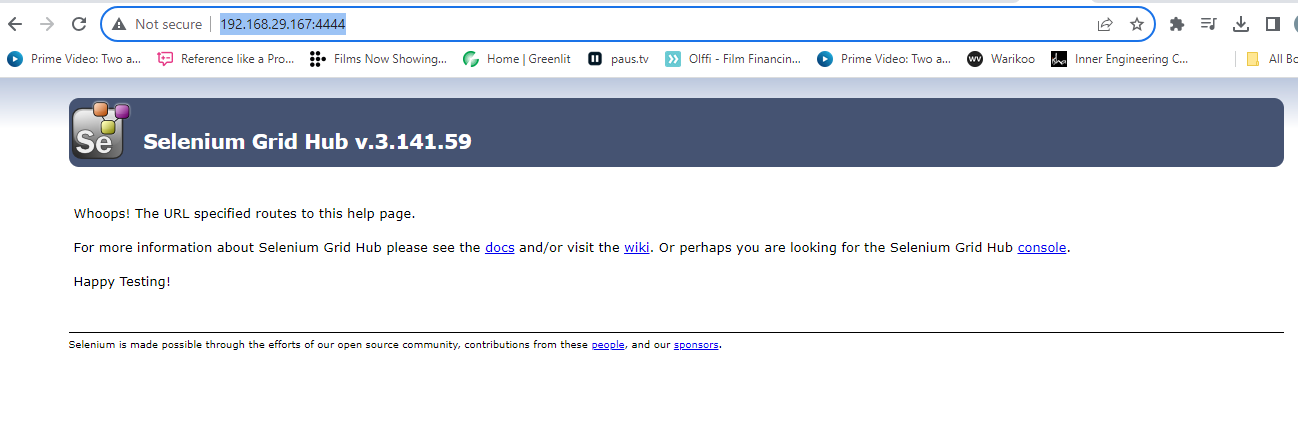
17:21:26.537 INFO [Hub.start] - Nodes should register to http://192.168.29.167:4444/grid/register/

17:21:26.538 INFO [Hub.start] - Clients should connect to <http://192.168.29.167:4444/wd/hub>

**Important note : Don't ever close your cmd after installing Hub, chances are there the hub config might be lost.**

The marked address is the address of your hub in your localhost.

Go to your browser and type the same address the following screen should be visible.



This confirms that your **Hub is up and running**

**Step 4** : **Node Configuration** .

Open another instance of cmd(Don't close the Hub installed cmd instance)

Type the following command :

java -Dwebdriver.chrome.driver="C:\Users\Dell\Documents\seleniumgrid\chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -hub http://192.168.56.1:4444/grid/register

After entering the command , we will get the following msg in cmd

20:17:45.784 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub

20:17:45.881 INFO [SelfRegisteringRemote$1.run] - Starting auto registration thread. Will try to register every 5000 ms.

20:17:46.284 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://192.168.29.167:4444/grid/register

20:17:46.386 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use

And also in other cmd the following msg will be displayed :

20:17:46.385 INFO [DefaultGridRegistry.add] - Registered a node http://192.168.29.167:22472

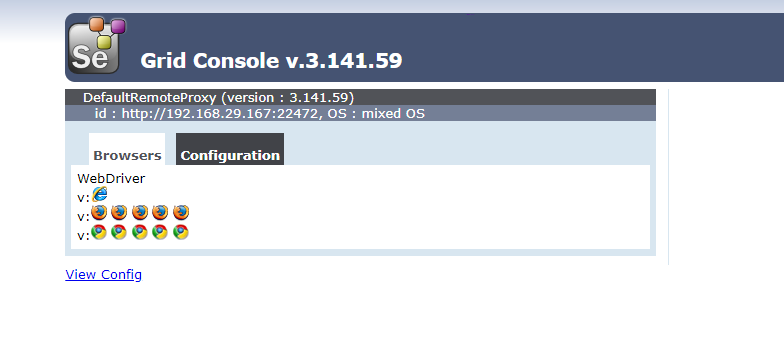
The node is registered in the 22472 port mentioned above , and hence :

**Hub address** : 192.168.29.167:4444

**Nodes address** : http://192.168.29.167:22472

And the following page should be displayed after clicking on Console option :

This is the **Selenium Grid Console**



**Step 5**

Now, It’s time to write the code in Eclipse

Create a Maven project and add selenium and testNg dependency (Avoid this step if already present)

**Important note : In your Maven Project, use the same dependency and version of the selenium which you have used for the standalone server jar file.**

**Step 6 :** In java class, add Desired Capabilities, set Capability browser and OS.

// Define Desired Capabilities

DesiredCapabilities des = new DesiredCapabilities();

des.setBrowserName("chrome");

des.setPlatform(Platform.WIN10);

**Step 7** : ChromeOptions Definition

ChromeOptions options = new ChromeOptions();

**Step 8 :**  Merging the ChromeOptions with the Desired capabilities, that the node is supposed to execute.

options.merge(des);

**Step 9** : We execute the Test with RemoteWebDriver providing the HubURL

String HubUrl = "http://192.168.29.167:4444/wd/hub";

WebDriver driver = new RemoteWebDriver(new URL(HubUrl),options);

driver.get("<https://www.selenium.dev/>");

Java class complete code :

public class GridTest {

public static void main(String[] args) throws MalformedURLException {

// Define Desired Capabilities

DesiredCapabilities des = new DesiredCapabilities();

des.setBrowserName("chrome");

des.setPlatform(Platform.WINDOWS);

//Chrome Options Definition.

ChromeOptions options = new ChromeOptions();

// here we are merging the ChromeOptions with the Desired capabilities, that the node is supposed to execute

options.merge(des);

// We execute the Test with RemoteWebDriver providing the HubURL and

String HubUrl = "http://192.168.29.167:4444/wd/hub";

WebDriver driver = new RemoteWebDriver(new URL(HubUrl),options);

driver.get("https://www.selenium.dev/");

System.out.println(driver.getTitle());

}

}

**Introduction to BDD-Cucumber**

**BDD – Behavioral Driven Development.**

**Approach to Develop,Test, maintain, Deliver the Software.**

**Other approaches :**

**TDD – Test Driven Development.**

# **What is BDD?**

BDD is a way for software teams to work that closes the gap between **business people** and **technical people** by:

BDD – 3 Amigos.

1. Amigos are – **Developers , Testers , Business Analyst/ Product Owner.**

Between these 3 Amigos , there is a constant Communication being conducted on daily basis.

Example –

Redbus

Encouraging collaboration across roles to build shared understanding of the problem to be solved

* Working in rapid, small iterations to increase feedback and the flow of value
* Producing system documentation that is automatically checked against the system’s behaviour

We do this by focusing collaborative work around concrete, real-world examples that illustrate how we want the system to behave. We use those examples to guide us from concept through to implementation, in a process of continuous collaboration.

# 

# 

# **What is Cucumber?**



Ok, now that you know that BDD is about discovery, collaboration and examples (and not testing), let’s take a look at Cucumber.

Cucumber reads executable specifications written in plain text and validates that the software does what those specifications say. The specifications consists of multiple *examples*, or *scenarios*. For example:

There are 3 Main Components of Cucumber Framework :

**Feature files** - Files that are written in Plain English Format.

**Gherkin** - language/Format that is used to Write Feature files.

**Step Definitions -** Are the Subsequent Java classes that are used to convert Feature files into Java selenium Code.

**This acts like a translator which translates English language into java.**

**Runner file-** Runner file takes the path of Feature file and Step Definition and Executes the code.

Ok, now that you know that BDD is about discovery, collaboration and examples (and not testing), let’s take a look at Cucumber.

Cucumber reads executable specifications written in **plain text** and validates that the software does what those specifications say. The specifications consists of multiple *examples*, or ***scenarios***

 For example:

**Scenario**: I want to test the Search box in my Youtube application.

**Given** I should open the browser enter url

**When** I type shahrukhkhan  in my search box

**And** I will press enter

**Then** All the songs related to srk should be displayed.

**Scenario** : I want to test the Amazon Fresh Feature In Amazon.in

**Given** : Open the browser and enter Amazon.in

**AND** make sure that you are using chrome browser

**When** Click on Fresh Link in top bar menu

**Then** The Amazon Fresh page should be displayed.

**AND** it should also display the SBI Card Advertisement.

Each scenario is a list of *steps* for Cucumber to work through. Cucumber verifies that the software conforms with the specification and generates a report indicating ✅ success or ❌ failure for each scenario.

Each scenario is a list of *steps* for Cucumber to work through. Cucumber verifies that the software conforms with the specification and generates a report indicating ✅ success or ❌ failure for each scenario.

In order for Cucumber to understand the scenarios, they must follow some basic syntax rules, called [Gherkin](https://cucumber.io/docs/gherkin/).

**Installation of Cucumber :**

1. Go to Eclipse Marketplace, Help-> Eclipse Marketplace -> Search Cucumber and install Cucumber.
2. Add following Dependencies to Pom.xml

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-java</artifactId>

<version>7.2.3</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit</artifactId>

<version>7.2.3</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-testng -->

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-testng</artifactId>

<version>7.14.0</version>

</dependency>

# **What is Gherkin?**

Gherkin is a set of grammar rules that makes plain text structured enough for Cucumber to understand. The scenario above is written in Gherkin.

Gherkin serves multiple purposes:

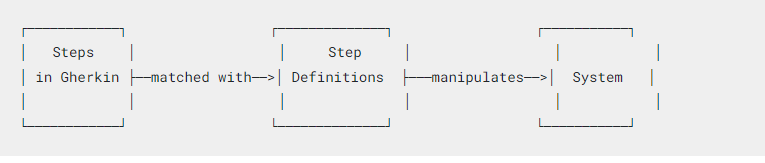
* Unambiguous executable specification
* Automated testing using Cucumber
* Document how the system *actually* behaves

Gherkin documents are stored in .feature text files and are typically versioned in source control alongside the software.

# 

# **What are Step Definitions?**

[Step definitions](https://cucumber.io/docs/cucumber/step-definitions) connect Gherkin steps to programming code. A step definition carries out the action that should be performed by the step. So step definitions hard-wire the specification to the implementation.



**Day 8 :**

**BDD-Cucumber implementation with Junit.**

1. Setting up the Project:

* Create a new Maven project in your preferred IDE (Eclipse, IntelliJ, etc.).
* Add the necessary dependencies to your pom.xml file:

<dependencies>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-java</artifactId>

<version>6.10.4</version> <!-- Use the latest version available -->

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit</artifactId>

<version>6.10.4</version> <!-- Use the same version as cucumber-java -->

<scope>test</scope>

</dependency>

</dependencies>

**2. Feature File:**

* Create a new directory named features in src/test/resources to store your feature files.
* Inside the features directory, create a feature file (e.g., sample.feature) with BDD scenarios:

Feature: Sample Feature

Scenario: Login with valid credentials

Given the user is on the login page

When the user enters valid username and password

Then the user should be logged in successfully

**3. Step Definitions:**

* Create a new package (e.g., stepdefinitions) in src/test/java to store your step definition classes.
* Inside the package, create a Java class (e.g., StepDefinitions) to define step definitions:

import io.cucumber.java.en.Given;

import io.cucumber.java.en.When;

import io.cucumber.java.en.Then;

public class StepDefinitions {

@Given("the user is on the login page")

public void navigateToLoginPage() {

// Implementation to navigate to the login page

}

@When("the user enters valid username and password")

public void enterValidCredentials() {

// Implementation to enter valid credentials

}

@Then("the user should be logged in successfully")

public void verifyLoginSuccess() {

// Implementation to verify successful login

}

}

**4. Test Runner:**

* Create a new Java class (e.g., TestRunner) in src/test/java to run your Cucumber tests:

import io.cucumber.junit.Cucumber;

import io.cucumber.junit.CucumberOptions;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)

@CucumberOptions(

features = "src/test/resources/features",

glue = "stepdefinitions"

)

public class TestRunner {

}

5. Running the Tests:

* Right-click on the TestRunner class and select "Run as JUnit Test" to execute your Cucumber tests.
* Cucumber will read the feature files, match the steps with step definitions, and execute the scenarios.

**BDD-Cucumber implementation with TestNG.**

BDD with Cucumber and TestNG combines the Cucumber framework, which uses Gherkin syntax for behavior-driven development, with TestNG for test execution in Java. The process involves writing feature files in Gherkin, implementing step definitions in Java using TestNG annotations, and executing the tests with a TestNG runner

1. Setting up the Project:

* Create a new Maven project in your preferred IDE (Eclipse, IntelliJ, etc.).
* **Add the necessary dependencies to your** pom.xml **file:**

**<dependencies>**

**<dependency>**

**<groupId>io.cucumber</groupId>**

**<artifactId>cucumber-java</artifactId>**

**<version>6.10.4</version> <!-- Use the latest version available -->**

**<scope>test</scope>**

**</dependency>**

**<dependency>**

**<groupId>io.cucumber</groupId>**

**<artifactId>cucumber-testng</artifactId>**

**<version>6.10.4</version> <!-- Use the same version as cucumber-java -->**

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

2. Feature File:

* Create a new directory named features in src/test/resources to store your feature files.
* Inside the features directory, create a feature file (e.g., sample.feature) with BDD scenarios:

**Feature: Sample Feature**

Scenario: Login with valid credentials

Given the user is on the login page

When the user enters valid username and password

Then the user should be logged in successfully

**3. Step Definitions:**

* Create a new package (e.g., stepdefinitions) in src/test/java to store your step definition classes.
* Inside the package, create a Java class (e.g., StepDefinitions) to define step definitions:

**i**mport io.cucumber.java.en.Given;

import io.cucumber.java.en.When;

import io.cucumber.java.en.Then;

public class StepDefinitions {

@Given("the user is on the login page")

public void navigateToLoginPage() {

// Implementation to navigate to the login page

}

@When("the user enters valid username and password")

public void enterValidCredentials() {

// Implementation to enter valid credentials

}

@Then("the user should be logged in successfully")

public void verifyLoginSuccess() {

// Implementation to verify successful login

}

}

4. Test Runner:

* **Create a new Java class (e.g.,** TestRunner**) in** src/test/java **to run your Cucumber tests:**

**import io.cucumber.testng.AbstractTestNGCucumberTests;**

**import io.cucumber.testng.CucumberOptions;**

**@CucumberOptions(**

**features = "src/test/resources/features",**

**glue = "stepdefinitions"**

**)**

**public class TestRunner extends AbstractTestNGCucumberTests {**

**}**

5. Running the Tests:

* Right-click on the TestRunner **class and select "Run as TestNG Test" to execute your Cucumber tests.**
* **Cucumber will read the feature files, match the steps with step definitions, and execute the scenarios.**

**Day 9 :**

**Extent Report**

### **What are Extent Reports?**

Extent Reports is an open-source reporting library useful for test automation. It can be easily integrated with major testing frameworks like [JUnit, NUnit](https://www.browserstack.com/guide/junit-vs-nunit), TestNG, etc. These reports are HTML documents that depict results as pie charts. They also allow the generation of custom logs, snapshots, and other customized details.

Once an automated test script runs successfully, testers need to generate a test execution report. While TestNG does provide a default report, they do not provide the details.

### Using Extent Reports in Selenium

Extent Reports in [Selenium](https://www.browserstack.com/selenium) contain two major, frequently used classes:

* ExtentReports class
* ExtentTest class

Syntax

ExtentReports reports = new ExtentReports("Path of directory to store the resultant HTML file", true/false);

ExtentTest test = reports.startTest("TestName");

The ExtentReports class generates HTML reports based on a path specified by the tester. Based on the Boolean flag, the existing report has to be overwritten or a new report must be generated. ‘True’ is the default value, meaning that all existing data will be overwritten.

The ExtentTest class logs test steps onto the previously generated HTML report.

Both classes can be used with the following built-in methods:

* **startTest**: Executes preconditions of a test case
* **endTest**: Executes postconditions of a test case
* **Log**: Logs the status of each test step onto the HTML report being generated
* Flush: Erases any previous data on a relevant report and creates a whole new report

A Test Status can be indicated by the following values:

* **PASS**
* **FAIL**
* **SKIP**
* **INFO**

Syntax

reports.endTest();

test.log(LogStatus.PASS,"Test Passed");

test.log(LogStatus.FAIL,"Test Failed");

test.log(LogStatus.SKIP,"Test Skipped");

test.log(LogStatus.INFO,"Test Info");

The Log method takes into account two parameters, the first being the test status and the second being the message to be printed onto the generated report.

**Assertions using TestNG**

## **What are Assertions in TestNG?**

*Assertions in TestNG are a way to verify that the expected result and the actual result matched or not***. If we could decide the outcome on different small methods using assertions in our test case, we can determine whether our test failed or passed overall.**

### ***Syntax for TestNG Assertions:***

**Although there are many methods for assertions (*later in this article*), the generic syntax is:**

*Assert.Method(actual, expected)*

**The parameter as you see contains three values:**

* *Actual***: *The actual value that the tester gets like if the tester's assertion is on the title of the page then what was the actual title of the page goes here*.**
* *Expected***: *The value that you expect like if the tester's assertion is on the title of the page then what value of title do you expect goes here*.**

**Introduction to GIT & GitHUB**

Git is a distributed version control system, and GitHub is a web-based platform utilizing Git for collaborative software development, offering features like repositories, branching.

**@Test (priority = -1)**

**public void OpenBrowser() {**

**driver.get("https://www.demoqa.com");**

**String expectedTitle = "Free QA Automation Tools For Everyone";**

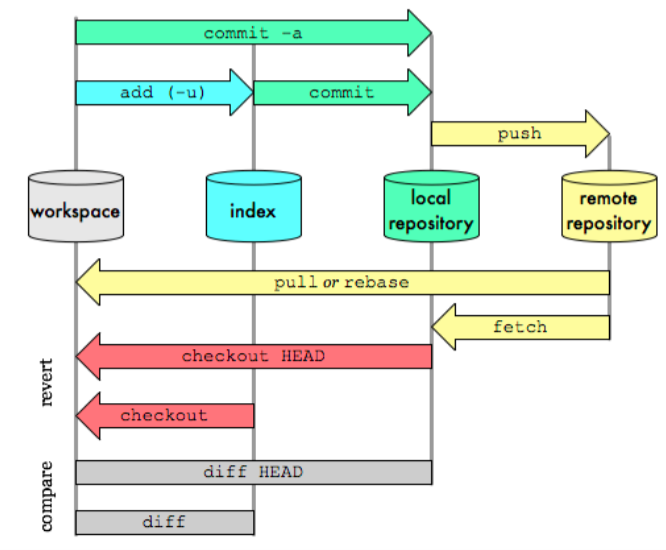
**String originalTitle = driver.getTitle();**

**Assert.assertEquals(originalTitle, expectedTitle);**

**Day 10:**

**Git & Github Commands Implementing**

**Git hub workflow and Architecture.**

****

**Git workflows can be divided into 2 sections :**

1. **Local area**
2. **Remote area**

**In Local area we have :**

1. **Working directory/ Working copy**

Whatever changes that need to be made like update in code/ adding new code is done in working copy. In IDE like Eclipse/VScode.

1. **Staging area / Index**

Staging area. Index is used as an intermediate storing of code/ files where staging maintains the files and also gives it a index.

The index is not in sequential order like(0,1.2,3)

1. **Local Repository :**

Local repository acts like an intermediate between Remote repository and Staging area.

If we want to send code to a remote repository then we can only do it from Local repository and not directly from the working copy or from Staging area.

Whatever code/ files that were added in staging area are committed to Local repository to be pushed to the Remote repository.

Local repository has a .git file that has all the data.

**In Remote area we have :**

**Remote Repository :** It is a repository which is stored in Cloud/ Internet using Github or some other storage repo softwares.

It has a master branches and different branches for different implementations.

ssh-keygen -t ed25519 -C "your\_email@example.com"

**Different workflows in Git :**

**Workflow in order – > 1)Git clone 2) Checkout to local branch**

**3) Git add 4) Git commit 5) Git Push 6) Merge with Pull request to Master**

Git commit -M ”Commit message”

1. **Cloning :**

Cloning is a process of downloading the code which is available in the remote repository to the Local repository.

Cloning is usually done when a new developer wants to work with an existing code, first he downloads the code from remote repository to local repository and then starts working on code in his working copy.

The command used is : **git clone <URL\_of\_RemoteRepo>**

1. **Git add <filename> / Git add \*(for all files)**

This command moves the code from working directory to Staging area.

1. **Git commit :**

This command is used when we wish to move the code from Staging area to the Local repository.

Command - **Git commit -m ”message to be given while committing”**

**Git Branching -**

Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes

1. **Fetch V/s Pull**

**Git Pull -** Git pull directly pulls the code from Remote repository to your working directory.

There is no involvement of Local repository in Git Pull.

**Git fetch** - Git fetch will fetch the code from your remote repository to your local repository.(with a .git file)

Now, if we want to get the code from Local repository to Working Directory, then we use **Merge**.

**Git Pull = Git Fetch + Merge**

**We can say that when we do Git Pull , it is equivalent to fetch and merge.’**

**Note : Merging can be done both at Remote repo level and also at Local level**

Lets try to understand the concepts with a real time scenarios using examples :

Assume we have in a team of 4 :

1. Team Lead - is developing scripts for Search feature
2. Senior QA1 - is developing scripts for Payment feature
3. Senior QA2 - is developing scripts for Cart feature
4. Associate QA - is developing scripts for Login feature

Now Lets start with SQA1 -

First he will Clone the code from the main branch to his working directory ,

Then he starts working on his assigned feature which is Payment feature.

So first clown ,then he checks out to a branch by creating a branch Payment feature in his Working Directory.

Note : **SQA1 cannot directly push the code to the master branch , it is a bad practice because there might be mistakes and corrections in his existing code, he cannot push the code directly to the main branch without a proper review**.

After checking out to his feature branch, SQA1 will add , commit and do Git push to the Payment feature in the Remote repository .

After this , he has to raise a Pull request to TL1 for Review of his code.

**Pull Request -** Pull request process is done when a developer need to get his code reviewed from his lead , he pushes the code(add+commit+push) from his working copy to the remote repository into the feature branch and then he sends a Pull request to his TL for the review.

**If the code is not accepted/ needs to make some changes**

**Then** TL will give review comments and say that the Pull request is rejected and requests him to make the changes.

**Then** SQA1 will make the necessary changes

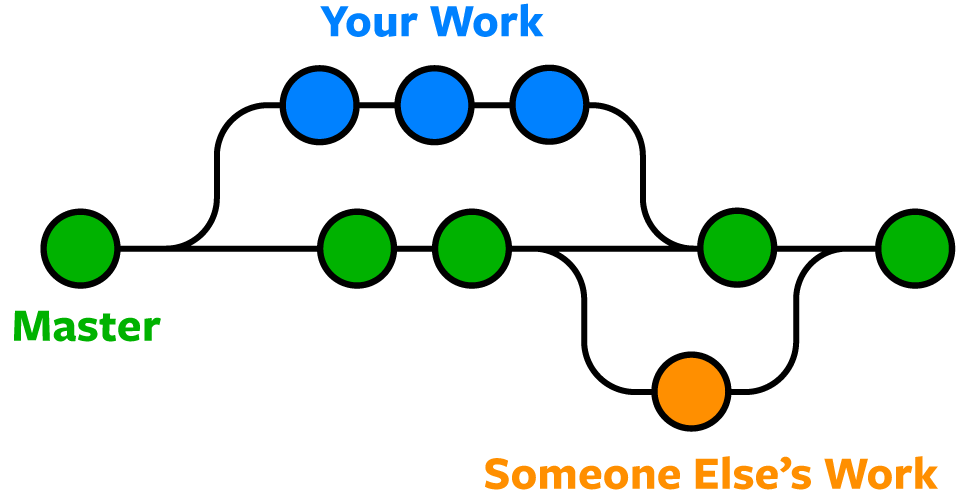
Then again SQA1 will again push the updated code to the remote repository(add+commit+ push) to the feature branch and sends a updated Pull request to his TL.

**TL will again review the code, now the code is ok to be accepted,**

**Then** TL will accept the pull request and he will merge the feature branch code to the Master branch.

**This is the process of Pull request in real time organisations.**

**Creating/Cutting the branches from master and Versions.**

****

**Lets assume that we have a master branch with a version v1.0**

Whenever , a new person lets say Associate QA, will start working on a new feature , they first have to cut/ create a new branch from the master branch, then push the code to the remote repo in that feature branch and then merge it to the master branch with a Pull Request process

WHen AQA pushes the code and with a PR merges the code to the master branch then we can say that the now the version of the master branch is V1.1

**Now whenever someone else writes a new code, he/she needs to pull the code from remote of the latest version that is available**

Lets say SQA2 wants to add new code in his feature then first he needs to make sure that the version is updated in his Working directory that is V1.1

So first he needs to Pull the latest version from Remote repo which is V1.1 and then make changes in his code then do the push(add+commit+Push) to feature branch then merge the code with Pull request process.

**Now this new merge will create a new version of Master called as V1.2.**

**Note - Creating a new branch can be done from any branch.**

**For example -** we can create a new branch out of master . or we can create a sub branch of a newly created branch also**.**

**At the end , we just need to make sure that all the branches with proper PR process are merged with master branch.**

**Link foe reference for Git commands and more notes on Git.**

https://krishnaiitd.github.io/gitcommands/git-workflow/

**Introduction to CI-CD Jenkins & Integration with Selenium**

Jenkins is a well-known open-source DevOps tool used in many organizations’ CI/CD pipelines. Selenium is a popular open-source automation tool for testing websites.

Because of its user-friendliness and community help desk, Selenium enjoys widespread adoption in the testing industry. There is no better example of continuous integration and deployment than the Jenkins Selenium combination.

Jenkins, created by Hudson Lab, is the most popular open-source continuous integration application. It’s cross-platform, meaning it works with Mac OS X, Linux, and Solaris. Jenkins is written in Java. Whether it’s an SVN checkout, a cron job, or the status of an application, Jenkin can keep tabs on it all. When a specific job condition is met, it will trigger the specified actions.

* Whenever you change your code, Jenkins can automatically perform your Selenium tests and then deploy your code to a new environment if the tests pass.
* Jenkins allows you to set a time and date for executing your tests.
* The Test Reports and performance logs can be saved for future reference.
* Jenkins can be used in a continuous integration setup with Maven to create and test a project.

### **Steps to install Jenkins & Selenium**

**Configuring Jenkins**

You can install Jenkins from the Jenkins website on your computer. Jenkins may be launched from the command line or run on a web application server.

**For command-line implementation, see the details below.**

* At the command prompt, execute java -jar followed by the location of a .war file.
* If your Jenkins.war file has begun to run, you can confirm this by pressing enter and viewing the console output.
* Now you can see if Jenkins is set to go by using the default port of 8080.
* Just type “http://localhost:8080” into your browser’s address bar. Your Jenkins user interface will load.

### **Integrating Jenkins with Selenium (using Jenkins Dashboard)**

* To initiate a fresh project, go to the Jenkins dashboard and select the New Item button.
* Choose the Freestyle Project choice and enter the project’s name. Proceed with the **OK** button.
* Fill out the Description box under **General** Section to briefly overview the project.
* Select “**None**” under “Source Code Management.”
* When deciding how to schedule the jobs, we can pick the appropriate choice from the provided option. To schedule the tasks, let’s go ahead and set a time limit.