Class: 11

- Java Input/output
- Java Collection Framework
- Array List
- Linked List

File Handling in Java

- File handling in Java implies reading from and writing data to a file.
- The File class from the java.io package, allows user to work with different formats of files.
- There are many ways to read a file in java. **We can use Buffered Reader, File Reader or Files class.**
- **We can use File Writer, Buffered Writer, Files or FileOutputStream to write file in java.**
- Provided below are few important methods for File Class

Method	Type	Description
canRead()	Boolean	It tests whether the file is readable or not
canWrite()	Boolean	It tests whether the file is writable or not
createNewFile()	Boolean	This method creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	It tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
mkdir()	Boolean	Creates a directory

Maintain Folder

Create a Folder

```
File fileObject = new File("D:\\SeleniumOnlineTraining");

fileObject.mkdir();
```

Check the existence of any Folder

```
File fileObject = new File("D:\\SeleniumOnlineTraining");

boolean a = fileObject.exists();

if (a == true){

System.out.println("Folder Exists");

}

else {

System.out.println("Folder Not Exists");

}
```

Delete a Folder

```
File fileObject = new File("D:\\SeleniumOnlineTraining");

fileObject.delete();
```

Maintain File

Create a Text File

```
File file = new File("D:\\Selenium\\Test1234.txt");

file.createNewFile()

System.out.println(file.canRead());

System.out.println(file.canWrite());
```

```
System.out.println(file.getAbsolutePath());

System.out.println(file.getName());

System.out.println(file.length());
```

Check the existence of any File

```
File fileObject = new File("D:\\Selenium\\Test1234.txt");

boolean a = fileObject.exists();

if (a == true){

System.out.println("File Exists");

}

else {

System.out.println("File Not Exists");

}
```

Delete a Files

```
File fileObject = new File("D:\\Selenium\\Test1234.txt");

fileObject.delete();
```

Write Data to a Text File

```
FileWriter file = new FileWriter("D:\\Selenium\\Test1234.txt");

BufferedWriter bw = new BufferedWriter(file);

String data = "Welcome to Java";

bw.write(data);

bw.close();

file.close();
```

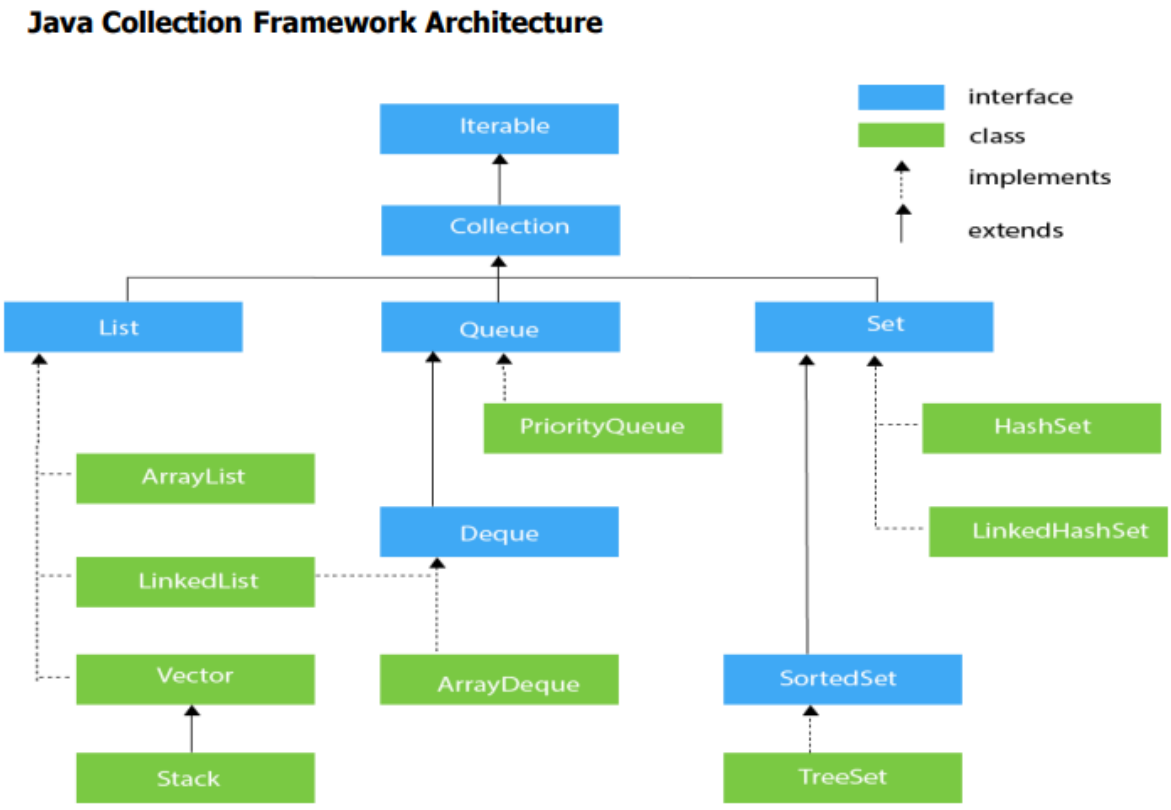
Read the data from a file

```
String line;
FileReader file = new FileReader("D:\\Selenium\\Test1234.txt");
BufferedReader br = new BufferedReader(file);
while ((line = br.readLine()) !=null){
System.out.println(line);
}
br.close();
file.close();
```

Java Collection Framework

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
- A collections framework is a unified architecture **for** representing and manipulating collections. All collections frameworks contain the following:
- Interfaces − These are **abstract** data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- Implementations, i.e., Classes − These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- Algorithms − These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection **interface**.
- All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion, etc. can be achieved by Java Collections.
- Java Collection means a single unit of objects. Java Collection framework provides many

interfaces (Set, List, Queue, Deque, etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet, etc.).



Java Collection Framework Interface Concepts

Iterable Interface: The Iterable **interface** is the ***root* interface for** all ***the*** collection ***classes***. The ***Collection* interface extends** the ***Iterable* interface** and therefore ***all*** the ***subclasses*** of ***Collection* interface** also implement ***the*** Iterable **interface**.

Collection *Interface*: The ***Collection* interface** is the **interface** which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can

say that the Collection **interface** builds the foundation on which the collection framework depends.

List Interface: List **interface** is the child **interface** of Collection **interface**. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

Set Interface: Set Interface in Java is present in java.util **package**. It **extends** the Collection **interface**. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one **null** value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

SortedSet Interface: SortedSet is the alternate of Set **interface** that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

Queue Interface: Queue **interface** maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which **implements** the Queue **interface**.

Deque Interface: Deque **interface extends** the Queue **interface**. In Deque, we can remove and add the elements from both the side. Deque stands **for** a **double**-ended queue which enables us to perform the operations at both the ends.

Java ArrayList

- Using ArrayList we can overcome the size issue. ArrayList is a resizable array.
- ArrayList inherits AbstractList **class** and **implements** List **interface**.
- ArrayList allows duplicate elements.
- Java ArrayList **class** maintains insertion order.

- In Java ArrayList **class**, manipulation is slow because a lot of shifting needs to be occurred **if** any element is removed from the array list.
- An ArrayList can be Sorted by using the sort() method of the Collections Class in Java. This sort() method takes the collection to be sorted as the parameter and returns a Collection sorted in the Ascending Order by **default**.

Example:

```
ArrayList<String> list=new ArrayList<String>(); //Creating arraylist
list.add("Ravi"); //Adding object in arraylist
list.add("Ram");
list.add("Ravi");
list.add("Shyam");
for(String obj:list)
System.out.println(obj);
```

Java LinkedList

- Java LinkedList **class** uses doubly linked list to store the elements. It provides a linked-list data structure.
- It inherits the AbstractList **class** and **implements** List and Deque interfaces.
- Java LinkedList **class** can contain duplicate elements.
- Java LinkedList **class** maintains insertion order.
- In Java LinkedList **class**, manipulation is fast because no shifting needs to be occurred.

Example:

```
LinkedList<String> llist = new LinkedList<String>();

// Adding elements using add method

llist.add("IDE");

llist.add("RC");

llist.add("WebDriver");

llist.add("Grid");

for(String str:llist)

System.out.println(str);

llist.set(2, “Selenium 3“); // Updating element using set method

list.remove(3); // Removing element using remove method

llist.addFirst(“Selenium 2"); // Adding element to first position using addFirst method

for(String str1:llist)

System.out.println(str1);
```

Class 12

- Hash Set
- Tree Set
- Hash Map
- Java Multi-Threading

Java Hash Set

- Java HashSet **class** is used to create a collection that uses a hash table **for** storage. It inherits the AbstractSet **class** and **implements** Set **interface**.
- HashSet doesn't allow duplicates. If you **try** to add a duplicate element, the old value would be overwritten.
- HashSet allows **null** values however **if** you insert more than one nulls it would still **return** only one **null** value.
- HashSet doesn't maintain any order, the elements would be returned in any random order.
- HashSet is the best approach **for** search operations.
- Java HashSet can be used to add elements from another Collection

Example:

```
HashSet<String> set=new HashSet <String>();
set.add("UFT");
set.add("Selenium");
set.add("Java");
set.add("SoapUI");
set.add("Silk");
```

```
java.util.Iterator<String> i = set.iterator();

while (i.hasNext())

System.out.println(i.next());
```

Java Tree Set

- TreeSet is one of the most important implementations of the SortedSet **interface** in Java that uses a Tree **for** storage.
- Java ***TreeSet* class implements** the ***Set* interface** that uses ***a*** tree **for** storage. It ***inherits*** AbstractSet .
- The objects of the TreeSet **class** are stored in ascending order
- Java TreeSet **class** contains unique elements only like HashSet.
- Java TreeSet **class** doesn't allow null element.

Example:

```
TreeSet<Integer> ts=new TreeSet<Integer>();

ts.add(10);

ts.add(20);

ts.add(10);

ts.add(30);

ts.add(20);

//Traversing elements

Iterator<Integer> itr=ts.iterator();

while(itr.hasNext()){
```

```
System.out.println(itr.next());
}
```

Java HashMap

Java HashMap **class implements** the map **interface** by using a hash table. It inherits AbstractMap **class** and **implements** Map **interface**.

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values.

HashMap doesn't allow duplicate keys but allows duplicate values. That means A single key can't contain more than 1 values but more than 1 key can contain a single value.

This **class** makes no guarantees as to the order of the map.

Example:

```
HashMap<String, String> langauges = new HashMap<String, String>();
// Add keys and values (State, Langage)
langauges.put("West Bengal", "Bengali");
langauges.put("UP", "Hindi");
langauges.put("Bihar", "Bhojpuri");
langauges.put("Tamilnadu", "Tamil");
System.out.println(langauges);
// Print keys and values
for (String i : langauges.keySet()) {
System.out.println( i + " " + langauges.get(i));
```

}

Multithreading in Java

- Multithreading is a mechanism that allows concurrent execution of two or more parts of a program **for** maximum utilization of CPU. Each part of such program is called a thread. Threads are light-weight processes within a process. Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.
- Threads can be created by using two ways:

1. Extending the Thread **class**

2. Implementing the Runnable Interface

Thread Life Cycle:

New: In **this** phase, the thread is created using **class** "Thread class". It remains in **this** state till the program starts the thread. It is also known as born thread.

Runnable: In **this** phase, the instance of the thread is invoked with a start method. The thread control is given to scheduler to finish the execution. It depends on the scheduler, whether to run the thread.

Running: When the thread starts executing, then the state is changed to "running" state. The scheduler selects one thread from the thread pool, and it starts executing in the application.

Waiting: This is the state when a thread has to wait. As there multiple threads are running in the application, there is a need **for** synchronization between threads. Hence, one thread has to wait, till the other thread gets executed. Therefore, **this** state is referred as waiting state.

Dead: This is the state when the thread is terminated. The thread is in running state and as soon as it

completed processing it is in "dead state".

After starting a thread, it can never be started again. If user does so, an IllegalThreadStateException is thrown. In such **case**, thread will run once but **for** second time, it will **throw** exception.

Multithreading in Java

For every Java program, there will be a **default** thread created by JVM which is nothing but Main Thread. The entry point **for** Main Thread is the main() method.

```
Thread ct = Thread.currentThread();
System.out.println(ct);
System.out.println(ct.getName());
System.out.println(ct.getPriority());
System.out.println(ct.getThreadGroup());
System.out.println(ct.getId());
System.out.println(ct.getState());
System.out.println(ct.isAlive());
ct.setName("Test");
ct.setPriority(5);
System.out.println(ct.getName());
System.out.println(ct.getPriority());
```

Multithreading in Java

```
class TestThread extends Thread{

public void run(){

System.out.println("Thread is started to run");

}

public static void main(String args[]){

TestThread t1=new TestThread();

t1.start();

}

}

class TestThread2 implements Runnable{

public void run(){

for(int i=1; i<=10; i++) {

System.out.println("User Thread Value:"+i);

}

} }

public static void main(String args[]){

TestThread2 m1=new TestThread2();

Thread t1 =new Thread(m1);

t1.start();

}

}
```

Daemon Thread in Java

- Daemon thread is a low priority thread (in context of JVM) that runs in background to perform tasks such as garbage collection etc., they **do** not prevent the JVM from exiting (even **if** the daemon thread itself is running) when all the user threads (non-daemon threads) finish their execution. JVM terminates itself when all user threads (non-daemon threads) finish their execution, JVM does not care whether Daemon thread is running or not, **if** JVM finds running daemon thread (upon completion of user threads), it terminates the thread and after that shutdown itself. Main Thread can not be set to Daemon Thread.
- To set a thread to be a daemon thread, user need to **do** is to call Thread.setDaemon(). In **case** user call the setDaemon() method after starting the thread (start() method), it would **throw** IllegalThreadStateException. This clearly means that user can call setDaemon() method only before starting a thread.
- To check **if** a thread is a daemon thread, user can simply call the method isDaemon().

Daemon Thread in Java

```
class TestThread extends Thread{
public void run(){
if(Thread.currentThread().isDaemon()){
System.out.println("Daemon thread executing");
}
else{
System.out.println("user(normal) thread executing");
}
```

```
public static void main(String args[]){

TestThread t1=new TestThread();

TestThread t2=new TestThread();

t1.setDaemon(true);

t1.start();

t2.start();

}
```

Class 13

- Handling Frames
- Capture Screenshots
- Implicit and Explicit Wait
- Headless Browser Testing

What is Frame?

- Frame is a web page which is embedded in another web page or an HTML document embedded inside another HTML document. The Frame is often used to insert content from another source, such as an advertisement, into a Web page. The tag specifies a frame.

How can user identify in case any Frame is available in a webpage?

- User cannot detect the frames by just seeing the page.
- Open the application using Internet Explorer or Chrome browser. Right click on the page, if user is able to find the option like 'Reload frame' then there must be Frame inside the web page.
- Click on F12 button from keyboard. Press Ctrl + F and try to search with word "iframe". If user is able to find some search result which means some Frame is available in the webpage.
- User can even identify total number of iframes available in web page by using below

code:

```
int size = driver.findElements(By.tagName("iframe")).size();

System.out.println(size);
```

Handling Frames

To Switch between Frames user, have to use the driver's switchTo().frame command. User can use the switchTo().frame() in below mentioned 3 ways:

- **switchTo.frame(int frameNumber):** Pass the frame index and driver will switch to that frame.
- **switchTo.frame(string frameNameOrId):** Pass the frame element Name or ID and driver will switch to that frame.
- **switchTo.frame(WebElement frameElement):** Pass the frame web element and driver will switch to that frame.

Capture Screenshot

- Capturing the screenshots for the test execution is one of the important scripts in any test automation which helps user to identify defects and problems by referring the screenshot. So, Selenium WebDriver provides TakesScreenshot interface to capture a screenshot of the webpage when an exception or error occurred during the execution of code.

- When a tester test any application, may be many of the functionalities not working compared to the expected result. So, tester logs these defects and tell the developers about the getting of defects in the code and suggest to fix the defects or bugs to retest

once again. However, developers never accept a defect without required evidence and details. Developers need the required evidence to analyze the issues. That's why taking the screenshot is one of the mandatory evidence to view and understand why the test has failed. User can take the screenshot in image files like png, jpeg, or jpg.

- Some of the possible scenarios where user might need to capture screenshot using Selenium WebDriver:

i. Application issues

ii. Assertion Failure

iii. Difficulty to find Webelements on the web page

iv. Timeout to find Webelements on the web page

v. Exception or error occurs during run time

Process to Capture Screenshot

Capturing Screenshot in Selenium WebDriver is a 3 Step process:

Since TakesScreenshot is an interface, user cannot create an object directly. User need to convert the web driver object to TakesScreenshot.

```
TakesScreenshot screenshot =((TakesScreenshot)driver);
```

User need to call getScreenshotAs() method to create an image file which is

provided by the TakesScreenshot interface to capture the screenshot of the web page displayed in the driver object.

File srcFile= screenshot.getScreenshotAs(OutputType.FILE);

Here, the argument “OutputType.FILE” specified in the getScreenshotAs() method will return the capture screenshot in the form of file. Therefore, user can store in the variable ‘srcFile’ with type File.

Save the file object returned by getScreenshotAs() method using copy() method of the FileHandler class.

FileHandler.copy(srcFile, destFile);

Implicit and Explicit Wait

- When a page is loaded by the browser the elements which user want to interact might take some time for complete load as most of the web applications are developed using Ajax and Javascript. It makes user difficult to identify the element and in case element is found due to page load, it will throw an "ElementNotVisibleException" exception. Using Waits, user can resolve this problem.

- **Implicit Wait:** Implicit waits are used to provide a default waiting time between each consecutive test step/command across the entire test script. Thus, the subsequent test step would only execute when that have elapsed after executing the previous test step/command. The default setting is 0. Once user set the time, web driver will wait for that time before throwing an exception.

  Syntax: driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

- **Explicit Wait :** Explicit wait is used to pause the execution until the time a particular condition is met or the maximum time has elapsed. Explicit waits are applied for a particular instance only. Explicit wait gives better options than that of an implicit wait as it will wait for dynamically loaded Ajax elements.

welement = wait.until(Syntax: WebDriverWait wait = new WebDriverWait(driver, timeout);

ExpectedConditions.elementToBeClickable(locator));

welement.click();

Usage of Explicit Wait

Explicit waits can be used in the following cases:

1. find single web element

2. find multiple web elements

3. check web page title and URL

4. check element's status

5. interact with frames

Expected Conditions user can use in Explicit Wait

alertIsPresent()

elementToBeClickable()

elementToBeSelected()

urlMatches()

presenceOfAllElementsLocatedBy()

presenceOfElementLocated()

textToBePresentInElement()

textToBePresentInElementLocated()

textToBePresentInElementValue()

titleIs()

titleContains()

visibilityOf()

visibilityOfAllElements()

visibilityOfAllElementsLocatedBy()

visibilityOfElementLocated()

Headless Browser Testing

- Performing test execution of the web applications without opening a browser is called headless browser testing. The Headless browser acts similar to a normal web browser.
- User have full control over the web pages loaded into the headless browsers. The Only difference is user will not be able to see a graphical user interface.

  User have no option other than using headless testing when uesr machine does not have a GUI, for instance, if user want to run tests in Unix.
- It is recommended to use the headless browser when tests are executed in parallel as User Interface based browsers consume a lot of Memory / resources. So headless browsers can be used for server-side performance testing too.
- Anyone can run test case using headless browsers then user will get the result just in seconds.
- When tester have to run the test case on the remote machine or server, which does not have any browser, but still user have to execute test case then they can try with headless browser.

Headless Browser Testing Examples

Some of the examples of Headless Drivers include:

- HtmlUnit
- Ghost
- PhantomJS
- Chrome
- ZombieJS

- Watir-webdriver

Headless Browser Testing using HTMLUnitDriver

- HTML UnitDriver is the lightest weight and fastest implementation headless browser for of WebDriver. It is based on HtmlUnit. It is known as Headless Browser Driver. It is same as Chrome, IE, or FireFox driver, but it does not have GUI.
- Key Features of HTML unit driver

❖ Support for the HTTPS and HTTP protocols

❖ Support for HTML responses (clicking links, submitting forms, walking the DOM model of the HTML document etc.)

❖ Support for cookies

❖ Proxy server support

❖ Excellent JavaScript support

❖ Ability to customize the request headers being sent to the server

❖ Ability to determine whether failing responses from the server should throw exceptions or should be returned as pages of the appropriate type

Syntax: WebDriver driver = new HtmlUnitDriver();

Class : 14

- Handle Cookies
- By All Locator
- How to select an option from auto suggest dropdown

Handling Cookies with Selenium Webdriver

- Cookies are files stored in local computers containing information submitted by the websites visited by a user. The information is stored in key-value pairs and allows a particular website to customize its content as per the user.
- Every time the user loads the website, the browser sends the cookie back to the server to notify the website of the user's previous activity.
- Cookies have a specific life span defined by their creators. At the end of this, a cookie becomes expired. Cookies often track information like how frequently the user visits, what are the times of visits, what banners have been clicked on, what button clicked, user preferences, items in the shopping cart, etc. This allows the site to present you with information customized to fit your needs.

- Cookies are domain-specific, i.e., a domain cannot read or write to a cookie created by another domain. This is done by the browser for security purposes.
- Cookies are browser-specific. Each browser stores the cookies in a different location. The cookies are browser-specific, and so a cookie created in one browser(e.g., in Google Chrome) will not be accessed by another browser(Internet Explorer/Firefox).
- Some browsers limit the number of cookies stored by each domain (20 cookies). If thelimit is exceeded, the new cookies will replace the old cookies.
- Cookie names are case-sensitive. E.g., UserName is different than username.

Methods to handle Cookies with Selenium Webdriver

addCookie(): This method is used create and add the cookie in the cookie file.

```
Cookie cookie = new Cookie("cookieName", "cookieValue");

driver.manage().addCookie(cookie);
```

getCookies(): This method is used to return all the cookies which are stored in the web browser.

```
driver.manage().getCookies();

Set <Cookie> listofcookies = driver.manage().getCookies();

for (Cookie cookie1 : listofcookies) {

System.out.println(cookie1.getName());

System.out.println(cookie1.getValue());

}
```

getCookieNamed(): User can use this method to return a specific cookie according to its name.

```
manage().getCookieNamed(arg0);
```

deleteAllCookies(): This method delete all the cookies stored for a specific browser.

```
manage().deleteAllCookies();
```

deleteCookieNamed(): This method delete specific cookie according Name.

```
manage().deleteCookieNamed(arg0);
```

ByAll Locator in Selenium WebDriver

- ByAll is an Extra Special Locator in selenium webdriver which helps user to find the element based on the given locators. Locator could be any other types like id, name, classname, link text, partial link text, XPath, CSS etc.
- ByAll locator tries to find the element using the first locator if the element is not present then it waits for given implicit wait time, once it reaches the maximum wait time and if there is no element, the ByAll method tries to find the element using the second locator and goes on untill it finds the element, or there are no more locators.

Syntax: driver.findElement(new ByAll(By.className("ElementClass Name"), By.id("Element Id"), By.name("Element Name"))).sendKeys("Test");

How to select an option from auto suggest dropdown in selenium

- While user navigating to any software website with search box and that search box shows auto suggest list when user type some words inside it. Simplest example of ajax auto suggest drop list Is Google search suggestion.

When user will type something inside, it will show them a list of suggestions form where option can be selected based on needs.

- Xpath pattern should be same for all ajax auto suggest drop list Items.
- User need to use for loop to feed that changing values to xpath of drop list different Items. Other one thing user need to consider is user don't know how many items it will show in ajax drop list. Some keywords show user 2 Items and some other will show them more than 2 Items In drop list.

```
List <WebElement> options = driver.findElements(By.xpath("//div[@class =
'sbtc']"));
int optionsize = options.size();
System.out.println(optionsize);
for (int i = 0; i<optionsize; i++)
{
if(options.get(i).getText().equalsIgnoreCase(""))
{
options.get(i).click();
break;
```

```
}

}
```

- TestNG Overview and Step by Step Installation Guide
- TestNG Annotations
- Enable and Disable Test
- Test Case Priority and Sequential Execution

What is TestNG?

TestNG (NG stands for "Next Generation“) is a JUnit inspired open-source automated testing framework providing several features like data driven testing, parameterization support, different utility, annotations and test case grouping that aid in creating robust and powerful testing projects.

TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc.

Using TestNG, user can generate a proper report, user caneasily identify how many test cases are passed, failed, andskipped. In case user need to execute the failed test caseseparately, it is possible to execute.

Why TestNG is useful in Selenium?

- The testing framework can be easily integrated with tools like Maven, Jenkins, etc.
- Generate the report in a proper format including a number of test cases runs, 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 without loops just by using keyword called 'invocation count.'

- Using testng, user can execute multiple test cases on multiple browsers, i.e., crossbrowser testing.
- Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest
- TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-tounderstand annotations that do not require methods to be static.
- Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.

What are the important features available inTestNG?

- **Different Types of Assertions** – TestNG has an Assert class that provides multiple methods supporting different types of assertions like checking equality of expected and actual result, asserting a conditionas True or False, asserting if a value is Null etc.

- **Run tests in Parallel** – Using its testNG.xml file we can run test cases in parallel, thus reducing the overall test execution time. The parallel execution can be done in multiple levels – method level, class level, suite level.
- **Make tests dependent on one another** – Using its 'dependsOnMethods' and 'dependsOnGroups' attribute with @Test annotation, user can make test dependent on other tests.
- **Prioritizing tests** – In TestNG user can assign a numerical priority value to the Tests with 0 being the default value.
- **Grouping of tests** – TestNG supports logical grouping of test cases. Thus, providing ability to run the test groups in parallel, performing certain operations before or after the execution of the tests belonging to a group.
- **Data Driven Testing** – Using @DataProvider user can create data driven tests in which the test data is passed to the test method through its data provider.
- **Reporting** – After test execution an HTML report gets created providing tabular reporting of test results. The reporting format is also configurable by implementing TestNG listeners.
- **Parameterization** – TestNG provides inherent support for parameterization using its @Parameters annotation. The value of the parameter defined using @Parameters annotation is passed to the test logic from the testng.xml file.

What are the main advantages of TestNG over Junit?

- Compared to JUnit annotations, TestNG are easy to understand
- TestNG allows user the grouping of test cases easily which is not possible in JUnit
- TestNG supports the following three additional setup:
- @Before/After Suite, @Before/After Test and @Before/After Group
- TestNG does not need to extend any class
- In TestNG, it is possible to run selenium web driver test cases in parallel
- Based on group TestNG allows user to execute the test cases
- TestNG allows user to determine the dependent test cases; each test case is autonomous to another test case

Step by Step TestNG Installation Guide

1. Open Eclipse, then navigate to Eclipse Marketplace option

2.Enter TestNG in the Find search box and click on the Search button

3. Install TestNG for Eclipse

4.Click “Restart Now” to restart the eclipse, if required

Associate TestNG to Project

1. Right Click on any Java Project & Click

Properties >Java Build Path

2. Navigate to Libraries > Add Library

3. Select TestNG > Click Next> Finish

4. Click on Apply and Close

Create a New TestNG Test File

1: Right-click on any Project already created then choose New > Other…

2: Click on the TestNG folder and select the "TestNG class" option. Click Next.

3: Type the values for Source folder, Package name and Class name and click Finish.

Annotations in TestNG

An annotation is a tag or meta-data that provides additional information about the class, interface, or method in TestNG. Annotations are used to keep the structure of the test classes; these annotations invoke the methods according to the invocation time.

@Test: @Test annotation marks a method as a Test method.

@BeforeClass: The annotated method will run only once before the first test method in the current class is invoked.

@AfterClass: The annotated method will run only once after all the test methods in the current class have been 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.

@BeforeMethod: The annotated method will be run before each test method.

@AfterMethod: The annotated method will be run after each test method.

@BeforeTest: The annotated method will run before any test method belonging to the classes inside the <test> tag is run.

@AfterTest: The annotated method will run after all the test methods belonging to the classes inside the <test> tag have run.

@BeforeSuite: The annotated method will run only once before all tests in this suite have run.

@AfterSuite: The annotated method will run only once after all tests in this suite have run.

Execution Sequence in TestNG

BeforeSuite

BeforeTest

BeforeClass

BeforeGroups

BeforeMethod

Test

AfterMethod

AfterGroups

AfterClass

AfterTest

AfterSuite

Enable and Disable Test in TestNG

- TestNG provides an attribute for enabling and disable our @Test. There are many times when user don't want to run particular @Test and it's not a better way to remove the whole Test method from the class and then add again in case user want to run the @Test again.
- Using **@Test(enabled = false)** method user will not run the method, it simply skip / ignore method and execute next one. In case user want to execute / run method then simply add true above the method e.g**: @Test(enabled = true) .**
- For example, there might be a situation where user might have written skeleton code while development for that feature is still in development, or the feature is already broken so user know it will fail because of which user might want to ignore those test cases from the suite or from the test class.
- By default, TestNG considers enabled=true, if user don't provide one
- **@Test denoted with enabled=false** will be ignored from suite not skipped. It is an optional parameter
- It is not mandatory for TestNG to run enabled=true methods, when these methods depend on other methods

Test Case Priority and Sequential Execution in TestNG

- In any automation project, most times tester are required to configure test suite to run test methods in a specific sequence or user must give precedence to certain test methods over others. TestNG allows user to handle scenarios like these by providing a priority attribute within @Test annotation. By setting the value of this priority attribute user can maintain order the test methods as per their need.
- In order to achieve, user need to add annotation as @Test(priority=??). In case user don't mention the priority, it will take all the test cases as "priority=0" and execute.
- If user define priority as "priority=any positive integer value", these test cases will get executed only when all the test cases which don't have any priority as the default priority will be set to "priority=0"
- Negative priorities are acceptable in TestNG. However, user can provide an integer value to the priority parameter, including zero.
- As default priority of test is integer value 0. So, in case user have one test case with priority 1 and one without any priority value then the test without any priority value will get executed first (as default value will be 0 and tests with lower priority are executed first).
- There may be a chance that methods may contain same priority. In those cases, TestNG

considers the alphabetical order of the method names whose priority is same.

- In case user are defining it in some other syntax say @Test (PRIORITY=??) then Eclipse will show it as a compilation error.

Class :16

- Hard and Soft Assertions in TestNG
- dependsOnMethods and alwaysRun attributes
- invocationCount, invocationTimeOut and skipFailedInvocations attributes
- Run Multiple Test Suites in TestNG

Assertions in TestNG

- Assertions in TestNG are used to verify whether the test is failed or not in TestNG framework and in automation framework also. Test case is considered to be passed only if all statement in test cases executes successfully and in case any exception is thrown or expected result do not match with actual results then test case considered to be failed.
- Assertions are used to verify the statements and test cases in TestNG to decide whether test case is pass or fail.

There are two types of assertions in TestNG

1) Hard Assertion

2) Soft Assertion

Hard assertions are the usual assertions provided by TestNG. In hard assertion in case of any failure, the test execution stops, preventing the execution of any further steps within

the test method and move on to the next test method

Some common assertions provided by TestNG

assertEquals(String actual, String expected, String message)

assertNotEquals(double data1, double data2, String message)

assertFalse(boolean condition, String message)

assertTrue(boolean condition, String message)

Soft assertion in TestNG

To deal with the disadvantage of Hard Assertions, customized error handler provided by TestNG is called Soft Assertion. Soft assertions (SoftAssert) allows user to have multiple assertions within a test method, even when an assertion fails the test method continues with the remaining test execution.

The result of all the assertions can be collated at the end using softAssert.assertAll() method.

```
@Test
public void softAssertionTest(){
SoftAssert softassert= new SoftAssert();
//when Assertion failing
softassert.fail();
```

```
System.out.println(“Assertion is getting Failed");

//when Assertion passing

softassert.assertEquals(1, 1);

System.out.println("Assertion is getting Passed“);

//Collates test results and marks them pass or fail

softassert.assertAll();

}
```

Here, even though the first assertion fails still the test will continue with execution and print the message below the second assertion.

In case user forget to call assertAll() at the end of test, the test will pass even if any assert objects threw exceptions or fail. So don’t forget to add assertAll()

dependsOnMethods attribute in TestNG

Dependency is a feature in TestNG that allows a test method to depend on a single or a group of test methods. This will help in executing a set of tests to be executed before a test method. Method dependency only works if the “depend-on-method” is part of the same class or any of the inherited base class (i.e. while extending a class).

dependsOnMethods attribute on a test method [test1 e.g.] specifies all the test methods [test2, test3,..] this test method depends on. It means test1 will start execution only after all the tests

it depends on executed successfully. If any of the tests specified via dependsOnMethods attribute failed, then test1 will be skipped.

Syntax:

```
@Test(priority=5,dependsOnMethods={"method1","method2"})
```

alwaysRun attribute in TestNG

In case “alwaysRun” attribute set to true for any @Test Annotation, this test method will always be run even when the dependent methods got failed or skipped.

Syntax: @Test(dependsOnMethods = {"method1"}, alwaysRun=true)

invocationCount, invocationTimeOut and skipFailedInvocations attributes in TestNG

- invocationCount is a TestNG attribute that defines number of times a test method should be invoked or executed before executing any other test method. If invocationCount = 4, then the test method will be executed 4 times before executing next test method.
- In case skipFailedInvocations set to true and invocationCount is specified with a value > 1, then all invocations after a failure will be marked as a SKIP instead of a FAIL.
- invocationTimeOut is the maximum number of milliseconds that the total number of invocations on this test method should take. Say in case, invocationTimeOut is set 30 seconds and invocationCount is set to 10 times, then this method should complete all 10 invocations within 30 seconds; otherwise, the test will be marked as a failed. This

annotation will be ignored if the attribute invocationCount is not specified on this method.

Run Multiple Test Suites in TestNG using Selenium WebDriver

- Click File > New > Java Project
- Enter Project Name then click Next.
- Click on the "Libraries" tab, and then "Add Library…"
- On the Add Library dialog, choose "TestNG" and click Next.
- Click Finish.
- Add the JAR files that contain the Selenium API
- Click Finish and verify that newly created project is visible on Eclipse's Package Explorer window.
- Right-click on the "src" folder and select New > Package
- Enter package name and click on finish button.
- Right click on newly created package and select New > Other
- Click on the TestNG folder and select the "TestNG class" option. Click Next.
- Enter values on the appropriate input boxes and click Finish.
- Create one more package and TestNG class
- Write some code for both of the class
- Select the project and right click, select the option called "TestNG" and then click on "Convert to TestNG".

- Generate testng.xml window will open. In this window, enter details like

❖ Location

❖ Suite Name

❖ Test Name

❖ Class Selection

❖ Parallel Mode

❖ Click on "Finish" button

The testng.xml file created under the java project

Class : 17

- How to Include and Exclude Groups during Execution in TestNG
- TestNG Parameterization
- How to Execute only Failed Test Case without Listeners

How to Include and Exclude Groups during Execution in TestNG

- Groups in TestNG denotes the process of grouping different test cases/test methods together into a straightforward group and running these tests/methods together by just running the group in a single command though these tests/methods are belonging to different classes.
- Grouping tests is another very good feature of testng using which user can create group of test methods. User can create a group of test methods based on functionality and features or based on modules or based on testing types like functional testing, regression testing, smoke testing etc.. This way user can differentiate specific group test methods from all test methods.
- Groups are specified in testng.xml file and can be used either under the or tag. Groups specified in the tag apply to all the tags underneath.

Syntax for specifying Groups in Test Method:

```
@Test(groups = { "Group1", "Group2" })
```

Groups are specified in testng.xml file before Class and following syntax can be used

<groups>

<run>

<include name="Group1" />

</run>

</groups>

If user want to run the multiple groups in a same time, then they need to add another <include> tag

inside the <run> tag for each groups.

User can mention the group name which they don't want to run by using the <exclude> tag.

In case user want to run all group with some matching keyword, same can be possible using the regular expression: **<include name="G.*" />**

TestNG Parameterization

- One of the important features of TestNG is parameterization. This feature allows user to pass parameters to tests as arguments. This is supported by using the testng @Parameters annotation.

- The @Parameters annotation can be used for any of the @Before, @After, and @Test annotated methods. It can be used to initialize variables and use them in a class, test, or may be for the whole test execution.
- In case user need to pass some simple values such as String types to the test methods at runtime, user can use this approach of sending parameter values through testng XML configuration files. User have to use the @Parameters annotation for passing parameter values to the test method.

```
@Test
@Parameters({"testparameter1", "testparameter2"})
public void multiplePrameter(String parameter1, String parameter2) {
System.out.println("First Parameter is: " + parameter1);
System.out.println("First Parameter is: " + parameter2);
}
```

The name values used in the testng.xml must match with a parameter used on @Parameters

```
<parameter name="testparameter1" value="First parameter value" />
<parameter name="testparameter2" value="Second parameter value" />
```

User can declare the variables at the suite level, and test level both. However, TestNG gives preference to the Parameters defined at the test level over the parameters set at the suite level.

How to Execute only Failed Test Cases in TestNG withoutListener Class

Most of the times, test cases may fail while executing automated test scripts. The reason may be anything (like Network issue, Element not found, Time out exception, System issue or browser issue) but user need to execute the test scripts again. Instead of running all the test scripts again and again, user can execute only the failed test cases again for any number of times and check for the updated results.

Steps to run only Failed Test Cases in TestNG:

1. After the first run of an automated test suite, right click on Project and then click on Refresh option or press F5.

2. User will be able to see a folder named “test-output” folder. Inside “test-output” folder, user could also find “testng-failed.xml”

3. Run “testng-failed.xml” to execute only the failed test cases again and again.

Class :18

- DataProvider in TestNG
- Listeners in TestNG
- How to Execute Failed Test Cases in TestNG with IRetryAnalyzer
- Parallel Test Execution in Selenium WebDriver using TestNG

DataProvider in TestNG

- In most of the situations, user need to run scenarios with different set of data. During that times, it is not possible to write a test case for each set of data, instead user can use DataProvider, which provides data in the form of an array of objects.
- To run the @Test multiple times with different inputs, user can use data provider option present in TestNG
- A DataProvider annotated method always returns an array of objects. It can be any dimensional.
- Number of rows present in the array will be the number of iteration of the @Test
- Number of columns present in the array will be the number of arguments of the @Test

```
@DataProvider

public Object[][] getTestData()

{

//Rows - Number of times your test has to be repeated and Columns - Number of parameters in test data.
```

```
Object[][] data = new Object[2][2];

data[0][0] ="user1";

data[0][1] = "password1";

data[1][0] ="user2";

data[1][1] = "password2";

return data;

}

public class TestNGDataProvider{

@Test(dataProvider="getTestData")

public void setData(String username, String password)

{

System.out.println("Entered Username is: "+username);

System.out.println("Entered Username is: "+password);

}
```

The main difference between passing data from the DataProvider method and passing data from testng.xml file, By using DataProvider, we can pass multiple sets data using DataProvider and test method will run on each set of data.

But using parameters and testng.xml, it is not possible to pass multiple sets of data.

Listeners in TestNG

- Sometimes user wish to modify the behavior of TestNG during script development and this can be done by Interfaces. These interfaces which help the user to do so are known as Listeners. As the name suggests, 'Listeners' primary task is to listen to an event defined and react according to that. The main purpose for which Listeners are used by the programmers is to create logs and create the custom reports according to the specific scenario defined.
- A TestNG listener always extends org.testng.ITestNGListener and TestNG provides us below listener types:

❖ IExecutionListener

❖ IAnnotationTransformer

❖ ISuiteListener

❖ ITestListener

❖ IConfigurationListener

❖ IMethodInterceptor

❖ IInvokedMethodListener

❖ IHookable

❖ IReporter

Listeners in TestNG

Each listener has specific methods that are overridden. Most popular listeners and the methods which are overridden by them are mentioned below:

1. **ITestListener**

   ITestListener is one of the most commonly used listeners in Selenium Webdriver. The programmer simply needs to implement the ITestListener interface and override all the methods of this interface in order to use it. It makes the call before and after every test present in the suite. There are several methods in it which are mentioned below:

- onStart: This is the first and foremost method that is called after the test class is instantiated. It can also be used to retrieve the directory from which the test is running.
- onFinish: This is the last method to be called after all the overridden methods are done.
- onTestStart(ITestResult result): This method is called each time before any new test method. It indicates that a required test method is started.
- onTestFailure(ITestResult result): This method is called when any test method is failed as it indicates the failures of the test. We can perform certain tasks on test failure like taking the screenshot when a particular test fails in order to get more deep insight into failure.
- onTestSkipped(ITestResult result): This method is called when any test method is skipped for execution.
- onTestSuccess(ITestResult result): This method is called when a particular test method is successfully executed. The programmer can perform any desired operation on test method success by writing code inside this method.

- onTestFailedButWithinSuccessPercentage(ITestResult result): This method is called when any test method is failed with some success percentage. For example, Say any test method is executed 20 times and failed 10 times. It takes 2 parameters, i.e. successPercentage and invocationCount. For the above case, successPercentage would be 50 and the invocationCount would be 20.

2. ISuiteListener

Unlike the ITestListener, which is implemented after every test method, ISuiteListener is implemented at the Suite level. It has two methods which are overridden:

onStart: This method is implemented before the invocation of the test suite which means all the code written inside it is run before the start of any suite.

onFinish: This method is implemented after the invocation of the test suite which means all the code written inside it is run after the whole test suite is run.

Listeners in TestNG

 There are 2 important ways of creating Listeners in TestNG:

I. User can use the @Listeners interface within the class:

Step 1: User to create a class for Listener which is implementing ITestListener and overriding all its methods.

public class TestNGListener implements ITestListener

System.out.println("Test Case Failed: " + result.getName());

Step 2: User need to implement the above Listener in the normal Java/TestNG class (before class) having

the @test methods using @Listeners annotation.

@Listeners(packagename.ListenerClassName.class)

Step 3: User can run the class from the XML file

II. User can use add Listeners in the XML file directly:

Step 1: User to create a class for Listener which is implementing ITestListener and overriding all its methods.

public class TestNGListener implements ITestListener

Step 2: User create a normal Java/TestNG Class having all the @test methods and there is no need to use

@Listeners annotation.

Step 3: User can add an entry of the listener and class in the XML file (After Suite and before Test)

<suite name="TestNGSuite">

<listeners>

<listener class-name="packagename.ListenerClassName"></listener>

</listeners>

<test name="TestNGListener">

How to Execute Failed Test Cases in TestNG with IRetryAnalyzer

Test methods will be failed because of many reasons like uncertain behavior by application, network issues..etc. To overcome the false failures we may want to re execute failed test methods.

• When a test fails, the retry() method gets called, and the retry() method will result in either true or false.

• In case user have a test method that is annotated with retryAnalyzer along with @Test,so based on the result from the retry() method @Test method decides whether it should try or not.

• If retry() results in true then @Test method will try to re-execute the test method. if results false then test method will not be re-executed

• retry() receives a parameter for TestResult, this will contain whether the test is passed or not and test method names, based on this increment the counter in RetryFailedTest.

• Create a class called RetryListener which should implement IRetryAnalyzer Initialize a two variables maxRetry=5, and currentTry=0

• Override the method called retry() and write condition where it should enter to if block

only if maxRetry > currentTry Inside if block returns true; after if block return false.

How to Execute Failed Test Cases in TestNG with IRetryAnalyzer

```
public class RetryListener implements IRetryAnalyzer {

int currentTry = 0;

int maxRetry = 5;

@Override

public boolean retry(ITestResult result) {

if(currentTry < maxRetry)

{

currentTry++;

return true;

}

return false;

}
```

Parallel Test Execution in Selenium WebDriver using TestNG

- Parallelism or multi-threading in software terms is defined as the ability of the software, operating system, or program to execute multiple parts or sub-components of another program simultaneously. TestNG allows the tests to run in parallel or multi-threaded mode. This means that based on the test suite configuration, different threads are started simultaneously, and the test methods are executed in them. This gives a user a lot of advantages over normal execution, mainly reduction in execution time and ability to verify a multi-threaded code.
- Parallel test execution is vastly used by the QA industry for functional automation testing. This feature helps QA to configure their tests to be executed easily in multiple browsers or operating systems simultaneously.
- TestNG provides an auto-defined XML file, where user can set the parallel attribute to method/tests/classes and by using the concept of multi-threading of Java, one can set the number of threads, one wants to create for parallel execution. Below is the structure for defining this attribute in

the TestNG XML:

<suite name="Parallel Testing" parallel="methods" thread-count="2">

Methods: Helps run methods in separate threads

Tests: Help to run all methods belonging to the same tag in the same thread

Classes: Helps to run all methods belonging to a class in a single thread

Instances: Helps run all methods in the same instance in the same thread

Along with the parallel attribute, the thread-count attribute helps in defining the number of threads one wishes to create while running the tests in parallel. For example, in case one has four methods, with thread count as three, then during execution, three threads shall start in parallel with the corresponding methods. As the first method execution is completed, and the thread gets free, it takes up the next method in the queue

Class :19

- Introduction of Maven, Installation and running sample program
- How to Execute Selenium Script through Maven and TestNG
- Maven Commands and running Maven Project through Command Lines

Introduction of Maven, Installation and running sample program

- Apache Maven is a software project management andcomprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting anddocumentation from a central piece of information.
- Maven is used to define project structure, dependencies, build, and test management.
- Using pom.xml user can configure dependenciesneeded for building testing and running code.
- Mavenautomatically downloads the necessary files from therepository while building the project.
- Maven is a dependency management tool, i.e., to execute the frameworks; we need the jar files. Maven helps the user to keep the up-to-date jar file on the framework.
- During runtime, the maven will check the version of the jarfiles present in the local system and compares it with the pom.xml file.

Steps for creating Maven project:

1. In Eclipse IDE, create a new project by selecting File | New | Other from

Eclipse menu.

2. On the New dialog, select Maven | Maven Project and click Next

3. On the New Maven Project dialog select the Create a simple project and click Next

4. Enter Group Id: and Artifact Id: and click finish. The groupId is the id of the project's group. Generally, it is unique amongst an organization. The artifactId is the id of the project. It specifies the name of the project.

5. Right-click on JRE System Library and select the Properties option from the menu

6. On the Properties for JRE System Library dialog box, make sure Workspace default JRE is selected and click OK

7. Select pom.xml from Project Explorer. pom.xml file will Open in Editor section

8. Add the Selenium, Maven dependencies to pom.xml in the <project> node.

How to Execute Selenium Script through Maven and TestNG

Create a Maven Project and add Dependency for TestNG in pom xml file

(https://mvnrepository.com/):

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.8</version>

<scope>compile</scope>

</dependency>

- Select Package and Create a New Classes. Add code in the classes.
- Right-click on the TestNG file, and select Run As TestNG Test
- Right-click on the Project and select TestNG and then select Convert to TestNG options.

Once we select the Convert to TestNG options, it will open the Generate testng.xml

window, and click on the Finish

User need to add the below plugins in the pom xml file

(https://maven.apache.org/surefire/maven-surefire-plugin/):

Maven surefire plugin

right-click on the MavenProject → Run As → Maven Test

How to Execute Selenium Script through Maven and TestNG

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M5</version>

<configuration>

<suiteXmlFiles>

<suiteXmlFile>testng.xml</suiteXmlFile>

</suiteXmlFiles>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

Maven Commands and running Maven Project through Command Lines

- Go to http://maven.apache.org/download.cgi to download installation file for Maven
- Ensure JAVA_HOME environment variable is set and points to user JDK installation
- Unzip apache-maven-3.6.3-bin.zip
- Select My Computer and select Properties by Right-Clicking on My Computer. Click on Advanced system settings. Click on 'Advanced' tab and then click on 'Environment Variables..'.
- Click on New button under 'System Variables..'
- Write 'MAVEN_HOME' in the Variable name box then enter apache-maven-3.6.3 Maven path in the Variable value box and click OK.

- Add the bin directory of the created directory apache-maven-3.6.3 to the Path environment variable

Add below maven compiler plugin to the POM XML file

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version>
<configuration>
<source>10</source>
<target>10</target>
</configuration>
</plugin>
```

Maven Commands

Open a command prompt window (Press Windows + R at the same time. Then enter cmd in the Run window and click on the Ok button).

mvn -version in command prompt to check the vesion of the Maven already installed user system

Go to the directory where user saved the maven project (cd location of the project)

mvn clean: This command cleans the maven project by deleting the target directory

mvn complie: It's used to compile the source Java classes of the project.

mvn test: This command is used to run the test cases of the project using the maven-surefire-plugin.

mvn package: This command builds the maven project and packages them into a JAR, WAR, etc.

mvn help: This command prints the maven usage and all the available options for us to use.

Class 20

- Repository and Repository Manager in Maven
- How to implement Headless Browser testing with Maven
- Maven Profiles to Control Build Execution

Importance of Snapshots in Maven

Maven repository is a directory where all the packages, JAR files, plugins or any other artifacts are stored with POM.xml. Repository in maven holds build artifacts and dependencies of various types.

User need to add multiple dependencies for any framework in Project POM.xml file. Project Object Model (POM) contains all the dependencies that are being used by their current project.

There are 3 types of maven repository:

- Local Repository
- Central Repository
- Remote Repository

Maven local repository is located in the local computer system. It is created by maven when the user runs any maven command. The default location is %USER_HOME%/.m2 directory. When maven build is executed, Maven automatically downloads all the dependency jars into the local repository. For new version maven will download automatically. If version declared in the dependency tag in POM.xml file it simply uses it without downloading. User can change the location of maven local repository by changing the settings.xml file. It is located in

MAVEN_HOME/conf/settings.xml.

Maven Central repositories are located on the web. It has been created by the apache maven itself. it contains a large number of commonly used libraries. It is not necessary to configure the maven central repository URL. Internet access is required to search and download the maven central repository. When maven cannot find a dependency jar file for local repository its starts searching in maven central repository using URL: https://repo1.maven.org/maven2/.

Repositories and Maven

Maven Remote Repository is stored in the organization's internal network or server. The company maintains a repository outside the developer's machine and are called as Remote Repository.

<dependencies>

<dependency>

<groupId>com.companyname.common-lib</groupId>

<artifactId>common-lib</artifactId>

<version>1.0.0</version>

</dependency>

<dependencies>

<repositories>

<repository>

<id>companyname.lib1</id>

<url>http://download.companyname.org/maven2/lib1</url>

</repository>

<repository>

<id>companyname.lib2</id>

<url>http://download.companyname.org/maven2/lib2</url>

</repository>

</repositories>

Repository and Repository Manager in Maven

Maven Dependency Checking Process:

Maven will scans through local repositories for all configured dependencies, in case it is found continues with further execution.

Maven scans in central repositories for that particular dependency in case of absence of dependencies in local repository. Available dependencies are downloaded in local repositories for future execution of the project.

In case dependencies are not found in Local Repository and Central Repository, Maven starts scanning in Remote Repositories.

In case dependencies are not available in any of the three Repositories- Local Repository, Central Repository,

Remote Repository, Maven throws an Exception “not able to find the dependencies & stops processing”.

It downloads all found dependencies into the Local Repository.

A repository manager is a dedicated server application designed to manage repositories. The usage of a repository manager is considered an essential best practice for any significant usage of Maven.

The followings are the open-source and commercial repository managers are known to support the repository format used by Maven.

- Apache Archiva
- CloudRepo
- Cloudsmith Package
- JFrog Artifactory Open Source
- JFrog Artifactory Pro
- Sonatype Nexus OSS
- Sonatype Nexus Pro
- packagecloud.io

How to implement Headless Browser testing with Maven

- Headless browsers are browsers like any other but without a graphical user interface (GUI). The GUI allows user to visualize an aesthetic representation of the page source code (usually in HTML, CSS, and Javascript). Headless browsers fetch the page the same way standard browsers do, and can render the page (screenshot), and save it as an image file if requested.
- The main advantages of using them are that these browsers are more lightweight and therefore require less computational resources, such as memory and processing. They are ideal to use in environments that don't have a GUI: for example, a Continuous Integration server.

Maven Dependency:

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>htmlunit-driver</artifactId>

<version>2.27</version>

</dependency>

 Creating WebDriver Instance for HtmlUnitDriver:

WebDriver driver = new HtmlUnitDriver();

Maven Profiles to Control Build Execution

- Maven profiles can be used to create customized build configurations, like targeting a level of test granularity or a specific deployment environment. Within each profile element, user can configure many elements such as dependencies, plugins, resources, finalName.
- Maven Profile is an alternative set of configurations which set or override default values. Profiles allow to customize a build for different environments. Profiles are configured in the pom.xml and each profile is identified with an identifier.

What are the different types of profile? Where is each defined?

Per Project

- Defined in the POM itself (pom.xml).

Per User

Defined in the Maven-settings (%USER_HOME%/.m2/settings.xml).

Global

- Defined in the global Maven-settings (${maven.home}/conf/settings.xml).

Build the project with command mvn compile -X. The option -X outputs Maven debug info to console and from the lengthy debug output, From the output, it is clear that Javac debug is still true and optimize is false which means that the production profile is not activated by Maven, and it is still using default configurations.

To activate any profile use command mvn compile -Pprofile. The option -P activates the profile.

From the Maven debug messages, user can confirm that profile is activated and Javac flags debug and optimize are set as configured in the profile.

Maven Profiles to Control Build Execution

By defining a profile user can override the values defined by project's Effective POM. Maven allows following element in an Profile.

```
<project>
<profiles>
<profile>
<id>smoke</id>
<build>
<defaultGoal>...</defaultGoal>
<finalName>...</finalName>
<resources>...</resources>
<testResources>...</testResources>
<plugins>...</plugins>
</build>
<reporting>...</reporting>
<modules>...</modules>
```

<dependencies>...</dependencies>

<dependencyManagement>...</dependencyManagement>

<distributionManagement>...</distributionManagement>

<repositories>...</repositories>

<pluginRepositories>...</pluginRepositories>

<properties>...</properties>

</profile>

</profiles>

</project>

To activate a profile, add -P option: mvn package -Pprofile

If user always want to execute a profile, we can make one active by default:

<profile>

<id>defaulttestingprofile</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

Then, user can run mvn package without specifying the profiles, and they can verify that the defaulttestingprofile profile is active.

Importance of Snapshots in Maven

During development of a large scale application, the process involves a series of changes to be made in the application till it is confirmed to be ready as a final release. Because, there could be multiple teams working on an application across different set of modules.Consider 2 teams A & B working on 2 different modules and lets say team B will be dependent on the services provided by team A. Say team A is involved in working on some major critical fixes and those fixes will be checked in every day. So team B has to be notified in a way that there are still some pending work by team A and the final version has not yet been released by the team A. This is where the maven's SNAPSHOT comes into the picture.

Snapshot is a special version which indicates the current development copy of the project which is being worked on. For each build, maven always checks out for a SNAPSHOT of the project.

Hence, whenever maven finds a newer SNAPSHOT of the project, it downloads and replaces the older .jar file of the project in the local repository.

Snapshot version always gets some updates/changes being made on the project. Also, Snapshot should exist only during development phase and it will not be having a release version. This implies that the build of the project will be changed at any time and it is still under the development process.

Snapshot Vs Version:

- In case of SNAPSHOT, Maven will automatically fetch the latest SNAPSHOT (data-service: 1.0-SNAPSHOT) every time Maven Test project is built.
- In case of Version, maven once downloaded the mentioned version say JavaSamples:1.0, then it will never try to download a newer 1.0 available in repository. To download the updated code,
- Maven Test Project version is to be upgrad d to 1.1