**Selenium:**

**findelement vs findelements:**

findElement method is used to access a single web element on a page. It returns the first matching element. It throws a NoSuchElementException exception when it fails to find If the element.

driver.findElement(By.xpath("Value of Xpath"));

findElements method returns the list of all matching elements.

the findElements method returns an empty list when the element is not available or doesn’t exist on the page. It doesn’t throw NoSuchElementException.

List link = driver.findElements(By.xpath("Value of Xpath"));

**relative path(//) vs obsolute path(/)**

A single slash at the start of XPath instructs XPath engine to look for element starting from a root node.

A double slash at the start of XPath instructs XPath engine to search look for matching element anywhere in the XML documen

Absolute path example– html/body/div[1]/div[2]/div[1]/div/div[1]/div/div/div[2]/div[2]/div/div/div/form[1]/div[1]/div[1]/div[1]/div/div/input

Relative path example – //\*[@id=’xz123′]

Partial X path, Using keyword contains, starts with, text()

**page factory & Page object model**

Selenium Page Factory Pattern is like an extension to Page Object Model , but Page Factory is much enhanced model. To start with, we just need to import package ‘org.openqa.selenium.support.PageFactory’

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

We use Page Factory pattern to initialize web elements which are defined in Page Objects.

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

@FindBy is used to identify Web Elements in the page.

@FindBy(id="username")

private WebElement userName;

**implicit wait , explicit wait**

Implicitly wait:

implecit wait implemented in driver level

Implicit waits are used to provide a default waiting time (say 30 seconds) between each consecutive test step/command across the entire test script. Thus, the subsequent test step would only execute when the 30 seconds have elapsed after executing the previous test step/command.

Syntax:

driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);

import java.util.concurrent.TimeUnit

Explicitly Wait:

explicit wait implemented at language level

Explicit waits are used to halt the execution until a particular condition is met or the maximum time has elapsed.

Syntax:

WebDriverWait wait = new WebDriverWait(drv,30);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[contains(text(),'COMPOSE')]")));

expected conditions types:

elementToBeClickable()

textToBePresentInElement()

alertIsPresent()

frameToBeAvailableAndSwitchToIt()

titleIs()

Each FluentWait instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition

Wait wait = new FluentWait(driver)

.withTimeout(30, SECONDS)

.pollingEvery(5, SECONDS)

.ignoring(NoSuchElementException.class);

**Navigate()**

driver.navigate().back()

driver.navigate().forward()

driver.navigate().refresh()

driver.navigate().to("http://google.com")

XPath is defined as XML path. It is a syntax or language for finding any element on the web page using XML path expression. XPath is used to find the location of any element on a webpage using HTML DOM structure

**Selenium Alert:**

driver.switchTo().alert().dismiss();

driver.switchTo().alert().accept();

driver.switchTo().alert().getText();

driver.switchTo().alert().sendKeys("Text"); for prompt alert box asking to enter value.

**preceding siblings following siblings**

**frame switch , window handles**

driver.switchTo().frame(Name / Id / index);

want to switch back to previous or parent of current frame then following code

driver.switchTo().parentFrame();

Want to switch to Main or parent most then following code

driver.switchTo().defaultContent();

**To find size of iframe**

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

how to take values from webtable

**Iteration of iframes:**

for(int i=0; i<=size; i++){

driver.switchTo().frame(i);

System.out.println(total);

driver.switchTo().defaultContent();}

**What is the difference between *driver.get()*and*driver.navigate.to(“url”)*?**

*driver.get():*To open an URL and it will wait till the whole page gets loaded  
*driver.navigate.to():*To navigate to an URL and It will not wait till the whole page gets loaded

**How to press ENTER key on text box In Selenium WebDriver?**

driver.findElement(By.xpath("xpath")).sendKeys(Keys.ENTER);

[How to handle login pop up window using Selenium WebDriver?](https://stackoverflow.com/questions/11522434/how-to-handle-login-pop-up-window-using-selenium-webdriver)

Answer:>>https://testtupp123:password123@preprod.creditsesame.com

driver.switchTo().alert();

//Selenium-WebDriver Java Code for entering Username & Password as below:

driver.findElement(By.id("userID")).sendKeys("userName");

driver.findElement(By.id("password")).sendKeys("myPassword");

driver.switchTo().alert().accept();

dependencies, always true

apache poi reader

read data & write

testNG:

testNG is a testing framework which is created over JUnit

testNG Annotations is easy to understand

it supports parallel execution

it supports to run multiple test cases using the testing xml file

it Support for data-driven testing (with @DataProvider).

Support for parameters.

TestNG generates reports

**TestNG Priority**

We can set the priorities for the testcases using @Test annotation (Ex: @test(Priority=1))

**Priority execution order:**

If priority set for 3 methods then priority not set for 2 methods

Then non priority methods executed based on alphabetical order then priority methods executed based on priority mentioned in it.

If same priority is set to 2 methods then it follows alphabetical order.

**testNG Depency**

It support to set the dependencies between testcases using dependsOnMethods

ex: @test(dependsOnMethods={“logintest”})

**alwaysRun:**

**testNg Annotations:**

**@BeforeSuite:**The annotated method will be run before all tests in this suite have run.   
**@AfterSuite:**The annotated method will be run after all tests in this suite have run.   
**@BeforeTest**: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.   
**@AfterTest**: The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.   
**@BeforeGroups**: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.   
**@AfterGroups**: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.   
**@BeforeClass**: The annotated method will be run before the first test method in the current class is invoked.   
**@AfterClass**: The annotated method will be run after all the test methods in the current class have been run.   
**@BeforeMethod**: The annotated method will be run before each test method.   
**@AfterMethod**: The annotated method will be run after each test method.

**@Parameters:**This annotation is used to pass parameters to test methods.

Example :

@Test

  @Parameters({ "sUsername", "sPassword" })

  public void test(String sUsername, String sPassword)

**@DataProvider:**If we use @DataProvider annotation for any method that means you are using that method as a data supplier. The configuration of @DataProvider annotated method must be like it always return Object[][] which we can use in @Test annotated method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.

**Groups**

(groups = { ” Group Name” })

TestNG allows you to perform sophisticated groupings of test methods. Not only can you declare that methods belong to groups,

**How do you run failed test cases in TestNg?:**

1. After the first run of an automated test run. Right click on Project – Click on Refresh.
2. A folder will be generated named “test-output” folder. Inside “test-output” folder, you could find “testng-failed.xml”
3. Run “testng-failed.xml” to execute the failed test cases again.

**how to disable particular test test case in testng?**

annotation **@Test(enabled = false)** helps to disable this test case.

If a test method is annotated with *@Test(enabled = false)*, then the test case that is not ready to test is bypassed.

**Java**

serialization

array list

hash map

in the given paragraph how many times a particular word repeats

find the 2nd highest number in the integer array

compare 2 array and print the identical value

Palindrome programme

Fibonacci program

Factorial program

Even or odd

Prime number program

string vs string buffer

mutuable & immutable

Final finally finalize

How to hide the password or SSN number in the TestNG report in the Selenium

<https://developers.perfectomobile.com/pages/viewpage.action?pageId=21430626>

StringEncrypt() Class

**encryptXOR (String**message***, String***key***)***

**decryptXOR(String**message***, String***key***)***

How to highlight the element in the page

Abstract class.

Read data from excel in selenium

Array and array list difference

Hash table

Heap

Difference between list and set ??

Difference between private & protected access modifiers

How can we access the Static method to our page???

**Can you inherit a final class in java?**

No, We can't extend the final class.

**Can you inherit final method of super class into a sub-class?**

Yes, we can inherit final method of a super class into a sub-class.

**Can you inherit private members of super class into a sub-class?**

No, we cannot inherit private member of a super class into a sub-class.

**Can you inherit private members of super class into a sub-class?**

No, we cannot inherit private member of a super class into a sub-class.

**Can you inherit static member into a sub-class.**

Yes, we can.

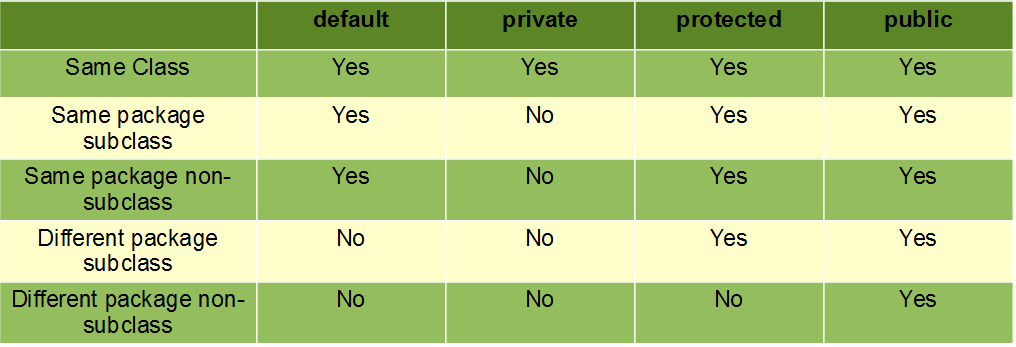
**What happens if super class and sub class having same field name?**

Super class field will be hidden in the sub class but you can access hidden field of a super class by using 'super' keyword in the sub-class.

**Can we inherit constructor in sub-class?**

No, we cannot inherit constructor in sub-class.

Access Modifiers:



**Protected**: The protected access modifier is specified using the keyword **protected**.

**Private**: The private access modifier is specified using the keyword **private**.

* The methods or data members declared as private are accessible only **within the class** in which they are declared.
* Any other **class of same package will not be able to access** these members.

**Default**: When no access modifier is specified for a class , method or data member – It is said to be having the **default** access modifier by default.

* The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.

**public**: The public access modifier is specified using the keyword **public**.

* The public access modifier has the **widest scope** among all other access modifiers.
* Classes, methods or data members which are declared as public are **accessible from every where** in the program. There is no restriction on the scope of a public data members.
* The methods or data members declared as protected are **accessible within same package or sub classes in different package.**