==

**************************************Automation Testing**

==

Automation Testing: testing application feature using automation tool & executing test script

--> preferably regression tests & Smoke tests are automated

--> automation of regression tests is carried when-

a. Build becomes stable (with minimal defects)

b. New feature is added on the top of existing stable build

Why Automation testing is required?

**Limitations of manual testing:

1. more human efforts are required
2. regression testing becomes time consuming
3. test cycle execution time increases
4. compatibility testing (system & browser) becomes difficult

**Advantages of automation testing:

1. less human efforts
2. regression becomes less time consuming
3. reduces project duration
4. reduces cost of project
5. reusable scripts
6. compatibility testing (System compatibility & cross-browsing) becomes easy & fast
7. more reliable & efficient (less errors & fast)

Automation Tools:

1. Selenium (for web-based application)**	-open source
2. Sahi(for web-based application)	-open source
3. Serenity(for web-based application)	-open source
4. Appium (for mobile application)**	-open source
5. Selendroid (for mobile application)	-open source
6. Robot Framework **	-open source
7. SahiPro(for web-based & mobile application)	-License
6. QTP (for web-based application)	-License
7. Rhenorex (for web-based application)	-License

Automation Testing using Selenium:

**Selenium- open source automation tool for web-based applications

Versions of Selenium:

1. Selenium IDE --> only runs on Firefox
   --> compatibility testing not possible
   --> only supports JAVA

2. Selenium-RC --> only supports JAVA
   --> compatibility testing is possible

3. Selenium Webdriver** --> multi-language support (JAVA, Groovy,C++,Python, Perl, PHP, Rubby etc.)
   --> compatibility testing is possible (cross-browsing)

4. Selenium-grid --> supports cross platform testing parallel on (Windows, Linux, macOS etc.)
   on multiple browsers

Additional automation tools:

** Selendriod --> supports only android applications

** Appium --> supports andriod & iOS mobile applications

Software required for Selenium:

1. Java JDK latest version (say JDK15 from oracle website)
2. Selenium Remote WebDriver "selenium-java-3.141.59" (.zip) file (https://www.selenium.dev/downloads/)
3. Eclipse IDE (latest)
4. latest browser specific driver .jar file (Platforms Supported by Selenium)

Create project-->Right Click on project --> Build path--> configure build path--> Libraries--> Add External jar

----

Selenium WebDriver: it is collection of open source APIs to automate web-based applications

```
                Architecture of Selenium-WebDriver:

                        "SearchContext(I)" <--Extends--  "WebDriver(I)" <--Implements- A. RemoteWebDriver(C) :  1. 
FirefoxDriver
                        a. findElement                  a. get()
                                                        2. ChromeDriver
                        b. findElements                 b. getCurrentUrl()
                                                3. IEDriver
                                                                        c. getTitle()
                                                                4. EdgeDriver
                                                                        d. Close()
                                                                        5. SafariDriver
                                                                        e. Quit()
                                                                        6. OperaDriver
                                                                        f. Manage()
                        B. ScriptExecutors         : JavaScriptExecutor(I)
                                                                        g. Navigate()
                C. ScreenShots             : TakesScreenshots(I)
                                                                        h. Maximize()
                                                                        i. setSize()
                                                                        j. setPosition()
                                                                        k. getWindowHandle()
                                                                        l. getWindowHandles()
                                                                        m. switchTo() etc.
```

```
---------------------------------------------------------------------------------------------------------
1. To set system properties
        System.setProperty("webdriver.<nameOfSupportedDriver>.driver", "<pathOf_Driver.exe>");
2. To instantiate the browser (create object)
        e.g. WebDriver driver=new ChromeDriver();
3. Browser Actions:
        a. open given URL
                driver.get("https://www.google.co.in/");
```

b. Setting size of browser window
```
Dimension D=new Dimension(100,200);
driver.manage().window().setSize(D);
```

c. Setting position of browser window
```
Point P=new Point(300,400);
driver.manage().window().setPosition(P);
```

d. Maximize browser window
```
driver.manage().window().maximize();
```

e. To Navigate to different URL
```
driver.navigate().to("http://facebook.com/");
```

f. To Navigate back to previous URL
```
driver.navigate().back();
```

g. To Navigate forword to desired URL
```
driver.navigate().forward();
```

i. To refresh the browser
```
driver.navigate().refresh();
```

j. To find the title of given page (extracted from Page-source)
```
String title = driver.getTitle();
System.out.println("Title of Page: "+title);
```

k. To find the url of current page
```
String currentURL = driver.getCurrentUrl();
System.out.println("URL of current page: "+currentURL);
```

l. To close the current tab
```
driver.close();
```

m. to release WebDriver object
```
driver.quit();
```

```
-----------------------------------------------------------------------------------------------------------
HTML : HyperText Markup Language

        <HTML>
                <body>
                        Label <tag attribute="attribute_value"> Text
                </body>
        </HTML>


Possible tag_names:
        1. Input: input --> values: text, radio, checkbox, buttons,
        2. Link:  <a href="value"> Text </a> --> value: heperlinks

Possible attributes:
        1. type
        2. id
        3. class
        4. name
        (& not limited.....)

Tables:
        1. tr = Table Row
        2. th = Table Header
        3. td = Table data (column)

        <table>
                <tbody>
                        <tr>
                                <th>.....</th>
                        </tr>
                        <tr>
                                <td>.....</td>
                        </tr>
                </tbody>
        </table>
-----------------------------------------------------------------------------------------------------------
```

```
Locators in Selenium: used to identify the unique element/s in browser web-page

**To find single element using locator --> driver.findElement(By.locator_method())     --> output: WebElement
**To find array of elements using locator--> driver.findElements(By.locator_method())  --> output: WebElement[]

Methods/types of Locators:
        1. tagName()                    --> tag
        2. id()                                 --> attribute
        3. className()                  --> attribute
        4. name()                               --> attribute
        5. linkText()                   --> text of link (full)
        6. partialLinkText()    --> text of link (partial)

  **7. xpath()                          --> XML path
                        7.a xpath by absolute path (/): parent to immediate child only
                                --> e.g. /html/body/div/input[5]
                        7.b xpath by relative path (//): parent to any child
                                --> e.g. //html//input[5]
                  **7.c xpath by attribute : use tag & attribute to find element
                                --> //tag[@attribute_name='attribute_value']
                        7.d xpath by index : subset of xpath by attribute but with index
                                -->(//tag[@attribute_name='attribute_value'])[index_value]
                        7.e xpath by text:      use text in given element
                                --> //tag[text()='text_value']
                        7.f xpath by contains: subset of xpath by text used for text with blank spaces/ nonbreakable
spaces
                                -->//tag[contains(text(),'text_value')]

                Additional xpaths to handle dynamic web-elements whose attribute values changes dynamic (xpath
axes):
                        7.g forward traversing (X-axis):
                                        i)xpath/child_node -->to navigate forward from current node to its immediate
child node
                                  **ii)xpath/child::node_name -->to navigate forward from current node to only its
child node (same level)
                                  **iii)xpath/descendant::node_name -->to navigate forward from current node to its
child & grand-child nodes
```

```
                                  **iv)xpath/following-sibling --> to navigate forward from current node to
following nodes of same name
                                  & at same level

                        7.f reverse traversing(Y-axis):
                                        i) xpath/..  -->to navigate back from current node to its immediate parent
node
                                  **ii) xpath/parent::node_name -->to navigate back from current node to its parent
node
                                  **iii)xpath/ancestor::ansestor_name -->to navigate back from current node to its
ancestor (root) node
                                        iv) xpath/preceeding::node_name --> to navigate back from current node to
all nodes at same level

        8. cssSelector()                -->

**To locate in web-page: Right click on element --> select 'Inspect'

-----------------------------------------------------------------------------------------------------
Synchronization in Selenium: matching selenium test script running time with web-application's speed using waits

1. Thread class         --> Thread.sleep(time_in_Milliseconds)

2. Implicit waits       --> always call on every use of WebDriver instance
                                        --> default polling time =500 milliseconds

driver.manage().timeouts().implicitlyWait(default_Timeout,TimeUnit.MILLISECONDS)

3. Explicit waits       --> more customized wait than implicit wait
                                        --> can have customized polling time
                                        --> can ignore the exceptions
                                        --> can apply on particular web-element
                3.a Fluent wait:
                                        --> It needs instance of FluentWait class to work
                                                        FluentWait wait=new FluentWait(driver);
                                        --> customize polling time & timeouts
                                                        wait.withTimeout(2000, TimeUnit.MILLISECONDS);
                                                        wait.pollingEvery(30, TimeUnit.MILLISECONDS);
                                          --> can ignore exceptions while waiting for particular webelement
                                                        wait.ignoring(NoSuchElementException.class);
                                          --> can wait till certain condition
                                                        wait.until(ExpectedConditions.alertIsPresent());

                3.b WebDriver wait: Specialized version of fluent wait that can handle multiple conditions at a time
                                                WebDriverWait wait=new WebDriverWait(driver,
default_timeout_SECONDS,polling_time);

----------------------------------------------------------------------------------------------------------
List-Box or Drop-down handling using Select class:

                1. find webelement for given list-box
                        WebElement ele=driver.findElement(By.------);

                2. Create object for Select class
                        Select sel=new Select(ele);

                3. Use appropriate method of select class from below-
                        3.a. selectByIndex(Index_value)
                        3.b. selectByValue("value_of_option")
                        3.c. selectByVisibleText("text_of_option")

                4. to extract all options under dropdown
                        sel.getOptions()

                5. Additional methods of Select class
                        5.a getFirstSelectedOption() --> It returns first/default selected option as element
                        5.b isMultiple() --> checks if multiple options are selected

----------------------------------------------------------------------------------------------------------
iFrame: an HTML document (Webpage) embedded within another HTML document (Webpage)

default content --> parent frame <--> child frame1,2,.... --> operation

e.g.<html>
```

```
        <body>
                UserName<input type='text'/><br>
                Password<input type='pass'/><br>
                <iframe src='basic.html' name='accept' id='accept'> </iframe> <br>
                <iframe src='basic.html' name='accept1' id='accept1'> </iframe> <br>
                <input type='button' value='login'/>
        </body>
</html>

                1. Default Contents --> Contents of Current-Webpage
                2. Parent frame --> Webpage embedded into current Webpage
                3. Child frame --> webpages embedded into parent frames

1. To switch from default webpage to iFrame --> driver.switchto().frame("...")

                --> index of frame

                --> id

                --> name
2. To switch from child frame to parent frame --> driver.switchto.parentFrame()
3. To Switch from frame to default content --> driver.switchto().defaultContent()

---------------------------------------------------------------------------------------------------------
PopUps: small/separate window displayed when we perform action on any component in WebPage

1. Alert PopUps --> cannot be inspected
                                --> can be drag & drop (depends)
                                --> contains buttons like OK/Accept/Enable... & Decline/Cancel/Disable....
                                --> can have text, special characters like ?, !, triangle sign
                                --> can be handled using "Alert" class

                                        Alert alt=driver.switchTo.alert();
                                        to click OK        --> alt.accept()
                                        to click cancel    --> alt.dismiss()
                                        to get Alert text  --> alt.getText()
                                        to send input text --> alt.sendKeys()
```

```
2. Hidden Division Popup --> colorful, can be inspected
                                                --> cannot be drag & drop
                                                --> can be directly handled by locating elements on it
                                                --> e.g. Flipkart login
                                                --> no separate method/class required, can be handled using regular
xpath

3. Child browser pop-up --> can be inspected, can be drag & drop (depends)
                                          --> can contain address fields, maximize, minimize, close etc. options
                                          --> Cnt+P on webpage
                                          --> no separate method/class required, can be handled using regular xpath
                                          --> Need to handle multiple windows & switch to child window to perform
operations

                                          Set<String> ids=driver.manage.getWindowHandles();
                                          ArrayList<String> id=ArrayList<String>(id);
                                          String tab=id.get(Index of child/desired window);
                                          driver.switchTo.window(tab);
                                          -->now user can perform regular operations i.e. locate element & perform
actions

4. Authentication pop-up --> cannot be inspected
                                                --> contains buttons like OK/Accept/Enable... &
Decline/Cancel/Disable....
                                                --> contains authentication fields like input, check-boxes,
radio-buttons etc.
                                                --> cannot be drag & drop
                                                --> can be handled using get() method  & pass
-->"username:password@URL"

5. File upload                  --> Need robot class or Auto-IT tool to handle it
6. File Download popup   --> Need robot class or Auto-IT tool to handle it

---------------------------------------------------------------------------------------------------------
Taking screen-shots in Selenium:
        1. Use "TakesScreenshot" interface for source screen
```

```
                -->  File scr=((TakesScreenshot)driver).getScreenshotAs(outputType.FILE)
        2. Set destination path & format for screen-shot file
                --> File dest=new file("destination_path\\file_name.jpg")
        3. Copy Screen-shot to source to destination
                --> FileHandler.copy(scr,dest)   OR
                --> FileUtils.copyFile(scr,dest)

----------------------------------------------------------------------------------------------------------
Parameterization-1: Fetching data from Excel Files (.xlsx, .csv)
        1. Import Apache.Poi dependency in project
        2. create excel file with desired data
        3. Create object for file
                --> fileInputStream file=new fileInputStream("file_path\\file.xlsx")
        4. open excel sheet using WorkbookFactory interface & fetch data
String value=WorkBookFactory.create(file).getSheet("Sheet_name").getRow("row_index")
                        .getCell("column_index").getStringCellValue()
                                                                OR      .getNumericCellValue()
important methods of WorkbookFactory:
        create(file)--> to open created file format
        getsheet("Sheet_name")--> select desired sheet in workbook
        getRow(row_index) --> select desired row to access data
        getCell(coumn_index) --> select desired column to access data
        getStringCellValue() --> to extract string value from pointed location
        getNumericCellValue() --> to extract string value from pointed location
        getLastRowNum()          --> to identify maximum limit of row count
        getLastCellNum()        --> to identify maximum limit of column count
----------------------------------------------------------------------------------------------------------

                                                         OR .getNumericCellValue()
Actions Class: To perform Keyboard & mouse operation smartly on web-page
        --> It is selenium based utility
        --> To handle drop-down, auto-suggestions, operation related to mouse & keyStroke
        1. Identify elements say-e.g. drop-down & store (list of result) into variable
        2. Create object for action class & pass Webdriver object into it.
                        --> Actions action=new Actions(driver);
        3. Handle drop-down using action class methods
                --> moveToelement(webelement_object)--> move cursor to element on which action is to be perform
                --> build()                             --> bind multiple actions together & allow to perform one by
one
                --> perform()                   --> execute operations one by one
                -------------------
                --> click()                             --> Clicks at the current mouse location.
                --> clickAndHold()              --> Clicks (without releasing) at the current mouse location.
                --> contextClick()              --> Right click operation
                --> doubleClick()               --> Performs a double-click at the current mouse location.
                --> dragAndDrop((source,target) --> click-and-hold at the source element, moves to the target &
release
                --> keyDown(Keys.keyStroke)                     -->  pressing Key on keyboard ( Keys.SHIFT, Keys.ALT
or Keys.CONTROL)
                --> keyUp(Keys.keyStroke)                       -->  release Key on keyboard ( Keys.SHIFT, Keys.ALT
or Keys.CONTROL)
                --> pause()
                --> release()                                                   -->  Release mouse left button
                --> sendKeys(Keys.keyStroke or string)  --> keyStrokes: ENTER,ALT,CONTROL,TAB,SHIFT,
                --> moveByOffset(xOffset,yOffset)               --> Moves the mouse from its current position by the
given offset

        4. Pass the web-element in moveToElement method & perform desired action on it
        Format for using methods of Action class
                action.moveToElement(pass).click().sendKeys("Password").build().perform()
---------------------------------------------------------------------------------------------------------
Robot Class: used to control keyboard & mouse actions to interact with OS windows e.g. Download popUps, Alerts,
                         Print PopUps or any native applications (installed on OS)
                        --> It is Java based utility
                        --> For mouse it uses InputEvent
                        --> For keyboard it uses KeyEvent
        **Instance of Robot class --> Robot robo=new Robot();
        **Useful methods of Robot class:
                1. keyPress(KeyEvent.VK_keyname)
                2. keyRelease(KeyEvent.VK_keyname)
                3. mousePress(InputEvent.BUTTON1_DOWN_MASK)
                4. mouseRelease(InputEvent.BUTTON1_DOWN_MASK)
                5. mouseMove(X_offset, Y_offset)
---------------------------------------------------------------------------------------------------------
```

```
JavascriptExecutor to perform Scrolling Operation:
                --> Classes of javaScript --> Document Class & Window Class
                --> Document class: Performs actions on web-elements
                --> **Window Class: Performs actions on Web-Browser window
                --> Left scroll  (H) / Up Scroll   (V)--> Negative index
                --> Right Scroll (H) / Down Scroll (V) --> Positive index
                --> Selenium Web-driver instance needs casting with JavascriptExecutor interface
                        JavascriptExecutor js=(JavascriptExecutor)driver;

                Application-1 : Scrolling Horizontal/Vertical on main browser window
                        js.executeScript("window.scrollBy(index-H,index-V)");

                Application-2 : Scrolling into Page Object
                        a. find webelement you want to scroll upto
                        b. use--> js.executeScript("argument[0].scrollIntoView(true)",element)
                        c. perform the operation on desired element
-------------------------------------------------------------------------------------------------------
Parameterization-2:     Using Properties File
                --> used to provide data to existing script without changing much in the script
                --> easy format to use & maintain
                --> uses 'Properties()' class
                --> needs to load property file object into load() method --> So read file using FileInputStream();
                --> read any desired input just by its label --> getProperty("label_of_data")
-------------------------------------------------------------------------------------------------------
Logging & Appending: (using log4j)
                Logging --> Show output on console (INFO, DEBUG, ERROR, WARN,FATAL, ALL)
                Appending --> continue logging in console or continue writing logs into desired log file

                1. Download log4j jar file & add as external jar into IDE
                2. store configutarion file into 'Classes' folder
                Step-1: Navigate to Build path --> Configure build path --> Resources tab -->
                        tick 'Allow output folder to sources folder'  --> click on created output folder --> click
Edit button
                        --> Select Specific output folder --> Browse --> select project -->create new folder named
as 'classes'
                        --> OK
                Step-2: right click on project --> select 'Proprty' --> Resources --> Click on arrow next to
```

```
'Location'
                        --> in opened location find 'classes' folder we created --> create file "log4j.properties"
or "log4j.xml"
                3. prepare configuration for log4j in file created
                4. create object of Logger class of log4j
                        --> Logger log=Logger.getLogger(className.class)
                5. using FileInputStream, select properties or xml file created
                        -->FileInputStream file =new FileInputStream("file_path")
                6. using PropertyConfigurator, load the configuration file
                        -->PropertyConfigurator.configure(file);
                5. use following methods of logging:
                        --> log.info("Message")
                        --> log.error("Customised error message")
                        --> log.warn("warning message") etc.

------------------------------------------------------------------------------------------------------
Handling Cookies:
                --> Cookie is small file that holds data specific to client & website used in client-server
communication
                --> to handle cookies in selenium use following methods:
                        driver.manage().getCookies();
                        driver.manage().getCookieNamed("name_of_cookie");
                        driver.manage().addCookie(obj);
                        driver.manage().deleteCookie(obj);
                        driver.manage().deleteAllCookies();
                        driver.manage().deleteCookieNamed("name_of_cookie");
                --> How to get cookies
                                Set<Cookie> cookie1=driver.manage().getCookies();
                --> Create a cookie
                                Cookie cookie2=new Cookie("test","1234");
                                driver.manage().addCookie(cookie2);
------------------------------------------------------------------------------------------------------
Page Object Module (POM): It is JAVA design pattern that follows the encapsulation
                                                  --> a. All data members (WebElements) declared globally with Private
access in pageClass
                                                          b. initialize all data-members within constructor with
Public Access
```

c. utilize/perform actions on WebElements within method with public access

--> TestClass : Class with main method
--> PageClass: Class for all web elements in respective class (without main method)

How to use POM structure??:

1. Create test class & all required Page classes
2. In Test Class --> create WebDriver instance
3. In Test Class --> pass WebDriver instance into constructor of each of Page Classes
4. In Page Class --> define all data members (WebElements) as private entity at global level
5. In Page Class --> declair all data memebers (webElemets)
6. In Page Class --> create method for each data memebers (webElemets) to utilize (perform action) it
7. In Test Class --> call methods of page classes using respective objects of that class

Limitation of POM structure : it initializes all componenets before performing actions, that component may be Hidden hence it creates either 'No Such Element Exception' or 'Stale Element Exception'

How to Overcome limitations of POM : Using PageFactory Interface

----

POM with PageFactory:

--> We do initialize webelements only on which action is to be performed.
--> It also works with hidden elements
-->a. Only data members (WebElements) initialised & declared globally with Private access in pageClass
b. Early initialization using pagefactory & Webdriver within constructor of Page class
c. utilize/perform actions on WebElements within method with public access
--> rest is same as POM
--> @FindBy =driver.FindElement(By.-----)

TestNG: Java unit Framework used to write test classes

```
Advantages of TestNG --> 1. All test classes will be executed in the form of test-suite
                                                 2. Generates reports (Emailable reports, extend reports etc...)
                                                 3. Execute only failed test cases in new build (failed.xml)

How to install TestNG Plugin: Eclipse-->Help--> Marketplace -->find TestNG for Eclipse --> Select & Install
How to add testNG.jar:  Maven repo --> find testNG --> download jar file --> Eclipse --> Build path --> add external
jar
How to add testNG library: Eclipse --> Build path--> Libraries --> Classpath --> Add Libraries--> select TestNG

Inside of TestNG : 1. Annotation
                                   2. Keywords
                                   3. Verification
                                   4. Reporter & Reports (emailable reports)
                                   5. testng.xml (Include/Exclude/groups)
                                   6. Parameterization (testng.xml/DataProviders)
                                   7. Parellel execution
                                   8. Listener

1.Annotation:
                                @BeforeSuite                                   --> execute before all @Test methods
in a suite
                                        @BeforeTest                                    --> executed before first
@Test method
                                                @BeforeClass                   --> executed before first @Test
method (once per class)
                                                        @BeforeMethod          --> will execute before every mothod
annoted by @Test
                                                                @Test                  --> it makes a method as a
part of the test
                                                        @AfterMethod           --> will execute after every mothod
annoted by @Test
                                                @AfterClass                            --> executed after all @Test
methods are over (once per class)
                                        @AfterTest                                     --> executed after all @Test
methods are executed
                                @AfterSuite                                            --> execute after all @Test
```

```
methods in a suite


2. Keywords:
                        @Test(priority=...-2/-1/1/2/3....)      --> to prioraties test cases (test methods), default
priority=0
                        @Test(invocationCount=1/2/3.....)       --> to re-execute test cases multiple times
                        @Test(enabled=false) OR @ignore --> to ignore execution of respective test case
                        @Test(timeOut=5000)                             --> to bound test case w.r.t. mentioned time
or skip it
                        @Test(dependsOnMethods={"method1","method2",....}) --> to wait until parent methods to
execute.

        --> if parent methods fails, skip dependents
                        @Test(groups={"group_Name"})            --> associate the test to specific group & run if
group execution
                                                                                                        is called
from testNG.xml
                        Assert.fail                                                     --> force to fail current
test case


3. Verification: Always belongs to test class of POM only
                        A. Verification using Hard Assert (Static Methods) of Assert class)--> Validation
                        --> For a method with multiple verifications, if any verification fails, execution of that
test fails

                                --> Assert.assertEquals()       OR  Assert.assertNotEquals()
                                --> Assert.assertTrue()         OR  Assert.assertFalse()
                                --> Assert.Null()                       OR      Assert.notNull()
                                --> Assert.fail()

                        B. Verification using Soft Assert (using object of SoftAssert class)--> Verification
                        -->  For a method with multiple verifications, if any verification fails it notifies &
execute rest of
                                 verifications
                        Softassert assert=new Softassert();
                                --> assert.assertEquals()       OR  assert.assertNotEquals()
                                --> assert.assertTrue()         OR  assert.assertFalse()
                                --> assert.assertNull()                 OR      assert.assertNotNull()
                                --> assert.fail()
                                --> assert.assertAll()          --> To combine multiple assertions together

4. Reporter: To display log on console and append the same in testNG report as well
                        method1: Reporter.log("Testing")                --> Display log only into console but dont
append in report
                        method2: Reporter.log("Testing",true)   --> Display log on console & append the same in
report

        Reports: Execution report contains details of test cases Passed/Failed/Skiped, associated error logs,
                        group execution etc.
                        e.g. emailable-report, index reports

 5. testng.xml: To control the execution from single place we need tesng.xml --> called test suite
                                suite --> test --> class --> methods

                ***How to create testNG.xml???
                                Right click on package containing test classes --> select TestNG --> convert to
TestNG
                                --> Enter suite name & test name --> refresh the project
                ***How to run testNG.xml???
                                Right click on testng.xml --> Run as --> testNG Suite

                <suite name="testNGSuite">
                  <test thread-count="3" parallel="methods" name="testNGTest">
                        <classes>
                          <class name="tesNGBasics.testNGKeywords"/>
                          <class name="tesNGBasics.test1Class"/>
                          <class name="tesNGBasics.testNGAssertions">
                                        <methods>
                                                <exclude name="test2"/>
                                                <include name="test1"/>
                                        </methods>
```

```
         </class>
        </classes>
  </test>
</suite>


** to exclude perticular method in a class --> <exclude name="test2"/>
** to include only perticular method/s in a class --> <include name="test2"/>
** to execute parellely include parallel="level_name" at test level --> parallel="methods"
```

6. Parameterization-3:

```
importing external data (test data) into script.
methods to import : a. using testng.xml
                                        b. using DataProvider
a. using testng.xml:
        **after suite declairation in testng.xml pass parameter as below-
                <parameter name="user" value="testUser"/>
        **extract parameter into test case using annotation--> @parameters
                @Test
                @Parameters("user")
                public void method(String user)
                {
                 ..............
                }
b. using DataProviders: it overcomes limitations of parameterization using testng.xml
        part-1 : declaire data provider method

        @DataProvider(name="testData")
        public Object[][] testData()
        {
                Object[][] data= new Object[2][2];

                data[0][0]="Prashant";
                data[0][1]="prash123";

                data[1][0]="Abhi";
                data[1][1]="abhi456";
```

```
                                return data;
                        }

                        part-2: use data provider at test level_name

                        @Test(dataProvider="testData")
                        public void method(String user, String password)
                                {
                                 ..............
                                }
8. testNG Listeners: To listen test activity, log that activity & take appropriate action
                                ITestResult --> FAILURE(), SUCESS(), SKIP()
        How to use listner?? :
                * in method with  AfterMethod  annotation pass --> ITestResult <variable>
                * use ITestResult method to validate --> e.g. ITestREsult.FAILURE==<variable>.getStatus()
                here <variable> is extracted as boolean value from each of test methods

---------------------------------------------------------------------------------------------------------
Maven Architecture:

1. Maven Lifecycle : Default (for Deployment)
                                         Clean   (to Clean previous build)
                                         Site    (for site's documentation)

2. Maven phases    : validate: check if all information necessary for the build is available
                                         compile: compile the source code
                                   **test: run unit tests
                                   **install: install the package to a local repository
                                         deploy: copy the package to the remote repository

3. Maven Goals     : compiler plugin:   "compile"               – from the compiler plugin is bound to the compile
phase
                                         surefire plugin:       "test"                  - is bound to test phase
                                         install plugin :       "install"               - is bound to install phase
```