30% - 50% -> Java

50% -> Selenium/TestNG/Maven/Framework etc (Real Time Questions )/Appium

Name – Aravinda HB

Email - [aru03.info@gmail.com](mailto:aru03.info@gmail.com)

Phone : +91 - 9945042504

Pre-Req:

Java

------

1. OOPS Concepts. (abstraction, encapsulation, inheritance, polymorphism)
2. Basics of Programming
   1. Data types and Variables
      1. Local variables
      2. Global variables
      3. Static variables
      4. Final variables
   2. Methods
      1. Default Methods
      2. Parameterised Methods
      3. Methods without return type
      4. Methods with return type
   3. Constructors
      1. Default constructors
      2. Parameterised constructors
   4. Static and instance blocks
3. Class
   1. Abstract class
   2. Final class
   3. Concrete class
4. Interface
5. Inheritance
6. Operators
7. Looping and conational statements
   1. For, while, do while , enhanced for loop
   2. If , if else, if elseif else, switch
8. Strings \*\*\*\*\*\*\*\*\*\*\*
   1. String functions
9. Exception Handling
   1. Try
   2. Catch
   3. Throw
   4. Throws
   5. Finally
10. File Handling in Java
    1. How to read a text file
    2. How to write into a text file
11. Threads
12. Collections
    1. List
    2. Set
    3. map

What is Testing

What are the different types of Testing we perform

What is Automation testing?

Why Automation testing is required!!!?

Different Tools available in Market ???

Why Selenium is Popular?

Selenium

* Selenium IDE
  + Record and play back
  + Add-on Chrome and FF
* ~~Selenium RC~~
* Selenium WebDriver
  + Writing xpath and css selector.
  + Lot of concepts – each and every thing
* Selenium Grid
  + Distributed Execution

Frameworks –

* Data driven Automation Framework
* Function driven Automation Framework
* Keyword driven Automation Framework
* Hybrid driven Automation Framework
* TestNG
  + Annotations
  + Grouping
  + Priority
  + Parallel execution
  + Lot more .....
* POM

Maven –

What is maven

Why maven is required

*VCT – Git*

*BDD - Cucumber*

JAVA : DAY-2

- OOPL

- free software (sun Microsystems)

***- platform(HW + OS ) independent*** programming language

- JAVA(Basic) => J2EE + J2ME + Android

1. Installation

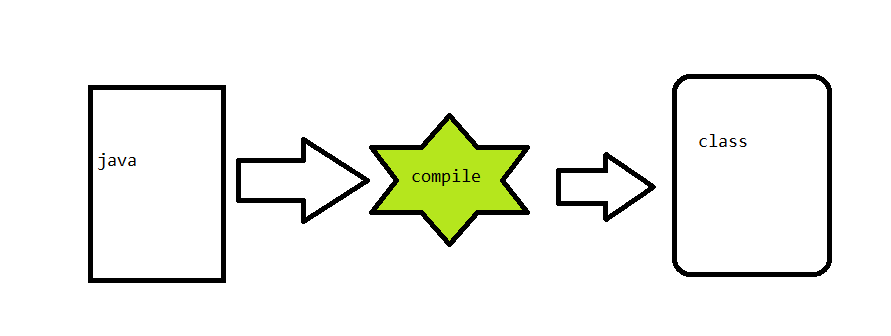
-jdk -> development

- jre -> execute java programs

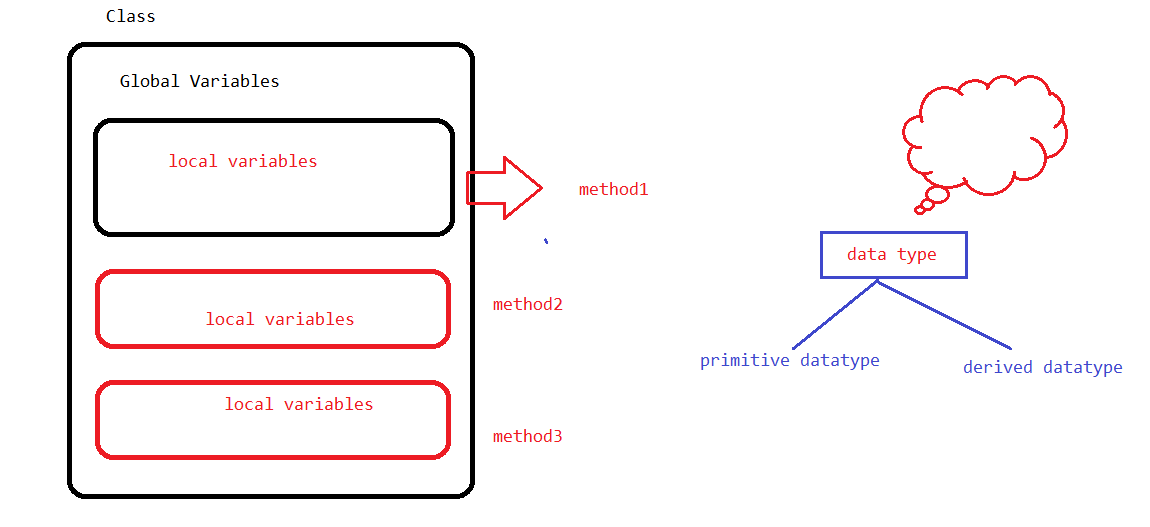
2. Configure Java using environment variables

Java Basics :

- packages ->



Variables and data types



Methods:

return\_type name([arg1, arg2])

{

//logic

}

Scope of a variable :

* Global variable – scope is throughout the class
* Local Variable - only inside the function (outside the function these variables are not accessible )

\* Class - blueprint (sketch of a House )

🡺 Members:

🡪variables

🡪methods

🡪constructor

🡪static IB

🡪instance IB

\* Objects - (House)

- Java is strictly type cast

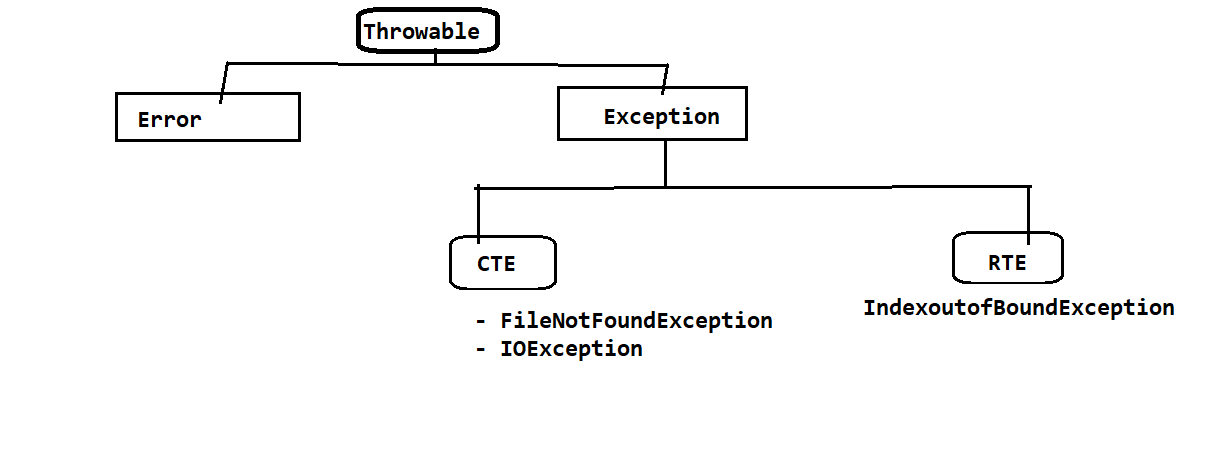
- for

- while

- do,while

- foreach

Exceptions:



Try

Catch

Throw

Throws

Finally

Selenium 30/05/2020

1. Free
2. Multi language (java,c#,ruby,python,perl,js)
3. Supports almost all the popular browsers
4. Supports parallel execution
5. No dedicated machine is required
6. Distributed Execution (using GRID)

Selenium IDE:

Implemented using java script

* Beginners
* IDE
* Record and Play

Verify and Assert :

Identification :

- id

- name

- className

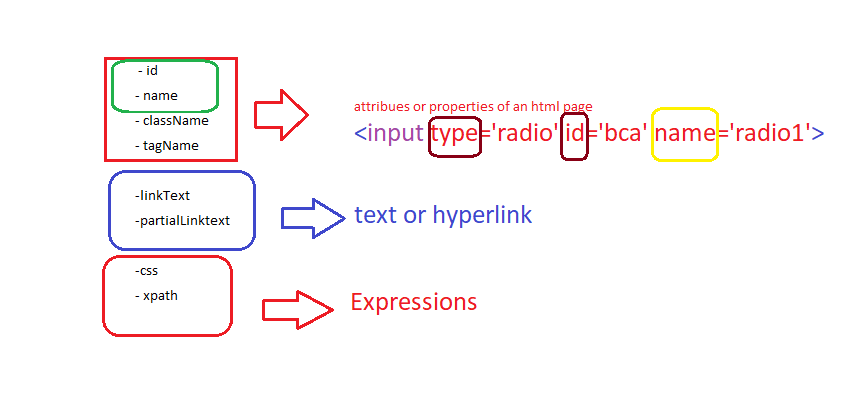
- tagName

-linkText

-partialLinktext

-css

- xpath



CSS:

**htmltag\_name**[attribute=’value’]

**htmltag\_name#idvalue OR #idvalue**

**htmltag\_name.classValue OR .classValue**

**PARENT TO CHILD – css\_expression > chid\_tag**

**XPATH:**

1. **BASIC** 
   1. **//html\_tag[@attibure=’value’]**
   2. **//a[@id='loginButton']/div**
2. **Functions :** 
   1. **text()**
      1. **LOGIN BUTTON IN ACTITIME - //div[text()='Login ']**
      2. **//td[text()='Please identify yourself']**
   2. **contains(arg1,arg2)**
      1. **arg1 -> attribute or text function**
      2. **arg2 -> partial value for attribute or text**
      3. **Ex: TEXT ON LOGIN SCREEN - //td[contains(text(),'Please')]**
      4. **Ex: ACTITIME LOGO - //img[contains(@src,'timer.png')]**
   3. **starts-with(arg1,arg2)**
      1. **arg1 -> attribute or text function**
      2. **arg2 -> partial value for attribute or text**
      3. **Ex: Google Search for Intellipaat - //h3[starts-with(text(),'Intellipaat')]**
3. **LOGICAL OPERATORS** 
   1. **AND - > when both the conditions are satisfied** 
      1. **//td[@class='nfo' and @data-spec='year']**
   2. **OR -> If any one condition is satisfied then it returns the element**
      1. **print all text boxes present in the page - //input[@type='text' or @type='password']**
      2. **Redbus.in cal🡪 //td[text()='2' and (@class='current day' or @class='we day' or @class='wd day')]**
      3. **OR THE SAME XPATH CAN BE USED WITH NOT - //td[text()='6' and not(@class='past day')]**
      4. **OR WITH OPERATOR - //td[text()='' and @class!='past day']**
4. **Traversing**
   1. **TRAVERSING FROM PARENT TO CHILD - to reduce the search portion in the HTML file**
      1. **//div[@data-hveid='CBAQAA']//h3**
   2. **TRAVERSING FROM CHILD TO PARENT – Dependent and independent element**
      1. **//div[div[div[p[text()='AI-401 | AI-775']]]]//span[@class='actual-price']**
5. **Axes Functions**
   1. **following-sibling**
      1. **Sample from Wikipedia page - //th[text()='Directed by']/following-sibling::td**
      2. **//div[@id='toc']//li[a[span[text()='Production']]]/following-sibling::li**
   2. **preceding-sibling**
      1. **//div[@id='toc']//li[a[span[text()='Production']]]/preceding-sibling::li**
   3. **parent**
      1. **//div[@id='toc']//span[text()='Production']/parent::a/parent::li/preceding-sibling::li**
   4. **child**
      1. **//th[text()='Directed by']/following-sibling::td/child::a**
   5. **ancestor** 
      1. **//p[text()='SG-905']/ancestor::div[@class='dept-options-section clearfix']//span[@class='actual-price']**
         1. **Write xpath to the independent element**
         2. **use ancestor and specify the ancestor tag**
         3. **traverse back to the required child element**
   6. **following**
      1. **//div[@id='toc']//span[text()='Production']/parent::a/parent::li/following::li**
   7. **preceding**
      1. **//div[@id='toc']//span[text()='Production']/parent::a/parent::li/preceding::li**

**Sample Xpath :**

**- GSM ARENA**

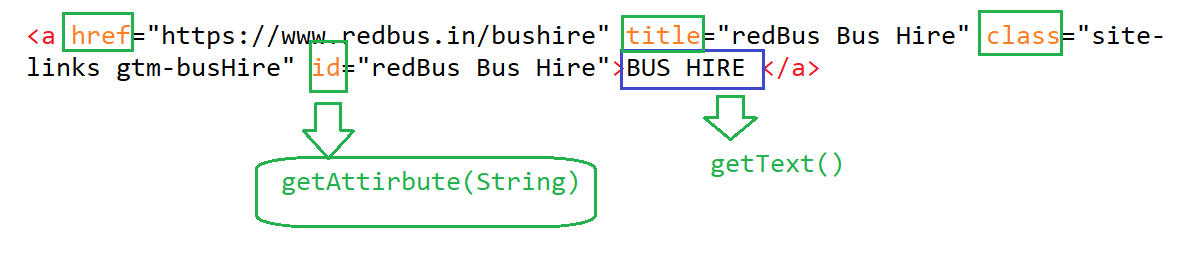
**//th[text()='Platform']/ancestor::tbody//td[@class='nfo']**

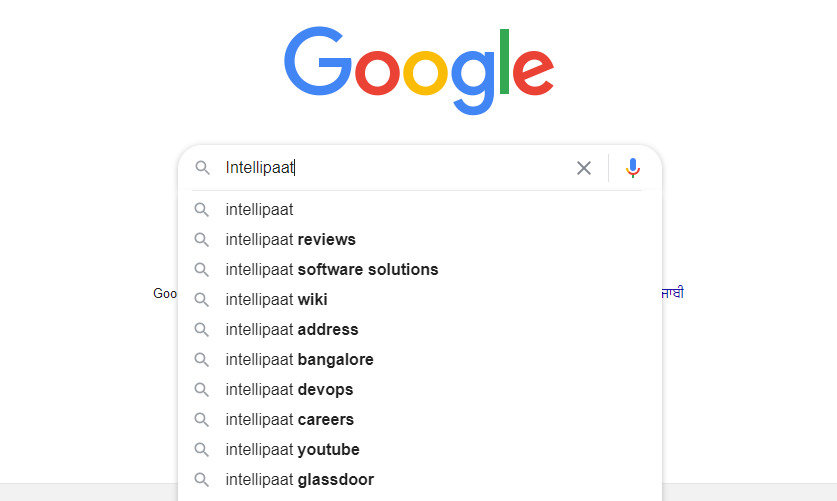
**SELENIUM WEBDRIVER SETUP.**

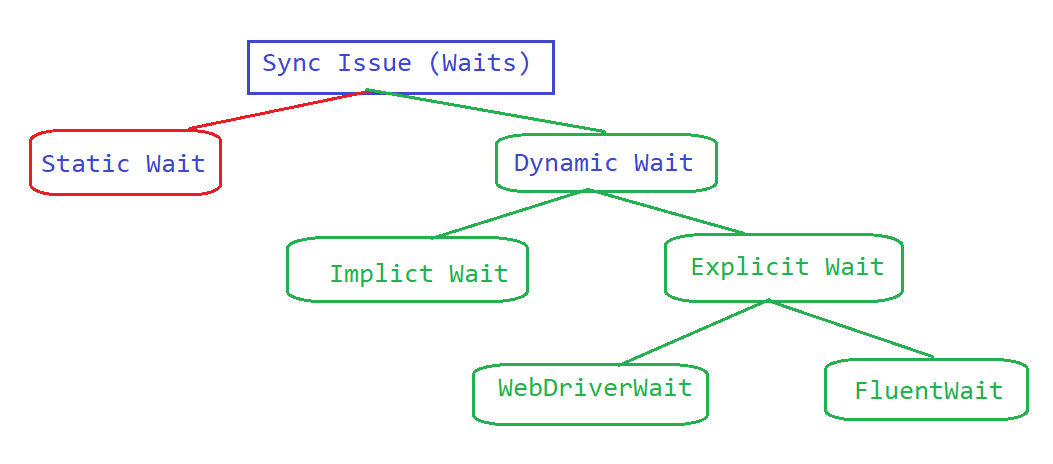
1. **Eclipse software**
2. **Create Java project and configure Selenium**
   1. **Attach java libs**

**Selenium Setup:**

1. **HelloWorld- Selenium**
2. **Handling Tool Tip in Selenium:**

* **Tool tip small information about the webElement.**
* **alt or title property**
* ****

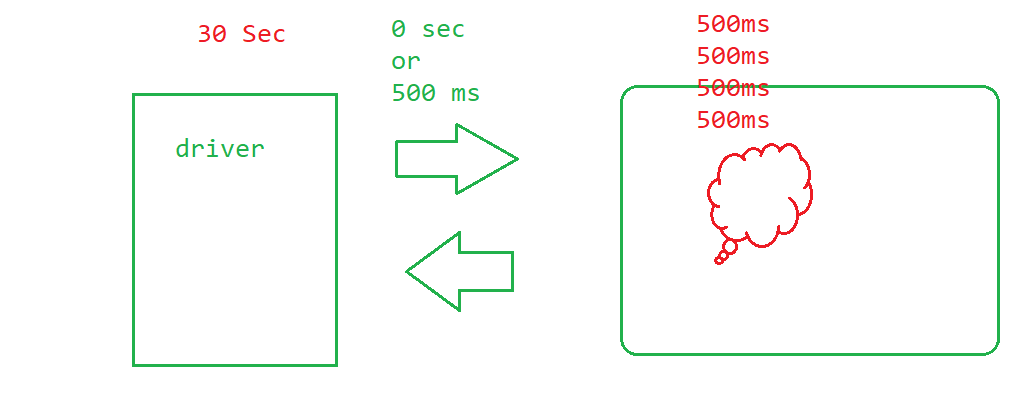
1. **Auto Suggestions:**
   1. **Suggestions given by application based on the user search criteria**
   2. 
   3. **we have to use findElements when we are playing with multiple Elements**
2. **Synchronization issue**

****

Usually Tool Execution speed is much faster than the Application execution speed. because of which some tests may fail.

We have to reduce the tool execution speed.

* 1. **Static Waits**
     1. Thread.sleep(ms)
     2. NEVER USE STATIC WAITS!!!
  2. **Dynamic Waits**
     1. **Implicit wait – Works on Page load. (JavaScript or AJAX Calls or ANGULAR JS)**
        1. Changing the default wait time of a driver Object.
        2. As soon as you create a Object we have to change the default time of driver

****

* + 1. **Explicit Wait**
       1. **WebDriverWait**

**- If** Time take is more ( More than implicit wait )

**-** Wait for change in state of the element ( Enabled, Element to be clickable, visible, invisible etc...)

**STEPS:**

1. create an Object to WebDriverWait be passing Max time to wait in sec
2. use wait Obj and call until method
3. Pass **ExpectedConditions** and use the corresponding method.

Drawbacks:

1. we have to rely on default functions available *( If the required option is not there or it is not working as expected* )
2. by default polling time is 500ms, and we can not change it in WebdriverWait
3. If any exception occurs we cannot ignore them.
   * + 1. **FluentWait**

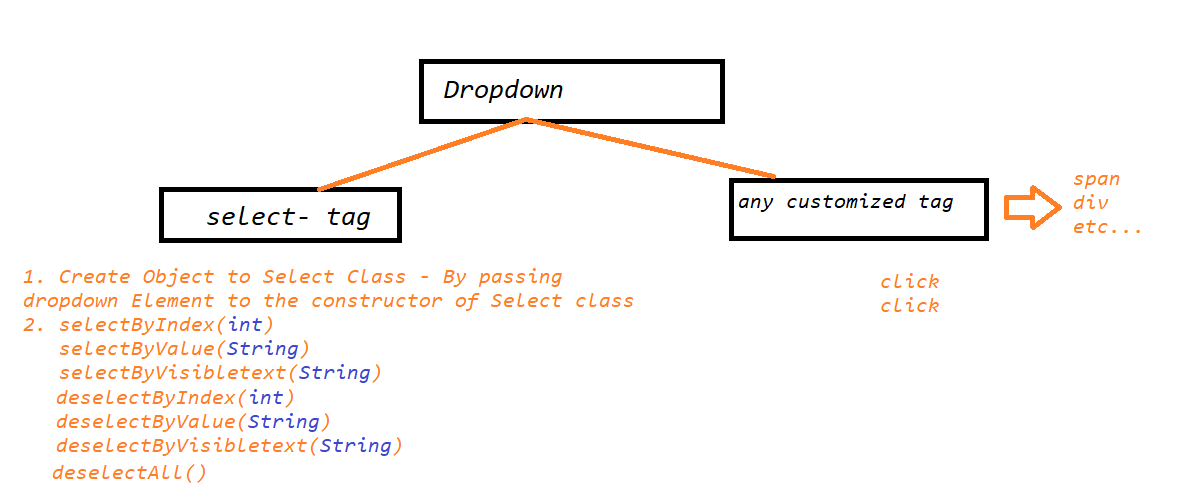
Its a another ExplictWait mechanism in which we can keep our own wait logic and it also allows us to have our own polling time.

95% webdriverWait will have the functions only 5% chances you may have to use FluentWait

1. Create a Fluentwait object by passing –
   1. WebElement
   2. What is polling time
   3. What is the max timeout
   4. Should it ignore any exceptions.
2. Function ->apply-> write your own logic (Anonymous inner class)
3. use wait object (FluentWait Object) call until and pass your function Object.

**NOTE** - if applilcation has more useage of **(JavaScript or AJAX Calls or ANGULAR JS)** then we have to rely on Explicit wait

**Handling DropDown :**

****

**Validations in Selenium:**

* getTitle() -> String
* getText() ->String
* getAttibute(attribute) ->String
* isEnabled() -> Boolean
* isDisplayed() -> Boolean
* isSelected() ->Boolean

**Automating Toast Messages:**

Messages which appears and disappear after few seconds.

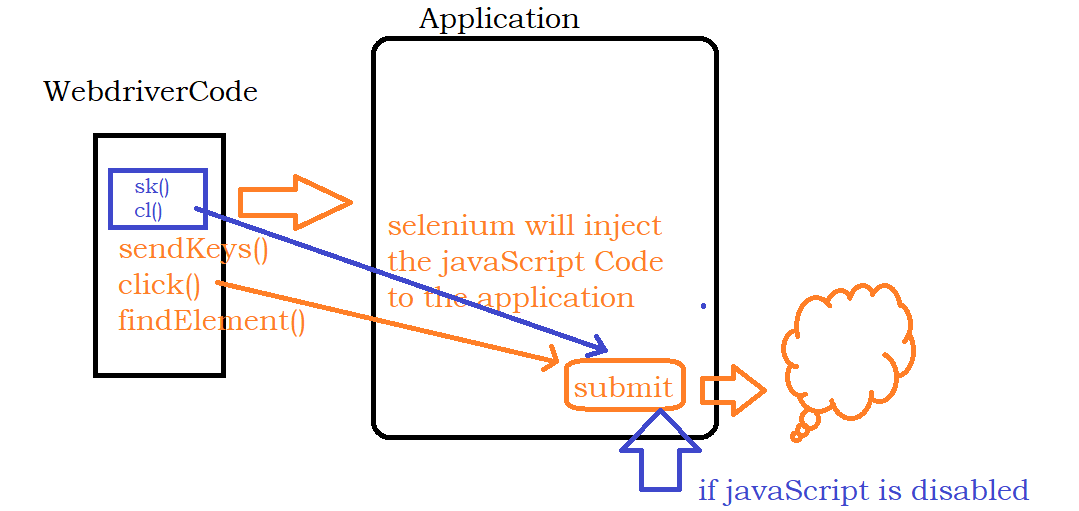
*1. Talk to Development Team and get the properties details*

*2. Start searching for a text Toast / Toasts in DOM*

*3. search for a toast message in DOM when you see the Toast Message*

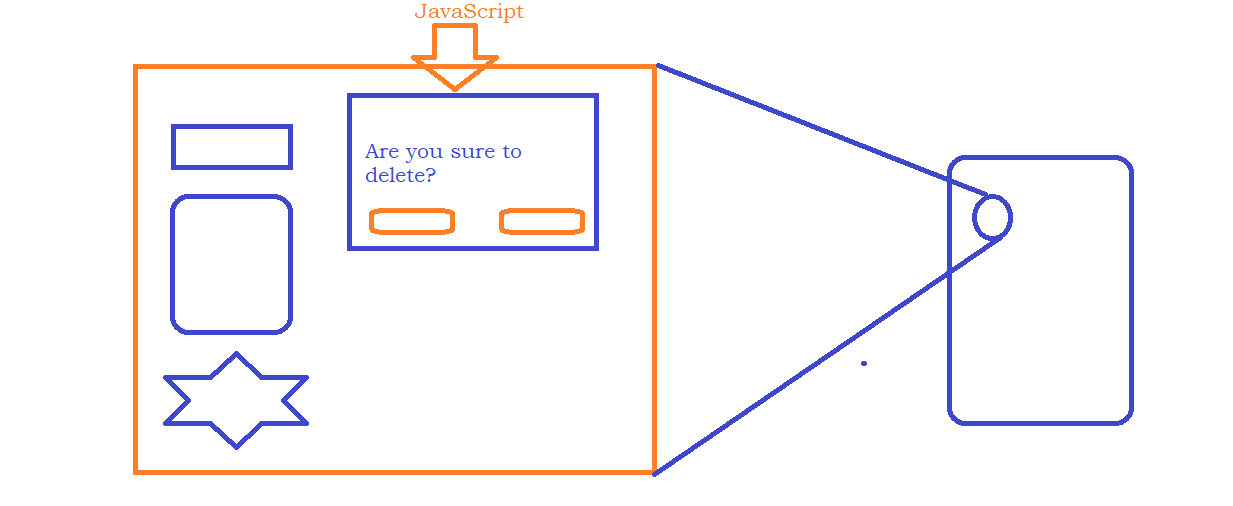
**Actions**

If we want to perform exact **keyboard** and **mouse Operations**

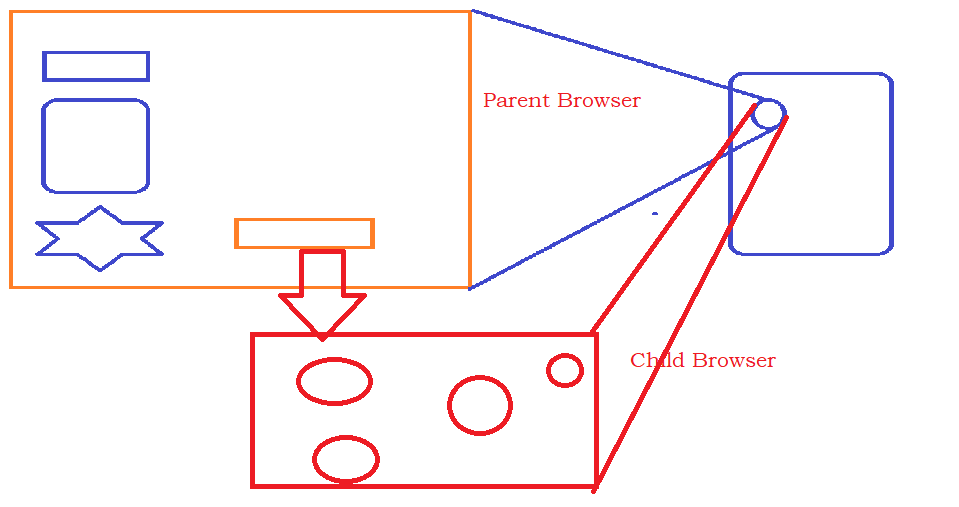


**SwitchTo()**

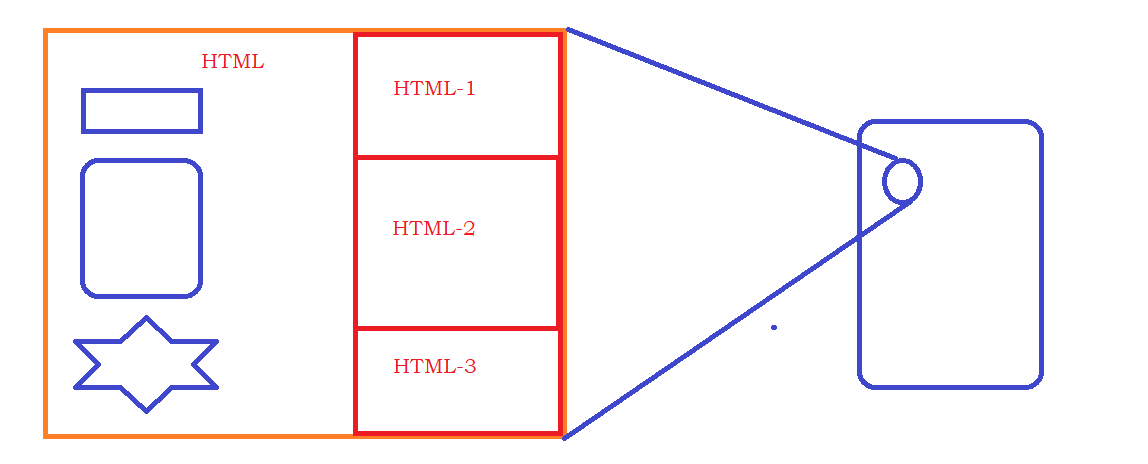
1. **Alerts / Confirmation popup – Due to java script**



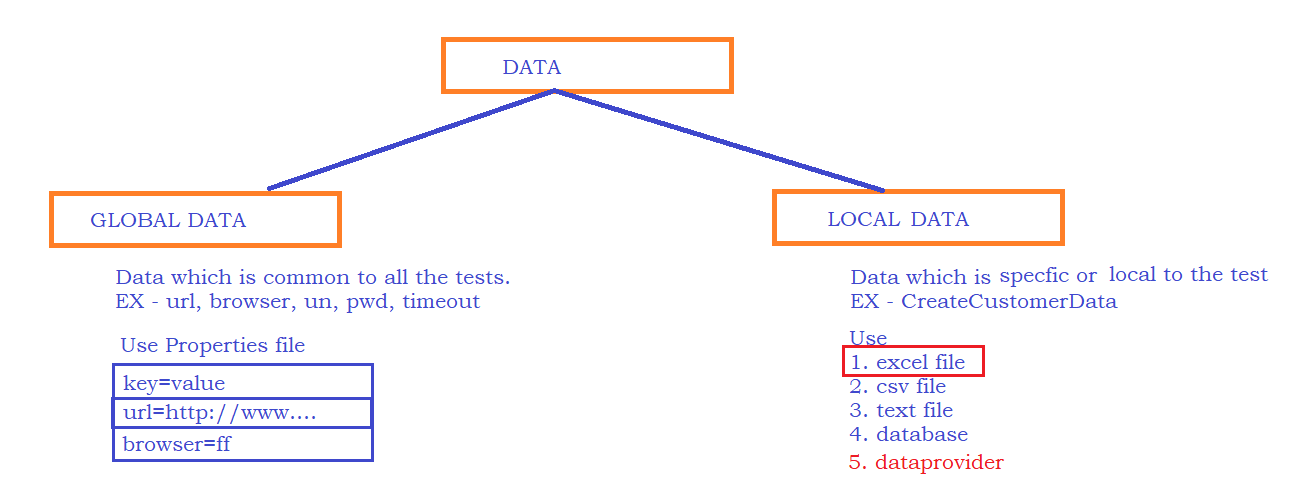
1. **Whenever there is a child Browser opened by the Application .**



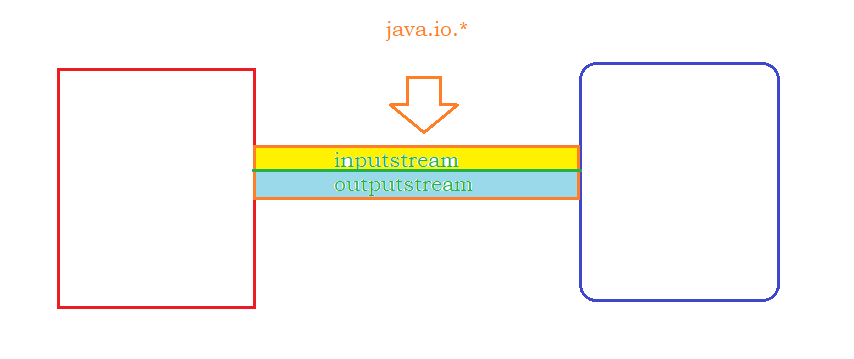
1. **Whenever there is a frame in the HTML File we have to go with switchto**
   1. **By index**
   2. **By Name**
   3. **By WebElement**

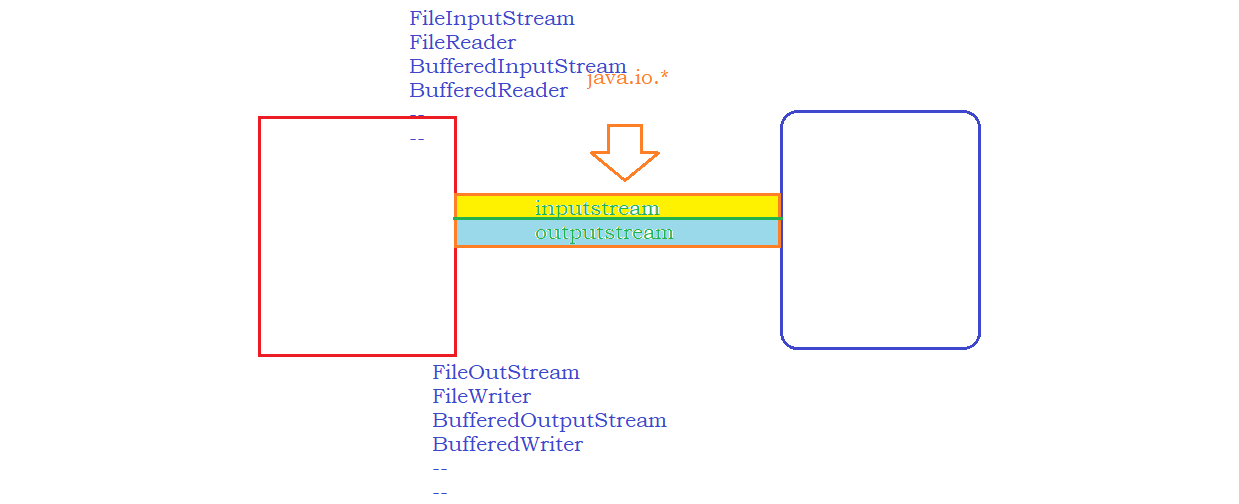
****

**Data driven Testing in Selenium**

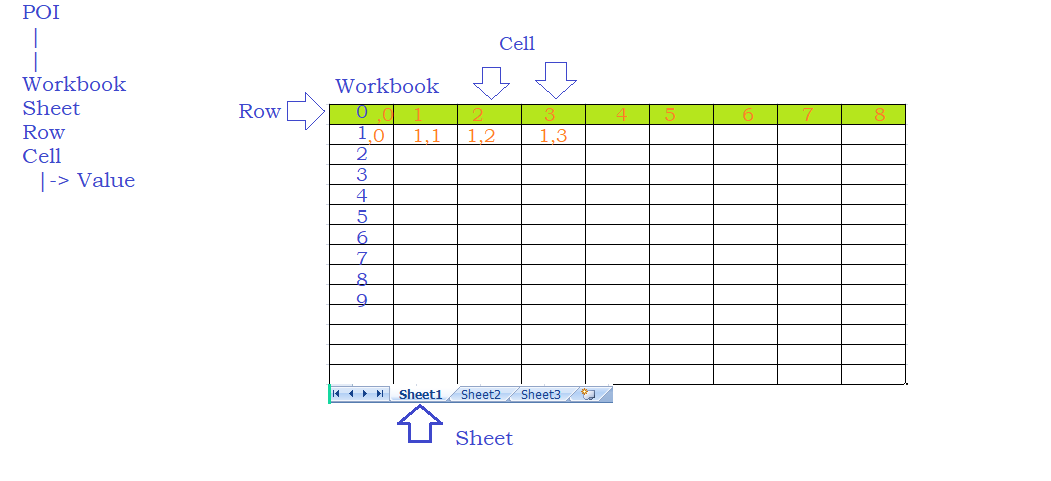
****

**File Reading and writing in JAVA**

****

****

**APACHE POI:**

****

**Handling PopUp In Selenium**

**1. Alerts –** using switchTo() we can handle this

**2. Confirmation –** using switchTo() we can handle this

**3. Hidden Division popup –** normal selenium code will work

**4. Authentication popup -**

**5. File Download popup -**

**6. File Upload popup –**

**FrameWorks –**

**Set of Guidelines or procedure one has to use while automating a test.**

* **How the tests should be written**
* **where we are storing the data (text, excel, db, dataprovider ... )**
* **How the reports should look like (html, pdf, excel)**
* **Execution (How tests should execute one after the other )**
* **Should it send email ???**
* **Re-executing the failed tests ???**
* **Create a defect for the failed test ?**
* **Capturing screenshot for the failed tests???**

**Types of Framework -**

* **Function Driven Automation Framework**
* **Data Driven Automation Framework**
* **KeyWord Driven Automation Framework ( Everything will be in excel )**
* **Hybrid Driven Automation Framework**
* **TestNG**
* **PageObjectModel**

**TestNG –**

**Ready made framework which act as Plugin for selenium tool.**

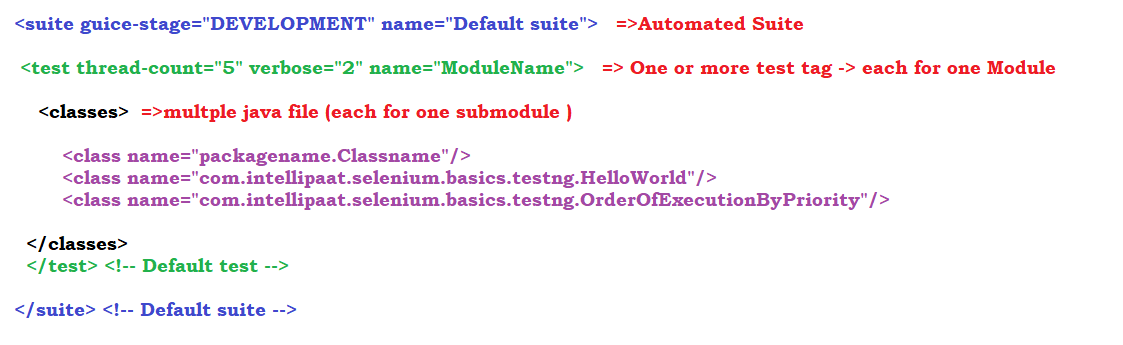
**junit and nunit ->**frameworks used by developers.

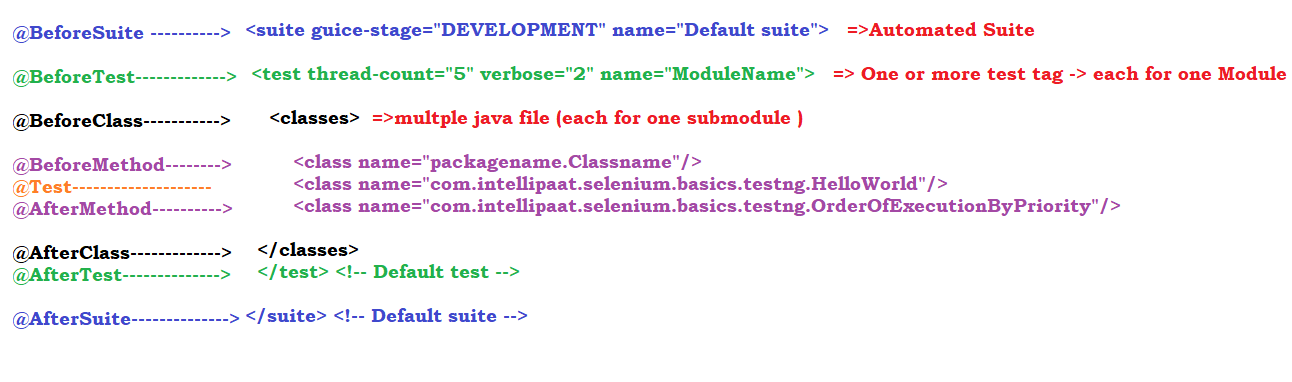
TestNG uses its own runtime environment, which search for annotation for the methods to execute.

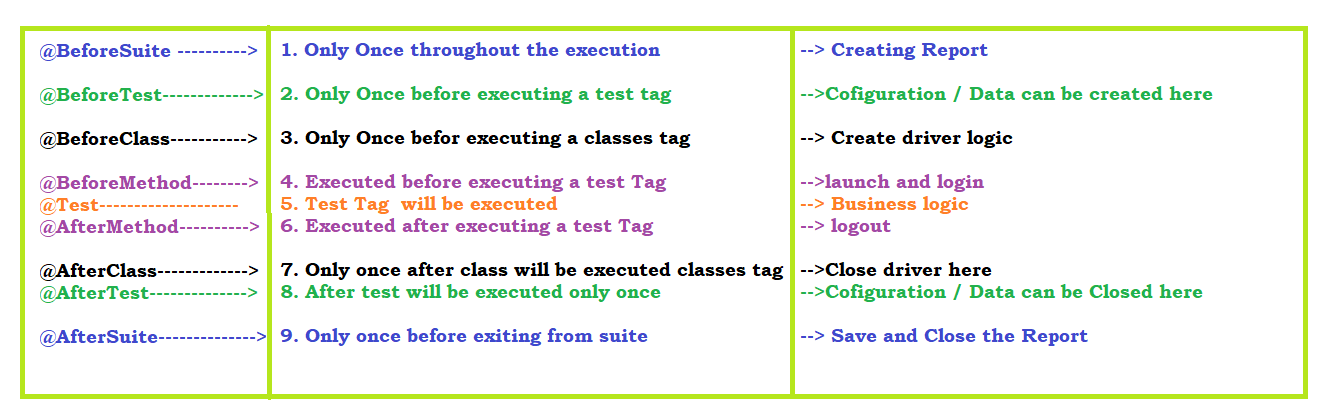
**OrderOfExecution ->** can be controlled by parameterising the test annotation

1. **dependsOnMethods**
2. **priority**

**TestNg xml file :**

****

**Annotations : **



**DATA PROVIDER –**

1. Parameterize you test with dataprovider and specify the data provider name (method name)

2. you should pass the same number of arguments in the same order whatever is returned by data provider.

GROUPS –

1. smoke

2. sanity

3. Regression

4. stage

5. dev

6. prod

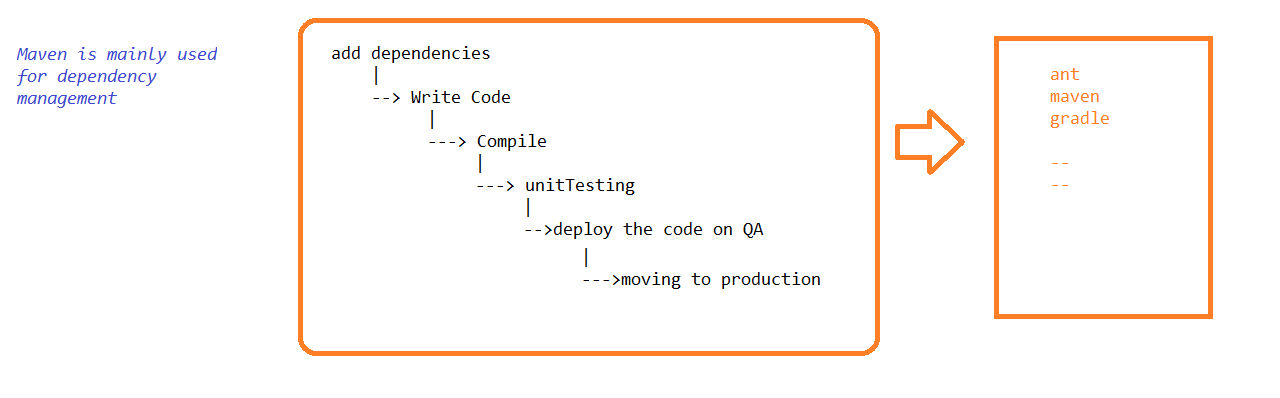
**Capturing Screenshot**

**Executing JavaScript in Selenium**

**Parallel Execution**

**Maven AND POM –**

Build Management Tool,

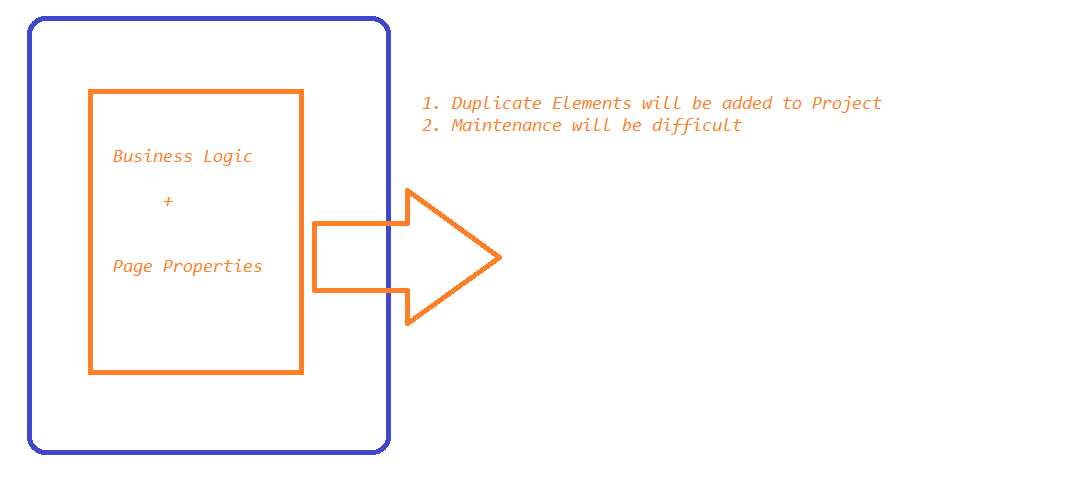


- Download and set environment Variables

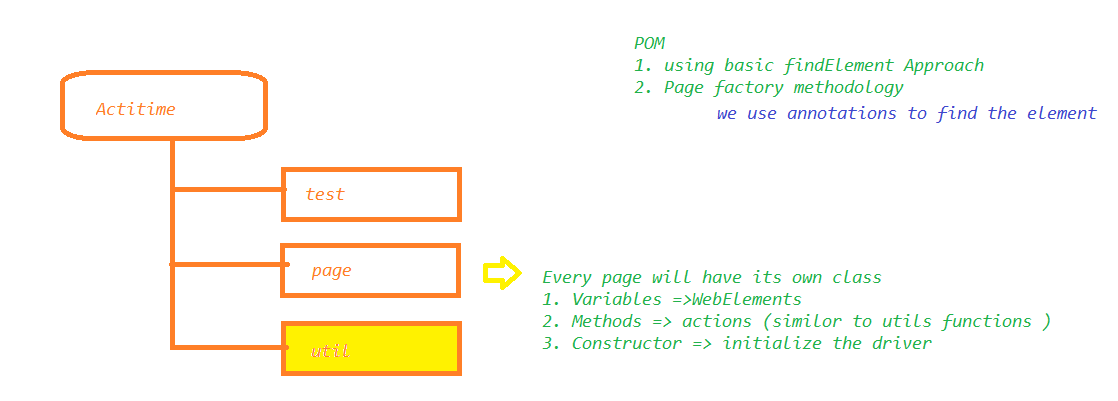
* <https://maven.apache.org/download.cgi> and download the latest zip file
* unzip the downloaded zip file
* MAVEN\_HOME - D:\maven\apache-maven-3.6.1
* M2 - D:\maven\apache-maven-3.6.1\bin
* PATH - D:\maven\apache-maven-3.6.1\bin

- Create Maven Project

POM –



POM



- MVN LifeCycle

- Executing code without opening eclipse

* validate - validate the project is correct and all necessary information is available
* compile - compile the source code of the project
* test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
* package - take the compiled code and package it in its distributable format, such as a JAR.
* verify - run any checks on results of integration tests to ensure quality criteria are met
* install - install the package into the local repository, for use as a dependency in other projects locally
* deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

**BDD**

**Cucumber**

**- gherkin**

**- cucumber framework is developed in ruby,**

Anyname.feature (DSL- Domain Specific Language)

Feature: Login Feature

Scenario : User Logs in to the application on valid credentials

**Given** when user is on login page

**And** he enter valid user name

**And** he enver valid password

**When** he click on login button

**Then** he will be able to see the dashboard page

Feature File

Gherkin KeyWords:

Feature:

Scenario:

Background

Given

When

Then

And

But

TestRunner

+

Junit Runner

Step Definition

Java, JavaScript, Ruby

Annotations

AUT

**TestRunner –**

Cucumber Options in test Runner:

@cucumberOptions(

**Features** =”location of the feature file from src”,

**Glue** = {“package name – location of your step definition ”},

**Monochrome** = Boolean ( to remove the junk cahrs in console output )

**dryRun** = Boolean ( to perform dryrun of steps { steps are not executed but only existence is verified } )

**plugin** / format = to generate Cucumber reports and to save Junit and Json Reports

**strict** = true 🡪 fail the TC and not continue if there is any failure in the middle of TC

)