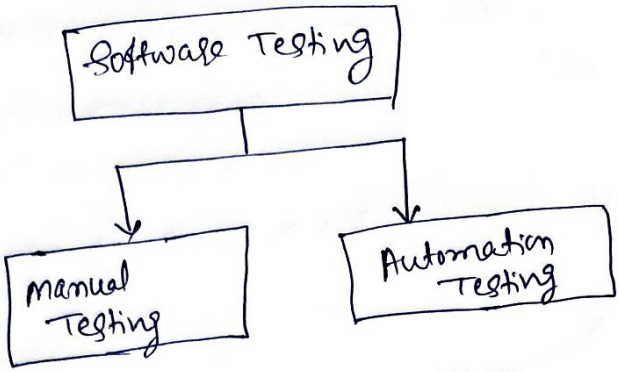
* Basics of Automation Testing *



Software Testing is divided into two parts

1. Manual Testing
2. Automation Testing

- Manual Testing is the process of testing the software manually to find the defects.
- manually the software will be tested and reports are generated without Automation tools.
- Types of manual testing, using them the tester will test the correctness of the application.
- Manual Testing is Best when you are performing
  - * Exploratory Testing
  - * usability Testing
  - * Ad-hoc Testing
  - * monkey Testing
- Manual Testing is prefered when the project is in initial development stage and if the project is a short term.
- To perform the manual testing the tester should not have knowledge on Automatin Tools.

Disadvantages of Manual Testing

1. Time consuming, mainly while doing regression testing
2. manual testing is less reliable – human involvement.
3. Expensive over automation testing in the long run.
4. No reuse of any testing process done.

- Automation Testing is the process of testing the software using an Automation tools to find the defects.

Examples of Automation Tools

1. selenium
2. HP QTP (Quick Test Professional) / UFT (Unified Functional Testing)
3. Load Runner
4. SilkTest
5. IBM Rational Functional Tester
6. TestComplete
7. WATIR
8. WinRunner
9. Ranorex studio

- Automation Testing will carried out during the following Testing areas.
  - → Regression Testing
  - → Load Testing
  - → Performance Testing
  - → Integration Testing
  - → System Testing
  - → Unit Testing
  - → Acceptance Testing
  - → Smoke Testing

What to Automate :→

- → High Risk Business Critical test cases
- → Test cases repeatedly executed
- → difficult to perform manually
- → Time consuming TC
- → To do Parallel Testing

* Advantages of Automation Testing :->

- Faster in execution, reliable, cheaper and reusable scripts.
- It does not requires human intervention.
- Compatibility testing
- Increase Test coverage.
- Testing reports are generated Automatically.

* Disadvantages of Automation Testing :->

- Best if your product is stable
- Automation testing is expensive initially.
- Automation testing has some limitations such as
  - Handling captcha
  - getting visual aspects of UI such as fonts, colors, sizes .... etc.
- Knowledge of Programming Language and Automation Tool Must

- When should one start test automation?

  one should start test automation only when all the manual testing process is carried out successfully.

  one need to be sure that Environment and build are stable.

- Which test cases should be considered for automation?

  (1) Tests which run for every build

  (2) Tests which use multiple values for the same action

  (3) Same tests to be performed on different hardware and software platforms

  (4) Regression tests.

Basics of Selenium

http://www.seleniumhq.org

Selenium

- Selenium is a web-based Automation tool, which automates the Browser
- Open source (free) Automation tool.
- Supports different Platforms and Browsers
- Selenium has four components namely.
  1. Selenium IDE (integrated development Environment)
  2. selenium RC (Remote control)
  3. selenium webDriver
  4. selenium Grid.

* Selenium IDE

- is a firefox plugin, allows us to record and playback the scripts.
- Even though we write scripts using Selenium IDE, we need to use selenium RC or webDriver to write more advanced and robust test cases.
- supports :→ windows, mac OS, Linux, Mozilla firefox.

* Selenium RC (selenium 1)

- Allowed users to use programming languages in creating complex test cases.
- supports Java, Javascript, Ruby, PHP, Python, Perl, C#
- supports all Browsers (Mozilla firefox, IE, chrome, safari, opera)

* Selenium webDriver

Selenium webDriver (Selenium 2) is a Browser Automation framework for automating the Browser.

- Allows test scripts to communicate directly with the Browser.
- Supports webdrivers for java, C#, PHP, Python, Perl, Ruby.
- Browsers support : Mozilla firefox, IE, Chrome safari, opera, Android, ios, HtmlUnit

Selenium Grid :→

- Used with RC to execute multipal test cases on different machines (Parallel test case execution) with different Browser and operating systems
- Selenium Grid is used manully for cross Browser testing.

Advantages of Selenium

(1) open source, supports multipal languages and platform
(2) Cross Browser testing and Headless testing
(3) Integrated with unit testing framework like TestNG, Junit, BDD, TDD.
(4) Avilable in the form of API's.
(5) Test cases can be exeuted on local machine as well as on cloud / virtual machine / servers

Ex: Browserstack, SouceLabs, Docker

(6) Easy to learn
(7) parallel Test execution
(8) Less Hardware usage.

Disadvantages of Selenium

(1) No direct support
- Mobile app ⇒ Selenium + Appium
- Reports ⇒ Selenium + TestNG or Extent reports
- Popup Handling ⇒ selenium + AutoIT (windows)
- Building project ⇒ maven
- CI-CD ⇒ Jenkins
- Testcase design ⇒ TestNG

(2) Need to learn languages

(3) can't automate captcha / Barcode

(4) can't automate windows desktop applications

(5) Selenium supports only web-based application automate.

Setup the Eclipse with selenium:

(1) install JDK

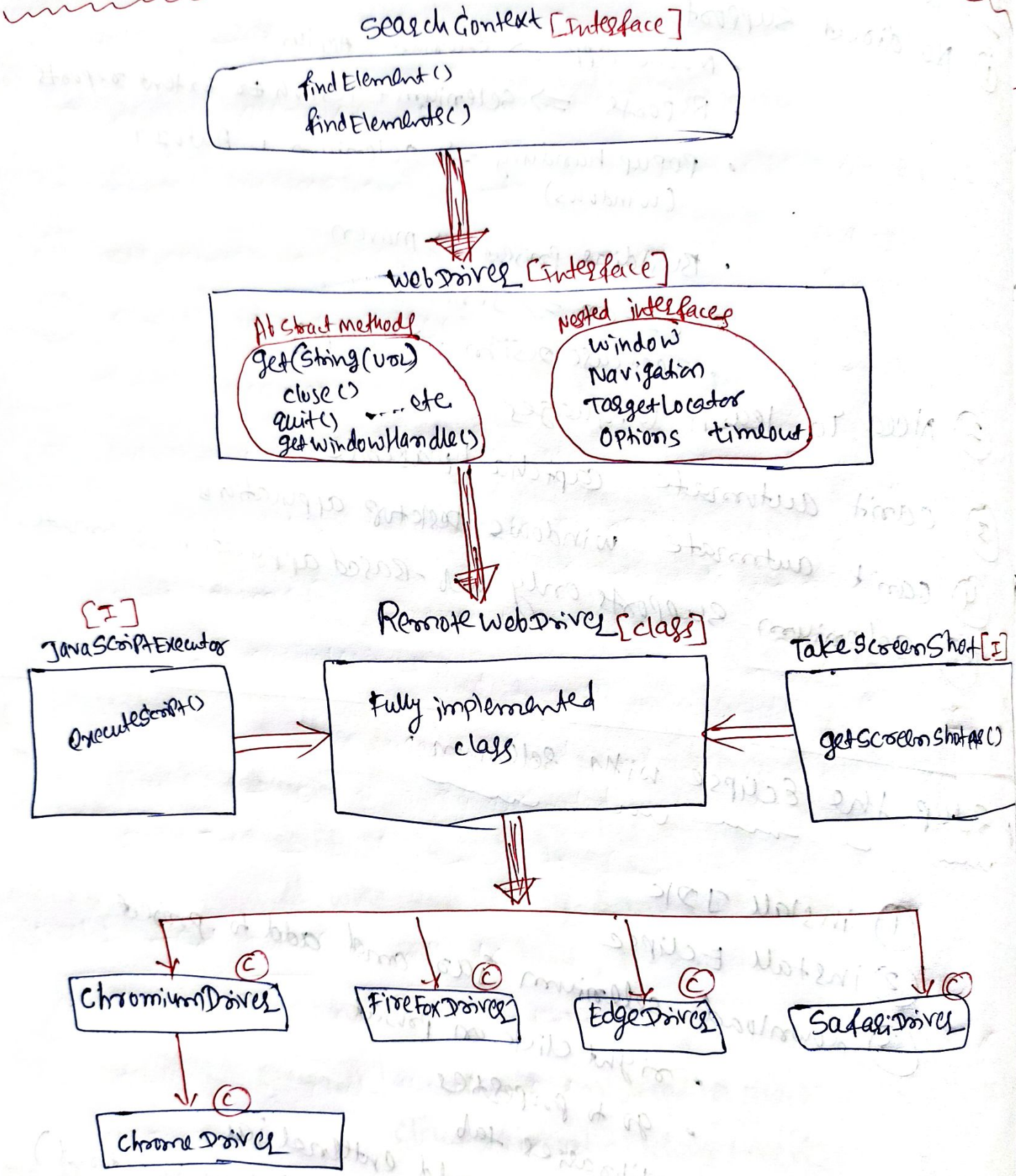
(2) install Eclipse

(3) download selenium jar and add to project
- right click on project
- go to properties
- libraries tab
- click on add external jar
- Browse selenium jar (where you stored)
- apply ⇒ OK

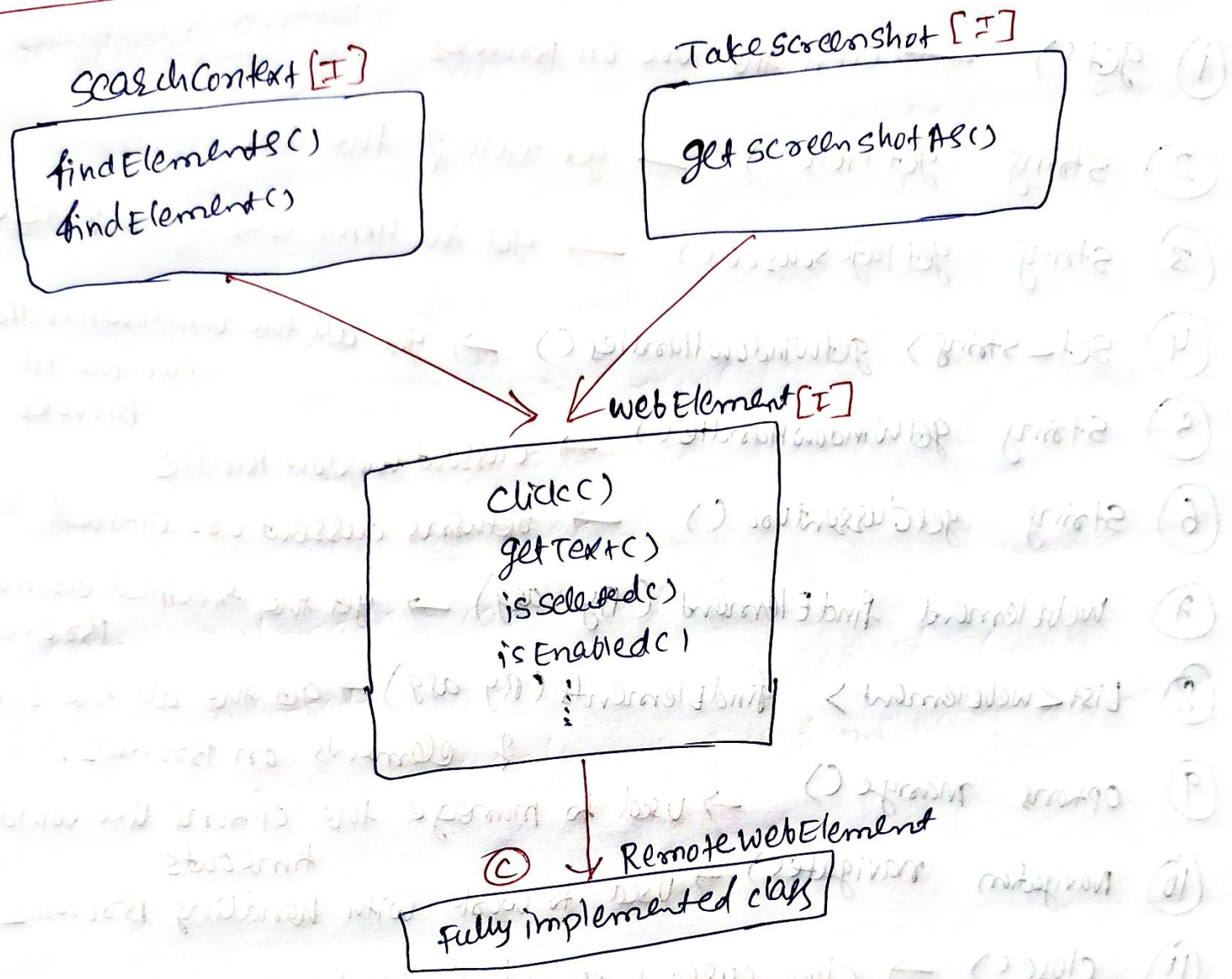
(4) see the jar added in Referenced libraries.

Similarly add "selenium-java" jar to explore the read the source code of selenium API.

WebDriver Interface Hierarchy



WebElement Interface Hierarchy



✱ WebDriver Interface Methods

① get() → Open the URL on Browser

② String getTitle() → get Title of the URL

③ String getPageSource() → get the source code of Page

④ Set<String> getWindowHandles() → get all the windowhandles (unique Id of Browser windows)

⑤ String getWindowHandle() → current window Handle

⑥ String getCurrentUrl() → returns current URL opened

⑦ WebElement findElement(By arg) → get the specified element on Browser

⑧ List<WebElement> findElements(By arg) → get the all the occurrence of elements on Browser.

⑨ Options manage() → used to manage the options like window, timeouts

⑩ Navigation navigate() → Used to work with handling Browser History

⑪ close() → close current opened window

⑫ Quit() → closing every associated window for an driver

⑬ SwitchTo() → Send commands to frame or window

✱ WebElement Interface methods

① clear() : if the element is a text entry, this will clear the value.

② click() : click the element

③ getText() : gets the visible text

④ getAttribute(String name) : returns the value of given attribute of the element.

⑤ getCssValue(String propertyName) : gets the value of a given css property.

⑥ getLocation() : gets the location of the rendered element in Points

⑦ getSize() : get the width and height of the element in dimension

(8) getTagName() : gets the tag name of the element

(9) boolean isDisplayed() : element displayed or not

(10) boolean isEnabled() : element enabled or not

(11) boolean isSelected() : element is selected or not

(12) sendKeys(String keystosend) : Sends on values to input fields.

(13) submit() : used to submit the form to the remote server

(14) findElement(By by) : finds the first webElement using given method

(15) findElements(By by) : find all elements within the current webpage using the method.

Locators :-

* Locators are the command (snippet code) used to find and match the elements of your webbrowser that is used to interact with.
- used to identify an HTML element
- To access all the locators Selenium provides the "By" class
- There are 8 - locators are in selenium to identify the web elements

(1) ID
(2) Name
(3) LinkText
(4) Partial LinkText
(5) Tag Name
(6) cssSelector
(7) Xpath

① Id

- id is used to select the element with a specified @id attribute.
- developer will assign a unique id for each element on the browser.
- Syntax:

driver.findElement(By.id("Enter here element id value"))

```
<input id="email" type="text" />
```

Example:

```
WebElement we = driver.findElement(By.id("email"));
```

② Name

- name is used to select the first element with specified @name attribute.
- many of the webelements have same name attribute value, it's not restriction for "name"
- Best to use when accessing similar types of elements.
- Syntax:

```
WebElement element = driver.findElement(By.name("user_name"));
```

```
<input id="user" name="user_name" type="text"/>
```

(3) linkText

- linkText is used to select the link element which contains the matching text.
- it is Best way to identify links on Browser.
- Only works with anchor tags
- used to find the Navigation within Browser.
- Syntax

 How to use ?

```
WebElement locator = driver.findElement(By.linkText(
                    "How to use ?"));
```

(4) PartialLinkText

- Used to select link (Anchor tags) elements which contains text matching the specified "partialLinkText"
- duplicate element possibility is more as we are partially passing the text.
- Syntax:

 How to use ?

```
WebElement locator = driver.findElement(By.partialLinkText(
                    "How to"));
```

(5) tagName

~~Those are~~

- Locates an element using the HTML tag.
- if we want to know the list of all the tags and their size, then go for tagName

Syntax:

<input id="email" type="text">

WebElement element = driver.findElement(By.tagName("input"));

(6) CSSSelector

- Locates an element using css selector.
- Syntax:

driver.findElement(By.cssSelector(<cssselector>));

Example:

driver.findElement(By.cssselector("#idvalue"));

- Relatively speedier than xpath.
- it's not easy to form cssselector and requires a deeper understanding of css/JS

(7) className

- Locates an element using the className attribute.
- Syntax:

WebElement e=driver.findElement(By.className("manage class"));

<input id="email" class="form-control" type="text">

Example

WebElement e=driver.findElement(By.className("form-control"));

(9) Xpath

- locates an Element using xpath query.
- we have to write an a xpath Expression to locate the element.
- Best way to traverse with DOM structure of webpage.

- Syntax:

```
WebElement ele = driver.findElement(By.xpath("xpath Expression"));
```

Example:

```
driver.findElement(By.xpath("//input[@id='email']"));
```

- it is an syntax for finding any element on the webpage using XML path expressions.

- Types of xpath
  1. Absolute path
  2. Relative path

* AbsoluteXpath:
  - direct way to find element
  - Any change in path xpath gets failed
  - starts with / (slash)
  - Adding or deleting in between tags will not access or locate desired element
  - path starts always form root node

Example:

```
html/body/div[2]/span/section/div[1]/div/div/h1[2]/b
```

* Relative Xpath:

- Path starts from choice of you node.
- starts with // (double Forward slash)
- if multipal elements in the path then it will select first element
- takes more time in identifying the element as we specify patial path
- Building relative xpath is time-consuming and difficult.

syntax: Xpath = //tagName [@attribute = 'value']

Ex:
```
//input [@id = 'email']
//* [@name = 'yourName']
```

* Syncronizing the Test cases with wait *

→ Test fails due to failure in loading page
→ Internet Issues, server down time.
→ Selenium throughs an exception "Element Not Visible Exception"
if element is not loaded due to some time interval.

→ Types of waits

1. Implicit wait
2. Explicit wait
3. Fluent wait

- Implicit wait applied on the driver instance, wait on every action we perform
- webdriver to wait for certain amount of time mentioned before it throws a "No such Element Exception"

- Even through high speed internet drivers holded with implicit wait

- waiting time to drive not on element.

Syntax:

```
driver.manage().timeouts().implicitlyWait(Timeout,
                TimeoutUnit.SECONDS);
```

Example:

```
WebDriver driver = new ChromeDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.
                SECONDS);
```

② Explicit wait

- ON elements
- Based on certain condition to be occured before throwing the exception.
- throws "ElementNotVisibleException"
- Combination of WebDriverWait and ExpectedConditions
- Better option then Thread.sleep(), where Program pause for mentioned time even through element is present.

- Syntax:

```
WebDriverWait wait = new WebDriverWait(driver, 15);
WebElement element = wait.until(ExpectedConditions.presenceOf
                     ElementLocated(By.id("EButton")));
```

Ex:

```
Ex: WebElement element2 = wait.until(ExpectedConditions.elementToBe
                          clickable(By.id("eleid")));
```

Note: Explore different methods of "ExpectedConditions" class

Imp

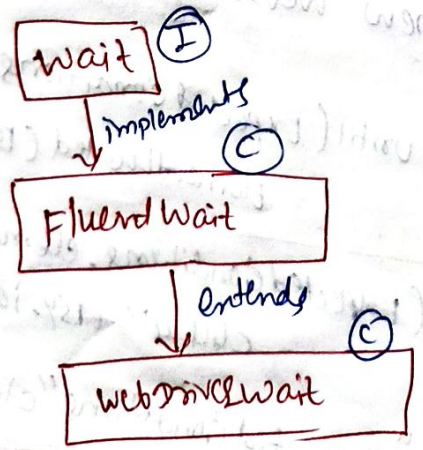
③ Fluent wait :

- The fluentwait in selenium is used to define maximum time for the webdriver to wait for condition as well as the frequency with which we want to check the condition before throwing an "ElementNotVisibleException" exception.
- Parameters → withTimeout : max timeout
  → pollingEvery : frequency/interval with which check for the condition.
- implemented through the wait class
- Explicit wait with configuration is fluent wait

Syntax :

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
        .withTimeout(30, TimeUnit.SECONDS)
        .pollingEvery(5, TimeUnit.SECONDS)
        .ignoring(NoSuchElementException.class);

wait.until(ExpectedConditions.ElementToBeClickable(By.
                        id("idvalue")));
```



Note : Explore the methods & the classes.
- Implicit vs Explicit vs Fluent

* Handling Dynamic Web Tables *

→ Table are the main element in the webpage, created with <table> tag
→ consists of rows and columns
→ Each row consists of table data called as <td> tag
→ rows are created using <tr> tag.
→ table headers are created with <th> tag.

Need: we need to validate the content of the table with Expected values.

- Using Selenium we can write a code which will displayes all the content of the table and we can locate each element within the table to find the Expected value.

Iterating the web table elements

```
WebElement table= driver.findElement(By.xpath("/html/body/table"));
List<WebElement> rows = table.findElements(By.tagName("tr"));
for(WebElement eachrow : rows) {
    List<WebElement> thead =eachrow.findElements(By.tagName
                                                 ("th"));
    for(WebElement hdata : thead) {
        Sop(" " +hdata.getText() +" ");
    }
List<WebElement> rowdata =eachrow.findElements(By.tagName
                                                 ("td"));
    for(WebElement cell : rowdata){
        Sop(" " + cell.getText() + " ");
    }
    Sopln();
}
}
```

HTML code for table structure

```
<table>
    <tr>
        <th> Name </th>
        <th> Mobile no </th>
        <th> City </th>
    </tr>
    <tr>
        <td> Kiran </td>
        <td> 34567 </td>
        <td> Germany </td>
    </tr>
    <tr>
        <td> Suresh </td>
        <td> 123890 </td>
        <td> Mexico </td>
    </tr>
    <tr>
        <td> puneet </td>
        <td> 12345 </td>
        <td> Australia </td>
    </tr>
    <tr>
        <td> Shradha </td>
        <td> 568906 </td>
        <td> UK </td>
    </tr>
    <tr>
        <td> Arati </td>
        <td> 24567 </td>
        <td> canada </td>
    </tr>
    <tr>
        <td> Pooja </td>
        <td> 111234 </td>
        <td> Italy </td>
    </tr>
</table>
```

table

tr one

row 1 data

tr two

row 2 data

tr three

row 3 data

tr four

row 4 data

tr five

row 5 data

tr six

row 6 data

tr seven

row 7 data

Handling the dropdown with Select class

- To perform the multiple operations on the dropdown object and multiple select object.
- Select class is in under the package ⇒ support.UI.select
- Step 1: create the WebElement reference for the dropdown object
- Step 2: create the object of the select class by passing the element object to constructor.
- step 3: using select class object reference perform the required operation on the dropdown.

Ex: select the dropdown option
deselect the dropdown option
find the options list for a dropdown
find the multiple option for a dropdown.

Select the dropdown option and deslect the dropdown options have many methods in select class.

methods from select class :⇒

① selectByValue("value");
② selectByIndex(int);
③ selectByVisibleText(String);
④ deselectByValue(String);
⑤ deSelectByVisibleText(String);
⑥ deselectByIndex(int);
⑦ isMultiple(); returns Boolean value
true: If Multiple option provided
false: If no multiple selection allowed
⑧ deselectAll() clear's all selected entry.

(9) List<WebElement> getOptions()

(10) List<WebElement> getAllSelectedOptions()

Sample HTML code for dropdown:

* The below list does not allow Multiple selection.

```
<form id="form1">
  <select width=300 style="width: 350px">
    <option value='blue'> Blue </option>
    <option value='green'> Green </option>
    <option value='red'> Red </option>
    <option value='yellow'> Yellow </option>
    <option value='black'> Black </option>
    <option value='' selected> Select a color </option>
  </select>
</form>
```

* The below list does allows multipleselections

```
<form id="form2">
  <select width=300 style="width: 350px" size="8"
          multiple>
    <option value='blue'> Blue </option>
    <option value='green'> Green </option>
    <option value='red'> Red </option>
    <option value='yellow'> Yellow </option>
    <option value='black'> Black </option>
  </select>
</form>
```

Selenium code :→

```
// No multi option allowed
WebElement dropdown = driver.findElement(By.xpath
                      ("//*[@id='form1']/select"));

Select sle = new Select(dropdown);
sle.selectByIndex(0);
sle.selectByVisibleText("Green");
sle.selectByValue("yellow");

List<WebElement> ele = sle.getoptions();
for(WebElement we : ele) {
    System.out.println(we.getText());
}

// with multi option allowed.
WebElement multisle = driver.findElement(By.xpath("//*[@id
                      ='form2']/select"));

Select multidrp = new Select(multisle);
multidrp.selectByIndex(0);
Thread.sleep(2000);
multidrp.selectByVisibleText("Green");
Thread.sleep(2000);
List<WebElement> ops = multidrp.getAllselectedoptions();
for(WebElement op : ops) {
    System.out.println(op.getText());
}
```

```
List<WebElement> optionselelist = multidrop.getoptions();

for (WebElement optionele : optionselelist) {

System.out.println(optionele.getText());

}
```

Handling cookies in selenium

- A small piece of codefile which consist of user information and preferences stored on webbrowser.
- Cookies assigned with name, domain, value, expiry, issecure.
- during the automation of smaller kind of scenarios where we have multipal login events like, place, viewcart, payment, address, order information, we have to login multipal time to avoid multilogin we can store small information in the form of cookie.
- values will be stored in the form of key-value pair
- we can Handle cookies with built in methods

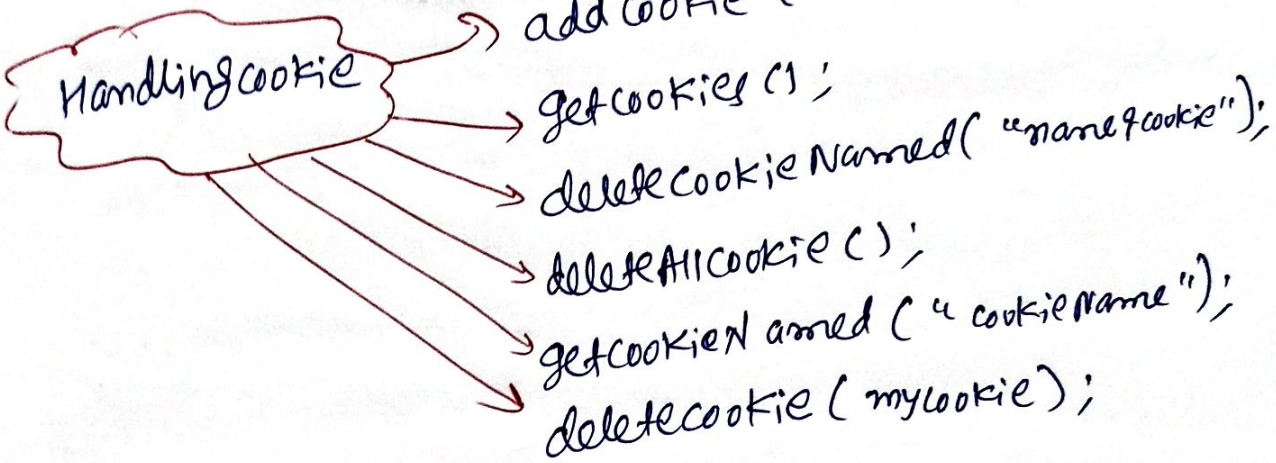
```
Cookie mycookie = new Cookie("key", "value");

driver.manage().addCookie(mycookie);
driver.manage().addCookie(new Cookie("foo", "oop"));

driver.manage().getCookieNamed("foo");
```

// Get all available cookies

```
Set<Cookie> cookies = driver.manage().getCookies();
    Sop(cookies);
driver.manage().addCookie(new Cookie("name", "mahader"));
driver.manage().deleteCookieNamed("foo");
driver.manage().deleteCookie(myCookie);
Set<Cookie> cookiesset = driver.manage().getCookies();
    Sop(cookiesset);
driver.manage().deleteAllCookies();
```



* Cookie class methods to get more information about cookie

Note: cookie object we can get all the information about the cookie

* Handling windows & Browser *

Webdriver provides us below methods to manage windows. We can control the size and position of the current window with the following methods.

Driver.manage.window()

→ .maximize() → it maximize the current window

→ .getSize() → it returns the size of the current window

→ .setSize() → it sets a new size of current window.

→ .setposition() → it moves the current window around.

→ .getposition() → it returns the current position of window.

* Selenium code:

```
driver.manage().window().maximize();
Dimension windowsize = driver.manage().window().getSize();
    Sop("default size:" + windowsize);
    Thread.sleep(2000);
Dimension setdim = new Dimension(windowsize.getwidth()/2,
                                 windowsize.getHeight()/2);
driver.manage().window().setSize(setdim);
Sop("New 1/2th size:" + driver.manage().window().getsize());
Thread.sleep(5000);
Point windowposition = driver.manage().window().getposition();
    SOP("current position" + "windowposition");
    Thread.sleep(5000);
```

```
Point newpoints = new Point(200, 200);
driver.manage().window().setPosition(newPoints);
Thread.sleep(5000);
SOP("New Position :" + driver.manage().window().getPosition());
```

Alert and Popup window Handling

- Alert are nothing But these are the small box which appears on the screen to give the user some information or notifications.

  used for
  - → Provide the information to end user
  - → To show the Error during application use.
  - → Permission taking.
  - → Take the input from the user.

- Types of Alert
  1. Simple Alert
  2. Prompt Alert
  3. Confirmation Alert
  4.

* Simple Alert : used to show information or warning on the screen.
* Prompt Alert : used for taking the inputs from the user
* Confirmation Alert : used for confirmation to perform certain operation, based on the permission provided.

Alert Interface provides the different methods are as follows.

① Void dismiss()

The dismiss method clicks on "cancel" Button on the Alert Popup.

Syntax: `driver.SwitchTo().alert().dismiss();`

② Void accept()

The accept method clicks on the "Ok" Button as soon as the Popup window appears.

Syntax: `driver.switchTo().alert().accept();`

③ String getText()

The getText method returns the text displayed on the alert box.

Syntax: `driver.SwitchTo().alert().getText();`

④ Void sendkeys (String value)

The sendkeys method enters the specified string pattern into alert box.

Syntax: `driver.switchTo().alert().sendkeys("Text");`

To Switch to the Alert box use the following Syntax.

```
Alert al= driver.switchTo().alert();
al.accept();
```

Selenium code for Alert box

```
driver.get("http://demo.automationtesting.in/Alerts.html");

driver.findElements(By.xPath("//*[@id='OkTab']")).click();
  sop("accept.....!!");
Alert alt = driver.switchTo().alert();
  sop(alt.getText());
  alt.accept();

driver.findElement(By.xPath("html/body/div[1]/div/div/div/div[1]/ul/
                                                    li[2]/a")).click();
driver.findElement(By.xPath("//*[@id='cancelTab']/button")).click();
  sop("dismiss.....!!");
Alert alt2 = driver.switchTo().alert();
  sop(alt2.getText());
  alt2.dismiss();

sop(driver.findElement(By.xPath("//*[@id='demo']")).getText());

driver.findElement(By.xPath("html/body/div[1]/div/div/div/div[1]
                                    /ul/li[2]/a")).click();
driver.findElement(By.xPath("//*[@id="cancelTab"]/button")).
                                                        click();
  sop("accept...!! (OK or Cancel)");
Alert alt3 = driver.switchTo().alert();
  sop(alt3.getText());
  alt3.accept();

sop(driver.findElement(By.xPath("//*[@id='demo']")).getText());

driver.findElement(By.xPath("html/body/div[1]/div/div/div/div[1]/ul
                                    /li[3]/a")).click();
driver.findElement(By.xPath("//*[@id='Textbox']/button")).click();
  sop("sendkeys.....!!");
Alert alt4 = driver.switchTo().alert();
  alt4.sendKeys("mahadev");
  alt4.accept();
sop(driver.findElement(By.xPath("//*[@id='demo1']")).
                                                    getText());

  driver.quit();
```

Handling Pop-up windows using WebDriver

→ Whenever we are using an application, where we have to switch between the multiple windows.

→ Once the operation has been done & user has to come back to main window (parent window).

→ Using the getWindowHandle() and getWindowHandles() method we can switch between the windows.

→ String getWindowHandle() : returns a unique identifier value of a current window.

→ String getWindowHandles() : returns a set of string of all the unique identifiers of the window that are opened.

Selenium code for window Handles

```
driver.get("http://demo.automationtesting.in/Windows.html");
String P_window = driver.getWindowHandle();
driver.findElement(By.xpath("//*[@id='Tabbed']/a/button")).click();
driver.findElement(By.xpath("/html/body/div[1]/div/div/div[1]/ul/li[2]/a")).
                                                    click();
Thread.sleep(2000);
Set<String> C_window = driver.getWindowHandles();
Sop("Parent window handle ---");
Sop(P_window);
Sop("parent window title:" + driver.getTitle());
```

```
Thread.sleep(2000);
Sop("List of all window Handles ---");
for (String win : C_window){
    if (!P_window.equalsIgnoreCase(win))
    // If Parent window is not equal to child window
    // Then get title of child window
    // and close child window
    {
    Sop(win);
    Thread.sleep(2000);
    drive.switchTo().window(win);
    Sop("window title:" +drive.getTitle());
    drive.close();
    }
drive.switchTo().window(P_window);
Sop("Parent window title :" +drive.getTitle());
drive.close();
```

* Handling frames *

The HTML frames are used to divide your Browser window into multiple sections, where each section can load a separate HTML document.

The window is divided into frames in similar way the table are organized into rows and columns.

The collection of frames in the Browser window is known as a frameset.

In General frames are nothing But HTML document in another HTML document.

frames are identified with <iframe> tag.

⇒ ways to locate the frames on Browser

1. frame id
2. frame index
3. frame xpath

Ex: driver.SwitchTo.frame("21410s");

driver.SwitchTo.frame(0);

driver.SwitchTo.frame(frameElement);

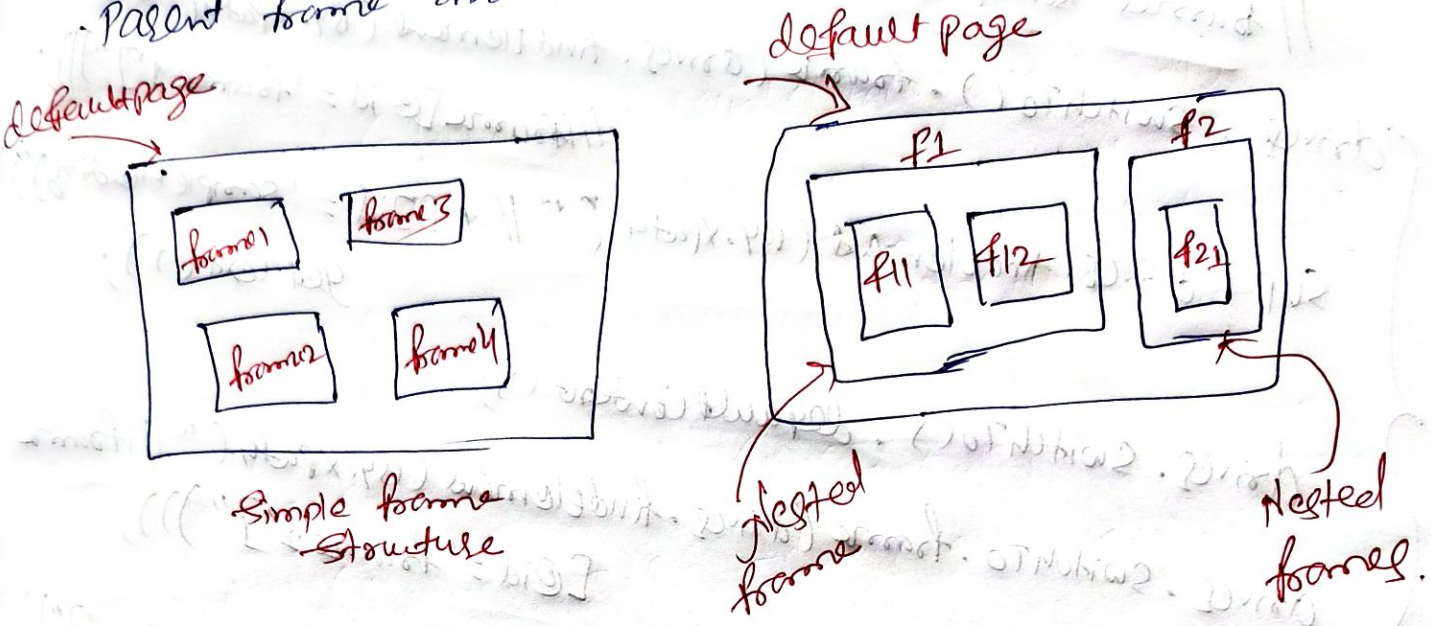
⇒ Methods:

* frame() : To Switch To a particular frame
* parentframe(): To SwitchTo Parent frame
* defaultContent(): To Switch To default page.

⇒ Exceptions:

* NoSuchFrame Exception: if the given element is neither an iframe nor frame element.
* StallElementReferenceException: if the element has gone stall.

- we can have a frame within another frame called as nested frames
- In the nested frame we need to switch to the parent frame and then we move to the child frame



- within the frames we can have all the tages that can be in HTML page.
- disadvantages of frame is that it's not supported by all Browser, screen resolution issue, Bookmark frames,

<frame> ⇒ HTML5 before
<iframe> ⇒ HTML5 and later.

- After working with the frames, main important is to come back to the default webpage using defaultContent() method. else Exception will be thrown.

P.T.O

Program ⇒

{Selenium code for frame handling}

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

//frames sample

driver.switchTo().frame(driver.findElement(By.xpath("//iframe[@id='frame1']")));

Sop(driver.findElement(By.xpath("//*[@id='sampleHeading']")).getText());

driver.switchTo().defaultContent();

driver.switchTo.frame(driver.findElement(By.xpath("//iframe[@id='frame2']")));

Sop(driver.findElement(By.xpath("//*[@id='sampleHeading']")).getText());

//Nested frames

driver.navigate().to("http://demo.automationtesting.in/Frames.html");

driver.findElements(By.linkText("Iframe with in an Iframe")).click();

driver.switchTo().frame(driver.findElement(By.xpath("//*[@id='Multiple']/iframe")));

driver.switchTo().frame(0);

driver.findElements(By.xpath("//input[@type='text']")).sendKeys("Hello");

Thread.sleep(5000);

//driver.switchTo().parentFrame();

driver.switchTo().defaultContent();

driver.findElement(By.linkText("Single Iframe")).click();