02/02/22 ①

Framework design - (imp) Learn how to explain

⇒ each & every page should have seperate page class. there are need to write BY locator

page Actn ⇒ features of that page

⇒ Assertions - should be written in TestNG.

⇒ selenium code written in page layer (1st layer)

⇒ DriverFactory.java ⇒ produce driver (2nd layer)

⇒ config.properties (3rd layer)

⇒ Testncr Layer (4th Layer) ⇒ Test class

(a) Test + assert

every Testncr class have some Base Test

assertion ⇒ test is passed or not (Not responsible for UI Action)

page can't write driver code in test layer.

⇒ Utility ⇒

⇒ Data Driven ⇒ (test data layer)

⇒ Reports

⇒ Test Runners

⇒ POM is used for web applicatn.

Framework → generic in nature.

① Page Layer ⇒

⇒ each & every page should have seperate class. (Java class). where we have 20 30 page so we need

⇒ In this, page class we will maintain By locator. we will use By locator approach.

Will create private By locator then create public page actions over here.

Page actions ⇒ feature of the page class.

for eg. login I do have login method for search, I have Search method for reg, I have reggi method.

Selenium code will be written in page layer

Inside page laye we will only define page methods and page locators defining with the help of Selenium.

page wise will write page actions.

② test layer (TestNG)

⇒ Assertions should be written in test classs.

Its a testNG responsibility.

⇒ test layer will contain all Assertions or Annotatn to validate the features.

3rd layer ⇒ DriverFactor.java → produce driver
New Chrome driver
New Firefox driver

init_prop()
init_driver()

4th ⇒ config.properties ⇒

⇒ we maintain config properties also known as environment variables.

eg. browser = chrome
url =
username =
password =
headless = true
highlight = true

* define ~~execute~~ given my test cases on QA, dev, stage, prod, UAT environment
* we have to write multi environment logic
* we have to maintain ~~multiple~~ no of configuratn file according to my env.
* same test should be executed on diff. environment.
* only thing is properties will be different ⇒ url will be diff.

⇒ It has to be read by init_prop()
↑
config prope

Config prope. को INITIALIZE driver factory को हम लिखेंगे यहाँ initialize होगा।

initialize driver ⇒ ① If I want to execute test cases on my local then I will be using driver = new ChromeDriver()

② But what if I want to execute my test cases from remote m/c like selenium grid then I will use remote webdriver here.

Run your test cases on Remote machine eg. on selenium grid with some desired Capabilities.

we will use concept like Thread local for better parallel execution.

ThreadLocal ~~method~~ Class

Test layer (testNG)

Annotation / test classes

* Every test class will have 2 things
  i) test class along with the assertions.

without assert? I can not write my test.

* Every testNG is having **Base test** which consist setup() & tearDown()

every test class/page extending BaseTest class

To write @Test, we need to call loginPage.java method call page action from page class → this method will return something. basis on that I will have to write Assertions

* we never use selenium code in testNG.

AccPageTest.java → it will login first then go to accpage.java then ResultPage.java
in test layer

call 3 class then whatever call results will get, ~~we~~ object which or something on which we will apply/write assertions.

so its called 'page object chaining model.

* driver is performing some actions on page
वो gA ves में EDIT दो driver handle करेगा।

whatever actions performed in class is correct or not it is validated through, assert. (that is what testNG is responsible for)

TestNG is not responsible for UI actions.

⑥ Utils/Libs, constants/java

In this we have to maintain utilities like element util, java script util.

How will you ~~create~~ click on button.

If I want to click ~~on~~ do for click I have to pass By locator to my element

util. get(), click(), synchronization(), wait()

whatever element utils we have created all utilities will maintain over there

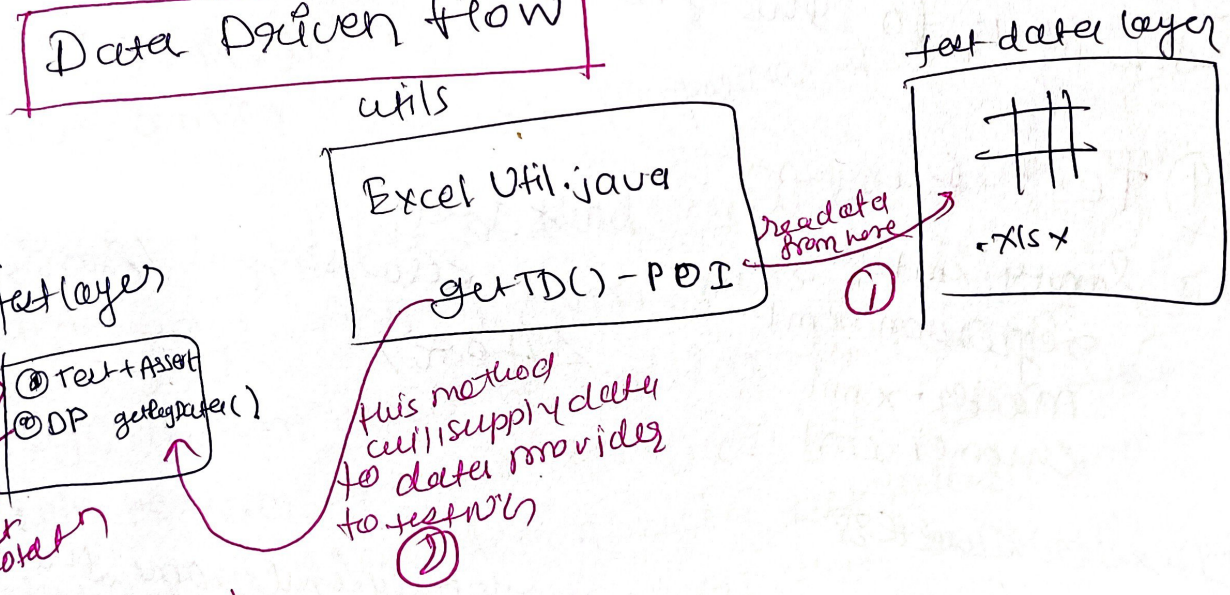
page layer will directly interact with this utils

* we have Javascript util for JS
* ExcelUtil.java with the help of Appache POI
* Constants.java
* library related utils like Error.java

⭐ maintain data driven approach ⭐

maintain data in excel file. ~~to no~~
then we have to read this excel data.
for this we have to use 3rd party library called POI library.

Data Driven flow



then test will execute against no. of rows available in ~~test~~ excel sheet

⑧ Test Reports / Logs

Selenium / Java does not provide any Report so we need to use our custom report.
eg. default TestNG Report, Extent Report

Extent Report → Listener

Allure Report → Listeners

Logs → log4j API

for every failure we will take Screenshot. we will attach this Screenshot to this file.

⑨ Test Runners.

Sanity.xml
regression.xml
master.xml
chrome.xml

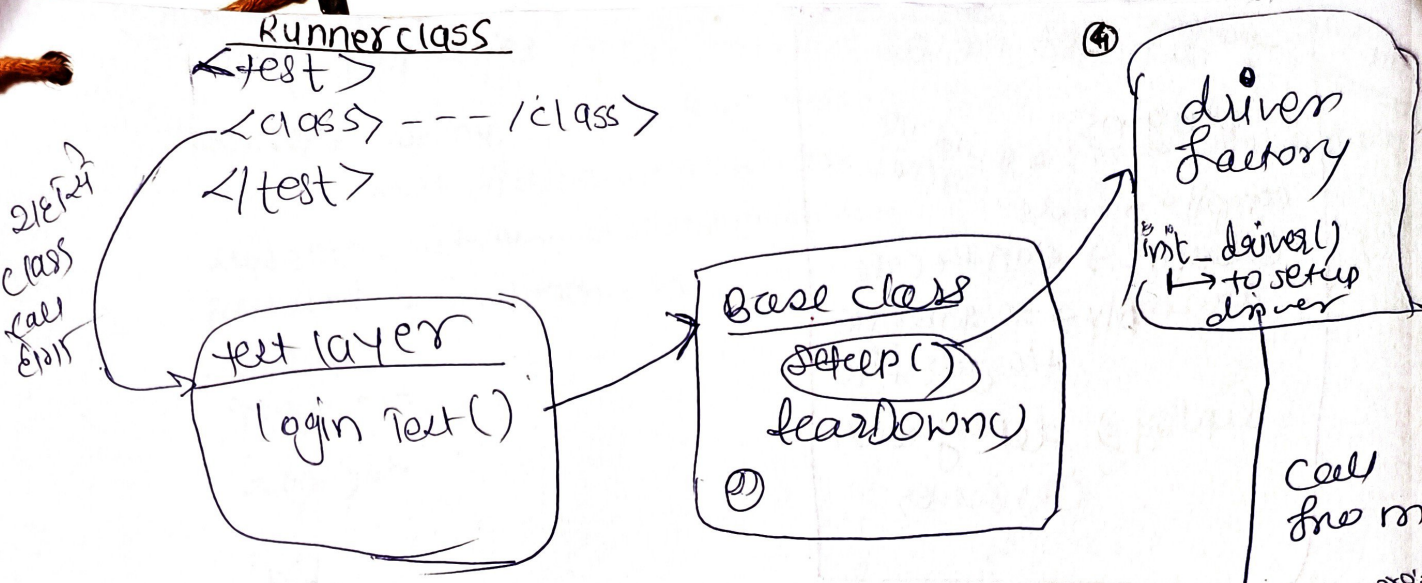
```
<test
    <class=---- /class>
</test>
```

Regression masters

chrome.xml (=> ~~only~~ test cases write only in chrome then run in a chrome.xml)
chrome specific test cases will run.

Runner class

<test>
<class> --- </class>
</test>



① Test Runner से class call होगा of test layer.

test layer में class B का class से function call होगा from driver factory like init_driver()

init_driver → config properties से → properties को get होगा

then login test method will be calling the page Action from page layer

page action will be calling to element util. will perform some action like get(), click() etc

then page layer Action will return something to test layer class to assert then test will be over.

Entire thing will be done through maven.

maven build life cycle ⇒ compile, test, assembly, deploy

Maven

- maintain dependencies
- plugins ↓ Build life cycle

compile → code
Surefire → Run the code
assembly → to generate the jar files
deploy → deploy build on server

Test Runners → Base Test → driver factory → config properties

Test layer → page layer → Element Utils

config.properties → In properties everything is String in key value pair

```
browser = chrome
url = https://demo.opencart.com/index.php?route=account/login
username = naveenanimation20@gmail.com
password = Selenium@12345
headless
incognito=
highlight=
```

DriverFactory.java

```
public class DriverFactory {
    public Properties prop;

    public Properties init_prop() {
        prop = new Properties();
        try {
            FileInputStream ip = new FileInputStream("./path");
            prop.load(ip);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return prop;
    }
```

// this method is used to initialize the properties
@return the properties class reference

```
public WebDriver init_driver(Properties prop) {
    String browserName = prop.getProperty("browser").trim();
```

(Ch) → driver.get(prop.getProperty("url").trim());

java class — helps to connect with config file — property

right click on path → path

./ path

./ ⇒ current project directory

Page object modal

- It is design pattern, used to design page classes. then there are such rules & requirements are defined that we have to follow those rules according to page object modal.

① create maven object (maven-archetype-quickstart) [file → new → maven project]

② groupID ⇒ com.opencart.NAT
(company, projectName, teamname)

ArtifactID ⇒ projectName eg. Bugdazz

③ delete dummy pkg. from src/main/java, src/test/java

④ add dependency (selenium, testNG)

⑤ right click → update → maven project

⑥ testNG code ⇒ src/test/java

⑦ other thing config file, xml, excel data → src/main/java

⑧ create pkg in src/main/java name as com.qa.opencart.factory

under factory create class → DriverFactory → public class DriverFactory { (component name)

class A → public WebDriver driver; (WebDriver) (Properties prop) → pointing to entire properties class object

public void init_driver(String browserName) {
    S.O.P("browser name is :" + browserName);
    if (browserName.equalsIgnoreCase("chrome")) {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
    }
    elseif(browserName.equalsIgnoreCase("firefox")){
        WebDriverManager.chromedriver().setup();
        driver = new FirefoxDriver();
    }
    else if (browserName.equalIgnoreCase("Safari")) {
        driver = new SafariDriver();
    }
    else{
        S.O.P("please pass the right broser name: "+ browserName);
    }
    driver.manage().deleteAllCookies();
    driver.manage().window().maximize();
    driver.get("");
    return driver();
}

this method is used to initialise the driver using browser name
@param browserName
return this returns the WebDriver

LoginP

⑤ create seperate pkg for page layer
src/main/java → create pkg → com.qa.opencart.pages → create class name as "LoginPage"

Page Layer → Always maintain by locator and page actions
By locator written with the private keyword so here, we achieve encapsulation.

write private variables is being used by the public methods
Called encapsulation. ⇒ public class LoginPage {
private ElementUtil eleUtil;
private WebDriver driver; # pointing to null
# help me to initialize my class variable that is driver variable will be initialized by given driver
public LoginPage(WebDriver driver) {
this.driver = driver;
eleUtil = new ElementUtil(driver);
}

driverfactory class will be responsible for initialize the driver

#write code.

```
private WebDriver driver;
//1. private by locator
private By emailId = By.id("input-email");
//  password = By.id("input-password");
//  loginBtn = By.xpath("//input[@value='login']");
//  forgotPwdLink = By.linkText("Forgotten Password");
```

2. page constructor:

```
public LoginPage(WebDriver driver) {
    this.driver = driver;
}
```
↓ same driver given to private WebDriver
initialized driver

3. page Actions (methods which are public)

```
public String getLoginPageTitle() {
    return driver.getTitle();
}
```
Constant.LOGIN_PAGE_TITLE, Constant.TIME_OUT
doGetPageTitle(actTitle, timeOut)
eleUtil → replace with method available in eleUtil

```
public String getLoginPageUrl() {
    return driver.getCurrentUrl();
}
```
eleUtil.getUrl()

```
public boolean isForgotPwdLinkExist() {
    return driver.findElement(forgotPwdLink).isDisplayed();
}
```

LoginPage.class

```
public void doLogin(String userName, String pwd){
    driver.findElement(emailId).sendKeys(userName);
    driver.findElement(password).sendKeys(pwd);
    driver.findElement(loginBtn).click();
}
```

LoginPage.class

```
public void doLogin(String userName, String pwd){
    driver.findElement(emailId).sendKeys(userName);
    driver.findElement(password).sendKeys(pwd);
    driver.findElement(loginBtn).click();
}
```

com.qa.opencart.pages
↳ LoginPage

```
public class LoginPage {
    # 1. private By locator

private By emailId = By.id("input-email");
private By
```

we can have multiple references for single object

create test class
① src/test/java → create package → com.qa.opencart.tests
→ create class → BaseTest.class (contains all pre annotation (launching browser url and post annotations)

write code

```
public class BaseTest {
    WebDriver driver;
    DriverFactory df;  # add pkg   properties prop;
    LoginPage loginpage;
```

reference is created

@BeforeTest
call this method using creating object
initialize df here

```
public void setUp() {
    df = new DriverFactory();  → prop = df.init_prop();
driver = df.init_Driver("chrome");
loginpage = new LoginPage(driver);
```

prop

we can have multiple references for single object
driverfactory का बार reference है
और वो return करे हर driver उसका अलग reference है
in BaseTest.java

```
loginPage = new LoginPage(driver);
```

write code

```
@AfterTest
public void tearDown() {
    driver.quit();
}
```

~~com.qa.pagetest.java~~
com.qa.opencart.tests → LoginPageTest.java

call page actions method

public class LoginPageTest extends BaseTest {

•

@Test
public void loginPageTitleTest() {
String title = loginPage.getLoginPageTitle();
S.O.P ("Login page title:" + title);
Assert.assertEquals (title, "Account Login");

→ Constants.LOGIN_PAGE_TITLE

Main purpose of test class is just to call the page class methods and asserts

Call the method & wherever the method is giving assert it.
without assertion test is not complete.

Page class never assert

@Test
public void loginPageURLTest(){
String url = loginPage.getLoginPageUrl();
S.O.P ("Login Page url:" + url);
Assert.assertEquals (title, "Account Login");
Assert.assertTrue (url.contains ("account/login"));
}

Constants.LOGIN_PAGE_URL_FRACTION

@Test
public void verifyForgotPwdLinkTest(){
Assert.assertTrue (loginPage.isForgotPwdLinkExist());
}

@Test
public void loginTest(){
loginPage.doLogin ("naveenanimation20@gmail.com", "Selenium@12345");

prop.getProperty("username").trim(),
prop.getProperty("password").trim());

}

}

Run

Config properties

its not java code

→ Create folder → right click on project → New → source folder → Name → src/test/resources

→ right click on it → folder (for files) → config (name) → create New file →

package → for java code

name as → config.properties (store value data in key and value pair format)

This are environment properties that is going to use in framework.

To read this ⇒ create function in DriverFactory for reading the properties from config.properties files

↳ reference create on top, then initialize

```
public void init_Prop() {
    prop = new Properties();
    FileInputStream ip = new FileInputStream("./src/test/resources/config.properties");
}
```

predefined class

copy path from right click on file → property → copy path

↳ help to connect with config.properties # import it

can throw exception

In properties file everything will be string.

Utils/libs → constant/java →

src/main/java → right click → package → com.qa.opencart.utils

→ create class → constants ⇒ expected values

public class constants {

same copy will be created & will be used in multiple test cases (common memory allocation)

public static final String Login_PAGE_TITLE = "Account Login";

public ——— URL = "account/login";

public ——— int Login_PAGE_URL_FRACTION DEFAULT_TIME_OUT = 5;

disad of Excel sheet ⇒ ① crashing issue
② maintain apache POI
③ constant data we never maintain in excel sheet

define constants here to ~~do~~ achieve remove hardcoded values. so can use this constants in login page.

Such dynamic data, registration page data, data getting change what if u have to maintain user friendly approach apart data, product search, registration data.

login page ~~is~~ will read all the methods from elementutil.java so fit it in copy existing java & paste it in com.qa.opencart.utils pkg.

→ creat object of ElementUtil.java in loginpage

generate html reports ⇒ index.html → right click → properties → copy path

What is use of maintaining element util class ?

Copy it to com.qa.opencart.utils.

page class will read all method of utility

create object in per LoginPage.java and use constants instead hardcoded value in utils

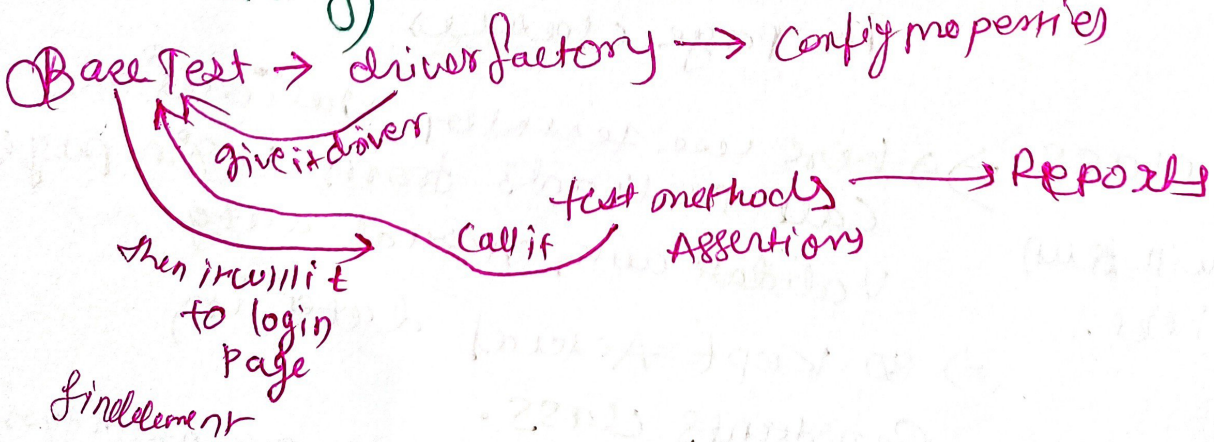
Element utils use in login page

copy utils paste in com.qa.opencart.utils (right click & paste)

Execution flow ⇒

Start from loginPage test → extending base test so go there → execute before test → call driver factory → & init driver

initialize chrome then rest of method will call like deleteAllCookies() --etc → return chrome driver → given to baseTest → Same driver pass to LoginPage → driver initialized in login page driver = new chromedriver → Setup method is over → navigate to loginPage test → Calling Login Page (login page reference is inheriting)



My Notes

⇒ Driverfactory मे driver initialize किया है but basetest class मे object बने हुए है उसके → driverfactory, loginpagetest, properties,

→ execution basetest से start होगा object create होकर driver initialize होगा driver factory call होगा aur driver initialize होगा then login page मे वही driver पास होगा aur operation perform होंगे

→ then test class से कुछ कुछ method of login page call होगा।

→ test class मे validate होगा using assertions

⇒ page class ⇒ ① BY locator से element मिलेगा (locators)
⇓
Automate your page
Selenium code

② perform actions on it

for performing Actions required Core Java [website मे कुछ भी आया तो उसको handle Java की help से करेंगे]

③ So make simple used elementUtils for code reusability in page classes

⇒ Test class ⇒⇒ here use testNG, validations
⇓ call methods from login page
program will run from here. Validate with Actual data

⇒ so kept Actual data in Constants class.
static final variables maintain these
So it is test driven approach
Not data driven

⇒ Driverfactory में properties class बनाया to define data in strings format eg. url, driver, username, pwd
it is one file in utils pkg

get (prop) object
↓
इसमे सब होगा हर जगह इसको pass करना होगा

to create object in base class & pass it to driverfactory

to extract data from it use

String browser = prop.getProperty (~~browser~~ driver)
↓
write key here

⇒ page pkg → AccountsPage.java

maintain By locator (which is private in nature)

testNG ⇒ call page class, get actual value & assert.

demo.opencart.com

```
private WebDriver driver;
private ElementUtil eleUtil;
private By header = By.cssSelector("div#logo a");
private By sections = By.cssSelector("div#content h2");
private By logoutLink = By.linkText("Logout");
private By search = By.name("search");
private By searchIcon = By.cssSelector("div#search button");

public AccountPage(WebDriver driver){
    this.driver = driver;
    eleUtil = new ElementUtil(driver);
}

public String getAccountPageTitle(){
    return eleUtil.doGetPageTitle(Constants.ACCOUNTS_PAGE_TITLE,
        Constants.DEFAULT_TIME_OUT);
}
```

→ test driven development approach

Constants.java ⇒ public static final String ACCOUNT_PAGE_TITLE = "My Account";

```
public String getAccountPageUrl(){
    return eleUtil.waitForUrlContains(Constants.ACCOUNT_PAGE_URL_FRACTION,
        Constants.DEFAULT);
}
```

public static final String ACCOUNT_PAGE_URL_FRACTION = "account/login";

ee route = account/account";

```
public String getAccPageHeader(){
    return eleUtil.doGetText(header);
}

public boolean isLogoutLinkExist(){
    return eleUtil.doIsDisplayed(logoutLink);
}
```

```
public void logout(){
    if(isLogoutLinkExist()){
        eleUtil.doClick(logoutLink);
        return true;
    }
    else{
        return false;
    }
}

public List<String> getAccPageSections(){
    eleUtil.waitForElementsVisible(locator, sections, 10);
    List<WebElement> sectionList =
    List<String> secValList = new ArrayList<String>();
    for(WebElement e: sectionList){
        String val = e.getText();
        secValList.add(val);
    }
    return secValList;
}
}

public boolean searchExist(){
    return eleUtil.doIsDisplayed(search);
}

public doSearch(String productName){
    if(searchExist()){
        eleUtil.doSendKeys(search, productName);
        eleUtil.doClick(searchIcon);
```

BaseTest.java ⇒ AccountPage accPage; // create reference

New test class ⇒ AccountPageTest.class extends base test

@BeforeClass
public void accPageSetUp(){
accPage = loginPage.doLogin(prop.getProperty("username"), prop.getProperty("password"));

→ return Acc class object
→ fetch it from basetest
→ it is returned here

Change → LoginPage.class ⇒ doLogin method में
public AccountPage doLogin

reference is attach here → return new AccountsPage(driver); # page class से Account page पर redirection
#written because it is new landing page

if you are navigating from some page to another then its method's (do login) responsibility to return that page object.

@Test
public void accPageTitleTest(){
String actTitle = accPage.getAccountPageTitle();
S.O.P(actTitle);
Assert.assertEquals(actTitle, Constants.Account_Page_Title);
}

@Test
public void accPageUrlTest(){
String actUrl = accPage.getAccountPageUrl();
S.O.P("Acc page url: " + actUrl);
Assert.assertTrue(actUrl.contains(Constants.Accounts_Page_Url_fraction));
}

@Test
public void accPageHeaderTest(){
String header = accPage.getAccPageHeader();
S.O.P("acc page header: " + header);
Assert.assertEquals(header, Constants.ACCOUNT_PAGE_HEADER);

add constant in Constants.java

```
public static final String ACCOUNTS_PAGE_HEADER = "Your Store";

@Test
public void logoutLinkTest() {
Assert.assertTrue(accpage.isLogoutLinkExist());
}
@Test
public void searchExistTest() {
    Assert.assertTrue(accPage.searchExist());
}
@Test
public void accPageSectionsTest() {
    List<String> accSectionsList = accPage.getAccPageSections();
    S.O.P("actual sec List:" + accSectionsList);
    Assert.assertEquals(accSectionsList, Constants.ACCOUNTS_PAGE_
                                        SECTIONS_LIST);
```

add in constants.java =>

```
public static final List<String> ACCOUNTS_PAGE_SECTIONS_LIST
    = Arrays.asList("My Account", "My Orders", "My Affiliate
                    Account", "Newsletter");
```

Change in → LoginPageTest.java

```
public void loginTest() {
    accPage = loginpage.doLogin(prop.getProperty("username").trim(),
            prop.getProperty("password").trim());
    Assert.assertTrue(accPage.isLogoutLinkExist());
```

data driven only for registeration form, search perform on diff data.

Contents are never hard coded. this is called hybrid

↓

combination of multiple design patterns

framework.

Registration page_POI API_Data Driven_Excel Data

LoginPage.java =>

add → private By registerLink = By.linkText("Register")

public RegisterPage gotoRegisterPage() {
eleUtil.doClick(registerLink);

after clicking on it where exactly we are landing on new page.

return new RegisterPage(driver)
next landing page object

this is test driven approach we need to create class of Register.

↳ Registeration Page-class + camera pic

public class RegisterPage {
private WebDriver driver;

copy paste code

public RegisterPage(WebDriver driver) {
this.driver = driver; import webdriver
eleUtil = new ElementUtil(driver);
}

public void accountRegistration(String firstName, String lastName, String email, String telephone, String password, String subscribe) {
eleUtil.doSendKeys(locator, value); # to fill register page

this locator parameter String

RegisterPageTest.java ⇒ extend base class to initialize driver

```
public class RegisterPageTest extends BaseTest{
    @BeforeClass
    public void regSetUp(){
```

Maintaing all page references in base class so,

BaseTest.java → RegisterPage regPage;

```
        regPage = loginPage.goToRegisterPage();
    }
    @Test
    public void userRegistrationTest(){
AssertTrue(regPage.accountRegistration("Ram", "Vaada", "r.h@gmail.com",
                                       "90966", "ram@123", "yes"));
```

#Run code

Data can't register with same data all time so implement data driven testing concept is in testNG

create sheet enter data there then move that file into src/test/resource → right click → folder ⇒ testdata

Inbuilt JDK provide library to read data from excel sheet

Java does not provide any library for data driven

we have to use 3rd party library. Apache POI API → based on java

Selenium does not provide any library to read excel sheet

Appache POI API Library → Powerful library to read data from excel sheet.

pom.xml ⇒ add dependencies → save → display in maven dependencies

⇒ Create excel utility in com.qa.opencart.utils

→ right click → new → class → ExcelUtil.java

maintain ~~data~~ method which will help me to read the data from excel sheet

public class ExcelUtil {
    private static Sheet sheet;
    private static Workbook book;
    public static String TEST_DATA_SHEET = "excelSheet", "./path";

    public static void getTestData(String sheetName) {
        try {
            FS ip = new FS(TEST_DATA_SHEET); # create connection with excel sheet with Java memory
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        book = WorkbookFactory.create(ip); # create a copy of excel sheet in Java memory with create method
        sheet = book.getSheet(sheetName);
    }
}

(FS) → FileInputStream # to make connect with file

right click on properties from src

↳ import org.apache.poi.ss.usermodel

RegisterPage.java ⇒

Multiple page chaining

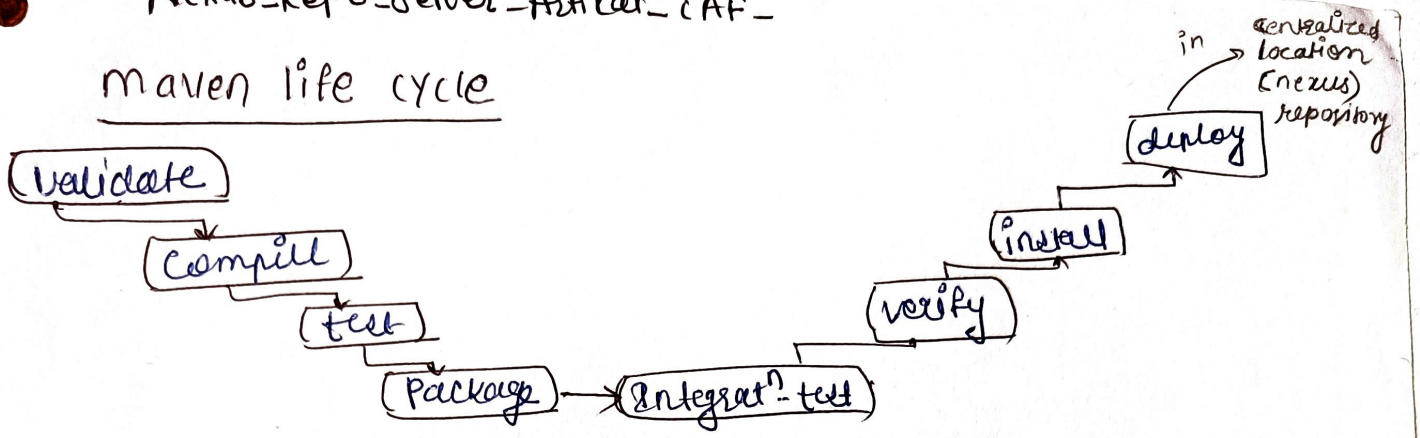
① Search macbook

its method responsibility to return next landing page

ResultsPage.class ⇒

Nexus_Repo_Server_Artifact_CAF_

maven life cycle



To do this we require plugin.

Common automat^n framework [CAF] handle by TCOE (center of excellence in testing)

→ provide packaging (package) in the form of jars.

maven centralized repository used for open source project

nexus repository for internal projects

Nexus → common repository management tool

Jar files are called "artifacts".

Nexus provide jar files.

① what is Automation process

existing framework → creating test cases

download jar & set dependency

git is for source code not for jar