Access specier:

public - it accseable any whre of the program.

proceted - it is accessable within a packegae .

it can be accessable outside but incase of inheritance.

default - it is accessable only inside the packegae where it is declared.

private - it is accessable only in the class where it is declared

variable :

int -> 0, float->0.0, char-> null,double -> 0.0000

String return type -> null

String str="sunil"

Method :

1. Concret mthod -> does have any body part and method declaration

<Access spe> <return type > <method NAme>()

{

body

}

2. Abstract method -> does not have any body part only method declaration is there

<Access spe> abstract <return type > <method NAme>();

EX: public abstract void abc();

Java Classes: A Class is like a "blueprint" for creating objects.

We can write variable and methods inside Class

public class Demo {

int a=2; - variable

static int b=3;

public static void main(String[] args) { -> methods

}

}

static keyword: If any method or varible is having static keyword we can call te method or variable directly by using there class name.

1. static member can be called directly from static block or non static block

public class Demo {

int a=2;

static int b=3;

public static void car() {

System.out.println("abc");

}

public static void main(String[] args) {

System.out.println(Demo.b);

Demo.car();

}

}

nonstatic : if you are not decalring static keyword it menas that is non -static method or variable.

1. non static member can not be called directly from static block .

public class Demo

{

int a=2; // non static variable

static int b=3; // static variable

public static void car() // static method

{

System.out.println("abc");

System.out.println("xyx");

}

public void cycle() // static method

{

System.out.println("abc");

System.out.println("xyx");

}

public static void main(String[] args) {

System.out.println(Demo.b); // 3

Demo.car(); // abc

//xyx

// calling Non static member (variable or method)

Demo d=new Demo(); //How TO OBJect:

System.out.println(d.a);

d.cycle();

}

}

2. non static member can call directly from non static block .

public void cycle() // non static method

{

System.out.println("abc");

System.out.println("xyx");

}

public void get() // non static method

{

System.out.println("abc");

cycle();

}

Constructor:

It is used to initialize a object.\

Type of Constructor:

1. Default Constructor
2. Parameterized Constructor
3. Constructor overloading ->

Creatiating multiple constructor in the same class but variation in the length,position and type of arugment.

1. Copy Constructor ->

**public** **class** Demo1 {

**public** Demo1()

{

System.***out***.println("aa");

}

**public** Demo1(**int** a) {

System.***out***.println("parametarized1");

}

**public** Demo1(**char** b) {

System.***out***.println("parametarized2");

}

**public** Demo1(**char** c,**int** b) {

System.***out***.println("parametarized3");

}

**public** **static** **void** main(String[] args) {

Demo1 d=**new** Demo1(); // aa

Demo1 d1=**new** Demo1(2); //parametarized

Demo1 d2=**new** Demo1('a');

Demo1 d3=**new** Demo1('b',3);

output:

aa

parametarized1

parametarized2

parametarized3

}

}

Method Ovrloading:

**public** **class** MethodOverloading {

**public** **static** **void** test()

{

System.***out***.println("test");

}

**public** **static** **void** test(**int** a,**int** b)

{

System.***out***.println("test1");

}

**public** **static** **void** test(**boolean** b)

{

System.***out***.println("test2");

}

**public** **void** test(**char** c)

{

System.***out***.println("test3");

}

**public** **static** **void** main(String[] args) {

MethodOverloading.*test*();

MethodOverloading.*test*(2, 3);

MethodOverloading mo=**new** MethodOverloading();

mo.test('c');

MethodOverloading.*test*(**true**);

}

}

output:

test

test1

test3

test2

Global variable & Local variable:

**public** **class** MethodOverloading {

**int** a=2; // global variable

**public** **static** **void** test()

{

**int** i=3; // local variable

System.***out***.println("test");

System.***out***.println(i);

}

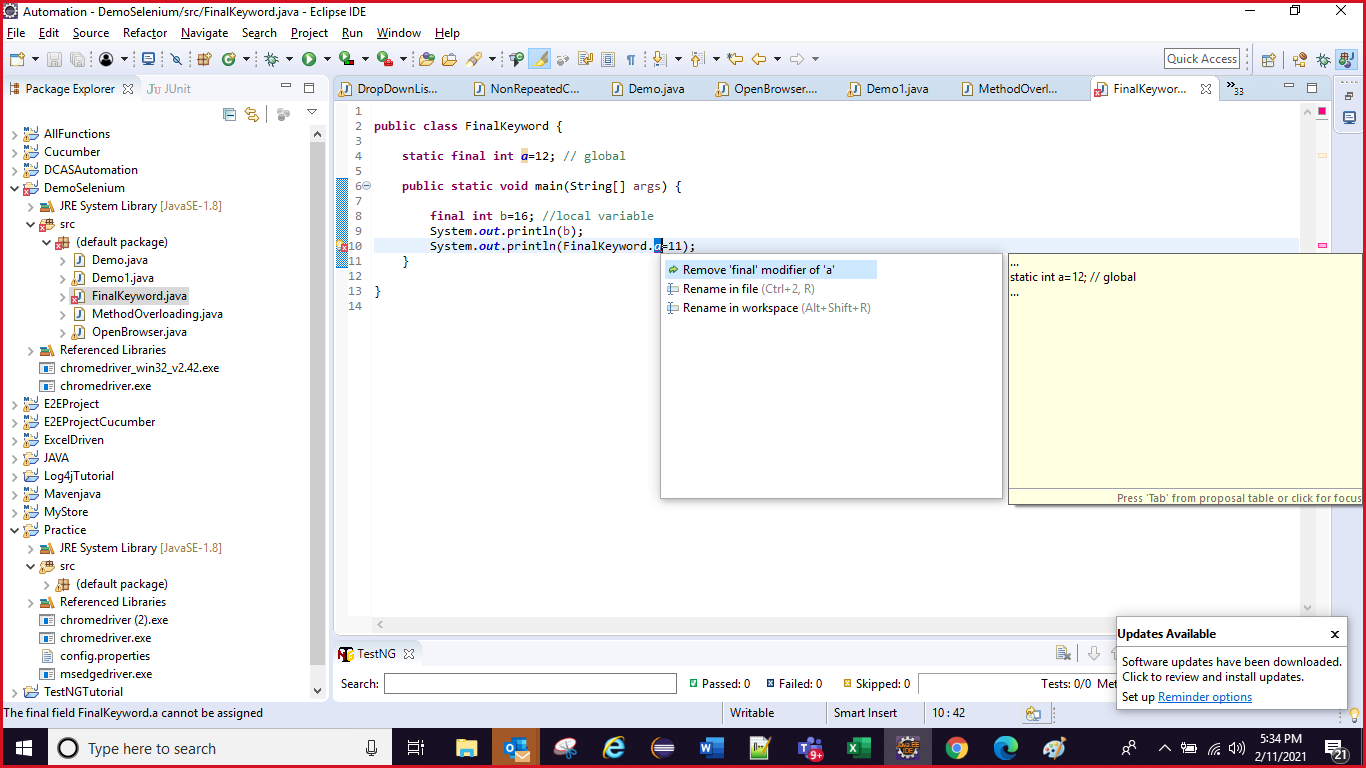
**public** **static** **void** test(**int** a,**int** b) //// local variable

{

System.***out***.println("test1");

}

# Final Keyword:



# this Keyword:

**public** **class** MethodOverloading {

**int** a; // global

**public** **void** test(**int** a)

{

**this**.a=a ;

}

**public** **static** **void** main(String[] args) {

MethodOverloading mo=**new** MethodOverloading();

mo.test(2);

}

this() statement:

**public** **class** ThisCAllingStatement {

**public** ThisCAllingStatement()

{

**this**(2);

System.***out***.println("aaa");

}

**public** ThisCAllingStatement(**int** a)

{

**this**(4,6);

System.***out***.println("BBB");

}

**public** ThisCAllingStatement(**int** a,**int** b)

{

**this**(3,'r');

System.***out***.println("CCC");

}

**public** ThisCAllingStatement(**int** a,**char** b)

{

System.***out***.println("DDD");

}

**public** **static** **void** main(String[] args) {

ThisCAllingStatement ts=**new** ThisCAllingStatement();

}

}

OOPs Concept:

Inheritance:

**class** Father

{

**public** **static** **void** swimmer()

{

System.***out***.println("good swimmer");

}

**public** **void** runner()

{

System.***out***.println("good runner");

}

}

**class** Son **extends** Father {

**public** **static** **void** singer()

{

System.***out***.println("good singer");

}

}

**public** **class** DemoInheritance

{

**public** **static** **void** main(String[] args) {

Father.*swimmer*();

Son.*singer*();

Son.*swimmer*();

}

}

2. Multiple Inheritance:

**class** Father

{

**public** **static** **void** swimmer()

{

System.***out***.println("good swimmer");

}

}

**class** Son **extends** Father {

**public** **static** **void** singer()

{

System.***out***.println("good singer");

}

}

**class** GrandSon **extends** Son {

**public** **static** **void** Painter()

{

System.***out***.println("good Painter");

}

}

**public** **class** DemoInheritance

{

**public** **static** **void** main(String[] args) {

Father.*swimmer*();

Son.*singer*();

Son.*swimmer*();

GrandSon.*swimmer*();

GrandSon.*Painter*();

GrandSon.*singer*();

}

}

3. Hierarchical Inheritance :

**class** Father

{

**public** **static** **void** swimmer()

{

System.***out***.println("good swimmer");

}

}

**class** Son **extends** Father {

**public** **static** **void** singer()

{

System.***out***.println("good singer");

}

}

**class** GrandSon **extends** Father {

**public** **static** **void** Painter()

{

System.***out***.println("good Painter");

}

}

**public** **class** DemoInheritance

{

**public** **static** **void** main(String[] args) {

Father.*swimmer*();

Son.*swimmer*();

GrandSon.*swimmer*();

GrandSon.*Painter*();

GrandSon.singer();

}

}

Super() calling state ment:

**class** Father

{

**public** Father()

{

//super();

System.***out***.println("Father is executing");

}

**public** **static** **void** swimmer()

{

System.***out***.println("good swimmer");

}

}

**public** **class** DemoInheritance **extends** Father

{

**public** DemoInheritance()

{

**super**();

System.***out***.println("DemoInheritance is executing ");

}

**public** **static** **void** main(String[] args) {

DemoInheritance d=**new** DemoInheritance();

}

}

Method Overriding:

**class** GrdandFather

{

**public** **void** swimmer()

{

System.***out***.println("Father is good swimmer");

}

}

**class** Son **extends** GrdandFather

{

**public** **void** swimmer()

{

System.***out***.println("Son is good swimmer");

}

}

**public** **class** MethodOverride {

**public** **static** **void** main(String[] args) {

Son s=**new** Son();

s.swimmer();

}

}

Interface:

**Ex 1:**

**interface** ABC

{

**void** set();

}

**interface** XYZ

{

**void** get();

}

**public** **class** Dustin **implements** ABC,XYZ

{

**public** **void** get()

{

System.***out***.println("get method");

}

**public** **void** set()

{

System.***out***.println("set method");

}

**public** **static** **void** main(String[] args) {

Dustin d=**new** Dustin();

d.get();

d.set();

}

}

Ex:2

**interface** ABC

{

**void** set();

**void** get();

}

**public** **class** Dustin **implements** ABC

{

**public** **void** get()

{

System.***out***.println("get method");

}

**public** **void** set()

{

System.***out***.println("set method");

}

**public** **static** **void** main(String[] args) {

Dustin d=**new** Dustin();

d.get();

d.set();

}

}

Encapsulation:

Variable:

**class** Demo3

{

**private** **int** a=4;

**public** **int** getter() {

**return** a; //5

}

/\*public void setter(int a)

{

this.a=a;

}\*/

}

**public** **class** Encapsulation {

**public** **static** **void** main(String[] args) {

//Encapsulation en=new Encapsulation();

Demo3 d3=**new** Demo3();

//d3.setter(5);

System.***out***.println(d3.getter());

}

}

2.

**class** Demo3

{

**private** **int** a=4;

**public** **int** getter() {

**return** **this**.a;

}

**public** **void** setter(**int** a)

{

**this**.a=a;

}

}

**public** **class** Encapsulation {

**public** **static** **void** main(String[] args) {

Demo3 d3=**new** Demo3();

d3.setter(5);

System.***out***.println(d3.getter());

}

}

**class** Demo3

{

**private** **void** test()

{

System.***out***.println("test method");

}

**public** **void** getter() {

test();

System.***out***.println("get method");

}

}

**public** **class** Encapsulation {

**public** **static** **void** main(String[] args) {

Demo3 d3=**new** Demo3();

d3.getter();

}

}

1.constructor will execute when you are creating any obj

2. problem is that our contructor is **private** so the scope is inside the **class** where it is declared.

3. our goal is to access the **private** constructor from another **class**.

4. so that creating method in the same **class** where **private** constructor is present

5. constructor will execute when you are creating any obj

6. so we are creating a test() method and **return** is **class** type it means **if** **return** type is **class** so it will **return** the obj of the **class**.

7. it means automatically consturctor is executed.

8. now calling the **static** test() method from another **class** by using the **class** name.

9. why we declared test() method as **static**?

1. our constructor is **private** so cannot create obj of the **class** and to call non-statict method we have to create an obj.

Private Constructor:

**class** Demo3

{

**private** Demo3()

{

System.***out***.println("constructor is executing");

}

**public** **static** Demo3 getter() {

System.***out***.println("get method");

**return** **new** Demo3();

}

}

**public** **class** Encapsulation {

**public** **static** **void** main(String[] args) {

Demo3.*getter*();

}

}

Output:

get method

constructor is executing

SELENIUM:

1. System.setProperty(1st, 2nd)

System.setProperty("webdriver.chrome.driver",

"C:\\Users\\sangitasadhu\\Documents\\Automation\\Practice\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

DROPDOWN:

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\sangitasadhu\\Documents\\Automation\\Practice\\chromedriver\_win32\_v2.42.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().window().maximize();

driver.get("https://www.spicejet.com/");

driver.manage().timeouts().implicitlyWait(5, TimeUnit.***SECONDS***);

WebElement element=driver.findElement(By.*id*("ctl00\_mainContent\_DropDownListCurrency"));

Select s=**new** Select(element);

List<WebElement> dropDwonLists= s.getOptions(); // number of dropdown options

**int** dropdownNumber=dropDwonLists.size();

System.***out***.println(dropdownNumber); // including select option from dropdown

//s.selectByIndex(1);

s.selectByIndex(dropdownNumber-1); //-> To get select the last option from dropdown list

//s.selectByIndex(dropdownNumber-2); -> To get select the second last option from dropdown list

s.selectByIndex(3);

s.selectByValue("AED");

s.selectByVisibleText("INR");

//various options to select values

# CheckBox:

List<WebElement> checkboxes = *driver*.findElements(By.*xpath*("//input[@type='checkbox']/following-sibling::label"));

//System.out.println(checkboxes.size());

**for**(**int** i=0;i<checkboxes.size();i++)

{

**if**(checkboxes.get(i).getText().equalsIgnoreCase("Sr. Citizen"))

{

**if**(checkboxes.get(i).isEnabled())

{

**if**(checkboxes.get(i).isSelected())

{

Sop(“elemet already selected”)

}

else

{

checkboxes.get(i).click();

**break**;

}

}

}

}

Radio Button:

List<WebElement> radio=driver.findElements(By.xpath("//label//input[@type='radio']"));  
  
for(int i=0;i<radio.size();i++)  
{  
  
if(radio.get(i).getText().equalsIgnoreCase("Radio2"))) - which radio button we are clicking  
  
{  
  
if(radio.get(i).isEnabled())  
{  
  
radio.get(i).click();  
break;  
  
}  
  
  
  
}  
  
}

WAIT:

**WAITS – to avoid Sync issues –**

**Types –**

Implicit wait

Explicit Wait

Thread.sleep

Fluent Wait

* Implicit Wait – It is defined globally.

Eg – I want my full script to wait globally for specific number of seconds before throwing exception on page load.

Syntax –

driver.manage().timeouts().implicitlywait(3,Timeunits.SECONDS);

driver – webdriverobject and 3 is seconds

Advantages –

1. It is defined globally. If I have 100 steps which involves multiple waits.

Script always wait globally on every page load before driver object is killed.

1. It listens continuously to DOM of web page.

Eg – we have defined implicit wait of 5 seconds. But, page is loaded within 2 seconds. So implicit wait objects is listening to DOM. If it loads in 2 seconds then this will not wait for 5 seconds. After 2 seconds, it will click on element.it will not wait for 5 seconds.

Disadvantages –

1. Suppose, we have to search hotel in a city and time taken for fetching hotel results is 5 secs. When I search hotels in a country it takes 15 seconds. If we specify the implicit wait of 15 seconds and assume, due to some performance issue in app it’s taking 12 seconds to load results for cities instead of 5 sec. Our script will pass because it will wait for 15 seconds but business req is of 5 seconds. So, we will not be able to test the scenario of 5 seconds.
2. Due to some technical issue in application, results are not loaded then driver will wait for 15 secs to throw exception. Longer time.

* Explicit Wait -

Is applied to specific webelement. It does not listen to the DOM like Implicit wait.

It can be achieved in Selenium in 2 ways – One with webdriver object and other as Fluent Wait.

Syntax –

WebDriverWait wait = new WebDriverWait(driver, 5);

// webdriverwait is class defined in selenium for explicit wait

wait.until(ExpectedConditions.visibilityofElementLocated(By.cssSelector(“.btn”));

//instead of visibilityOfEle… we have many methods in java to work on.

ExpectedConditions contains a list of rules for expectations implemented by default:

visibilityOfElementLocated(By locator)

visibilityOf(WebElement element)

textToBePresentInElement(By locator, String text)

titleContains(String title)

presenceOfElementLocated(By locator)

presenceOfAllElementsLocatedBy(By locator)

invisibilityOfElementLocated(By locator)

invisibilityOfElementWithText(By locator, String text)

elementToBeClickable(By locator)

stalenessOf(WebElement element)

alertIsPresent()

* Thread.Sleep – is part of java not Selenium. It will completely halt the execution for that specified time. It will not interact with DOM.

Thread.sleep(3000); - means execution is halted for 3 seconds and if page loads within 2 seconds. It will not resume execution. It will only resume after 3 seconds.

* Fluent Wait – It listens to DOM repeatedly but at regular intervals whereas explicit wait listens to DOM continuously.

There is some polling time allocated with this wait. Suppose, 10 seconds is time given and 2 sec is polling time. After every 2 seconds, fluent wait listens to DOM.

Usage –

Suppose, I have to verify confirmation messages on a payment page of a website. Messages are displaying in a particular order…

Firstly, your payment is being processed.

Secondly, your payment is successful.

Third, order is placed successfully.

I have to check if second message is displayed on webpage. But, all three messages have same attributes (i.e. same locators).

If we use explicit wait, then it will only capture first message n pass. But, in case of fluent wait, we give some polling time. After fixed polling time, it will check and if my polling time is 4 seconds and second message displays on 8th sec. I can validate that easily with fluent wait.

ACTIONS CLASS – It handles various user gestures performed on web application by mouse or keyboard.

* Context Click 🡪 right click on webelement

Actions actions = new Actions(driver);

WebElement elementLocator = driver.findElement(By.id("ID"));

actions.contextClick(elementLocator).perform();

* Double click on element

Actions actions = new Actions(driver);

WebElement elementLocator = driver.findElement(By.id("ID"));

actions.doubleClick(elementLocator).perform();

* Drag and Drop the element

Syntax –

Actions a = **new** Actions(driver);

/Element which needs to drag.

WebElement From=driver.findElement(By.xpath("//\*[@id='credit2']/a"));

//Element on which need to drop.

WebElement To=driver.findElement(By.xpath("//\*[@id='bank']/li"));

a.dragAndDrop(From,To).perform()

build() -> Ready to execute and its use when we have multiple operation.

Perform() -> exceute

* Composite Actions – Multiple Actions concatenated on a single webelemnt –

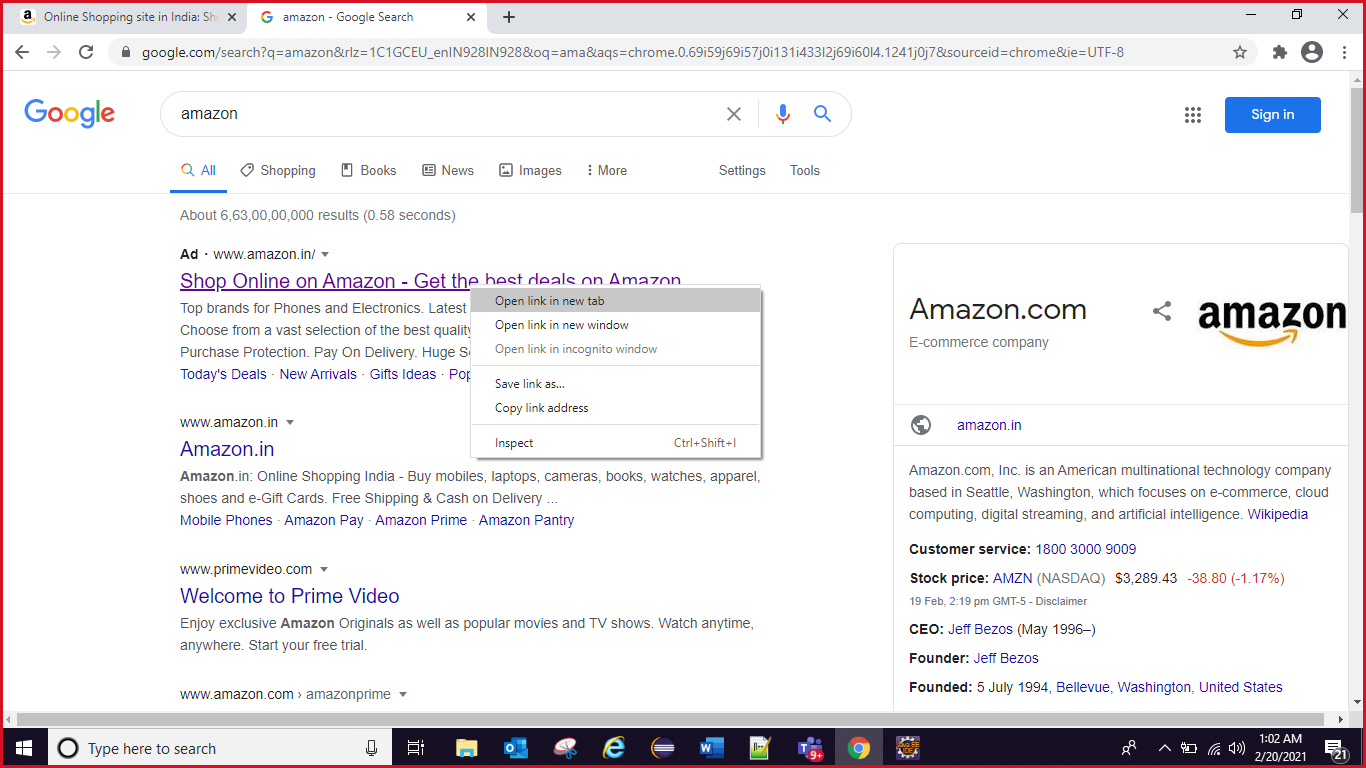
E.g. -

a.moveToElement(driver.findElemet(By.xpath()) -> xpath of text box).click().keyDown(Keys.***SHIFT***).sendKeys("hello").doubleClick().build().perform();

click on webelement then text provided in send keys should display in website in caps because of SHIFT down key then the text should be selected as double click is performed.

1. Move to element
2. Click on SHIFT from down to write in CAPS keyDown(Keys.***SHIFT)***
3. Send the text
4. Do double click to select entire text

SCENARIOS –



1.[How to right click and open the application in new window.](http://www.ufthelp.com/2014/11/close-vs-quit-in-selenium.html)  
  
Code:-

Actions oAction=new Actions(driver);  
 oAction.contextClick(//div[contains(text(),'Shop Online on Amazon - Get the best deals on Amazon')]).perform();

oAction.sendKeys("w").perform();

How to [refresh a application window](http://www.ufthelp.com/2014/11/Methods-Browser-Refresh-Selenium.html) using Action class? (Ctrl+F5)  
  
Code:-

Actions oAction=new Actions(driver);  
 oActions.keyDown(Keys.CONTROL).sendKeys(Keys.F5).perform();

1. COLOR and BACKGROUNDCOLOR:

String col=driver.findElement(xpath).getCssValue("color");

System.***out***.println(col); // rgba(60, 60, 60, 1)

String hex = Color.*fromString*(col).asHex();

System.***out***.println(hex); // #3c3c3c actual value

String bagCol= driver.findElement(xpath).getCssValue("background-color");

System.***out***.println(color);

String hex1 = Color.*fromString*(bagCol).asHex();

System.***out***.println(hex1);

7. how to crop images using selenium webdriver

Syntax: act.moveToElement(WebElement toElement, int xOffset, int yOffset).click().build().perform();

act.moveToElement(xpath of webElement, 10, 25).click().build().perform();

FRAMES – This is just hosted on webpage and prepared somewhere else. It has no html source on current page. We have to handle it separately because they are not part of webpage. The html that go inside a frame may not be from the same webpage.

<iframe> tag is identification that whether a container on a webpage is in frames or not.

We can switch to frames using index, weblement and id.

If we have to find it on the basis of id, then id has to be specifically defined by developer in html source with <iframe> tag.

driver.switchTo().frame(driver.findElement(By.*cssSelector*("iframe.demo-frame")));

driver.switchTo().defaultContent(); - to come out of frames to normal page.

driver.switchTo().parentFrame(); - immediate parent frame.

Mutiple frames –

driver.switchTo().frame(driver.findElement(By.*cssSelector*("iframe.demo-frame"))).size();

Methods Provided by Selenium for Handling iFrames

**Selenium provides the following built-in methods to switch back and forth from iframes.**

switchTo.frame(int frameNumber)

switchTo.frame(string frameName)

switchTo.frame(WebElement frameElement)

switchTo().defaultContent()

Cookies Handling oe delete Browser session –

driver.manage().deleteAllCookies();

driver.manager().deleteCookieNamed(“<cookiename>”);

* Scroll of Page:

System.setProperty("webdriver.chrome.driver", "E://Selenium//Selenium\_Jars//chromedriver.exe");

WebDriver driver = new ChromeDriver();

JavascriptExecutor js = (JavascriptExecutor) driver; -> doing type casting of driver

JavascriptExecutor -> interface

// Launch the application

driver.get("http://demo.guru99.com/test/guru99home/")

// This will scroll down the page by 1000 pixel vertical

js.executeScript("window.scrollBy(0,1000)");

* OR js.executeScript("window.scrollTo(0,document.body.scrollHeight);");
* Multiple WINDOW HANDLING –

Parent and Child window handling. If we are on a webpage and on clicking specific element new window (child window) is opened. Then, selenium will not work with child window. It will always stick to Parent window. We have to explicitly switch our control to child windows from parent window.

Webelement windowid = driver.getWindowHandle(); // It will store the address of parent window, to get windowhandle of current window.

Set <String> windowids= driver.getWindowHandles();

All windowids are fetched and stored in Set. First id stored will be parent window. Second one will be immediate child and third will be next child and so on..

We can iterate from one window to another using Iterator.

.next() will fetch first window ID

Next .next() will fetch next window ID and after that switch to child window.

Set<String> windowids= driver.getWindowHandles(); // set will store all windowids

Iterator<String> itr= windowids.iterator(); // Iterate between parent and child window

While(itr.hasNext())

{

driver.switchTo().window(itr.next());

sop(driver.getTitle());

}

//3. get the number of link present in one colum of footer

WebElement column=footerdriver.findElement(By.*xpath*("//div[@id='gf-BIG']//tr//td[1]//ul"));

System.***out***.println(column.findElements(By.*tagName*("a")).size());

//4. click on each of the link

**for**(**int** i=1;i<column.findElements(By.*tagName*("a")).size();i++)

{

String clickonlinkTab=Keys.*chord*(Keys.***CONTROL***,Keys.***ENTER***); // key work

column.findElements(By.*tagName*("a")).get(i).sendKeys(clickonlinkTab);//open the link using sendkeys

}// opens all the tabs

Set<String> set=driver.getWindowHandles();

System.***out***.println(set.size());

Iterator<String> itr=set.iterator();

// hasNext will check if at 0 index is having any window or not

**while**(itr.hasNext())

{

driver.switchTo().window(itr.next());

System.***out***.println(driver.getTitle());

}

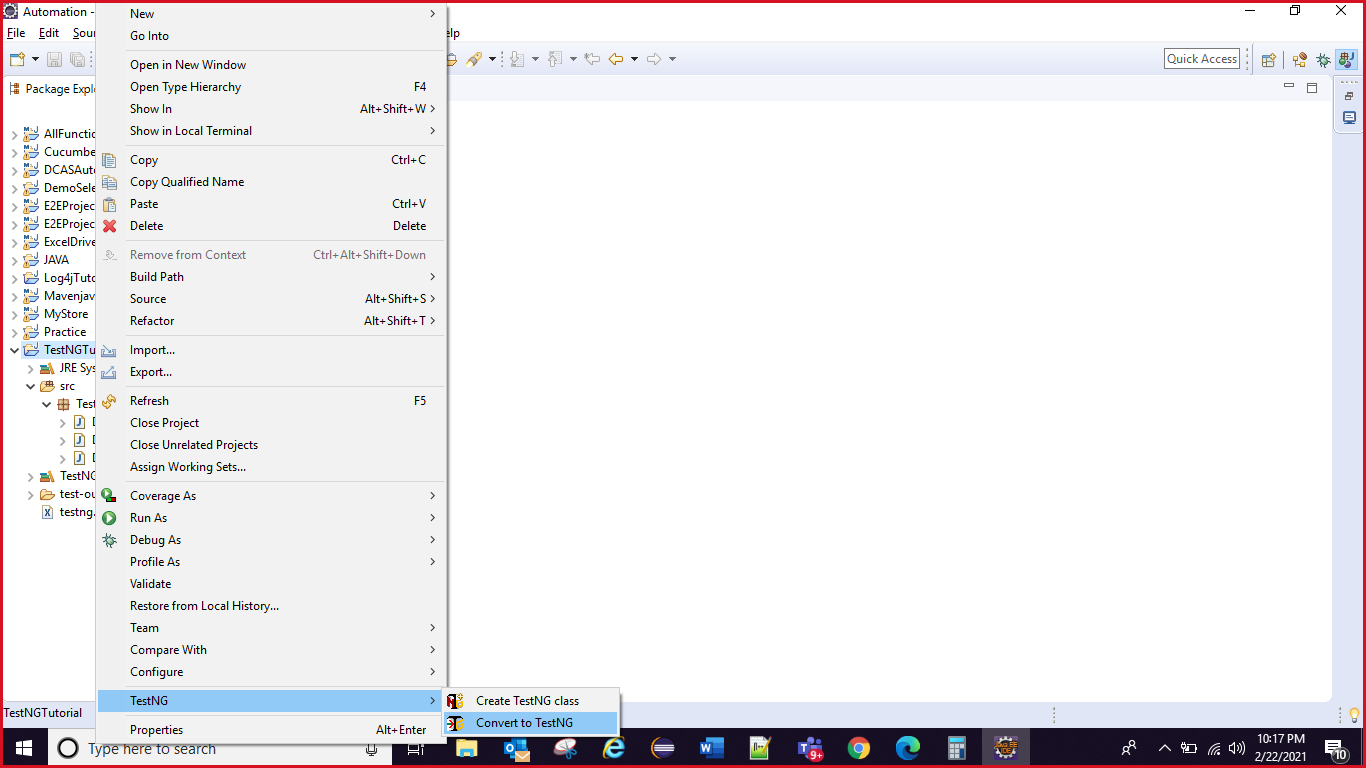
}

}

* Help to open all the link at a time:
* **public** **class** LinkOneInNewTab {
* **public** **static** **void** main(String[] args) {
* System.*setProperty*("webdriver.chrome.driver",
* "C:\\Users\\sangitasadhu\\Documents\\Automation\\Practice\\chromedriver\_win32\_v2.42.exe");
* WebDriver driver = **new** ChromeDriver();
* driver.manage().window().maximize();
* driver.manage().timeouts().implicitlyWait(5,TimeUnit.***SECONDS***);
* driver.get("http://qaclickacademy.com/practice.php");
* //1. get the number of link present in UI
* System.***out***.println(driver.findElements(By.*tagName*("a")).size());
* //2. get the number of link present in footer of UI
* WebElement footerdriver=driver.findElement(By.*id*("gf-BIG")); //limitting of driver
* System.***out***.println(footerdriver.findElements(By.*tagName*("a")).size());
* //3. get the number of link present in one colum of footer
* WebElement column=footerdriver.findElement(By.*xpath*("//div[@id='gf-BIG']//tr//td[1]//ul"));
* List<WebElement> columnLinks=column.findElements(By.*tagName*("a"));
* System.***out***.println(columnLinks.size());

* //4. click on each of the link
* **for**(**int** i=0;i<columnLinks.size();i++)
* {
* String clickonlinkTab=Keys.*chord*(Keys.***CONTROL***,Keys.***ENTER***); // It will help to open all the link
* columnLinks.get(i).sendKeys(clickonlinkTab);//open the link using sendkeys
* }// opens all the tabs
* Set<String> set=driver.getWindowHandles();
* System.***out***.println(set.size());
* Iterator<String> itr=set.iterator();
* // hasNext will check if at 0 index is having any window or not
* **while**(itr.hasNext())
* {
* driver.switchTo().window(itr.next());
* System.***out***.println(driver.getTitle());
* }
* }
* }

TESTNG:



TestNG – It is a testing framework which gives control over test cases.

* Download Testng plugIn from testNG.org website
* Go to TestNG.org
* Click on Installation
* Follow the steps –
* Go to eclipse and click on Help -> install new software -> open a window
* Paste Url mentioned in testNG.org to eclipse in the window and enter
* TestNG will be loaded , check the checkbox -> Next -> accept licence -> finish
* Go to window tab in eclipse -> prefernces -> type testNG in search. It will tell you if installed successfully.

TestNg does not require main method. It itself acts as java complier.

TestNG needs all executable code in a method and after that define TestNG anntations on top of method. Run as TestNG Test not java application this time.

Syntax –

@Test // import the annotation package to support this in current code

Public void method1()

{

//executable code

}

* In one class, we can define multiple cases in a single class file.

To Create TestNG XML ->

Go to Eclipse -> Go to project Created -> Right Click -> TestNG-> Convert to TestNg-> Next-> Finish

Structure of XML-

Test Suite – Set of test cases (can have multiple test cases)

Test

Classes

Class names

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name=*"FirstSuite"*>

<test name=*"Test"*>

<classes>

<class name=*"TestNG\_Package.BasicTestNG1"*/> // packagename.classname

<class name =*"TestNG\_Package.CLass2"*/> //classes can contain n number of test cases

</classes>

</test> <!-- Test -->

<test name=*"Test"*>

<classes>

<class name=*"TestNG\_Package.Class3"*/>

<class name =*"TestNG\_Package.Class4"*/>

</classes>

<class name =TestNG\_

</suite> <!-- FirstSuite -->

XML FORMATE:

<suite name="sute">

<test name="test">

<groups>

<run>

<include name = "" />

</run>

</groups>

<classes>

<class name ="packagename.classname" >

<methods>

<include name="aaa"/>

</methods>

</class>

</classes>

</test>

</suite>

<suite name="sute">

<test name="test">

<groups>

<run>

<include name = "" />

</run>

</groups>

<classes>

<class name ="packagename.classname" />

</classes>

</test>

</suite>

Run this XML as testNG Suite.

* Exclude methods from execution.

Exclude particular test cases while execution.

<Suite name =’abc’>

<test name =’day1’>

<classes>

<class name = ‘package1.test1>

<methods>

<exclude name =’loan1/>

</methods>

</class>

</classes>

* Similarly include can also be used to include a specific test

<Suite name =’includeTest’>

<test name =’my test’>

<classes>

<class name =’aa’/>

<methods>

<include name = ‘bookmark’/>

</methods>

</classes>

</test>

</suite>

Exclude or include test cases with Regex (Regular expression) while running.

E.g. – I have a class where all methods related to API are written with naming convention as – APILoginIn, APIRegister, APILogout.

If I want to exclude all cases related to API then I can use Regular expression.

<class name =’package1.testNG1’>

<methods>

<exclude name =’API.\*’/>

</methods>

</class>

* If I want to all run test cases from a package. E.g Regression ones.

<Suite name =’test1’>

<test name =’boo’>

<Packages>

<package name =’package name’/>

</packages>

<test>

</suite>

ANNOTATIONS in TestNG –

* If we have to excute a particular test before all other test cases then,

@BeforeTest Annotation is used in TestNG. – this can be used in case data has to be cleared before running test cases or set up connection

It will execute before the execution of all Test method present in the particular test folder at once .

Scope of this annotation is at Test Folder label

@AfterTest Annotation is used when we want something to run at end. E.g – delete cookies code, close connections

It will execute after the execution of all Test method present in the particular test folder at once .

Scope of this annotation is at Test Folder label

There scope is only limited to one test folder.

<Classes>

<class =”b”/>

<class = “r”/> // contains before test

</classes>

Before test is executed at first from a set of classes defined in test.

@BeforeSuite and @After Suite –It will execute before executing of all the tests methods present in the suite. Scope is in entire xml.

@BeforeSuite – we can store all the environment related data in test case with this annotation.

@BeforeMethod and @AfterMethods – can be used to exceute code before each and every Test method which is defined in a class.

Scope is limited to a class.

@BeforeClass and AfterClass – scope is limited to a class. @BeforeClass It will execute before executing of all the Test methods in a class.

https://stackoverflow.com/questions/30587454/difference-between-beforeclass-and-beforetest-in-testng#:~:text=As%20we%20know%20from%20official,tag%20is%20run. -Reference

**Methods are executed in a class in an alphabet order**

**Scenarios – If I have 100 test cases in a project and I want to run 10 test cases. How can we achieve that?**

**Solution – Groups tag**

**In a project, it is used to run specific number of smoke test cases when new build comes.**

**Syntax –**

In java class –

@Test(groups = {“Smoke”})

Public void method1()

{

{

**TestNG XML –**

<suite name =”’>

<test name =”Smoke Testing”>

<groups>

<run>

<include name =”Smoke”/> //same defined in class (group tag name in a class is Smoke)

</run>

</groups>

<classes>

<class name = “packagename.Class1”/>

<class name = “packagename.Class2”/>

</classes>

**In case you want to exclude Smoke Test cases and Run others –**

<groups>

<run>

<exclude name = “Smoke”/>

</run>

</groups>

**In case, we want to execute One method before other because of dependency between Methods –**

**In java class**

**@Test(dependsOnMethods = {“Method name on which it depends” , “another method name on which it depends”})**

**//a test case can depend on one or more methods.**

**This is testng helper attribute**

**public void login()**

**{**

**}**

**@Test(dependsOnMethods="login")**

**public void registrain()**

**{**

**}**

**# SKIP a Test case during execeution because of some bug which is already reported –**

**@Test(enabled = false) // enabled is testng helper attribute**

**TimeOut helper attribute – it can be provided if a test cases is taking too long to load.**

**@Test(timeOut = 4000) //miliseconds (this is 40 seconds here)**

**It will wait for 40 sec before throwing error.**

**Parameterization in TestNG – Can be achieved in 2 ways.**

* **@parameters – be defined in XML**
* **@DataProvider**

1. **Parameters** - Drive common variables from central place to our test cases. Examples – website URL will be same for all test cases. So, we will not hardcode this in our code, instead place at a centralized place so that all tests can access. Here, centralized place is **XML**.

We can declare these at either Suite level (applicable to all test cases) or at test folder level (applicable to specific test cases).

### whole application wise

<suite name=*"Suite"*>

<parameter name =*"URL"* value =*"dcas4444.com"*/>

<parameter name =*"username"* value =*"navdeep"*/>

<parameter name =*"password "* value =*"nav! "*/>

<test name=”SOMKE”>

</test>

</suite>

###Module wise:

<suite name=*"Suite"*>

<test name=”FOOD”>

<parameter name =*"URL"* value =*"dcas4444.com"*/>

<parameter name =*"username"* value =*"navdeep"*/>

<parameter name =*"password "* value =*"nav! "*/>

</test>

<test name=”CASH” >

<parameter name =*"URL"* value =*"dcas4444.com"*/>

<parameter name =*"username"* value =*"nav"*/>

<parameter name =*"password "* value =*"n! "*/>

<classes>

<class name=/>

</classe>

</test>

</suite>

In testNg class –

@Parameters( {"URL", “*username*”, “*password*”} )

@Test

**public** **void** login(String URL, String *username, String password*)

{

driver.get(*URL*);

driver.findElemet(Byy.xpath(“usertext xath”)).sendKeys(*username*);

driver.findElemet(Byy.xpath(“usertext xath”)).sendKeys(*password*);

}

This parameters are applicable to just one method, same process to be followed if we have to mention for others.

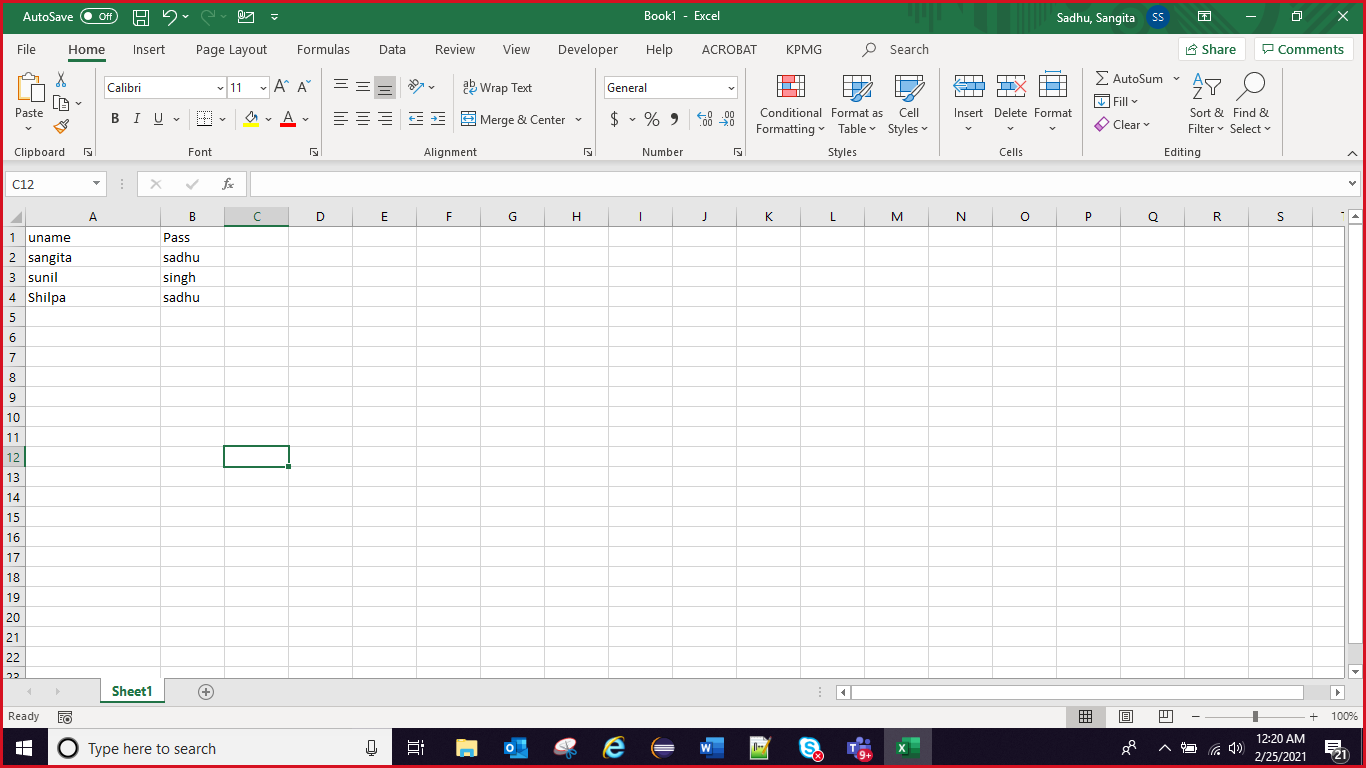
1. DATA PROVIDERS in TestNg – we are not writing anything in XML related to this.

These are used if I have to pass different set of data for a specific test case.

E.g- I have to test my application with different combinations of Username and Password to verify whether only specific type of users have access to application. Suppose we have to take screenshot at every failed step.

We cannot write this at each and every step. But, we can achieve this by testng listeners.

On test failure, script is routed to a block where screenshots code is written.



@Test(dataProvider = "getData")

**public** **void** Test1( String uname, String pwd) {

driver.findElemet(Byy.xpath(“usertext xath”)).sendKeys(uname);

driver.findElemet(Byy.xpath(“usertext xath”)).sendKeys(pwd);

}

@DataProvider

**public** Object[][] getData->Method name () {

// Declare multi-dimensional object array with rows as set of data i.e. number

// of times data has to be passed

// and column as how many parameters in one set

Object[][] data = **new** Object[3][2]; // 3(Row) times test will run with 2(columns) data values

// first set values

data[0][0] = "firstUsername";

data[0][1] = "firstpassword";

// 2nd set values

data[1][0] = "secondUsername";

data[1][1] = "secondpwd";

// 3rd set values

data[2][0] = "thirdusername";

data[2][1] = "thirdpwd";

// return this data so that it can be passed to method for execution

**return** data;

LISTENERS INTERFACE in TestNG – it listens to execution results.

Screenshot:

TakesScreenshot ts=TakesScreenshot(driver);

File source=ts.getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(source,new File(System.getProperty("user.dir")+//src//screeshot.png"))

new File(System.getProperty("user.dir")+//src//screeshot.png") - Destination folder

System.getProperty("user.dir") -> coming to project directory directly

TESTNG Priority:

**package** Test;

**import** org.testng.annotations.Test;

**public** **class** Day1

{

@Test (priority=1)

**public** **void** login()

{

System.***out***.println("Hello");

}

@Test(priority=-1)

**public** **void** YYY()

{

System.***out***.println("YYY");

}

@Test(priority=0)

**public** **void** a\_login()

{

System.***out***.println("AZero");

}

@Test(priority=-2)

**public** **void** ZZZ()

{

System.***out***.println("ZZZ");

}

@Test

**public** **void** ZAA()

{

System.***out***.println("ZAA");

}

@Test

**public** **void** YZAA()

{

System.***out***.println("YZAA");

}

}

Output:

ZZZ

YYY

YZAA

ZAA

AZero

Hello

-2 -> -1 -> without priority wala check between alphabetically -> 0 priority -> 1->2

2. **package** Test;

**import** org.testng.annotations.Test;

**public** **class** Day3 {

@Test(priority=-3)

**public** **void** b\_login()

{

System.***out***.println("Hello");

}

@Test(dependsOnMethods="b\_login")

**public** **void** a\_logout()

{

System.***out***.println("YYY");

}

@Test(priority=-2)

**public** **void** ZZZ()

{

System.***out***.println("ZZZ");

}

@Test

**public** **void** AAA()

{

System.***out***.println("AAA");

}

}

Output:

Hello

ZZZ

AAA

YYY

3.

**package** Test;

**import** org.testng.annotations.Test;

**public** **class** Day2 {

@Test

**public** **void** DefaultA()

{

System.***out***.println("Default");

}

@Test(priority=1)

**public** **void** a\_login()

{

System.***out***.println("Hello");

}

@Test(priority=0)

**public** **void** b\_login()

{

System.***out***.println("Zero");

}

}

Default

Zero

Hello

=====================================================================================

JAVA Programing:

MAP interface:

**public** **class** MapNullKey {

**public** **static** **void** main(String[] args) {

Map<String,String> map = **new** HashMap<>();

//map.put("Football", "A");

map.put("Squash", "A");

map.put("cricket", "B");

map.put("Football", "C");

map.put("Football", "M");

}

}

Output:

{cricket=B, Football=M, Squash=A}

1. In map we can store duplicate values
2. But we cannot have duplicate key
3. If any duplicate key trying insert then it will update with lasts value
4. Each key and values of map is called as Entry set
5. In map we can store value corresponding key using put method
6. We can have null key in map
7. The **get**() **method** of **Map** interface in **Java** is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the **map** contains no such **mapping** for the key.
8. NumberOfStringCount:

**import** java.util.\*;

**public** **class** NumberOfStringCount {

**public** **static** **void** main(String[] args)

{

String str="java is session session java";

String splitword[]=str.split(" ");

HashMap<String, Integer> hm=**new** HashMap<>();

**for**(String duplicate:splitword)

{

Integer j=hm.get(duplicate);

hm.put(duplicate, (j==**null**)?1:j+1);

}

//System.out.println(hm);

//System.out.println(hm.entrySet());

**for**(Map.Entry<String,Integer> map:hm.entrySet())

{

System.***out***.println(map.getKey()+" "+map.getValue());

}

}

}

Explanation:

java 2

session 2

is 1

{java=2}

java 2

String str="java is session session java";

String splitword[]=str.split(" ");

splitword[0] = "java"

splitword[1] = is

splitword[2] = session

splitword[3] = session

splitword[4] = java

HashMap<String, Integer> hm=new HashMap<>(); declaring HashMap

key as String aa variable

value as Integer

for(String aa:splitword) 1st -> java

{

Integer j=hm.get(is); null

hm.put(aa,(j==null)?1:j+1)

hm.put(key,value)

}

condition ? 1 : j+1

1. Number Of Character count:

import java.util.HashMap;

import java.util.Map;

public class NumberOfCharCount {

public static void main(String[] args) {

String str="aaabbaaac";

char ch[]=str.toCharArray();

HashMap<Character, Integer> hm=new HashMap<>();

for(Character duplicate:ch)

{

Integer j=hm.get(duplicate); //2

hm.put(duplicate, (j==null)?1:j+1);

}

// iteratating the map

for(Map.Entry<Character,Integer> map:hm.entrySet()) {

System.out.println(map.getKey()+" "+map.getValue());

}

}

}

Explanation:

String str="aaabbc";

char ch[]=str.toCharArray();

ch[0]=a

ch[1]=a

ch[2]=aaabbc

HashMap<Character,Integer> hm=new HashMap<Character,Integer>();

3. I am iterating the character array

for(Character dup(variable name): ch)

{

Integer j =hm.get(dup) 2

hm.put(key,value)

hm.put(dup,(j==null)? 1: j+1)

hm.put(a,1)

}

a -3

{a=3,b=2}

if(j==null)

{

1

}

hm =

elese

j+1

2+1 =3

1. Number of Integer count:

**public** **class** IntegerCount {

**public** **static** **void** main(String[] args) {

**int**[] arr= {1,4,1,2,2,3};

// converting integer array into String format:

String st[]=**new** String[arr.length];

**for**(**int** i=0;i<arr.length;i++)

{

st[i]=String.*valueOf*(arr[i]); //converting integer to String

}

HashMap<String, Integer> hm=**new** HashMap<String, Integer>();

**for**(String duplicate:st)

{

Integer j=hm.get(duplicate);

hm.put(duplicate, (j==**null**)?1:j+1);

}

for(Map.Entry<Character,Integer> map:hm.entrySet()) {

System.out.println(map.getKey()+" "+map.getValue());

}

}

}

1. Non Repetative character:

import java.util.HashMap;

import java.util.Map;

public class NonRepeatedValueFromString {

public static void main(String[] args) {

String str="java is session session java";

String splitword[]=str.split(" ");

HashMap<String, Integer> hm=new HashMap<>();

for(String duplicate:splitword)

{

Integer j=hm.get(duplicate);

hm.put(duplicate, (j==null)?1:j+1);

}

for(Map.Entry<String,Integer> map:hm.entrySet())

{

//System.out.println(map.getKey()+" "+map.getValue());

int count=map.getValue(); //2,2,1

if(count==1)

System.out.println("1st repeated string "+map.getKey());

}

}

}

1. First Reparative Character:

import java.util.HashMap;

import java.util.Map;

public class FirstRepeatative {

public static void main(String[] args) {

String str="aaabbc";

char[] ch=str.toCharArray();

HashMap<Character, Integer> hm=new HashMap<>();

for(Character duplicate:ch)

{

Integer j=hm.get(duplicate);

hm.put(duplicate, (j==null)?1:j+1);

}

for(Map.Entry<Character,Integer> map:hm.entrySet()) {

int count=map.getValue(); // 3,2,1

//System.out.println(count);

if(count>1)

{

System.out.println("non repeated character " +map.getKey()+"---"+count);

break; // because of break

}

}

}

}

1. First Non Repetitive character:

import java.util.HashMap;

import java.util.Map;

public class FirstNonRepeatativeChar {

public static void main(String[] args) {

String str="aabbcf";

char[] ch=str.toCharArray();

HashMap<Character, Integer> hm=new HashMap<>();

for(Character duplicate:ch)

{

Integer j=hm.get(duplicate);

hm.put(duplicate, (j==null)?1:j+1);

}

for(Map.Entry<Character,Integer> map:hm.entrySet()) {

//System.out.println(map.getKey()+" "+map.getValue());

int count=map.getValue();

System.out.println(count);

if(count==1)

{

System.out.println("non repeated character: "+map.getKey()+"----> "+count);

break;

}

}

}

}

1. Maximum and minimum value:

**public** **class** MaxMin {

**public** **static** **void** main(String args[]){

**int** arr[] = {10, 11,3,120};

**int** max=arr[0], min=arr[0];

**for**(**int** i=0;i<arr.length;i++)

{

**if**(arr[i]>max)

max=arr[i];

**if**(arr[i]<min)

min=arr[i];

}

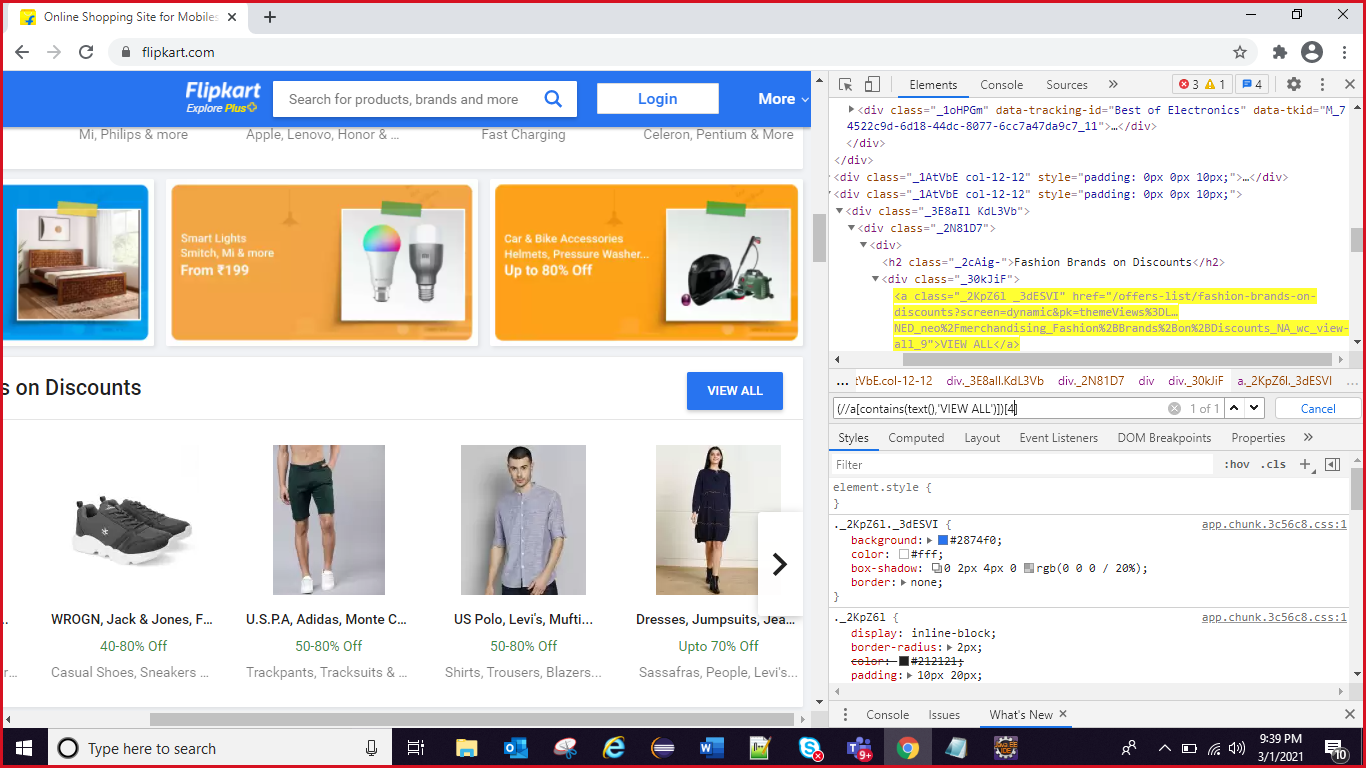
System.***out***.println(max);

System.***out***.println(min);

}

}

Xpath:



1. Palindrome String”

**public** **class** Plaindrom {

**public** **static** **void** main(String[] args) {

String str="MAM";

**for**(**int** i=0,j=str.length()-1;j>i;i++,j--)

{

**if**(str.charAt(j)!=str.charAt(i))

{

flag=1;

**break**;

}

}

**if**(flag==0)

{

System.***out***.println("palindrom");

}

**else**

System.***out***.println("non-palindrom");

}

}

Output:

Palindrom

1. Palindrome for integer number:

**public** **class** Plaindrom {

**public** **static** **void** main(String[] args) {

**int** a=454;

String str=String.*valueOf*(a);

**int** flag=0;

**for**(**int** i=0,j=str.length()-1;j>i;i++,j--)

{

**if**(str.charAt(j)!=str.charAt(i))

{

flag=1;

**break**;

}

}

**if**(flag==0)

{

System.***out***.println("palindrom");

}

**else**

System.***out***.println("non-palindrom");

}

}

Output:

Palindrom

10.

**public** **class** RemoveDuplicateChar {

**public** **static** **void** main(String[] args) {

String str="Sangita";

**char**[] ch=str.toCharArray();

LinkedHashSet<Character> set=**new** LinkedHashSet<Character>();// order of insertion will be same

**for**(**int** i=0;i<ch.length;i++)

{

set.add(ch[i]);

}

System.***out***.println(set);

**for**(Character s:set)

{

System.***out***.println(s);

}

}

}

Output:

[S, a, n, g, i, t]

S

a

n

g

i

t

11.

**public** **class** RemoveDuplicatetInteger {

**public** **static** **void** main(String[] args) {

**int**[] arr= {4,1,3,4,1,2,7,8,6,8,6};

//Stream.of(4,1,3,4,1,2,7,8,6,8,6).distinct().sorted().forEach(s->System.out.println(s));

LinkedHashSet<Integer> set=**new** LinkedHashSet<Integer>();

**for**(**int** i=0;i<arr.length;i++)

{

set.add(arr[i]);

}

System.***out***.println(set);

**for**(Integer s:set)

{

System.***out***.println(s);

}

}

}

Output:

[4, 1, 3, 2, 7, 8, 6]

4

1

3

2

7

8

6

12.

public class RemoveDuplicatFromString {

public static void main(String[] args) {

String str="Java Session Java";

String[] splitWord=str.split(" ");

HashSet<String> set=**new** HashSet<String>();

**for**(**int** i=0;i<splitWord.length;i++)

{

set.add(splitWord[i]);

}

System.***out***.println(set);

**for**(String word:set)

{

System.***out***.println(word);

}

}

}

Output:

[Java, Session]

Java

Session

13.

**public** **class** ReverseOfString {

**public** **static** **void** main(String[] args) {

String str="My name is sangita";

String[] splitString=str.split(" "); // split the string

List<String> words=Arrays.*asList*(splitString); // converting array to arrayList

Collections.*reverse*(words); //interface is Collections

//System.out.println(words);

**for**(String reverseWord:words)

{

//StringBuilder st=new StringBuilder(reverseWord);

//System.out.print(st.reverse()+" ");

System.***out***.print(reverseWord+" ");

}

}

}

//output: sangita is name My

14.

**public** **class** WordReverseInString {

**public** **static** **void** main(String[] args) {

String str="My name is sangita";

String[] splitString=str.split(" "); // split the string

List<String> words=Arrays.*asList*(splitString); // converting array to arrayList

//Collections.reverse(words); //interface is Collections

**for**(String reverseWord:words)

{

StringBuilder st=**new** StringBuilder(reverseWord);

System.***out***.print(st.reverse()+" "); // output: yM eman si atignas

//System.out.print(reverseWord+" ");

}

}

}

15.

String reverse:

**public** **class** WordReverse {

**public** **static** **void** main(String[] args) {

String reverseWord=”Sunil”;

StringBuilder st=**new** StringBuilder(reverseWord);

System.***out***.print(st.reverse());

}

}

16. String reverse and order reverse:

**public** **class** ReverseOfStringAndWords {

**public** **static** **void** main(String[] args) {

String str="My name is sangita";

String[] splitString=str.split(" ");

List<String> words=Arrays.*asList*(splitString); // converting

Collections.*reverse*(words); //interface it will revers the order

**for**(String reverseWord:words)

{

StringBuilder st=**new** StringBuilder(reverseWord);

System.***out***.print(st.reverse()+" "); }

}

}

// output: atignas si eman yM

14. Prime Number:

**public** **class** PrimeNumber {

**public** **static** **void** main(String[] args) {

**int**[] a= {1,2,3,4,5,6,7};

**int** fg=0;

**for**(**int** i=0; i<a.length; i++)

{

fg=0;

//for(int j=2;j<=(int)Math.sqrt(a[i]);j++)

**for**(**int** j=2;j<=a[i]/2;j++)

{

**if**(a[i]%j==0)

{

fg=1;

**break**;

}

}

**if**(fg==0 && a[i]!=1)

System.***out***.println(a[i]);

}

}

}

15.

# Can we execute a java program without a main method?

Yes, we can execute a java program without a main method by using a static block.

Static block in Java is a group of statements that gets executed only once when the class is loaded into the memory by Java ClassLoader,

class StaticInitializationBlock{

   static{

      System.out.println("class without a main method");

      System.exit(0);

   }

}

In the above example, we can execute a java program without a main method (works until Java 1.6 version). Java 7 and newer versions don’t allow this because JVM checks the presence of the main method before initializing the class.

class without a main method.

# Can we overload Java main method?

Yes, we can overload the main method of Java. But JVM will only call the default main method only. See the example below.

## Example

public class Tester {

   public static void main(String args[]) {

     System.out.println("Default Main");

   }

   public static void main(String args) {

      System.out.println("Overloaded Main");

   }

}

# Can We declare main() method as Non-Static in java?

The public **static void main(String ar[])** method is the entry point of the execution in Java. When we run a .class file JVM searches for the main method and executes the contents of it line by line.

You can write the main method in your program without the static modifier, the program gets compiled without compilation errors.

But, at the time of execution JVM does not consider this new method (without static) as the entry point of the program.  It searches for the main method which is public, static, with return type void, and a String array as an argument.

public static int main(String[] args){

}

If such a method is not found, a run time error is generated.

## Example

In the following Java program in the class Sample, we have a main method which is public, returns nothing (void), and accepts a String array as an argument. But, not static.

import java.util.Scanner;

public class Sample{

   public void main(String[] args){

      System.out.println("This is a sample program");

   }

}

On executing, this program generates the following error −

Error: Main method is not static in class Sample, please define the main method

# Can we declare a main method as private in Java?

Yes, we can declare the main method as private in Java.

It compiles successfully without any errors but at the runtime, it says that the main method is not public.

## Example:

class PrivateMainMethod {

   private static void main(String args[]){

       System.out.println("Welcome to Tutorials Point");

    }

}

The above code is working successfully at compile time but it will throw an error at the runtime.

## Output:

Error: Main method not found in class PrivateMainMethod, please define the main

method as:

public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

## Overriding main method

You cannot override static methods and since the public static void main() method is static we cannot override it.

String Builder – it is not thread safe. It is non-synchronized. Means that it allows multiple threads to act on a single object at same time. It is faster as compared to StringBuffer because it does not have thread safe mechanism.

String Buffer – It is thread safe which means that StringBuffer will give control to one object at a time to work on specific String variable when there are multiple objects trying to access string at same time. This avoids false results.

STRING:

Comparison in Strings – there are 2 methods to compare strings –

* .equals
* ==

.equals compares the content stored in two strings but

== compares the reference of strings.

**public** **class** ABC {

**public** **static** **void** main(String[] args)

{

String s1 =**new** String("HELLO");

String s2 =**new** String("HELLO");

System.***out***.println(s1 == s2); //false

System.***out***.println(s1.equals(s2)); //true

String a1 ="HELLO";

String a2 ="HELLO";

System.***out***.println(a1 == a2); //true

System.***out***.println(a1.equals(a2)); //true

}

}

1. Converting String to Integer:

**public** **class** StringToIntExample1{

**public** **static** **void** main(String args[]){

//Declaring String variable

String s="200";

//Converting String into int using Integer.parseInt()

**int** i=Integer.parseInt(s);

//Printing value of i

System.out.println(i);

}}

## Broken Link:

**public** **static** **void** main(String[] args) **throws** MalformedURLException, IOException {

System.*setProperty*("webdriver.chrome.driver",

"C:\\Users\\sangitasadhu\\Documents\\Automation\\Practice\\chromedriver.exe");

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(10,TimeUnit.***SECONDS***);

driver.get("https://rahulshettyacademy.com/AutomationPractice/");

List<WebElement> links=driver.findElements(By.*xpath*("//li[@class='gf-li']//a"));

SoftAssert as=**new** SoftAssert();

**for**(WebElement link:links)

{

String urlvar=link.getAttribute("href"); // getAttribute will return url(value) of href attribute

URL u\_var=**new** URL(urlvar);

HttpURLConnection conn=(HttpURLConnection) u\_var.openConnection(); // it will call url and get the response in conn obj -> conn obj will open connection

conn.setRequestMethod("HEAD"); // sending request as "HEAD"

conn.connect(); // make a connection

**int** respCode=conn.getResponseCode(); // get the response code or status

System.***out***.println(respCode);

as.assertTrue(respCode<400, "broken link" +link.getText()+ " Response code is " +respCode);

}

as.assertAll(); // it will return if as.assertTrue() got any failure after verifying every links

}

}

## HTTPConnection:

**public** **class** HTTPConnection {

**public** **static** **void** main(String[] args) {

DesiredCapabilities ch=DesiredCapabilities.*chrome*(); // to create general chrome profile

ch.setCapability(CapabilityType.***ACCEPT\_INSECURE\_CERTS***, **true**);

//ch.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);

ChromeOptions option=**new** ChromeOptions(); // L

option.merge(ch); //

System.*setProperty*("webdriver.chrome.driver",

"C:\\Users\\sangitasadhu\\Documents\\Automation\\Practice\\chromedriver.exe");

WebDriver driver = **new** ChromeDriver(option);

driver.manage().window().maximize();

driver.get(" https://sitiam04.dcas.dc.gov:4446/Curam/");

}

}

To develop connection between eclipse and database –

* Download jar named – MySqlConnectorJava.jar and configure this in project thru Build Path -> external jars. This jar develops connection between MySql and java.
* Drivermanager.getConnection(url, user, password)

This user is username and password which was entered while setting up the database.

URL here is connection url with syntax as below–

“jdbc.mysql://”+host+”:”+port+”/databasename”;

* 1. Make a connection with DB
  2. Load JDBC driver using Class.forname()
  3. execute the query using executeQuery() method
  4. fetching the records and using those records in our test method.

Class Database

{

public static void main(String[] args) throws SQLException, ClassNotFoundException

{

String host="localhost";

String port= "3306";

Username=root;

Password=root

//Connection con=DriverManager.getConnection("jdbc:mysql://" + host + ":" + port +, "root", "root");

Connection con=DriverManager.getConnection(dbURL,host,+port+,+username+,+password)

Class.forname(“com.mysql.jdbc.Driver”);

Statement s=con.createStatement();

ResultSet rs=s.executeQuery("select \* from credentials where emp=12 ");

EmpID Name dept

1 2 sunil ECE

while(rs.next())

{

int empID=rs.getInt(1); //12

String name=rs.getString(2); //sunil

String dept=rs.getString(3); // ECE

}

//After executing quesries we are using those records in our test case.

Public void empDetails(int empID,String name,String dep)

{

driver.findElement(By.xpath(".//\*[@id=’empID’]")).sendKeys(empID));

driver.findElement(By.xpath(".//\*[@id='password']")).sendKeys(name);

driver.findElement(By.xpath(".//\*[@id='password']")).sendKeys(dept)’

}

* Alternative:

while(rs.next()) // 0 index will hasve no values, always give .nex() i.e. 1st index.

{

WebDriver driver= new FirefoxDriver();

driver.get("https://login.salesforce.com");

driver.findElement(By.xpath(".//\*[@id=’empID’]")).sendKeys(rs.getInt(1));

driver.findElement(By.xpath(".//\*[@id='password']")).sendKeys(rs.getString(2));

driver.findElement(By.xpath(".//\*[@id='password']")).sendKeys(rs.getString(3));

}

}

}

Excel sheet:

**public** **class** DataDriven {

**public** ArrayList<String> getdata(String testData) **throws** IOException

{

ArrayList<String> data=**new** ArrayList<String>();

FileInputStream fis=**new** FileInputStream("C:\\Users\\sangitasadhu\\Documents\\Demodata.xlsx");

XSSFWorkbook workbook=**new** XSSFWorkbook(fis);

**int** sheetCount=workbook.getNumberOfSheets(); // number of sheets present in exel

**for**(**int** i=0;i<sheetCount;i++)

{

**if**(workbook.getSheetName(i).equalsIgnoreCase("testdata")) // checking sheet name what we want using getSheetName(i)

{

XSSFSheet sheet=workbook.getSheetAt(i); // enter that particular sheet

Iterator<Row> rows=sheet.iterator(); // sheet is collections of Rows

Row firstrow=rows.next(); //using this rows moving to the firstrow

//from this 1st row want to go to cell

Iterator<Cell> cells=firstrow.cellIterator(); // row is collection of cells/collumn

**int** k=0;

**int** collumn=0;

**while**(cells.hasNext()) // checking has next object or not

{

Cell value=cells.next(); //move to cell

**if**(value.getStringCellValue().equalsIgnoreCase("TestCases"))

{

collumn=k;

}

k++;

}

System.***out***.println(collumn); //getting the collumn's index from which column wants to fetch data 0

//after getting 1st column data and get any colunm's data

**while**(rows.hasNext()) //checking next row are present or not

{

Row r=rows.next(); // moving to next row collumn=0

**if**(r.getCell(collumn).getStringCellValue().equalsIgnoreCase(testData)) //get the 1st column and particular row's data and checking

{

Iterator<Cell> cv=r.cellIterator(); // again get all the cell's value from this row

**while**(cv.hasNext())

{

Cell c=cv.next();

**if**(c.~~getCellTypeEnum~~()==CellType.***STRING***)

data.add(c.getStringCellValue());

**else**

data.add(NumberToTextConverter.*toText*(c.getNumericCellValue())); // converting integer to string

}

}

}

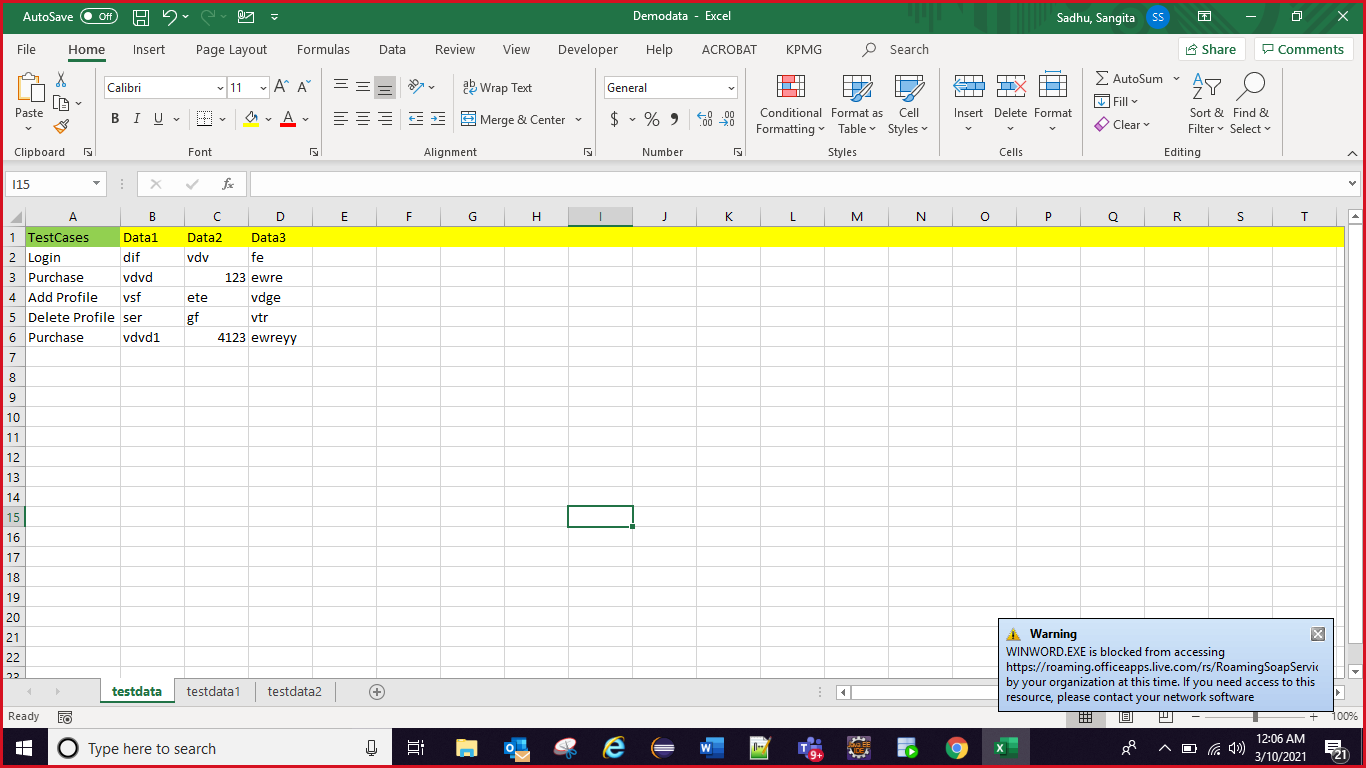
}

}

**return** data;

}

}



**public** **class** TestSample {

**public** **static** **void** main(String[] args) **throws** IOException {

DataDriven d=**new** DataDriven();

ArrayList finaldata=d.getdata("login");

System.***out***.println(finaldata);

}

}

Exception:

The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

### **Types of Java Exceptions**

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**public** **class** JavaExceptionExample {

**public** **static** **void** main(String[] args) {

**try**{

//code that may raise exception

**int** data=100/0;

}

**catch**(ArithmeticException e)

{

System.***out***.println(e);

}

//rest code of the program

System.***out***.println("rest of the code...");

}

}

## **Common Scenarios of Java Exceptions**

There are given some scenarios where unchecked exceptions may occur. They are as follows:

### **1) A scenario where ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

1. **int** a=50/0;//ArithmeticException

### **2) A scenario where NullPointerException occurs**

If we have a null value in any [variable](https://www.javatpoint.com/java-variables), performing any operation on the variable throws a NullPointerException.

1. String s=**null**;
2. System.out.println(s.length());//NullPointerException

### **3) A scenario where NumberFormatException occurs**

The wrong formatting of any value may occur NumberFormatException. Suppose I have a [string](https://www.javatpoint.com/java-string) variable that has characters, converting this variable into digit will occur NumberFormatException.

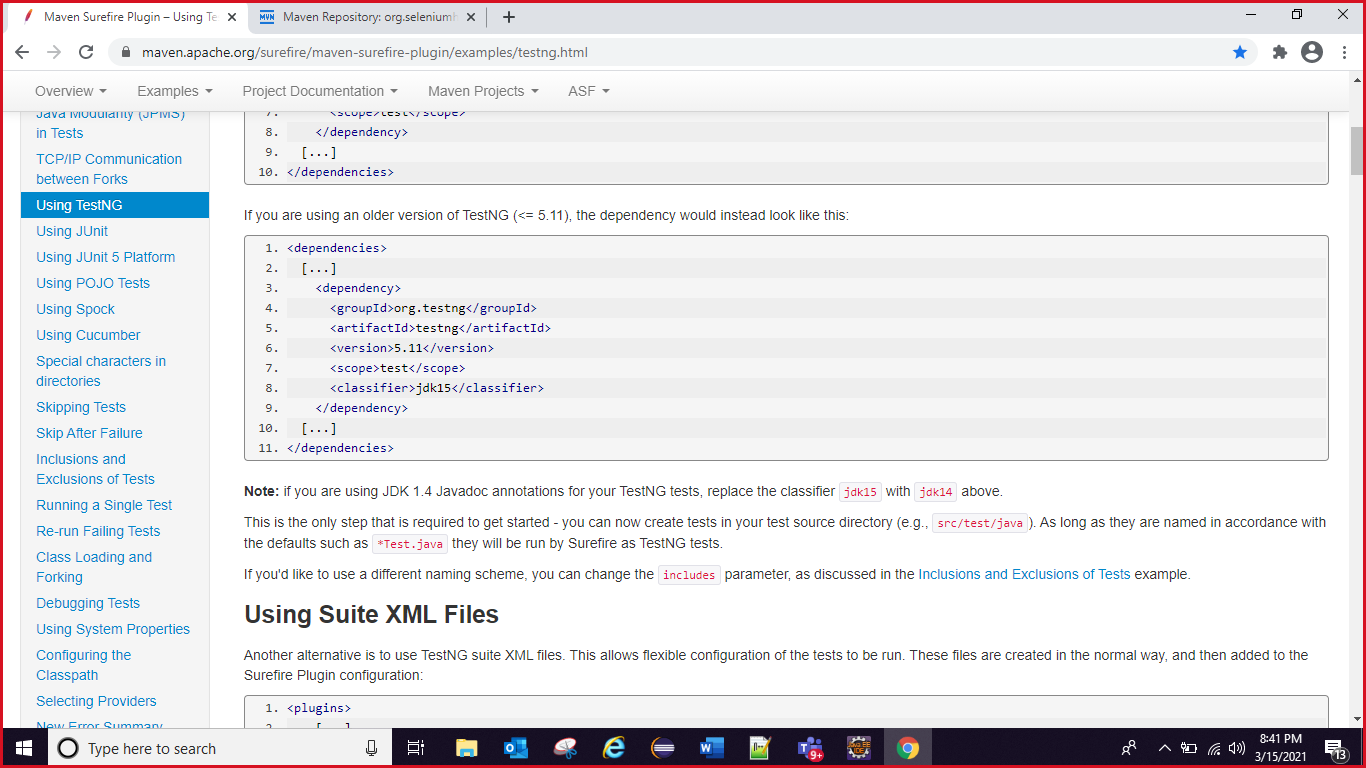
1. String s="abc";
2. **int** i=Integer.parseInt(s);//NumberFormatException

### **4) A scenario where ArrayIndexOutOfBoundsException occurs**

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

1. **int** a[]=**new** **int**[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException

Maven:



1. <plugins>
2. [...]
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-surefire-plugin</artifactId>
6. <version>3.0.0-M5</version>
7. <configuration>
8. <suiteXmlFiles>
9. <suiteXmlFile>testng.xml</suiteXmlFile>
10. </suiteXmlFiles>
11. </configuration>
12. </plugin>
13. [...]
14. </plugins>

cd means change directory:

To run test cases through maven we have execute our test cases from cmd promt.

Give

mvn test

