//creating java script executor

JavascriptExecutor js = JavascriptExecutor(Webdriver)

//Clicking a element in java script executor

js.executeScript("arugument[0].click();",element

//taking screenshot when a error occrs

WebDriver drivre= new ChromeDriver();

EventFiringWebDriver eventDriver = new EventFiringWebDriver(driver).register(new AbstractWebDriverEventLListener()){

@override

public void onException(Throwable thro,WebDriver driver){

File screen= ((TakeScreenshot)drviver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screen,new File("c:\\slenium Testing\screen.png");

});

}

//

//Selecting a particular text in browser

driver.get("/");

Webelement ele = driver.findElement(By.xpath(""));

Actions act = new Actions(driver);

act.dragAndDrop(source\_locator,target\_Locator).perform();

//writting values using javascriptExecutor in selenium

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("document.getElementById('User').value="Test");

Actions dummy = new Actions(driver);

dummy.doubleClick(ele).bulid().perform();

//Handling popup

String srcWin= driver.getWindowhandle();

Alert pop = driver.switchTo().alert();

pop.dismiss();

pop.accept();

driver.switchTo().window(srcWin);

//Invoking chrome

System.setProperty('webdriver.Chrome.driver","filepath.exe");

WebDriver driver = new ChromeDriver();

//Right click on context menu

Actions act = new Actions(driver);

act.moveToElement(ele).perform();

act.contextClick().perform();

//Mouse move over

Actions act =new Actions(driver);

act.moveToElement(ele).built().perform();

//setting prority and enable in Test NG

@Test(priority=1,enabled=false)

//Logging in Log4j

private static Logger log = Logger.getLogger(Log.Class.getName());

log.info("Implicit wait applied");

//parameter in TestNG

<suite name = "test Suite">

<test name= "fist">

<parameter name="userName" value ="Naren">

<parameter name="password" value="Tester">

<classes>

<class name="automatio.testcase">

</classes>

</test>

</suite>

//Reading data from an Excel sheet

private static XSSFSheet ExcelWorkSheet;

private static XSSFWorkbook ExcelWorkBook;

private statis XSSFCell cell;

private static XSSFRow row;

FileInputStream ExcelFile= new FileInputStream(filepath);

ExcelWorkBook = new ExcelWorkbook(ExcelFile);

ExcelWorkSheet = ExcelWorkBook.getSheet(sheetName);

int startRow = 1;

int startCol=1;

int ci,cj;

int totalRows = ExcelWoorksheet.getLastRowNum();

int totalCols = getColumnCount();

tabArray=new String[totalRows][totalCols];

ci=0;

for (int i=startRow;i<=totalRows;i++, ci++) {

cj=0;

for (int j=startCol;j<=totalCols;j++, cj++){

tabArray[ci][cj]=getCellData(i,j);

System.out.println(tabArray[ci][cj]);

}

}

}

public int getColumnCount(String sheetName)

{

sheet = workbook.getSheet(sheetName);

row = sheet.getRow(0);

int colCount = row.getLastCellNum();

return colCount;

}

//paralle execution multi browser

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

<suite name="Suite" parallel="tests">

<test name="FirefoxTest">

<parameter name="browser" value="firefox" />

<classes>

<class name="automationFramework.MultiBrowser" />

</classes>

</test>

<test name="IETest">

<parameter name="browser" value="ie" />

<classes>

<class name="automationFramework.MultiBrowser" />

</classes>

</test>

</suite>

////maven life cycle

Phase Handles Description

prepare-resources resource copying Resource copying can be customized in this phase.

validate Validating the information Validates if the project is correct and if all necessary information is available.

compile compilation Source code compilation is done in this phase.

Test Testing Tests the compiled source code suitable for testing framework.

package packaging This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.

install installation This phase installs the package in local/remote maven repository.

Deploy Deploying Copies the final package to the remote repository.

defect life cycle