How to take screenshot in Selenium WebDriver?

To capture screenshots in Selenium, one has to utilise the Interface **TakesScreenshot** and use **getScreenshotAs**() This notifies WebDriver that it should take a screenshot in Selenium and store it.

Let's explore this function with a three-step process –

Step 1 – Convert web driver object to TakeScreenshot instance

```
TakesScreenshot scrShot =(TakesScreenshot) driver;
```

Step 2 – Call getScreenshotAs method to create image file

```
File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);
```

Step 3 : Create a file to store screenshot image

File destFile = new File("//pathofmyfile");

Step 4 – Copy file to Desired Location

```
FileUtils.copyFile(SrcFile, DestFile);
```

How to Upload File in Selenium

File upload is performed when there is a need of uploading any file or a document on a specific website such as forms, registration pages, document uploaders, etc.

Uploading a file process includes browsing a file from the desired location or from your computer and uploading it to the website

```
WebElement uploadElement=driver.findElement(By.id("uploadfile_0"));

   // enter the file path onto the file-selection input field

        uploadElement.sendKeys("C:\\newhtml.html");

           // click the "UploadFile" button

     driver.findElement(By.name("send")).click()
```

Remember following two things when uploading files in WebDriver

1. There is no need to simulate the clicking of the “Choose File” button. WebDriver automatically enters the file path onto the file-selection text box of the <input type=”file”> element
2. When setting the file path in your Java IDE, use the proper escape character for the back-slash.

;

Download file in Selenium

Locate the download link:

Find the element representing the link to the file you want to download. You can use various locators, such as By.id, By.xpath, etc

```
WebElement downloadLink = driver.findElement(By.id("download-link"));
```

Simulate a click on the link:

Click the download link to initiate the file download:

```
downloadLink.click();
```

Day – 5

Introduction to Data Driven Testing using Apache POI & implementation.

Data-driven testing in Selenium using **Apache POI** involves reading test data from external Excel files, enhancing test robustness and flexibility by **separating** data from the test scripts.

Data-driven testing using Apache POI (Poor Obfuscation Implementation) in Java Selenium allows you to read test data from external sources like Excel files.

Add Apache POI dependencies:

Include the Apache POI dependencies in your project. You can add these dependencies to your Maven pom.xml

For Maven:

```
<dependency>
   <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>5.0.0</version>
</dependency>
<dependency>
   <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>5.0.0</version>
</dependency>
```

Components :

1) External Excel File.
2) Apache POI jars.
3) Test Script – java Selenium script.

Step 1 : Create an Excel file with test data:

Create an Excel file with test data. For example, let's say you have a sheet named "TestData" with columns "Username" and "Password."

Read data from Excel using Apache POI:

Create a method to read data from the Excel file using FileInputSTream and WorkbookFactory. Here's a simple example:

```
// Fetch the data from Excel

// Step 1 - FIleInputStream to access the excel file with path

FileInputStream fis = new
FileInputStream("C:\\Users\\Dell\\Documents\\RestAssuredWorkspace\\JanuarySelenium\\src\\test\\resources\\facebbokdata.xlsx");

// Step 2 - Use Apache Poi , WorkBookFactory- Creating workbookfactory instance


Workbook workbook = WorkbookFactory.create(fis);
```

Use the test data in your Selenium tests:

In your test script, use the data obtained from the Excel file:

Junit

Installation Steps –

1) Add junit dependency into your pom.xml.

Why not **main method** for Automation testing?

How many main methods can be executed at once?

1000 test cases.- 1000 different test scripts.

1 Main method can be executed at once.

a.We have 100 test scripts, since we cant execute 1000 main methods at once we will not work with main method.

b.And also we cannot write all our scripts in a single main method

What is Junit ?

Junit is a Unit Testing framework available in java used to write unit test scripts by developers.

Who uses Junit?

Developers writing Unit Test cases will make use of Junit to write and Execute Unit Test cases.

Also Test engineers make use of Junit To write Automation Test Scripts.

Introduction to TestNG annotations

TestNG (Test Next Generation) is a testing framework for Java that is widely used in Selenium automation testing. It provides powerful features for test configuration, parallel execution, grouping, and reporting.

Installation steps of TestNG:

Goto Eclipse window

Click on help option-> Eclipse Marketplace

1. Write TestNG in find edit box and click on go button.
2. Find TestNG for eclipse division and click on install button
3. Click on confirm button and I accept the terms and conditions and click on finish

To use TestNG with Selenium, you need to add the TestNG library to your project. If you're using Maven, add the following dependency to your pom.xml file:

```
<dependency>

   <groupId>org.testng</groupId>

   <artifactId>testng</artifactId>

        <version>7.4.0</version> <!-- Use the latest version available -->

   <scope>test</scope>

</dependency>
```

TestNG Annotations:

TestNG uses annotations to define the behaviour of test methods. Some commonly used annotations in Selenium automation are:

@**Test**: Marks a method as a test method.

@**BeforeMethod**: For every @test , it will open a new browser.

Executes before each test method.

@**AfterMethod**: Executes after each test method.

@**BeforeClass**: Executes once before the test class.

@**AfterClass**: Executes once after the test class.

@**BeforeTest**: This will execute all the @test methods in a single browser instance.

Executes before a test (a group of classes).

The @BeforeTest annotation is used to designate a method that should run before any test method belonging to a <test> tag in the XML suite file.

@**AfterTest**: Executes after a test (a group of classes).

@**dataprovider** – Helps us to run a single testcase with Multiple sets of data.

@**beforesuite** – Configure and write scripts at suite level.

@**aftersuite** - Configure and write scripts at suite level

New functionalities are:

->Html report

->Parallel execution - 1) Executing the separate scripts at the same time.

2) Executing the same scripts with different browsers.- Compatibility

->Grouping execution - We can group certain scripts like smoke, regression.

->Additional annotations like @dataprovider helps in Data driven testing.

->batch execution is easier

Difference between Junit & TestNG.

JUnit and TestNG are both popular testing frameworks for Java, and they share similarities in terms of functionality, but they also have some differences. Here's a comparison between JUnit and TestNG:

1.Annotation Support:

JUnit: JUnit 4 uses annotations such as @Test, @Before, @After, etc.

TestNG: TestNG provides a broader set of annotations, including @Test, @BeforeMethod, @AfterMethod, @BeforeClass, @AfterClass, @BeforeTest, @AfterTest, etc. TestNG's annotations offer more fine-grained control over test execution.

2.Parameterization:

JUnit: JUnit supports parameterized tests using the @Parameterized annotation.

TestNG: TestNG has built-in support for parameterization using the @DataProvider annotation, making it easier to run tests with different sets of data.

3.Dependencies:

JUnit: JUnit does not have built-in support for test dependencies. Developers often use @Before and @After annotations for setup and teardown.

TestNG: TestNG provides a **dependsOnMethods** attribute that allows you to specify test dependencies explicitly, ensuring the order of test execution.

4.Groups and Prioritization:

JUnit: JUnit doesn't have built-in support for test groups or prioritization. Developers often use naming conventions or custom runners for grouping.

TestNG: TestNG allows you to group tests using the **groups** attribute in annotations. It also supports test **prioritization** using the priority attribute.

5.Parallel Execution:

JUnit: JUnit does not provide native support for parallel execution. Developers often rely on external tools for parallel test execution.

Parallel Execution – Execution of Multiple test scripts at once wrt different browsers/ platform

TestNG: TestNG has built-in support for parallel test execution at various levels (methods, classes, suites), making it easy to parallelize test runs.

6.Suite Configuration:

JUnit: JUnit relies on test suite classes or test runners for suite-level configuration.

TestNG: TestNG allows you to define suite-level configurations in XML files, providing more flexibility in setting up test environments.

7.Listeners:

JUnit: JUnit has limited support for test listeners.

TestNG: TestNG provides a rich set of listeners that allow you to customise test execution and report generation.

Listeners – With the help of Listeners we can rerun some failed test scripts and also used for screenshot implementation.

8.Reporting:

JUnit: JUnit's reporting capabilities are basic, and developers often use additional tools for more detailed reports.

TestNG: TestNG generates detailed HTML reports by default, providing comprehensive information about test results.

In summary, while both JUnit and TestNG are powerful testing frameworks, TestNG often offers more features out of the box and provides additional capabilities for test configuration, parameterization, and parallel execution.

Programs :

```
package Day5;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class uploadtest {

	public static void main(String[] args) {

		WebDriver driver = new ChromeDriver();

		driver.get("https://demo.guru99.com/test/upload/");

		//driver.findElement(By.name("uploadfile_0")).click();

		// Uplaod handled by sendkeys and mention the path of file in sendkeys

	driver.findElement(By.name("uploadfile_0")).sendKeys("C:\\Users\\Dell\\Desktop\\gdt
_order.pdf");


		driver.findElement(By.id("terms")).click();
```

```
                driver.findElement(By.id("submitbutton")).click();

        }

}

package Day5;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Screenshotdemo {

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

                WebDriver driver = new ChromeDriver();

                driver.get("https://www.ebay.com/sellercenter/selling/how-to-sell");

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


                JavascriptExecutor j = (JavascriptExecutor)driver;

                Thread.sleep(2000);


                WebElement threesteps = driver.findElement(By.linkText("3 Steps to sell"));


                String link = threesteps.getAttribute("href");

                System.out.println(link);


                j.executeScript("arguments[0].scrollIntoView();", threesteps);
```

```
		// Taking screenshot after scrolling

		// Step 1 - Cast driver to TakesScreenshot interface
				TakesScreenshot ts = (TakesScreenshot)driver;


		// Step 2 - call getScreenshotAs() and take a screenshot, storing it inside a file
variable.

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


		// Step 3 - Creata a new File in Local machine

			File dest = new
File("C:\\Users\\Dell\\Documents\\screenshots\\janbatch.png");


		// Step 4 - Copy file from src to dest


			FileUtils.copyFile(src, dest);


			driver.close();


	}

}

package Day5;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Downloadtest {

	public static void main(String[] args) {

		WebDriver driver = new ChromeDriver();



	driver.get("https://demo.automationtesting.in/FileDownload.html#google_vignette");
```

```
            WebElement downlaodlink = driver.findElement(By.linkText("Download"));


            JavascriptExecutor j = (JavascriptExecutor)driver;


            j.executeScript("arguments[0].scrollIntoView();", downlaodlink);

            // SImple click() is used to download
            downlaodlink.click();


            driver.findElement(By.id("textbox")).sendKeys("ENter some data");


            //
            driver.findElement(By.id("createTxt")).click();

            // SImple click() is used to download
            driver.findElement(By.id("link-to-download")).click();





        }

}

package Day5;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class JunitDemo {
        WebDriver driver;


        @Before // Preconditions that we have to perform
        public void setup() {

             driver = new ChromeDriver(); // open the browser
```

```
	}



		driver.get("https://the-internet.herokuapp.com/login"); // passing the URL




	}



	@Test
	public void firsttest() {
		driver.findElement(By.name("username")).sendKeys("Omkar");

		driver.findElement(By.id("password")).sendKeys("omkar@2002");

		driver.findElement(By.cssSelector("i.fa-sign-in")).click();

	}

	@Test
	public void textboxtest() throws InterruptedException {


		driver.findElement(By.name("username")).sendKeys("Omkar");

		Thread.sleep(2000);


	}


	@Test
	public void passwordtest() {

		driver.findElement(By.id("password")).sendKeys("omkar@2002");


	}

	@After
	public void closeoperation() {

		driver.close();

	}
```

```
}

package Day5;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class TestNGdemo {

	WebDriver driver;


	@BeforeTest
	public void setup() {

		 driver = new ChromeDriver(); // open the browser

		driver.get("https://the-internet.herokuapp.com/login"); // passing the URL

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

	}

	@Test
	public void usernametest() throws InterruptedException {

		driver.findElement(By.name("username")).sendKeys("Omkar");

		Thread.sleep(2000);

		//driver.findElement(By.cssSelector("i.fa-sign-in")).click();


	}


	@Test(dependsOnMethods = {"usernametest"}) // this passwordtest is dependent on
usernametest, so usernametest should execute first
	public void passwordtest() throws InterruptedException {
```

```
                driver.findElement(By.id("password")).sendKeys("omkar@2002");

                driver.findElement(By.cssSelector("i.fa-sign-in")).click();
                Thread.sleep(2000);




        }


        @AfterTest
        public void teardown() {


                driver.close();
        }

}

package Day5;
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.poi.EncryptedDocumentException;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class Datadrivenfacebook {

        WebDriver driver;


        @BeforeTest
        public void setup() {

                 driver = new ChromeDriver(); // open the browser

                driver.get("https://www.facebook.com/"); // passing the URL

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

        }
```

```
	@Test

	public void logintest() throws EncryptedDocumentException, IOException,
InterruptedException {

		// Fetch the data from Excel

		// Step 1 - FIleInputStream to access the excel file with path

		FileInputStream fis = new
FileInputStream("C:\\Users\\Dell\\Documents\\RestAssuredWorkspace\\JanuarySelenium\\src\\test\\re
sources\\facebbokdata.xlsx");

		// Step 2 - Use Apache Poi , WorkBookFactory- Creating workbookfactory instance


		Workbook workbook = WorkbookFactory.create(fis);

		// accessing 'akshay' value from row 2 col 1
		String username =
workbook.getSheet("Sheet1").getRow(2).getCell(0).getStringCellValue();

		String password =
workbook.getSheet("Sheet1").getRow(2).getCell(1).getStringCellValue();


		// We are done with reading of data, now we have to just send the data into
webelements.

		driver.findElement(By.id("email")).sendKeys(username);

		driver.findElement(By.id("pass")).sendKeys(password);

		driver.findElement(By.name("login")).click();

		Thread.sleep(2000);

	}


	@AfterTest
	public void teardown() {


		driver.close();
	}

}
```