SYNTAX TECHNOLOGIES

Selenium

Class 11

Agenda

Upload File using Selenium

Taking Screenshots in Selenium

Javascript Executor

Writing an Advanced Xpath

How to find broken links/images

Upload File using Selenium Webdriver

There are different ways to handle file uploads with Selenium Webdriver.
The first and the Easy way is simple case of just finding the element and typing the absolute path of the document into it.

```
WebElement element= driver.findElement(By.name("datafile"));
element.sendKeys("C:\Users\Sandesh\Desktop\testfile.txt");
```

Screenshot using Selenium

Screenshots are desirable for bug analysis. In automation, it is necessary to take the screenshot for verification so we can prove also that our test case has covered certain functionality or not.

Test cases may fail while executing the test cases. While we are executing the test cases manually we just take a screenshot and place in a result repository. The same can be done by using Selenium WebDriver.

It's very important to take screenshot when we execute a test script. When we execute huge number of test scripts, and if some test fails, we need to check why the test has failed.

It helps us to debug and identify the problem by seeing the screen shot.

Screenshot using Selenium

Failures may occur because of below reasons:

- Actual and Expected values are not matching
- When there is no element
- When page takes more time to load
- When an unexpected alert comes in to focus
- When there is Assertion issues

There is an interface called **TakesScreenshot** which provides **getScreenshotAs** method and which selenium uses to take screenshot.

Screenshot using Selenium

Our browser classes (Like ChromeDriver, FirefoxDriver ...) extends RemoteWebdriver and RemoteWedriver implements TakesScreenshot Interface along with Webdriver Interface.

We can take screenshot of a webpage using getScreenshotAs method from the TakesScreenshot, but we cannot initialize TakesScreenshot as it is an interface.

So to take screenshot of the page we have to cast our driver object to TakesScreenshot interface type:

TakesScreenshot scrShot =((TakesScreenshot)webdriver)

Note: casting is a process of converting 1 dataType/objectType to another

Screenshot using Selenium

Convert webdriver object to TakesScreenshot

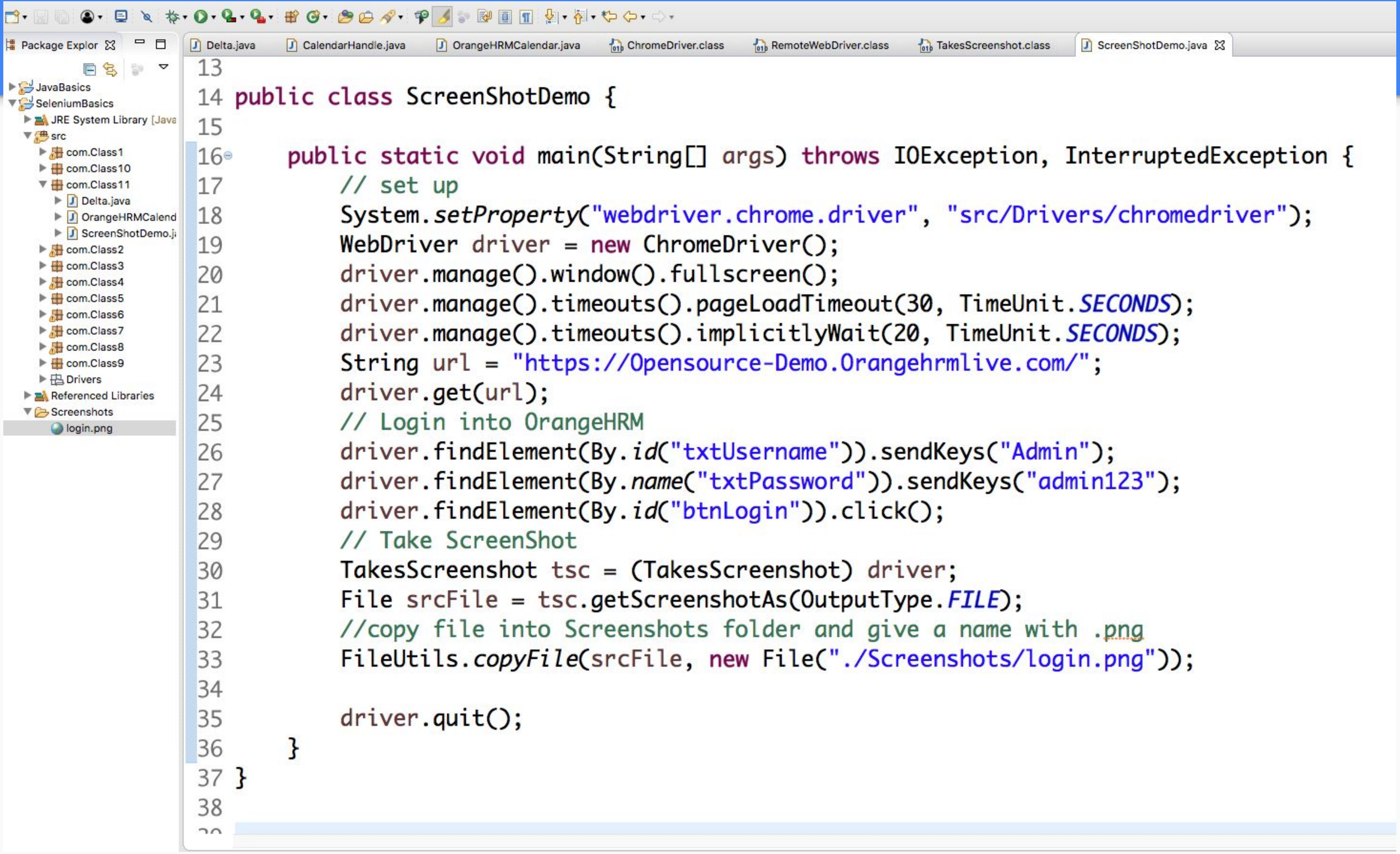
TakesScreenshot scrShot =(TakesScreenshot)webdriver;

Call getScreenshotAs method to create image file

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

Copy file at destination

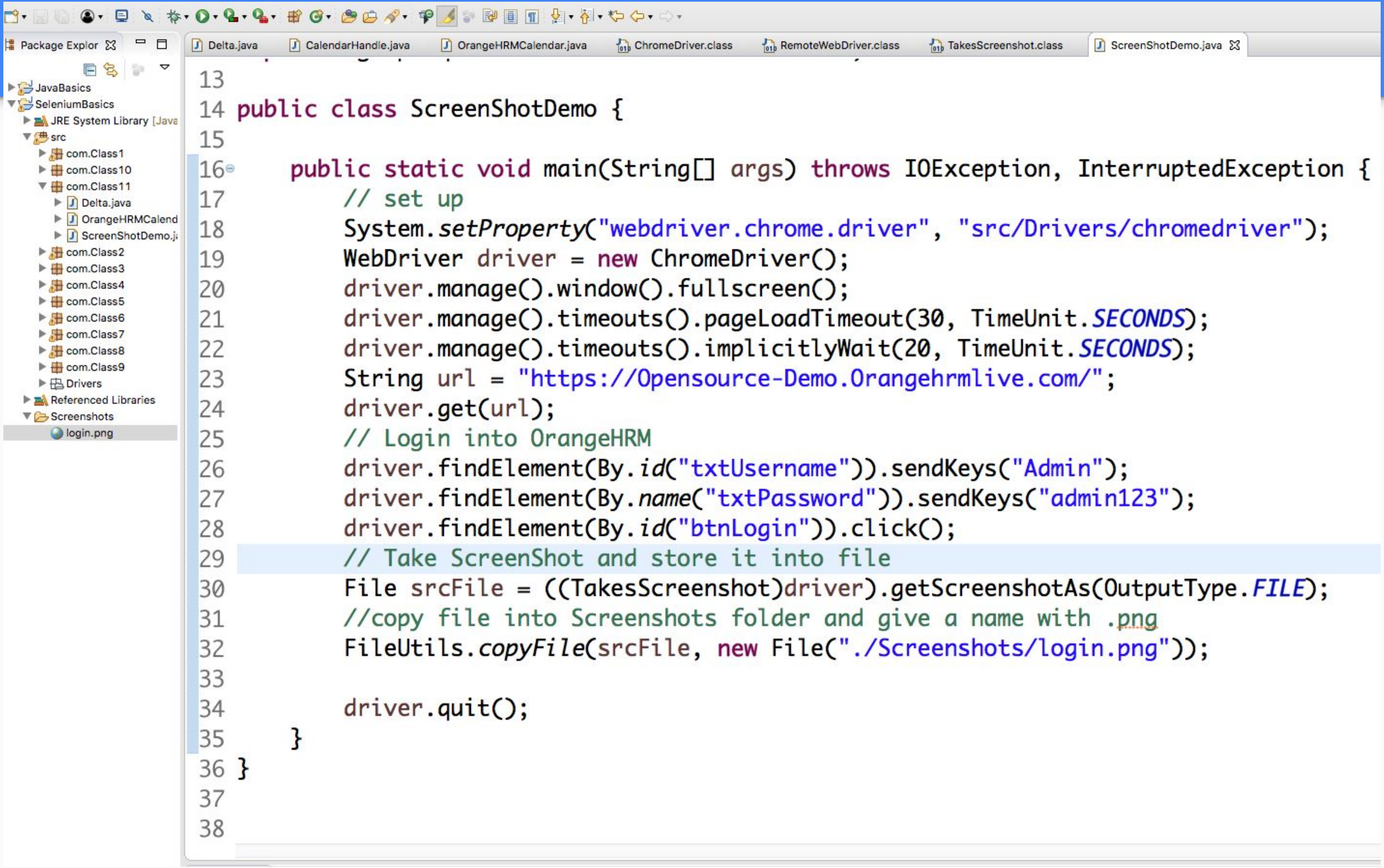
FileUtils.copyFile(SrcFile, DestFile);



```java
public class ScreenShotDemo {

    public static void main(String[] args) throws IOException, InterruptedException {
        // set up
        System.setProperty("webdriver.chrome.driver", "src/Drivers/chromedriver");
        WebDriver driver = new ChromeDriver();
        driver.manage().window().fullscreen();
        driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        String url = "https://Opensource-Demo.Orangehrmlive.com/";
        driver.get(url);
        // Login into OrangeHRM
        driver.findElement(By.id("txtUsername")).sendKeys("Admin");
        driver.findElement(By.name("txtPassword")).sendKeys("admin123");
        driver.findElement(By.id("btnLogin")).click();
        // Take ScreenShot
        TakesScreenshot tsc = (TakesScreenshot) driver;
        File srcFile = tsc.getScreenshotAs(OutputType.FILE);
        //copy file into Screenshots folder and give a name with .png
        FileUtils.copyFile(srcFile, new File("./Screenshots/login.png"));

        driver.quit();
    }
}
```



```java

public class ScreenShotDemo {

    public static void main(String[] args) throws IOException, InterruptedException {
        // set up
        System.setProperty("webdriver.chrome.driver", "src/Drivers/chromedriver");
        WebDriver driver = new ChromeDriver();
        driver.manage().window().fullscreen();
        driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        String url = "https://Opensource-Demo.Orangehrmlive.com/";
        driver.get(url);
        // Login into OrangeHRM
        driver.findElement(By.id("txtUsername")).sendKeys("Admin");
        driver.findElement(By.name("txtPassword")).sendKeys("admin123");
        driver.findElement(By.id("btnLogin")).click();
        // Take ScreenShot and store it into file
        File srcFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
        //copy file into Screenshots folder and give a name with .png
        FileUtils.copyFile(srcFile, new File("./Screenshots/login.png"));

        driver.quit();
    }
}


```

Test Case

TC 1: Upload file and Take Screenshot

1. Navigate to "http://samples.gwtproject.org/samples/Showcase/Showcase.html#!CwFileUpload"
2. Upload file
3. Verify file got successfully uploaded and take screenshot

What is Javascript?

A Javascript is a small chunks of program that makes a website interactive. For example, a javascript could create a pop-up alert box , or provide a dynamic drop down menu.

JavaScript code can listen to event on the web page like reacting to a click on a button, reacting to when check a checkbox, or when we enter any value in to the text bar.

Javascript can change the content of the html page dynamically, it can also change the attributes of a web element like change the a button from disabled state to enabled state.

JavascriptExecuto r

JavaScriptExecutor is something which present in every selenium tool and in all the languages we can execute it on the browser.

JavaScriptExecutor allows you to run pure Javascript code irrespective of the Selenium language binding(Java, C#, Python etc.) you are using.

JavaScriptExecutor is an interface which provides the mechanism to execute Javascript through selenium driver.

JavaScriptExecutor provides “**execute script**” & "**executeAsyncScript**" methods, to run JavaScript in the context of the currently selected frame or window.

We should go for JavaScriptExecutor **only** when we are not able to perform a particular task with our selenium like sometimes we may not be able click a element

JavascriptExecuto r

We have to cast driver object into JavascriptExecutor type to use the methods present in the JavascriptExecutor interface.

cast the driver object to JavascriptExecutor
JavascriptExecutor js = (JavascriptExecutor) driver;

access the methods:
js.executeScript("javascript command");

Note: casting is a process of converting 1 dataType/objectType to another

Methods present in JavascriptExecutor

Perform Scroll on application using Selenium

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("window.scrollBy(0,150)");
//Vertical scroll - down by 150 pixels
```

To scroll an app to specified elements

```
JavascriptExecutor je = (JavascriptExecutor) driver;
je.executeScript("arguments[0].scrollIntoView(true);", element);
```

Click Button using JavaScriptExecutor

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("arguments[0].click();", element);
```

Advanced xPath

Sometimes multiple elements will have the same attribute values or no attributes defined in its html and in this case we need to work on creating unique xpath.

To find an element which is not having an identity, We find the related elements of that element
and access that element using a relation between identified element

Advanced xPath

1. If Parent is having unique identification
   parentXpath/childTagName

2. If immediate child is having unique identification
   childXpath/..

3. If the next element is having unique identification
   nextElementXpath/preceding-sibling::tagName

4. If the previous element is having unique identification
   previousElementXpath/following-sibling::tagName

Test Case

TC 1: OrangeHRM Login

1. Navigate to “https://opensource-demo.orangehrmlive.com/”
2. Login to the application by writing xpath based on “parent and child relation”

TC 1: OrangeHRM Login

3. Navigate to “https://opensource-demo.orangehrmlive.com/”
4. Login to the application by writing xpath based on “following and preceding siblings”

What is a broken link?

Broken links are links or URLs that are not reachable. They may be down or not functioning due to some server error.

Due to existence of broken links, website reputation gets damaged and there will be a negative impact on the business.

It’s mandatory to find and fix all the broken links before release. If a link is not working, we face a message as 404 Page Not Found.

Find broken links using selenium

To find broken links using selenium it means we need to check the link which is pointing to wrong URL or invalid URL.

An URL will always have a status with 2xx which is valid. There are different HTTP status codes which are having different purposes. For an invalid request, HTTP status is 4xx and 5xx.

4xx class of status code is mainly for client side error, and 5xx class of status codes is mainly for the server response error.

Some of the HTTP status codes:

200 – Valid Link
404 – Link not found
400 – Bad request
401 – Unauthorized
500 – Internal Error

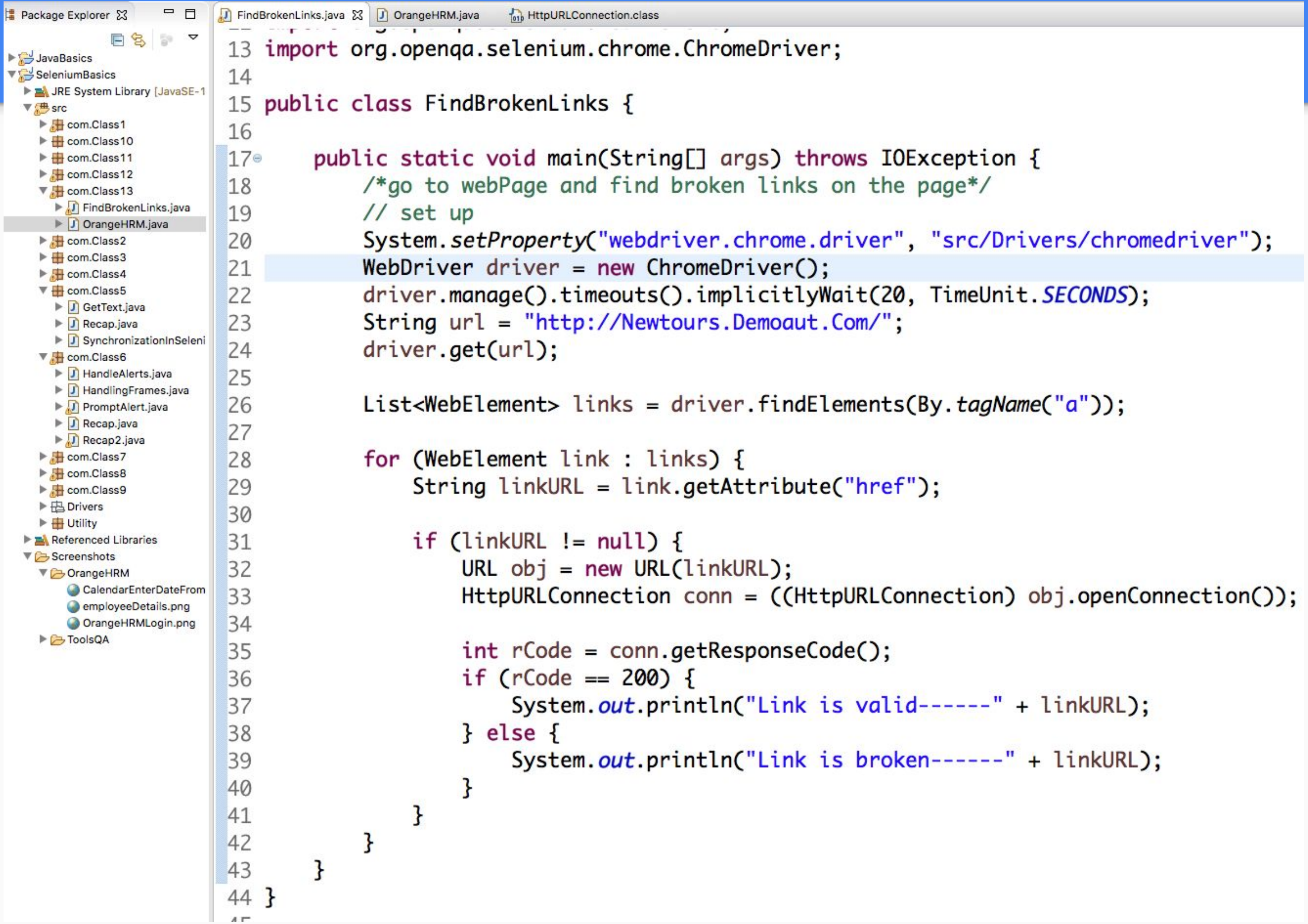
Find broken links using selenium

While doing validation you only have to verify status

200- Success- ok

Logic to find broken links:

- Collect all the links in the web page based on **<a>** tag.
- Loop/Iterate through each link and get **‘href’** attribute.
- Create object of the **URL** Class and pass the url as parameter.
- Create **HttpURLConnection** connection using URL object.
- Get and read HTTP response code.
- HTTP response code is not 200→ link is broken.
- Repeat this for all the links captured.



```java
import org.openqa.selenium.chrome.ChromeDriver;

public class FindBrokenLinks {

    public static void main(String[] args) throws IOException {
        /*go to webPage and find broken links on the page*/
        // set up
        System.setProperty("webdriver.chrome.driver", "src/Drivers/chromedriver");
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        String url = "http://Newtours.Demoaut.Com/";
        driver.get(url);

        List<WebElement> links = driver.findElements(By.tagName("a"));

        for (WebElement link : links) {
            String linkURL = link.getAttribute("href");

            if (linkURL != null) {
                URL obj = new URL(linkURL);
                HttpURLConnection conn = ((HttpURLConnection) obj.openConnection());

                int rCode = conn.getResponseCode();
                if (rCode == 200) {
                    System.out.println("Link is valid-----" + linkURL);
                } else {
                    System.out.println("Link is broken------" + linkURL);
                }
            }
        }
    }
}
```