AUTOMATE AN ECOMMERCE WEB APPLICATION

SOURCE CODE:

FlipKartDemo.java

```
package com.simplilearn.testng;
import java.util.NoSuchElementException;
import java.util.concurrent.TimeUnit;
import java.util.function.Function;
 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;
 import org.openqa.selenium.support.ui.FluentWait;
import org.openqa.selenium.support.ui.Wait;
import org.testng.annotations.AfterMethod;
 import org.testng.annotations.BeforeMethod;
 import org.testng.annotations.Test;
 public class FlipKartDemo
 {
WebDriver driver;
 @Test
public void flipkart() throws Exception
{
 driver.get("https://www.flipkart.com/search");
 driver.findElement(By.name("q")).sendKeys("iphone 13");
 driver.findElement(By.cssSelector("#container > div > div._1kfTjk >
div._1rH5Jn > div._2Xfa2_ > div._1cmsER > form > div > button")).click();
 Thread.sleep(3000);
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("window.scrollBy(0,document.body.scrollHeight)");
System.out.println("Reached At bottom of the page");
//identify image
WebElement i = driver.findElement
(By.xpath("//*[@id=\"container\"]/div/div[3]/div[1]/div[2]/div[2]/div/div/div/a/
div[1]/div[1]/ div/div/img"));
// Javascript executor to check image
 Boolean p = (Boolean) ((JavascriptExecutor)driver) .executeScript("return
arguments[0].complete " + "&& typeof arguments[0].naturalWidth !=
\"undefined\" " + "&& arguments[0].naturalWidth > 0", i);
```

```
 //verify if status is true if (p)
 {
System.out.println("Logo present");
 }
 Else
 {
System.out.println("Logo not present");
}
Thread.sleep(3000);
driver.navigate().refresh();
Wait wait = new FluentWait(driver) .withTimeout(30, TimeUnit.SECONDS)
.pollingEvery(5, TimeUnit.SECONDS)
.ignoring(NoSuchElementException.class);
WebElement clickseleniumlink = wait.until(new Function()
{
 @Test public
WebElement apply(WebDriver driver )
{
// return driver.findElement(By.cssSelector("#container > div > div._1kfTjk >
div._1rH5Jn > div._2Xfa2_ > div._3_C9Hx > div > a:nth-child(1) > img"));
WebElement element= driver.findElement(By.cssSelector("#container > div >
div._1kfTjk > div._1rH5Jn > div._2Xfa2_ > div.go_DOp._2errNR > div > div >
div > a"));
String getTextOnPage= element.getText();
 if(getTextOnPage.equals("Login"))
{
System.out.println(getTextOnPage);
 System.out.println("Passed"); return element;
}
else
{
System.out.println("Fluent Wait Fail!, Element Not Loaded Yet");
return null;
   }
 }
}
);
}
@BeforeMethod
 public void beforeMethod()
{
 System.setProperty("webdriver.chrome.driver",
C:\\Users\\kartheek\\Documents\\chromedriver.exe"); driver= new
ChromeDriver();
 driver.manage().window().maximize();
 }
@AfterMethod
public void afterMethod()
{
//driver.close();
//driver.navigate().refresh();
driver=null;
}
}
```

1. Import statements:
   The code starts with importing necessary packages and classes from Selenium WebDriver and TestNG.

2. Class and WebDriver initialization:
   - The class `FlipKartDemo` is defined to hold the test methods.
   - The WebDriver instance (`driver`) is declared at the class level.

3. `@BeforeMethod`:
   - The `@BeforeMethod` annotation denotes a setup method that will be executed before each test method.
   - In the `beforeMethod()`, the Chrome WebDriver is set up by providing the path to the ChromeDriver executable and initializing the WebDriver instance as `new ChromeDriver()`.
   - The window is maximized using `driver.manage().window().maximize()`.

4. `@Test` method `flipkart()`:
   - The `@Test` annotation marks the `flipkart()` method as a test method that will be executed when running the TestNG suite.
   - The method starts by navigating to the Flipkart search page with `driver.get("https://www.flipkart.com/search")`.
   - It then finds the search input field by its name and enters "iphone 13" using `driver.findElement(By.name("q")).sendKeys("iphone 13")`.
   - The search button is clicked using `driver.findElement(By.cssSelector("#container > div > ... > button")).click()`.
   - A 3-second pause is introduced with `Thread.sleep(3000)` (not a recommended practice, used for demonstration purposes).

- A JavaScriptExecutor object `js` is created to perform scrolling down to the bottom of the page using `js.executeScript("window.scrollBy(0,document.body.scrollHeight)")`.
- The status of the logo image at the bottom of the page is checked using JavaScriptExecutor, and the result is printed.

5. `Thread.sleep(3000)`:
   - Another pause is introduced for 3 seconds before refreshing the page.

6. Page refresh:
   - The page is refreshed using `driver.navigate().refresh()`.

7. FluentWait setup:
   - A FluentWait object `wait` is created, specifying a timeout of 30 seconds and polling every 5 seconds, ignoring `NoSuchElementException`.

8. `WebElement clickseleniumlink = wait.until(...)`:
   - A WebElement is declared and initialized using FluentWait.
   - The `wait.until(...)` method is used to wait for an element to be present on the page. The condition for waiting is defined in the provided `Function`.

9. Function `apply()`:
   - The `apply()` method of the `Function` interface is implemented to define the condition to wait for an element.
   - The target element is located using a CSS selector, and its text is retrieved using `element.getText()`.
   - If the element's text is "Login," it prints "Passed"; otherwise, it prints "Fluent Wait Fail!, Element Not Loaded Yet."

10. `@AfterMethod`:
    - The `@AfterMethod` annotation denotes a cleanup method that will be executed after each test method.
    - The WebDriver is closed and set to null to release resources and ensure a fresh instance is used for the next test.

Overall, this code performs the following steps:
- Opens the Flipkart search page, enters "iphone 13" in the search box, and clicks the search button.
- Scrolls to the bottom of the page and checks the presence of the logo image.
- Refreshes the page.
- Uses FluentWait to wait for the "Login" element to appear and prints the result.

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.simplilearn.testng</groupId>
 <artifactId>FlipKart</artifactId>
<version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>
 <name>FlipKart</name>
<url>http://maven.apache.org</url>
<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
 <dependencies>
 <dependency>
 <groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
 <version>7.5</version>
<scope>test</scope>
 </dependency>
<dependency>
 <groupId>org.seleniumhq.selenium</groupId>
 <artifactId>selenium-java</artifactId>
 <version>3.5.1</version>
 </dependency>
<!--ReportNg 1.1.4-->
<dependency>
<groupId>org.uncommons</groupId>
 <artifactId>reportng</artifactId>
<version>1.1.4</version>
<scope>test</scope>
 </dependency> <!--Velocity-dep 1.4--> <dependency>
<groupId>velocity</groupId>
```

```
<artifactId>velocity-dep</artifactId>
<version>1.4</version>
</dependency>
<!--Guice 3.0-->
 <dependency>
<groupId>com.google.inject</groupId>
 <artifactId>guice</artifactId>
<version>3.0</version>
</dependency>
</dependencies>
</project>
```

Make sure the `chromedriver.exe` path is correct in the `beforeMethod()` of `FlipKartDemo.java`.
Instead of using the `Thread.sleep()` method, consider using explicit waits for better test reliability. For example, you can use `WebDriverWait` to wait for elements to be clickable or visible.
Remove the nested `@Test` annotation inside the `apply()` method in `FlipKartDemo.java` as nested test methods are not supported by TestNG and may cause unexpected behavior.
It's a good practice to organize the test methods and set up/teardown methods using appropriate TestNG annotations like `@BeforeMethod` and `@AfterMethod` instead of declaring them within the test method.
Check if all the dependencies mentioned in the POM.xml are required for your project. For example, you can remove `velocity-dep` and `guice` dependencies if they are not used in your test automation.
Consider adding TestNG HTML reports or Allure report for better test reporting and visualization.
Make sure the Selenium WebDriver version mentioned in the POM.xml is compatible with the version of the browser you intend to use for automation.
Before running the test, ensure that the correct version of ChromeDriver is present at the specified path, and it is compatible with the installed Chrome browser.