**Documentation**

**1. Project Setup**

Environment Setup:

1. *Install Eclipse IDE and JDK***:**
   * Download the latest version of Eclipse IDE from the official website.
   * Download and install the Java Development Kit (JDK) from the [Oracle website](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html).
   * After installation, configure the JDK in Eclipse by navigating to Window > Preferences > Java > Installed JREs, and add the path to your JDK installation.
2. *Set Up Selenium WebDriver:*
   * Download the Selenium WebDriver JAR files from the Selenium HQ website.
   * Set up the WebDriver in Eclipse by creating a new Java project and adding the Selenium JAR files to the project’s build path.
   * Additionally, download the appropriate WebDriver binaries for Chrome and Firefox and set them up by adding them to your system’s PATH variable or by specifying the path in your test scripts.

Browser Drivers:

* ChromeDriver
* GeckoDriver (for Firefox)

**2. Usage Instructions**

Running Tests:

1. *Basic Navigation and Interaction:*
   * The scripts demonstrate how to interact with web elements using identifiers like id, name, linkText, className, and xpath. To run these scripts:
     + Open the Java class file containing your test script in Eclipse.
     + Run the file as a Java application or as a TestNG/JUnit test if annotations are used.
2. *Advanced Element Handling:*
   * This script involves interacting with various types of web elements such as text fields, buttons, checkboxes, and dropdowns. Ensure the web page (e.g., DemoQA Forms page) is accessible.
   * Execute the script to see the automation of form filling, clicking buttons, and other user actions.
3. *Synchronization Techniques:*
   * The scripts use implicit and explicit waits to handle page loading issues. Ensure these waits are correctly set up to avoid NoSuchElementException errors during execution.
4. *Page Object Model (POM):*
   * Scripts refactored with POM should be executed similarly but will involve interactions abstracted into Page Object classes. These scripts are more maintainable and easier to extend.
5. *Keyboard and Mouse Actions:*
   * This script uses the Actions class to perform keyboard operations like entering text and handling search suggestions in Google Search.
   * Run this script to observe simulated user interactions with the keyboard and mouse.

**3.** **Script Explanations**

1. ***Opening the Google Chrome browser***

### Objective:

### The objective of the provided Java code is to automate the process of opening the Google Chrome browser and navigating to the YouTube website using Selenium WebDriver.

**Script:**

package Part1;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class ThirdChrome {

public static void main(String[] args) {

System.out.println("Opening Google!!!");

System.setProperty("webdriver.gecko.driver","C:\\Users\\admin\\Downloads\\geckodriver-v0.34.0-win64\\geckodriver.exe");

WebDriver driver = new ChromeDriver() ;

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

}

}

**Explanation:**

1. Package Declaration:
   * The code is part of a package named Part1. In Java, packages are used to group related classes together, making the code more organized and manageable.
2. Import Statement:
   * These import statements bring in the necessary Selenium classes:
     + WebDriver: This is the main interface that defines the methods used to interact with a web browser.
     + ChromeDriver: This class implements the WebDriver interface for Google Chrome, enabling control of the Chrome browser through Selenium.
3. Main Class and Method:
   * The class ThirdChrome contains the main method, which is the entry point of the program.
   * Inside the main method, a message "Opening Google!!!" is printed to the console using System.out.println().
4. Setting the WebDriver System Property:
   * The System.setProperty() method is used to set the system property for the geckodriver, which is the WebDriver for Firefox (not Chrome). This line mistakenly sets the path for Firefox's geckodriver, but the code later uses ChromeDriver instead of FirefoxDriver.
   * This mismatch is an issue and can be fixed by either:
     + Using the correct WebDriver (geckodriver) if intending to use Firefox.
     + Setting the path for chromedriver if intending to use Chrome.
5. Instantiating the WebDriver:
   * A new instance of ChromeDriver is created and assigned to the WebDriver reference driver. This action opens a new Google Chrome browser window.
6. Navigating to a URL:
   * The get() method of the WebDriver interface is called to navigate to the specified URL, "<https://www.youtube.com>". This will load the YouTube homepage in the opened Chrome browser.
7. ***Navigation through various links***

**Objective:**

The purpose of the above Java code is to use the Firefox browser to automatically go through the different links on the "https://example.com/" website. In order to mimic user engagement and account for possible loading times, the script simulates a sequence of clicks on various links and adds delays between activities.

**Script:**

package Part1;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Fourth {

public static void main(String args[]) {

try {

WebDriver driver =new FirefoxDriver();

System.*setProperty*("webdriver.gecko.driver", "C:\\Users\\samar\\Downloads\\geckodriver-v0.34.0-win64\\geckodriver.exe");

driver.get("https://example.com/");

WebElement link = driver.findElement(By.*linkText*("More information..."));

link.click();

link = driver.findElement(By.*linkText*("Domains"));

link.click();

link = driver.findElement(By.*linkText*(".ARPA Registry"));

link.click();

link = driver.findElement(By.*linkText*("Protocols"));

link.click();

link = driver.findElement(By.*linkText*("Performance"));

link.click();

Thread.*sleep*(1000);

link = driver.findElement(By.*linkText*("Numbers"));

link.click();

Thread.*sleep*(1000);

link = driver.findElement(By.*linkText*("IPv6 (AFRINIC)"));

link.click();

Thread.*sleep*(1000);

link = driver.findElement(By.*linkText*("About"));

link.click();

Thread.*sleep*(2000);

link = driver.findElement(By.*linkText*("Performance Reporting"));

link.click();

Thread.*sleep*(2000);

}

catch(Exception e) {

e.printStackTrace();

}

}

}

**Explanation:**

1. Package Declaration:
   * This declares that the class Fourth is part of the Part1 package, which helps in organizing the project files.
2. Import Statements:
   * These imports bring in necessary classes from the Selenium library:
     + By: This class provides mechanisms to locate elements within a webpage.
     + WebDriver: The main interface to control the browser.
     + WebElement: Represents an element on the web page.
     + FirefoxDriver: An implementation of WebDriver for Firefox, used to automate Firefox browser.
3. Main Class and Method:
   * The class Fourth contains the main method, which serves as the entry point for the program.
4. Setting Up WebDriver:
   * The FirefoxDriver object driver is instantiated to control the Firefox browser.
   * The System.setProperty() method sets the path to the geckodriver executable, which is necessary for Selenium to interact with Firefox. Note that this line should ideally come before the instantiation of FirefoxDriver.
5. Navigating to a URL:
   * The get() method is used to navigate to the URL "<https://example.com/>", which loads the Example Domain website.
6. Interacting with Web Elements:
   * The script interacts with various links on the page using the By.linkText() method, which locates elements by the visible text of the link.
   * The same pattern is repeated to interact with other links like "Domains", ".ARPA Registry", "Protocols", etc.
7. Introducing Delays:
   * Thread.sleep(1000); pauses the execution for 1 second (1000 milliseconds) before proceeding to the next action. This is used to ensure that the page has sufficient time to load before the script interacts with the next element.
8. Error Handling:
   * The entire code is wrapped in a try-catch block to handle any exceptions that might occur during execution. If an error occurs, the stack trace is printed for debugging purposes.

1. ***Practice form on the DemoQA website***

**Objective:**

Using a variety of Selenium WebDriver instructions, this Java Selenium script aims to automate the process of completing a practice form on the DemoQA website ("https://demoqa.com/automation-practice-form"). Form inputs, radio buttons, checkboxes, date pickers, file uploads, dropdown menus, and JavaScript execution are all demonstrated.

**Script:**

package Part2;

import java.time.Duration;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.Select;

import org.openqa.selenium.support.ui.WebDriverWait;

public class First {

public static void main(String[] args) {

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

System.*setProperty*("webdriver.gecko.driver","C:\\Users\\admin\\Downloads\\geckodriver-v0.34.0-win64\\geckodriver.exe");

WebDriver driver = new ChromeDriver();

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

try {

driver.get("https://demoqa.com/automation-practice-form");

driver.findElement(By.*id*("firstName")).sendKeys("Nitish");

driver.findElement(By.*id*("lastName")).sendKeys("Mahant");

driver.findElement(By.*id*("userEmail")).sendKeys("nitishmahant@gmail.com");

WebElement radio = driver.findElement(By.*cssSelector*("Label[for='gender-radio-1']"));

((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", radio);

radio.click();

driver.findElement(By.*id*("userNumber")).sendKeys("9876578743");

WebElement dateElement = driver.findElement(By.*id*("dateOfBirthInput"));

dateElement.click();

WebElement yearDropdown = driver.findElement(By.*className*("react-datepicker\_\_year-select"));

Select selectYear = new Select(yearDropdown);

selectYear.selectByVisibleText("2002");

WebElement monthDropdown = driver.findElement(By.*className*("react-datepicker\_\_month-select"));

Select selectMonth = new Select(monthDropdown);

selectMonth.selectByVisibleText("August");

WebElement day = driver.findElement(By.*className*("react-datepicker\_\_day--002"));

day.click();

WebElement subjectInput = driver.findElement(By.*cssSelector*(".subjects-auto-complete\_\_input input"));

((JavascriptExecutor) driver).executeScript("arguments[0].click();", subjectInput);

subjectInput.sendKeys("Physics");

Thread.*sleep*(1000);

subjectInput.sendKeys(Keys.*TAB*);

subjectInput.sendKeys("English");

Thread.*sleep*(1000);

subjectInput.sendKeys(Keys.*TAB*);

subjectInput.sendKeys("Computer Science");

Thread.*sleep*(1000);

subjectInput.sendKeys(Keys.*TAB*);

WebElement checkbox1 = driver.findElement(By.*cssSelector*("label[for='hobbies-checkbox-1']"));

((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", checkbox1);

checkbox1.click();

WebElement checkbox2 = driver.findElement(By.*cssSelector*("label[for='hobbies-checkbox-2']"));

((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", checkbox2);

checkbox2.click();

driver.findElement(By.*id*("uploadPicture")).sendKeys("C:\\Users\\admin\\Downloads\\Passport\_photo.jpg");

driver.findElement(By.*id*("currentAddress")).sendKeys("Gurdaspur");

WebElement ele=driver.findElement(By.*id*("react-select-3-input"));

ele.sendKeys("Haryana");

WebDriverWait wait = new WebDriverWait(driver,Duration.*ofSeconds*(10));

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*cssSelector*("div[id^='react-select-3-option']")));

WebElement suggestions=driver.findElement(By.*xpath*("//div[@id='react-select-3-option-2']"));

((JavascriptExecutor)driver).executeScript("arguments[0].scrollIntoView();",suggestions);

((JavascriptExecutor)driver).executeScript("arguments[0].click();",suggestions);

WebElement city = driver.findElement(By.*id*("react-select-4-input"));

city.sendKeys("Panipat");

WebDriverWait wait2 = new WebDriverWait(driver, Duration.*ofSeconds*(5));

wait2.until(ExpectedConditions.*visibilityOfElementLocated*(By.*cssSelector*("div[id^='react-select-4-option'")));

WebElement suggestions2 = driver.findElement(By.*xpath*("//div[@id='react-select-4-option-1']"));

((JavascriptExecutor)driver).executeScript("arguments[0].scrollIntoView();",suggestions2);

((JavascriptExecutor) driver).executeScript("arguments[0].click();",suggestions2);

WebElement button = driver.findElement(By.*id*("submit"));

((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", button);

button.click();

}

catch (Exception e) {

e.printStackTrace();

}

}

}

**Explanation:**

1. Package Declaration:
   * This declares that the class First is part of the Part2 package, helping in organizing the project files.
2. Import Statements:
   * These imports bring in the necessary classes from Selenium and Java for web automation, including classes for handling elements (By, WebElement), browser interaction (WebDriver), and wait operations (WebDriverWait, ExpectedConditions).
3. Main Class and Method:
   * The First class contains the main method, which serves as the entry point for the program.
4. Setting Up WebDriver:
   * The System.setProperty() method sets the system property for the geckodriver, which is incorrect here since ChromeDriver is used. The path should instead be set for chromedriver.
   * A new instance of ChromeDriver is created and maximized using driver.manage().window().maximize().
5. Navigating to the URL:
   * The get() method is used to navigate to the DemoQA practice form URL.
6. Filling Out the Form:
   * Entering Text Fields:
     + Text fields for first name, last name, and email are filled using the sendKeys() method.
   * Selecting Radio Button:
     + The radio button for gender is selected using a CSS selector. The JavascriptExecutor scrolls the element into view before clicking it.
   * Entering Mobile Number:
     + The mobile number field is filled with the user's phone number.
   * Selecting Date of Birth:
     + The date of birth is selected using dropdowns for year and month, and clicking on the day.
   * Entering Subjects:
     + The subjects are entered by interacting with the input field and selecting options using the keyboard (Keys.TAB).
   * Selecting Checkboxes:
     + The script clicks on the checkboxes for hobbies, scrolling them into view first.
   * Uploading a File:
     + A file is uploaded by sending the file path to the file input element.
   * Entering Address:
     + The current address is entered into the respective field.
   * Selecting State and City:
     + The state and city are selected using auto-complete dropdowns, with WebDriverWait ensuring that the options are visible before clicking.
   * Submitting the Form:
     + The form is submitted by scrolling to and clicking the submit button.
7. Error Handling:
   * Any exceptions that occur during the execution are caught and printed to the console for debugging.
8. ***Extract all links and other web elements from the Wikipedia Main Page***

**Objective:**

This Java Selenium script's goal is to obtain all of the links and images that are available on the Wikipedia homepage ("https://www.wikipedia.org/"). Additionally, it illustrates how to manage synchronization in Selenium using both implicit and explicit waits.

**Script:**

package Part2;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

import java.util.List;

import java.util.concurrent.TimeUnit;

public class Second {

public static void main(String[] args) {

System.setProperty("webdriver.gecko.driver","C:\\Users\\admin\\Downloads\\geckodriver-v0.34.0-win64\\geckodriver.exe");

WebDriver driver = new ChromeDriver();

try {

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

driver.get("https://www.wikipedia.org/");

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(20));

WebElement centralContent = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("www-wikipedia-org")));

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

System.out.println("Total number of links on the page: " + links.size());

for (WebElement link : links) {

System.out.println(link.getText() + " - " + link.getAttribute("href"));

}

List<WebElement> images = driver.findElements(By.tagName("img"));

System.out.println("Total number of images on the page: " + images.size());

for (WebElement img : images) {

System.out.println("Image source: " + img.getAttribute("src"));

}

} finally {

driver.quit();

}

}

}

**Explanation:**

1. Package Declaration:
   * Declares that the Second class is part of the Part2 package.
2. Import Statements:
   * These imports bring in necessary classes from Selenium for web automation, including classes for handling elements (By, WebElement), browser interaction (WebDriver), and synchronization (WebDriverWait, ExpectedConditions, TimeUnit).
3. Main Class and Method:
   * The Second class contains the main method, serving as the entry point for the program.
4. Setting Up WebDriver:
   * The System.setProperty() method sets the system property for the geckodriver (again, incorrectly set here as geckodriver instead of chromedriver).
   * A new instance of ChromeDriver is created.
5. Implicit Wait:
   * An implicit wait of 10 seconds is set. This makes the WebDriver wait for up to 10 seconds before throwing a NoSuchElementException if an element is not found.
6. Navigating to Wikipedia:
   * The get() method is used to navigate to the Wikipedia homepage.
7. Explicit Wait:
   * An explicit wait of 20 seconds is applied here. The WebDriver waits until the central content of the Wikipedia homepage (identified by the id="www-wikipedia-org") becomes visible.
8. Extracting and Printing Links:
   * All <a> tags (links) on the page are found using findElements(By.tagName("a")).
   * The total number of links is printed to the console.
   * A loop iterates over each link, printing the link text and the URL (href attribute).
9. Extracting and Printing Images:
   * All <img> tags (images) on the page are found using findElements(By.tagName("img")).
   * The total number of images is printed to the console.
   * A loop iterates over each image, printing the source URL (src attribute).
10. Cleanup:
    * The finally block ensures that the WebDriver quits and closes the browser, even if an exception occurs during the execution.
11. ***Refactor existing scripts to use the Page Object Model. Use the DemoQA Forms page for this task***

**Objective:**

This Page Object Model (POM) design pattern-based Selenium script aims to automate a form submission on the Demo QA website. The division of test logic from page interaction logic is demonstrated in this script, which improves readability and maintainability.

**Script:**

package POM;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class NewTest {

private WebDriver driver;

private Test1 formsPage;

@BeforeClass

public void setUp() {

System.out.println("Demo QA form(POM)");

System.setProperty("webdriver.gecko.driver", "C:\\Users\\admin\\Downloads\\geckodriver-v0.34.0-win64\\geckodriver.exe");

driver = new FirefoxDriver();

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

driver.get("https://demoqa.com/automation-practice-form");

formsPage = new Test1(driver);

}

@Test

public void testFormSubmission() throws InterruptedException {

formsPage.setFirstName("Nitish");

formsPage.setLastName("Mahant");

formsPage.setEmail("nitish@gmail.com");

formsPage.selectGender();

formsPage.setUserNumber("1234567890");

formsPage.setDateOfBirth("August", "2002", "02");

formsPage.setSubjects("Physics");

formsPage.selectHobbies();

formsPage.uploadPicture("C:\\Users\\admin\\Downloads\\8th Sem.jpg");

formsPage.setAddress("Gurdaspur");

formsPage.selectState("Punjab");

formsPage.selectCity("Gurdaspur");

formsPage.submitForm();

Thread.sleep(5000);

}

@AfterClass

public void tearDown() {

driver.quit();

**}**

**}**

**Explanation:**

1. Package Declaration:
   * This declares that the NewTest class is part of the POM package, following the convention for organizing code in Java.
2. Import Statements:
   * These import statements bring in the necessary classes from Selenium and TestNG to create a test class (NewTest), manage the WebDriver, and define test methods.
3. Class Declaration and Variables:
   * NewTest is the main test class.
   * driver: This WebDriver instance will be used to control the browser.
   * formsPage: An instance of the Test1 class (representing the form page in the Page Object Model) that encapsulates interactions with the form.
4. @BeforeClass Method (Setup):
   * The @BeforeClass annotation indicates that this method should run before any test methods in the class.
   * The WebDriver is initialized as a FirefoxDriver, with the system property set to locate geckodriver.
   * The browser window is maximized, and the URL for the form page is loaded.
   * An instance of the Test1 class is created and associated with the formsPage variable. This instance will handle interactions with the form.
5. @Test Method (Test Execution):
   * The @Test annotation marks this method as a test method.
   * This method calls various methods from the Test1 class to interact with the form fields, including setting text fields, selecting options, and submitting the form.
   * Each method call corresponds to a specific action on the form, like entering the first name, selecting a gender, etc.
   * Thread.sleep(5000) is used to wait for 5 seconds after form submission, which is generally not recommended. Instead, WebDriverWait should be used for synchronization, but this is a quick solution to observe the form submission.
6. @AfterClass Method (Teardown):
   * The @AfterClass annotation ensures that this method is executed after all test methods have been run.
   * driver.quit() closes the browser and ends the WebDriver session, cleaning up any resources used during the test.

**4.Conclusion**

This documentation offers thorough explanations of the scripts used in the project, as well as usage guidelines and a thorough explanation of the setup procedure. Every script is made to emphasize a distinct facet of web automation with Selenium, ranging from simple navigation to sophisticated element handling, synchronization, and POM implementation. Users can better understand and utilize Selenium in a variety of web automation scenarios by following this guide.