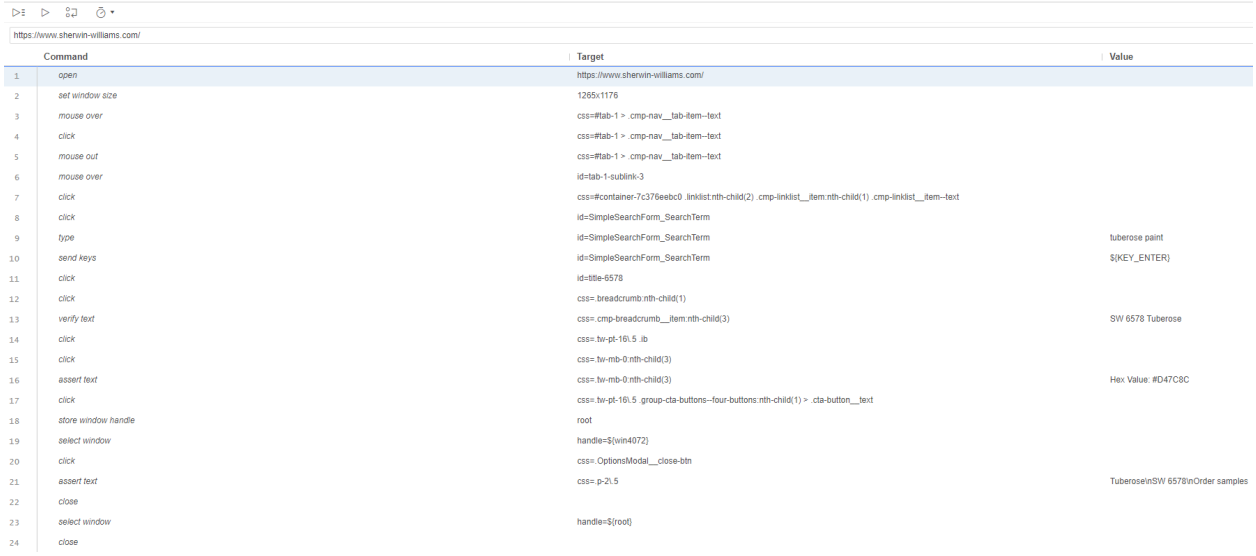
Cloud test automation interview task

The first step was to install Selenium IDE. Selenium IDE is a browser extension that allows the user to record automated tests in real time and then export these tests as code.

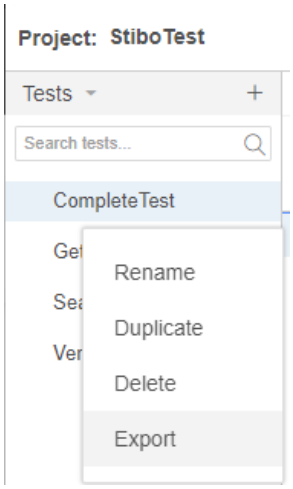
Once the test had been recorded and verified to work in Selenium IDE, the test was then exported as java code to NetBeans IDE.

For this task, I focused on writing the scenario as one test case following a happy path scenario. In a real life work environment, it would be important to consider a negative scenario, for example where the wrong color of paint was selected such as **SW 6851 | hibiscus | Interior / Exterior**.

First, the test scenario is recorded in Selenium IDE.



The test case is then exported as java code and then imported into NetBeans.



In NetBeans, the POM file containing the various repositories and directories necessary to get the project up and running is present. A JUnit test file is also present. This will contain the actual test code.

At the beginning, the various libraries that will be utilized are defined.

```
1  import org.junit.Test;
2  import org.junit.Before;
3  import org.junit.After;
4  import static org.junit.Assert.*;
5  import static org.hamcrest.CoreMatchers.is;
   import static org.hamcrest.core.IsNot.not;
7  import org.openqa.selenium.By;
8  import org.openqa.selenium.WebDriver;
   import org.openqa.selenium.firefox.FirefoxDriver;
10 import org.openqa.selenium.chrome.ChromeDriver;
   import org.openqa.selenium.remote.RemoteWebDriver;
   import org.openqa.selenium.remote.DesiredCapabilities;
13 import org.openqa.selenium.Dimension;
14 import org.openqa.selenium.WebElement;
15 import org.openqa.selenium.interactions.Actions;
   import org.openqa.selenium.support.ui.ExpectedConditions;
   import org.openqa.selenium.support.ui.WebDriverWait;
   import org.openqa.selenium.JavascriptExecutor;
   import org.openqa.selenium.Alert;
20 import org.openqa.selenium.Keys;
21 import java.util.*;
   import java.net.MalformedURLException;
   import java.net.URL;
24 import org.openqa.selenium.WindowType;
   import org.openqa.selenium.safari.SafariDriver.WindowType;
```

As part of setUp, the ChromeDriver is referenced. As part of tearDown the ChromeDriver is then closed.

```
27 /**
28  *
29  * @author Lasse
30  */
31 public class CloudAutomationTest {
32
33     public CloudAutomationTest() {
34     }
35
36     private WebDriver driver;
37     private Map <String,Object> vars;
38
39     @Before
40     public void setUp() {
41         System.setProperty(key:"webdriver.chrome.driver", value: "C:\\Users\\Lasse\\ChromeDriver\\chromedriver.exe");
42         driver = new ChromeDriver();
           vars = new HashMap<String, Object>();
44
45     }
46
47     @After
48     public void tearDown() {
49         driver.quit();
50     }
```

The website is opened and the screen resolution is set.

```
52      @Test
        public void completeTest() throws InterruptedException {
54        // Test name: CompleteTest
55        // Step # | name | target | value
56        // 1 | open | https://www.sherwin-williams.com/ |
57        driver.get(url:"https://www.sherwin-williams.com/");
58        // 2 | setWindowSize | 1265x1176 |
59        driver.manage().window().setSize(new Dimension(width: 1265, height: 1176));
60        // 3 | mouseOver | css=#tab-1 > .cmp-nav__tab-item--text |
```

Shop Products & Color is clicked.

```
66        // 4 | click | css=#tab-1 > .cmp-nav__tab-item--text |
67        driver.findElement(by: By.cssSelector(cssSelector:"#tab-1 > .cmp-nav__tab-item--text")).click();
68        // 5 | mouseOut | css=#tab-1 > .cmp-nav__tab-item--text |
69        {
70          WebElement element = driver.findElement(by: By.tagName(tagName:"body"));
71          Actions builder = new Actions(driver);
72          builder.moveToElement(target: element, xOffset:0, yOffset:0).perform();
```

Homeowner Paints, Stains & Supplies is clicked. In order to achieve this, the **mouse over** function in Selenium IDE was utilized.

```
68        // 5 | mouseOut | css=#tab-1 > .cmp-nav__tab-item--text |
69        {
70          WebElement element = driver.findElement(by: By.tagName(tagName:"body"));
71          Actions builder = new Actions(driver);
72          builder.moveToElement(target: element, xOffset:0, yOffset:0).perform();
73        }
74        // 6 | mouseOver | id=tab-1-sublink-3 |
75        {
76          WebElement element = driver.findElement(by: By.id(id: "tab-1-sublink-3"));
77          Actions builder = new Actions(driver);
78          builder.moveToElement(target: element).perform();
79        }
```

Under **Home interiors**, **Paint** is clicked.

```
79      }
80      // 7 | click | css=#container-7c376eebc0 .linklist:nth-child(2) .cmp-linklist__item:nth-child(1) .cmp-linklist__item--text |
81      driver.findElement(by: By.cssSelector(cssSelector:"#container-7c376eebc0 .linklist:nth-child(2) .cmp-linklist__item:nth-child(1) .cmp-linklist__item--text")).click();
```

The search field is selected and filled out with **tuberose paint**. The enter button is pressed.

```
85        // 8 | click | id=SimpleSearchForm_SearchTerm |
86        driver.findElement(by: By.id(id: "SimpleSearchForm_SearchTerm")).click();
87        // 9 | type | id=SimpleSearchForm_SearchTerm | tuberose paint
88        driver.findElement(by: By.id(id: "SimpleSearchForm_SearchTerm")).sendKeys(keysToSend: "tuberose paint");
89        // 10 | sendKeys | id=SimpleSearchForm_SearchTerm | ${KEY_ENTER}
90        driver.findElement(by: By.id(id: "SimpleSearchForm_SearchTerm")).sendKeys(keysToSend: Keys.ENTER);
```

Out of the search results, the product **SW 6578 | tuberose | Interior / Exterior** is clicked.

```
95          // 11 | click | id=title-6578 |
96          driver.findElement(by: By.id(id: "title-6578")).click();
```

To verify that I am on the correct page, the URL is checked. This functionality was not available in Selenium IDE and therefore this code was added manually.

```
98      if(driver.getCurrentUrl().endsWith(suffix: "https://www.sherwin-williams.com/en-us/color/color-family/red-paint-colors/SW6578-tuberose"))
99      {
100       System.out.println(x: "URL is correct");
101     }
102
103     else
104     {
105         System.out.println(x: "URL is not correct");
106     }
```

To open a new tab, the following code is added.

```
108         driver.switchTo().newWindow(typeHint: WindowType.TAB);
109
110         driver.get(url:"https://www.sherwin-williams.com/en-us/color/color-family/red-paint-colors/SW6578-tuberose");
```

The product name **SW 6578 Tuberose** is verified.

```
114     // 12 | click | css=.breadcrumb:nth-child(1) |
115     driver.findElement(by: By.cssSelector(cssSelector:".breadcrumb:nth-child(1)")).click();
116     // 13 | verifyText | css=.cmp-breadcrumb__item:nth-child(3) | SW 6578 Tuberose
117     assertThat(actual: driver.findElement(by: By.cssSelector(cssSelector:".cmp-breadcrumb__item:nth-child(3)")).getText(), matcher:is(value: "SW 6578 Tuberose"));
```

FULL DETAILS are clicked.

```
118         // 14 | click | css=.tw-pt-16\.5 .ib |
119         driver.findElement(by: By.cssSelector(cssSelector:".tw-pt-16\\.5 .ib")).click();
```

Hex Value: #D47C8C is verified.

```
123     // 15 | click | css=.tw-mb-0:nth-child(3) |
124     driver.findElement(by: By.cssSelector(cssSelector:".tw-mb-0:nth-child(3)")).click();
125     // 16 | assertText | css=.tw-mb-0:nth-child(3) | Hex Value: #D47C8C
126     assertThat(actual: driver.findElement(by: By.cssSelector(cssSelector:".tw-mb-0:nth-child(3)")).getText(), matcher:is(value: "Hex Value: #D47C8C"));
```

The **Color sample** link is clicked.

```
127         // 17 | click | css=.tw-pt-16\.5 .group-cta-buttons--four-buttons:nth-child(1) > .cta-button__text |
128         vars.put(key:"window_handles", value: driver.getWindowHandles());
```

The popup window is closed.

```
129     // 18 | storeWindowHandle | root |
130     driver.findElement(by: By.cssSelector(cssSelector:".tw-pt-16\\.5 .group-cta-buttons--four-buttons:nth-child(1) > .cta-button__text")).click();
131     // 19 | selectWindow | handle=${win4072} |
132     vars.put(key:"win4072", value: waitForWindow(timeout:2000));
133     // 20 | click | css=.OptionsModal__close-btn |
134     vars.put(key:"root", value: driver.getWindowHandle());
135     // 21 | assertText | css=.p-2\.5 | Tuberose\nSW 6578\nOrder samples
136     driver.switchTo().window(nameOrHandle: vars.get(key:"win4072").toString());
137
138     Thread.sleep(millis: Long.parseLong(s: "5000"));
139
140     // 22 | close |  |
141     driver.findElement(by: By.cssSelector(cssSelector:".OptionsModal__close-btn")).click();
```

The search field is selected and filled out with **tuberose**.

```
145        // 23 | click | css=.ml-0\.5 > span |
146        driver.findElement(by: By.cssSelector(cssSelector:".ml-0\\.5 > span")).click();
147        // 24 | type | id=SearchBarInput1 | tuberose
148        driver.findElement(by: By.id(id: "SearchBarInput1")).sendKeys(keysToSend: "tuberose");
149        // 25 | click | css=.p-2\.5 |
```

innerText is fetched from the tuberose rectangle, the value is verified and the browser is closed.

```
153        driver.findElement(by: By.cssSelector(cssSelector:".p-2\\.5")).click();
154        // 26 | assertText | css=.p-2\.5 | Tuberose\nSW 6578\nOrder samples
155        assertThat(actual: driver.findElement(by: By.cssSelector(cssSelector:".p-2\\.5")).getText(), matcher:is(value: "SW 6578\\n\\nTuberose\\n\\nOrder samples"));
156        // 27 | close |  |
157        driver.close();
158    }
```

Notes

When starting this assignment, my first intention was to use the Jetbrains Aqua IDE, which is an IDE made specifically for automated testing. The main reason for this is the embedded Web Inspector, which makes it easy and quick to capture page elements and add them to the code

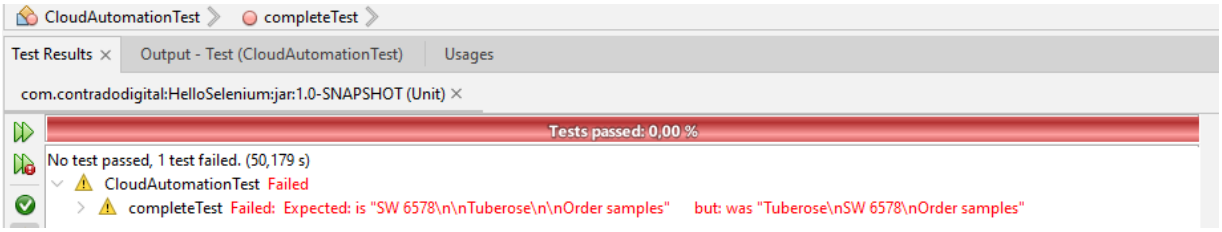
This was the main tool that I had used for writing automated test scenarios, but because of technical difficulties and the lack of support due to the IDE being relatively new, I was unable to get any tests up and running. I tried to utilize a more baseline solution, but due to my inexperience, this would be too difficult and time consuming within the scope and deadline of this assignment. My decision to use Selenium IDE in conjunction with NetBeans was made because of it being a middle-of-the-road solution which would allow me to quickly set up a selenium test while also requiring some technical understanding in order to fit the specific needs and goals of this assignment.

When using the record function of the Selenium IDE, the default screen resolution is set to 1265x1176. This was not changed during recording or when importing the code into NetBeans. Changing this to 1920x1040 should be an easy fix, but this proved to cause too many errors. In order to avoid this, the resolution was kept at 1265x1176, but in the future it would be more conducive to set a proper full screen resolution.

At some points, race conditions would occur in the test script. To avoid this, waits to be implemented at these steps.

```
Thread.sleep(millis: Long.parseLong(s: "5000"));
```

When executing the script, the test case would continually fail. The reason for this is because, when verifying the value of the **innerText** of the tuberose rectangle.



This was not a compiler error, but was caused by the actual value **Tuberose\nSW 6578\nOrder samples** of the tuberose triangle differing from the expected value **SW 6578\\n\\nTuberose\\n\\nOrder samples**. It is unclear whether this is due to an error in the website code or due to an error in the script.

If we assume that this was due to a website error, it is relevant to consider how this should be handled in an organizational manner. In such a case, my approach would be to create a bug report with as much relevant info as possible (Test steps, screenshots, browser info, system info etc.) and send it to a relevant party, such as a developer. The developer would either accept or decline the bug report. If the bug report is accepted, the development team would work on fixing the bug. Once this was done, the test would be rerun and if successful, the bug report would be closed. If the bug report is declined, the next step would be to have a dialogue with the product owner and/or stakeholders in order to make a final decision on how to proceed forward.