1- Add WebDriver Executables for Browsers

To start a WebDriver using WebDriverManager, you can avoid manually setting the path to the WebDriver executable by using the WebDriverManager library, which automatically manages the browser driver binaries required for Selenium. Here's how to set up WebDriverManager with Selenium in Java:

1. Add WebDriverManager Dependency

If you're using Maven, add WebDriverManager as a dependency in your `pom.xml` file:

```
xml
Copy code
<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>5.3.0</version> <!-- Check for the latest version -->
</dependency>
```

For Gradle, add it to your `build.gradle` dependencies:

```
gradle
Copy code
dependencies {
    testImplementation 'io.github.bonigarcia:webdrivermanager:5.3.0' // or
latest version
}
```

2. Initialize WebDriver with WebDriverManager

With the dependency in place, you can now set up WebDriver without specifying the WebDriver path directly. Here's a simple Java code example using WebDriverManager:

```
java
Copy code
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;

public class SeleniumTest {
    public static void main(String[] args) {
        // Setup ChromeDriver using WebDriverManager
        WebDriverManager.chromedriver().setup();

        // Initialize ChromeDriver
        WebDriver driver = new ChromeDriver();

        // Open a webpage
        driver.get("https://www.google.com");

        // Print the page title
```

```
        System.out.println("Page title is: " + driver.getTitle());

        // Close the browser
        driver.quit();
    }
}
```

Explanation

- **`WebDriverManager.chromedriver().setup();`**: This line automatically downloads the ChromeDriver binary compatible with the local Chrome browser version and sets up the path, so there's no need to set it manually.
- **Code Execution**: The code opens Chrome, navigates to Google, prints the page title, and closes the browser.

Benefits of WebDriverManager

- **Automatic Driver Management**: WebDriverManager ensures the WebDriver binary is compatible with the installed browser version.
- **Cross-Browser Support**: You can easily switch to other browsers (e.g., Firefox, Edge) by using `WebDriverManager.firefoxdriver().setup();` or `WebDriverManager.edgedriver().setup();`.

2- Browser Actions and WebElement Interactions.

Browser Actions in Selenium

These actions allow you to interact with the browser window itself or navigate between pages. Here are some common actions:

1. **Navigating to URLs**:

   ```
   java
   Copy code
   driver.get("https://www.example.com"); // Opens the specified URL
   driver.navigate().to("https://www.example.com"); // Alternative
   navigation method
   ```

2. **Managing Browser Navigation**:
   - `driver.navigate().back();` // Moves back one page in browser history
   - `driver.navigate().forward();` // Moves forward one page
   - `driver.navigate().refresh();` // Refreshes the current page
3. **Window Management**:
   - Switching windows:

```
java
Copy code
String originalWindow = driver.getWindowHandle();
for (String windowHandle : driver.getWindowHandles()) {
    driver.switchTo().window(windowHandle); // Switches to a new
window
}
driver.switchTo().window(originalWindow); // Returns to the
original window
```

- Managing full screen and resizing:

```
java
Copy code
driver.manage().window().maximize(); // Maximizes the browser
window
driver.manage().window().fullscreen(); // Sets browser to full
screen
driver.manage().window().setSize(new Dimension(1024, 768)); //
Custom size
```

WebElement Interactions in Selenium

These interactions involve manipulating elements on a web page, such as clicking buttons or typing into fields.

1. **Locating Elements**: Selenium provides multiple methods to locate elements, such as:

```
java
Copy code
WebElement element = driver.findElement(By.id("element_id"));
```

2. **Performing Actions on Elements**:
   - **Click**:

```
java
Copy code
WebElement button = driver.findElement(By.id("button_id"));
button.click(); // Clicks the button
```

   - **Send Keys (Typing Text)**:

```
java
Copy code
WebElement inputField = driver.findElement(By.name("username"));
inputField.sendKeys("sample text"); // Types text into a field
```

   - **Clear**:

```
java
Copy code
inputField.clear(); // Clears text from a field
```

3. **Advanced Interactions Using `Actions` Class**: The `Actions` class allows for complex interactions such as hovering, right-clicking, dragging and dropping, etc.

```
java
Copy code
Actions actions = new Actions(driver);
actions.moveToElement(element).perform(); // Hovers over an element
actions.contextClick(element).perform(); // Right-clicks an element
actions.dragAndDrop(sourceElement, targetElement).perform(); // Drags
and drops
```

4. **Getting Element Attributes and Text**:
   - **Retrieve text**:

     ```
     java
     Copy code
     String text = element.getText(); // Gets the visible text of the
     element
     ```

   - **Retrieve attribute value**:

     ```
     java
     Copy code
     String attributeValue = element.getAttribute("href"); // Gets an
     attribute value
     ```

5. **Handling Drop-Downs**: Selenium supports selecting drop-down options by index, visible text, or value.

```
java
Copy code
Select dropdown = new Select(driver.findElement(By.id("dropdown_id")));
dropdown.selectByVisibleText("Option 1");
dropdown.selectByIndex(2);
dropdown.selectByValue("option3");
```

3- WebElement Interactions and locators in Selenium in details

Selenium Locators in Detail

Locators in Selenium help identify elements uniquely. There are multiple locator strategies in Selenium:

1. By ID

This is often the fastest and most reliable locator strategy since IDs are unique to each element on a page.

```
java
Copy code
WebElement element = driver.findElement(By.id("uniqueId"));
```

2. By Name

Locates elements by the `name` attribute.

```
java
Copy code
WebElement element = driver.findElement(By.name("elementName"));
```

3. By Class Name

Locates elements by the `class` attribute. This locator may return multiple elements if multiple elements share the same class.

```
java
Copy code
WebElement element = driver.findElement(By.className("className"));
```

4. By Tag Name

Finds elements by their HTML tag, useful for grouping elements or working with tags like `<table>`, `<tr>`, `<td>`, etc.

```
java
Copy code
WebElement element = driver.findElement(By.tagName("tagname"));
```

5. By Link Text and Partial Link Text

- **Link Text**: Finds links based on their exact visible text.

  ```
  java
  Copy code
  WebElement link = driver.findElement(By.linkText("Exact Link Text"));
  ```

- **Partial Link Text**: Finds links containing a part of the visible text, useful for dynamic links.

  ```
  java
  Copy code
  WebElement link = driver.findElement(By.partialLinkText("Partial
  Text"));
  ```

6. By CSS Selector

CSS selectors allow for highly specific element selection and can be very versatile.

- **Example**: Finding an element with a specific class and attribute.

  ```
  java
  Copy code
  ```

```
WebElement element =
driver.findElement(By.cssSelector("input.className[type='submit']"));
```

7. By XPath

XPath provides a powerful way to navigate through elements and attributes in an XML or HTML document, especially useful when dealing with complex and nested elements.

- **Absolute XPath**: Starts from the root of the document (not recommended due to its fragility).

```
java
Copy code
WebElement element =
driver.findElement(By.xpath("/html/body/div/div[2]/input"));
```

- **Relative XPath**: More flexible and preferred, often using attributes or text values.

```
java
Copy code
WebElement element =
driver.findElement(By.xpath("//input[@type='submit']"));
```

1. Basic XPath Syntax

- `//`: Selects nodes in the document from the current node that match the selection, regardless of their location.
- `/`: Selects nodes in the document from the current node that match the selection at an exact position.

```
java
Copy code
// Selecting any element with the tag name `button`
WebElement button = driver.findElement(By.xpath("//button"));

// Selecting a specific element with an exact path
WebElement specificButton =
driver.findElement(By.xpath("/html/body/div/button"));
```

2. XPath Attributes

Select elements based on the value of an attribute.

- **Single Attribute**: Selects based on one attribute value.

```
java
Copy code
WebElement button = driver.findElement(By.xpath("//button[@id='submit-
button']"));
```

- **Multiple Attributes**: Uses `and` or `or` to combine multiple attribute conditions.

```
java
Copy code
WebElement button =
driver.findElement(By.xpath("//button[@type='submit' and
@class='primary']"));
```

3. Text-Based XPath

Use the `text()` function to locate elements based on their exact or partial text content.

- **Exact Text**: Locate by exact text.

```
java
Copy code
WebElement loginButton =
driver.findElement(By.xpath("//button[text()='Login']"));
```

- **Contains Text**: Use the `contains()` function to locate elements containing specific text.

```
java
Copy code
WebElement loginButton =
driver.findElement(By.xpath("//button[contains(text(),'Log')]"));
```

4. XPath Functions

XPath provides several functions that add flexibility to the locator strategy.

- **Contains**: Matches elements with partial attribute values or text content.

```
java
Copy code
// Locate by partial ID match
WebElement item =
driver.findElement(By.xpath("//*[contains(@id,'item')]"));
```

- **Starts-With**: Matches elements whose attribute values start with a specific sequence.

```
java
Copy code
WebElement emailField = driver.findElement(By.xpath("//input[starts-
with(@id, 'email')]"));
```

- **Normalize-Space**: Trims whitespace, which is useful for elements with extra spaces.

```
java
Copy code
WebElement trimmedText = driver.findElement(By.xpath("//*[normalize-
space(text())='Example Text']"));
```

5. Hierarchical (Relative) XPath

Relative XPath helps in locating elements based on their relationships to other elements, such as parent, child, or sibling.

- **Child Nodes**: Selects direct child elements.

```
java
Copy code
WebElement listItem = driver.findElement(By.xpath("//ul/li[1]"));  //
First <li> in a <ul>
```

- **Parent Nodes**: Navigate to a parent element.

```
java
Copy code
WebElement parent = driver.findElement(By.xpath("//span[@id='child-
element']/.."));
```

7. Advanced Filters

- Use conditions and indexing within `[]` to refine selection within a node set.

```
java
Copy code
// Select the third element in a list of divs
WebElement thirdDiv =
driver.findElement(By.xpath("(//div[@class='item'])[3]"));
```

- Combine conditions for more refined filtering.

```
java
Copy code
// Select element with dynamic ID containing 'start' that also has a
visible class
WebElement dynamicItem =
driver.findElement(By.xpath("//div[contains(@id, 'start') and
contains(@class, 'visible')]"));
```

WebElement Interactions in Selenium

1. Basic Interactions with Web Elements

- **Click**: Clicks on an element, such as a button, link, or checkbox.

```
java
Copy code
WebElement button = driver.findElement(By.id("submit-button"));
button.click();
```

- **Send Keys (Typing Text)**: Inputs text into a text field or text area.

```
java
Copy code
WebElement textField = driver.findElement(By.name("username"));
textField.sendKeys("sampleUser");
```

- **Clear**: Clears any text already present in an input field.

```
java
Copy code
textField.clear();
```

2. Advanced Interactions Using the `Actions` Class

The `Actions` class allows for interactions like hovering, double-clicking, and dragging and dropping.

- **Hover Over Element**:

```
java
Copy code
Actions actions = new Actions(driver);
WebElement menu = driver.findElement(By.id("menu"));
actions.moveToElement(menu).perform();
```

- **Double Click**:

```
java
Copy code
WebElement button = driver.findElement(By.id("double-click-button"));
actions.doubleClick(button).perform();
```

- **Drag and Drop**:

```
java
Copy code
WebElement source = driver.findElement(By.id("draggable"));
WebElement target = driver.findElement(By.id("droppable"));
actions.dragAndDrop(source, target).perform();
```

3. Retrieving Text and Attributes

- **Get Text**: Gets the visible text within an element.

```
java
Copy code
String buttonText = button.getText();
```

- **Get Attribute**: Retrieves the value of a specified attribute of the element, such as `href`, `src`, etc.

```
java
Copy code
String linkHref = link.getAttribute("href");
```

4. Handling Checkboxes and Radio Buttons

- **Select Checkbox**:

```
java
Copy code
WebElement checkbox = driver.findElement(By.id("checkbox1"));
if (!checkbox.isSelected()) {
    checkbox.click();
}
```

Advanced Interactions with details:

1. Hover (Mouse Hovering)

Hovering over an element can reveal dropdowns, tooltips, or trigger other actions on the page.

```
java
Copy code
Actions actions = new Actions(driver);
WebElement element = driver.findElement(By.id("hoverElementId"));
actions.moveToElement(element).perform();
```

Here, `moveToElement()` positions the cursor over the specified element.

2. Right-Click (Context Click)

A right-click on an element, also known as a context click, can bring up a context menu.

```
java
Copy code
WebElement element = driver.findElement(By.id("rightClickElement"));
actions.contextClick(element).perform();
```

3. Double Click

Double-clicking can trigger certain actions, such as selecting text or opening files.

```
java
Copy code
WebElement element = driver.findElement(By.id("doubleClickElement"));
actions.doubleClick(element).perform();
```

4. Click and Hold (Drag Without Release)

Clicks and holds an element without releasing it, which is useful for drag-and-drop interactions where a drop event is separated from the drag.

```
java
Copy code
WebElement element = driver.findElement(By.id("draggableElement"));
actions.clickAndHold(element).perform();
```

5. Drag and Drop

This interaction involves clicking an element, dragging it to a new location, and then releasing it.

```
java
Copy code
WebElement source = driver.findElement(By.id("sourceElement"));
WebElement target = driver.findElement(By.id("targetElement"));
actions.dragAndDrop(source, target).perform();
```

Alternatively, you can break it down into individual steps:

```
java
Copy code
actions.clickAndHold(source).moveToElement(target).release().perform();
```

6. Keyboard Actions (KeyDown and KeyUp)

Keyboard actions are useful for simulating the pressing and releasing of keys, such as shift, control, or alt, as well as typing sequences.

```
java
Copy code
// Holding down the SHIFT key
actions.keyDown(Keys.SHIFT).sendKeys("selenium").keyUp(Keys.SHIFT).perform();
```

This will type "selenium" in uppercase.

7. Scrolling to an Element

For pages with a large amount of content, you might need to scroll down to make an element visible.

```
java
Copy code
WebElement element = driver.findElement(By.id("scrollToElement"));
actions.moveToElement(element).perform();
```

8. Multi-touch Actions

For mobile testing, Selenium's `TouchActions` class supports swipe, tap, and other gestures, though it works best with Appium.

```
java
Copy code
TouchActions touchActions = new TouchActions(driver);
WebElement element = driver.findElement(By.id("element"));
touchActions.singleTap(element).perform();
```

9. Composite Actions (Combining Actions)

Composite actions involve chaining multiple actions together. For instance, drag and drop with key modifiers or simulating complex gestures.

```
java
Copy code
actions.moveToElement(element).click().sendKeys("Selenium
WebDriver").doubleClick().perform();
```