Selenium

Michael Inden

Agenda

- Selenium at a glance
- Selenium WebDriver API
- Selenium in combination with JUnit 5
- Selenium Best Practices
- Further possibilities

Selenium at a glance

Selenium Worth knowing

- Selenium is open source and considered a quasi-standard in web test automation.
- It allows testing for different browsers and platforms
- Provides drivers for many programming languages such as Java, C#, Python, etc.
- All over the world, Selenium is used in projects of different sizes for web application test automation.
- Test automation increases the quality of your software and frees testers and developers from boring and repetitive tasks.
- Selenium enables functional testing and regression testing
- Reminder: Test pyramid => as few GUI tests as possible by hand

- Current is Selenium 4, succesor of Selenium 3, which was released 2016 (https://www.selenium.dev/documentation/webdriver/getting_started/upgrade_to_selenium_4/)

Selenium IDE

- Selenium IDE is an Integrated Development Environment (IDE) and a browser plugin.
- It allows you to easily record and replay interactions in the browser to create test cases, but not program with Java.
- There are variants for Chrome and Firefox
- https://www.selenium.dev/selenium-ide/

Selenium WebDriver

- Selenium WebDriver is a tool for testing web applications.
- It is quite easy to understand and use
- It provides an object-oriented API for accessing elements of web pages
- Communicates directly with the browser
- Simple programmatic automation
- Works well in combination with JUnit

Selenium Worth knowing

To be able to use Selenium and the `WebDriver`, some preliminary work has to be done:

1. Language-specific libraries are needed, in our case those for Java
   https://www.selenium.dev/downloads/
2. It is easier to specify them as Maven dependencies.

3. Download and install the desired native driver
   - Firefox: https://github.com/mozilla/geckodriver/releases
   - Chrome: https://chromedriver.chromium.org/downloads

4. Start the driver in its own terminal (not necessary for Chrome)

```
michaeli$ /opt/WebDriver/bin/geckodriver
//1600178692541 geckodriver INFO Listening on 127.0.0.1:4444
```

Selenium Client Downloads

Selenium Client & WebDriver Language Bindings

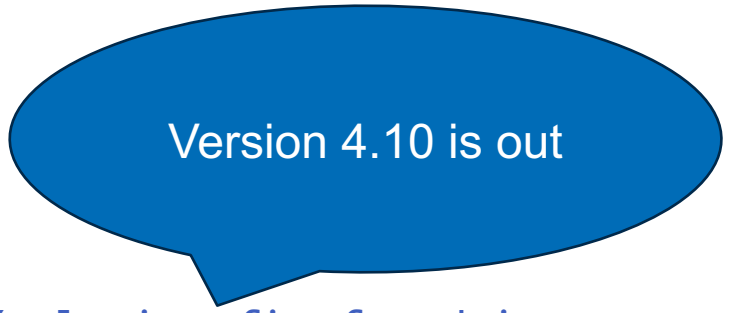
In order to create scripts that interact with the Selenium Server (Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers.

While language bindings for **other languages exist**, these are the core ones that are supported by the main project hosted on GitHub.

LANGUAGE	STABLE VERSION	RELEASE DATE	ALPHA VERSION	ALPHA RELEASE DATE	LINKS
Ruby	3.142.6	October 04, 2019	4.0.0alpha6	May 28, 2020	Download Alpha Download Changelog API Docs
Java	3.141.59	November 14, 2018	4.0.0-alpha-6	May 29, 2020	Download Alpha Download Changelog API Docs
Python	3.141.0	November 01, 2018	4.0.0a6.post1	May 28, 2020	Download Alpha Download Changelog API Docs
C#	3.14.0	August 02, 2018	4.0.0-alpha05	March 18, 2020	Download Alpha Download Changelog API Docs
JavaScript	3.6.0	October 06, 2017	4.0.0-alpha.7	March 05, 2020	Download Alpha Download Changelog API Docs

Selenium Maven Dependencies

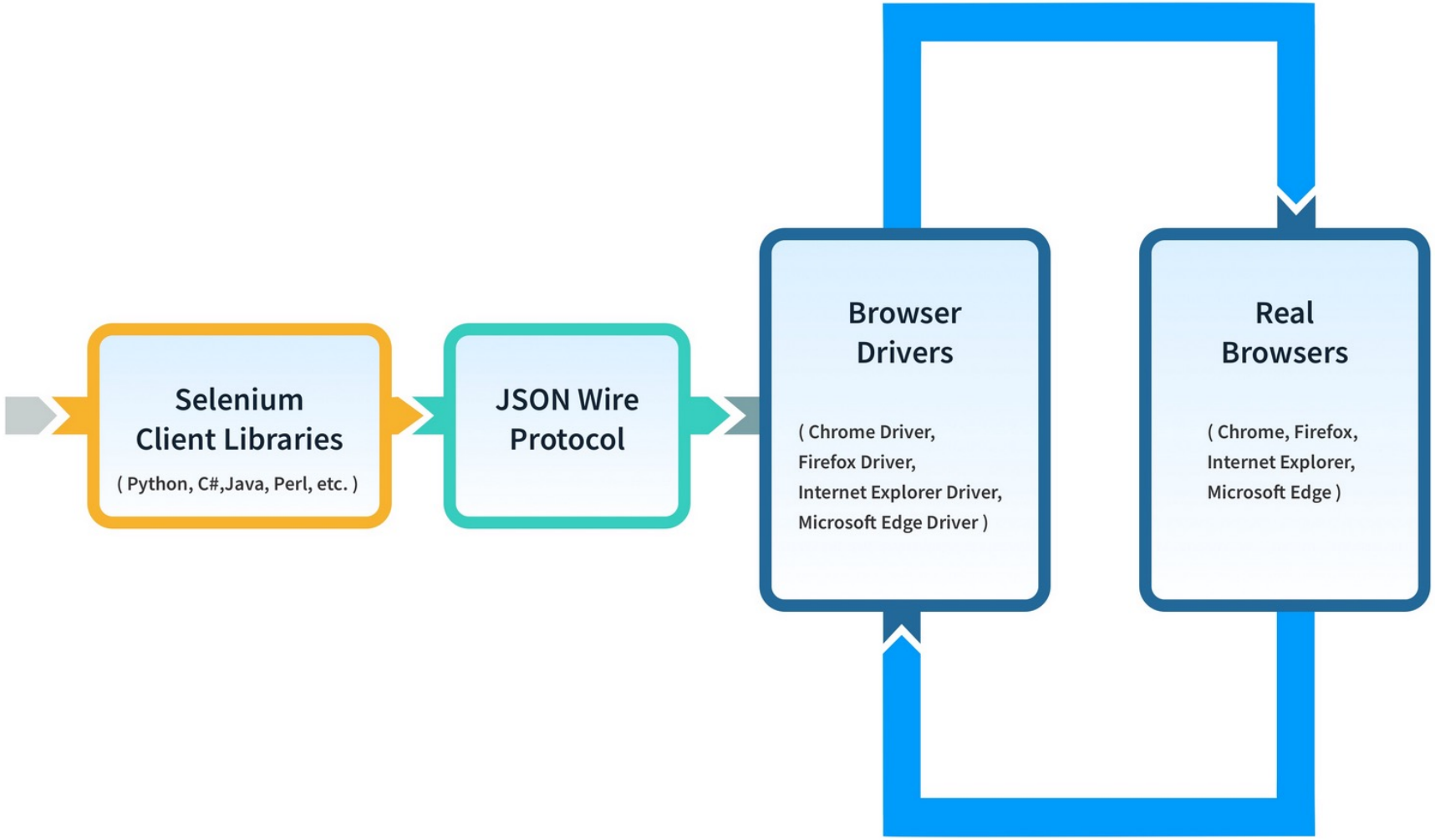
```
<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>3.141.59</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-firefox-driver -->
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-firefox-driver</artifactId>
<version>3.141.59</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-chrome-driver -->
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-chrome-driver</artifactId>
<version>3.141.59</version>
</dependency>
```



Selenium WebDriver API

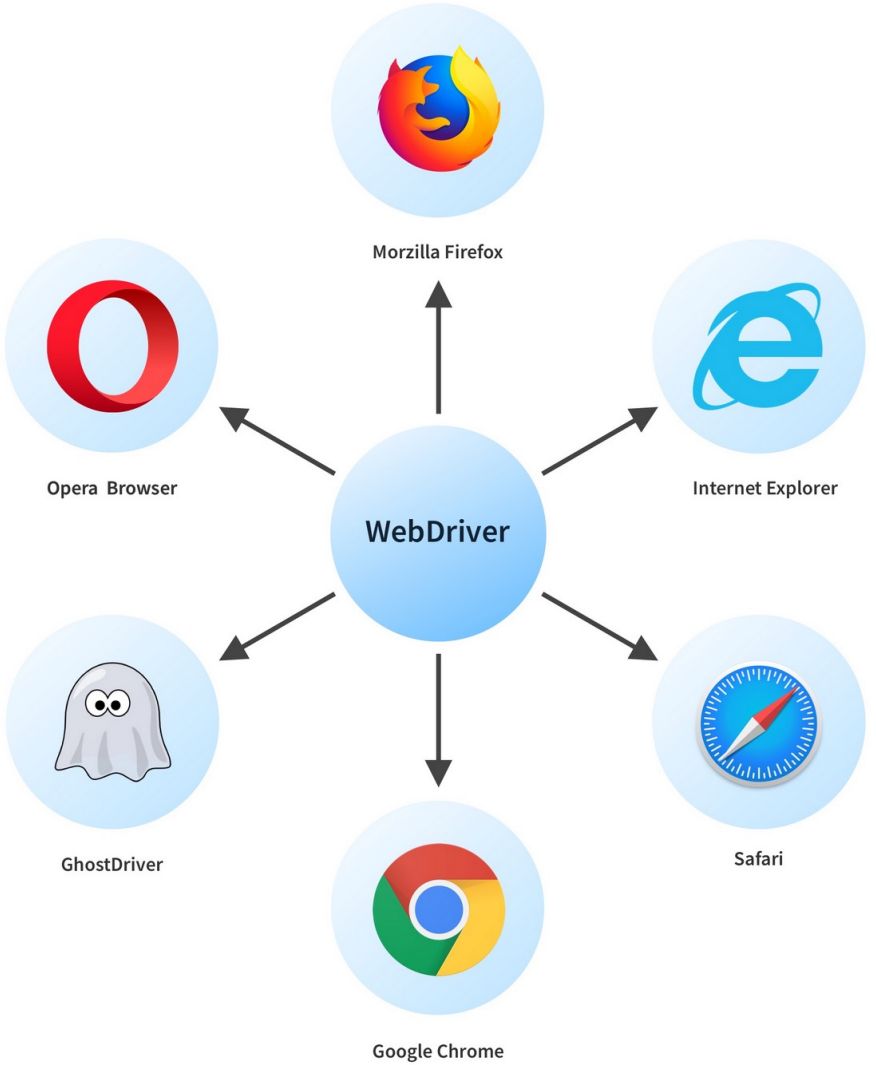
- Basics
- Locating web elements (locators)
- Interaction with different controls

Selenium Architecture



https://www.browserstack.com/guide/selenium-webdriver-tutorial

Selenium Supported Browsers



https://www.browserstack.com/guide/selenium-webdriver-tutorial

Selenium Typical Flow

- **After installing a native driver, it is necessary to set properties:**

```
System.setProperty("webdriver.gecko.driver", "/opt/WebDriver/bin/geckodriver");
System.setProperty("webdriver.gecko.driver", "C:\\geckodriver.exe");
```

- **After that, an abstraction of the particular browser is required:**

```
WebDriver driver = new FirefoxDriver();
// or
WebDriver driver = new ChromeDriver();
```

- **After having a `WebDriver` object, one is independent of the browser.**

- **For automation, Selenium needs access to a web page. A call to `get()` serves this purpose:**

```
driver.get("http://www.my-desired-page.com");
```

Selenium Things to know - What happens without a driver?

```
WebDriver driver = new FirefoxDriver();
```

```
Exception in thread "main" java.lang.IllegalStateException: The path to the driver executable must be set by the webdriver.gecko.driver system property; for more information, see https://github.com/mozilla/geckodriver. The latest version can be downloaded from https://github.com/mozilla/geckodriver/releases
```

```
WebDriver driver = new ChromeDriver();
```

```
Exception in thread "main" java.lang.IllegalStateException: The path to the driver executable must be set by the webdriver.chrome.driver system property; for more information, see https://github.com/SeleniumHQ/selenium/wiki/ChromeDriver. The latest version can be downloaded from http://chromedriver.storage.googleapis.com/index.html
```

Basic 8 steps

There are the following main steps for test cases or applications under test (AUT):

1. Provide driver suitable as system property
2. Create a `WebDriver` instance
3. Navigate to the desired web page
4. Identify the desired HTML element(s) on the web page
5. Perform actions on the HTML element(s)
6. Wait for the browser response
7. Check the results / perform test validations
8. Close the `WebDriver` and release resources

Selenium Navigation / Browser Handling

- `driver.navigate().to(<URL>)`
- `driver.navigate().refresh()`
- `driver.navigate().back()` / `driver.navigate().forward()`

- `driver.close()` – close the current window
- `driver.quit()` – close all windows

Selenium Navigation / Browser Handling

```
public class NavigationExample
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver", "/opt/WebDriver/bin/geckodriver");

        final WebDriver driver = new FirefoxDriver();

        try
        {
            driver.get("http://www.facebook.com");

            driver.navigate().to("http://www.amazon.de");
            driver.navigate().back();
        }
        finally
        {
            driver.close();
        }
    }
}
```

Selenium Trick: Opening new tabs

- **Selenium does not support opening tabs yet, coming in version 4**

- **Trick with JavaScript, but hacky (Set -> List):**

```
/* Tricky: Open new tab in browser with JavaScript */
public static void openNewTab(final WebDriver driver) throws InterruptedException
{
    JavascriptExecutor jse = (JavascriptExecutor) driver;
    jse.executeScript("window.open()");

    Thread.sleep(200);

    final List<String> tabs = new ArrayList<>(driver.getWindowHandles());
    driver.switchTo().window(tabs.get(tabs.size() - 1));
}
```

Locating web elements (Locators)

Selenium WebElements

- **A web page usually consists of a large number of elements and nested structures such as `div`.**

- **The following common elements are modeled as `WebElements`:**
  - Text box
  - Button
  - Drop Down
  - Hyperlink
  - Check Box
  - Radio Button

- **To access the elements of a web page, they must be suitably identified.**

- **So-called LOCATORs serve this purpose**

Selenium WebElements and Locators

- What is a locator? A locator is a way to uniquely access web elements on a web page.
- Normally elements are found in Selenium via Ids, names, CSS or the XPath.
- Conveniently, there is an understandable API with `findElement` / `findElements` and `By`:

```
WebElement element = driver.findElement(By.name("q"));
WebElement element = driver.findElement(By.id("rcnt"));

List<WebElement> results = driver.findElements(By.xpath("//*[@id='rso']//a"));
List<WebElement> results = driver.findElements(By.className("rc"));
```

- However, sometimes a bit tricky, because cryptic abbreviations

Example Google Search

```
public class SimpleGoogleSearch
{
    public static void main(String[] args)
    {
        System.setProperty("webdriver.gecko.driver", "/opt/WebDriver/bin/geckodriver");

        WebDriver driver = new FirefoxDriver();

        try
        {
            driver.get("https://www.google.com");
            WebElement searchTextField = driver.findElement(By.name("q"));
            WebElement searchButton = driver.findElement(By.name("btnK"));

            // Aktion
            searchTextField.sendKeys("JUnit 5");

            //searchButton.click();
            searchButton.submit();
        }
        finally
        {
            driver.quit();
        }
    }
}
```

CAUTION FRAGILITY

- Example Google search shows the fragility of the tests when the structure of the web pages changes (even slightly)

- **Suchbutton**
  - In the past: `btnG`
  - Nowadays: `btnK`

- **Result list**
  - In the past :

```
List<WebElement> results =
                    driver.findElements(By.xpath("//*[@id='rso']//h3/a"));
```

  - Nowadays :

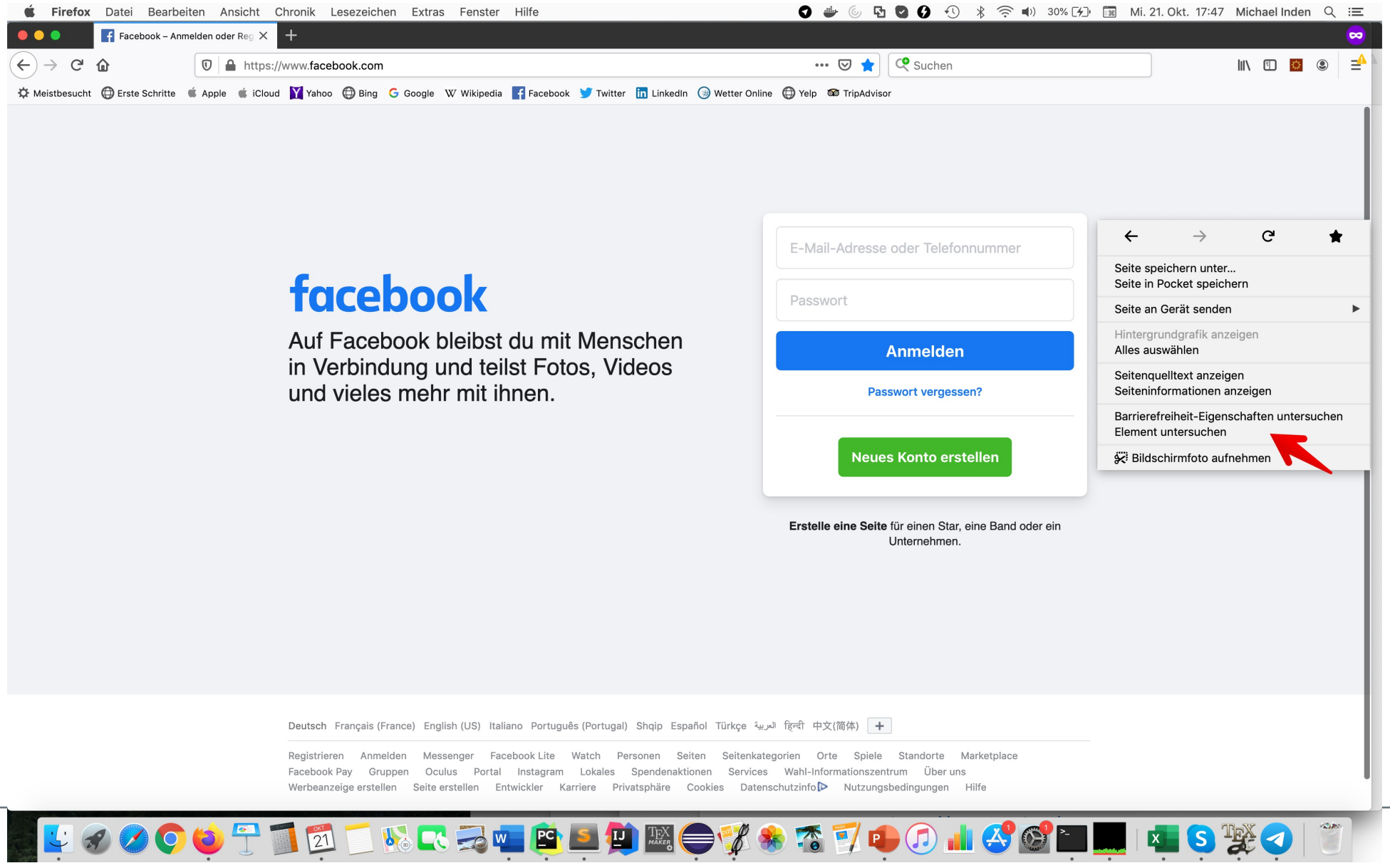
```
List<WebElement> results = driver.findElements(By.xpath("//*[@id='rso']//a"));
```

id > name > css > xpath

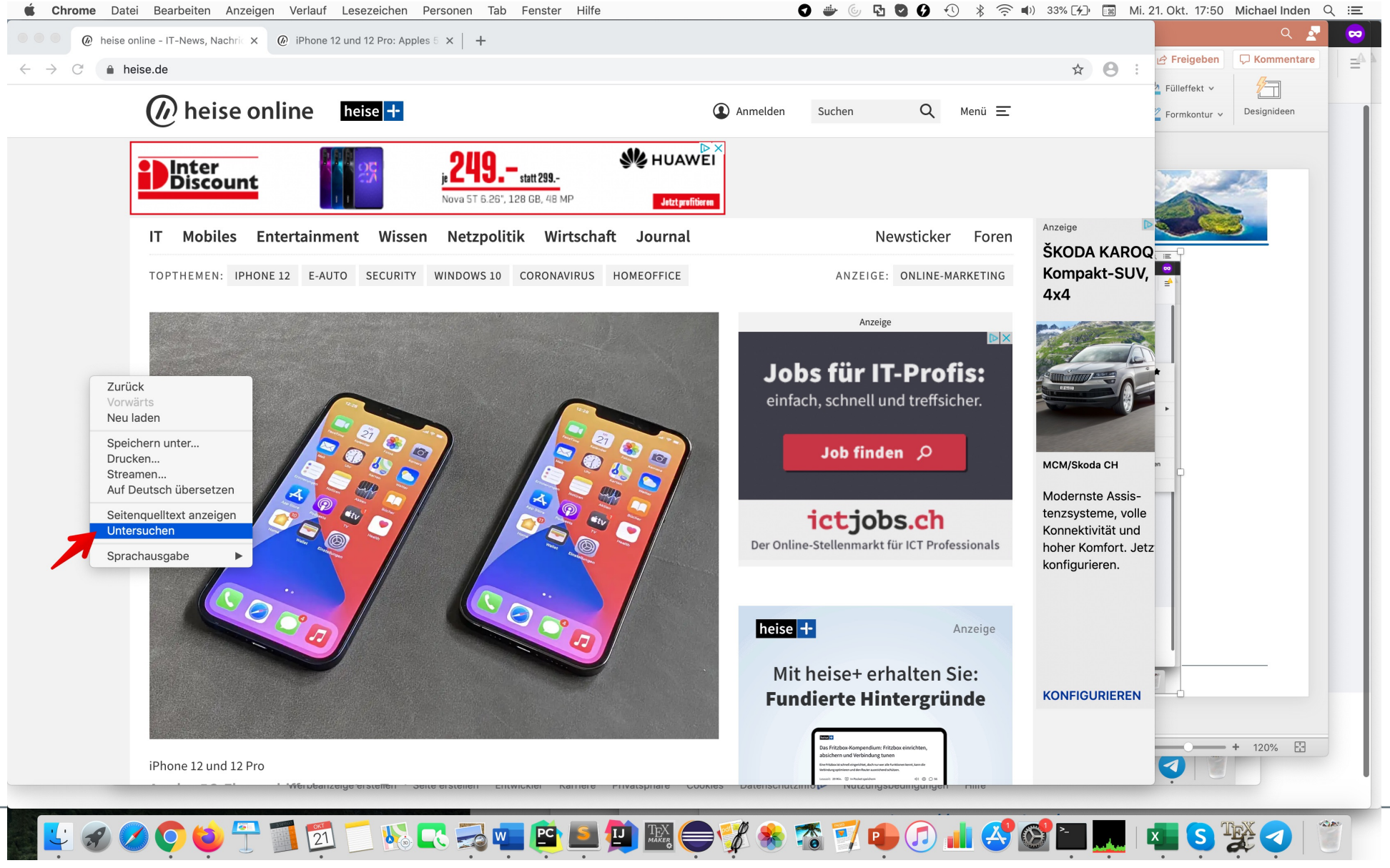
Why?

- **It is often easiest to use id and name (nevertheless sometimes problematic, if these change)**
- CSS / XPath almost always slightly cryptic, sometimes problematic because you reference wrong things (maybe one table of several is just not displayed)
- Even worse: changing controls when data is present (tab vs . tab with list)

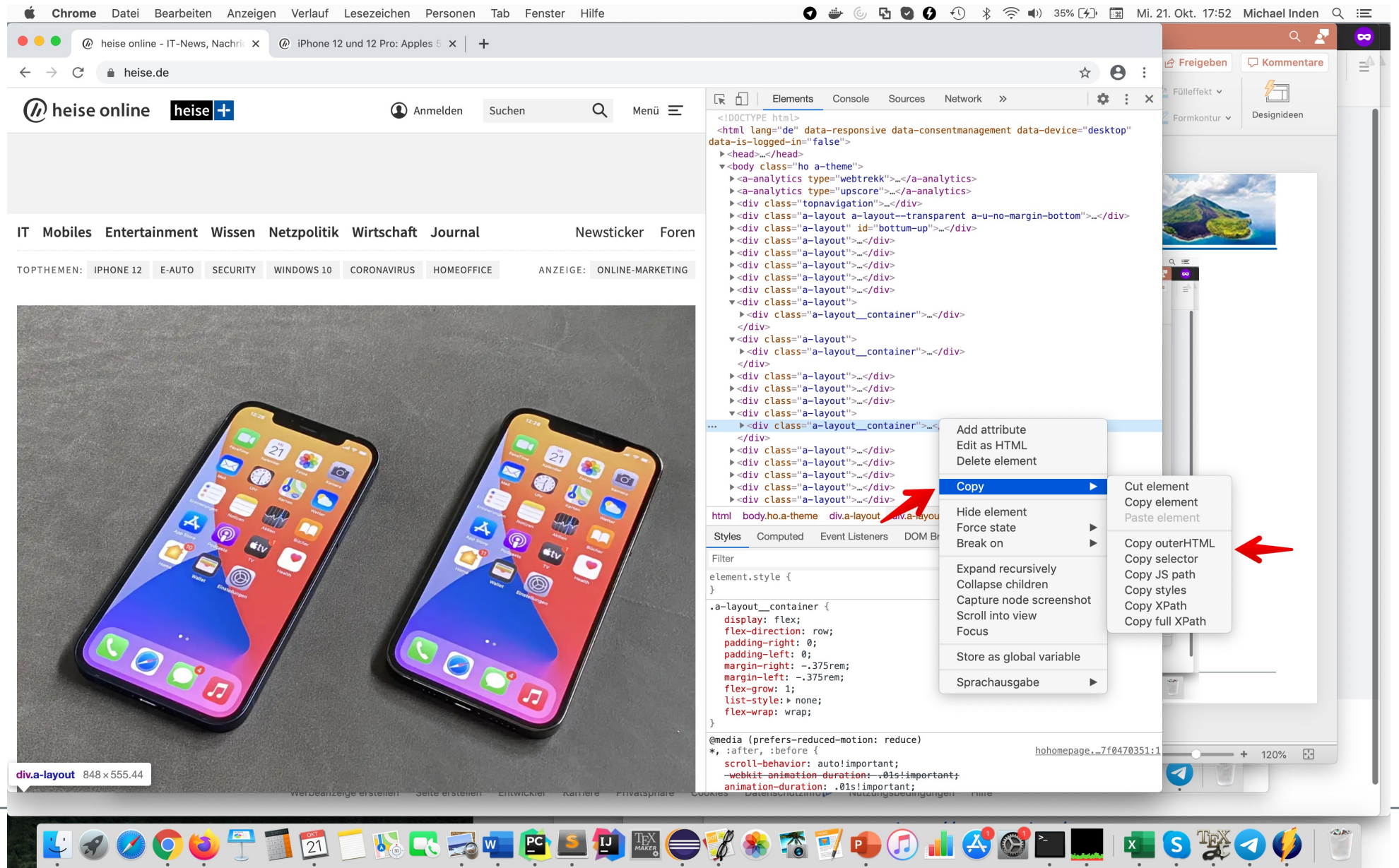
As a remedy: use developer tools



As a remedy: use developer tools



As a remedy: use developer tools



Waiting mechanisms

Selenium Waits

- **Sometimes elements are not (yet) visible ... 2 ways to wait:**
  - **Implicit** - default waiting time for all actions.
  - **Explicit-** Specific wait time for single actions

- **Implicit Wait - Attention: Sets the default wait time between EVERY single action, so a wait time of 10 seconds would slow down the test execution significantly:**

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

- **Explicit Wait - Intended to halt execution until a specific condition is met.**

```
// Initialize and wait till element(link) became clickable - timeout in 10 seconds
WebElement firstResult = new WebDriverWait(driver, 10).
            until(ExpectedConditions.elementToBeClickable(By.xpath("//a/h3")));
```

```
(new WebDriverWait(driver,10)).until(ExpectedConditions.presenceOfElementLocated(By.id("rcnt")));
```

Selenium Waits

Avoid Thread.sleep prefer Wait or FluentWait

Instead of sleep

```
1  public void clickOnContactType() {
2      getDriver().findElement(By.id("contacttypelink")).click();
3      sleep(500);
4  }
```

Prefer this fluentWait

```
1  public void clickOnContactType() {
2      getDriver().findElement(By.id("contacttypelink")).click();
3      SeleniumUtil.fluentWait(By.name("handle"), getDriver());
4  }
```

It's more robust, deterministic, in case of element not found... the exception will be clearer.

Another alternative is

```
1  (new WebDriverWait(driver, 30)).until(new ExpectedCondition() {
2                  public Boolean apply(WebDriver d) {
3                      return d.getTitle().toLowerCase().startsWith("java developer");
4                  }
```

https://mestachs.wordpress.com/2012/08/13/selenium-best-practices/

Actions on web elements

Selenium Actions on elements

- **Performing actions**
  - First you have to identify the WebElements with `findElement()` / `findElements()`
  - You get `WebElement` objects
  - After that you can interact with them by method calls

- **Important methods**
  - `sendKeys()` allows the text input
  - `clear()` deletes the entered text
  - `Submit()` sends a form

- ***`sendKeys()`* is probably the most used of them**
- ***`submit()`* can be called on the button or form**

Selenium Actions on elements

- **Input Box**
  - Text Fields for entering textual information
  - Password Field for entering passwords with masked representation
  - method(s): `sendKeys(), clear()`

- **Buttons**
  - Button to trigger an action
  - Method(s): `click()`

- **Submit Buttons**
  - Buttons to send the form to the server
  - method(s): `click(), submit()`
  - Sometimes `click()` does not work like `submit()` => see Google search example

Selenium Actions on elements

- **Radio Button**
  - Radio button to select one option among many, by selecting one option the other selected one will be turned off
  - Method: `click()`
- **Check Box**
  - Check Box to activate / deactivate a selection
  - Method: `click()`
- **Drop Down (Select)**
  - Combo box (HTML Select) to select from a list of possibilities (Multi Select supported)
  - Custom type Select

```
Select selectColors = new Select(driver.findElement(By.id("colors")));
```

  - Methods: `selectByValue()`, `selectByVisibleText()`, `selectByValue()`, `selectByIndex()`, `deselectXyz()`, `getAllSelectedOptions()`, ...

Selenium Links

- So far we have considered rather classic controls ... What is missing are LINKs!
- Examine concrete known elements:

```
WebElement link1 = driver.findElement(By.linkText("Heise News"));
WebElement link2 = driver.findElement(By.partialLinkText("Goog"));
```

- We can click the links and be redirected to the corresponding page:

```
link1.click();
```

- Examine unknown web elements in advance, as an example: find all links on a page:

```
private static List<WebElement> getAllLinksFromPage(final WebDriver driver)
{
    List<WebElement> links = driver.findElements(By.tagName("a"));
    return links;
}
```

Selenium Status queries

It is possible to query whether web elements (buttons, drop boxes, checkboxes, radio buttons, labels etc.) are visible or activated:

- `isDisplayed()`
- `isSelected()`
- `isEnabled()`

```
boolean buttonPresence = driver.findElement(By.id("filterBtn")).isDisplayed();

boolean buttonSelected = driver.findElement(By.id("filterBtn")).isSelected();

boolean searchIconEnabled = driver.findElement(By.id("searchBtn")).isEnabled();
```

Example

RadioCheckExample.html

```
<html>
<head>
    <title>Radio Button and Check Box Example</title>
</head>
<body>
    <strong>Radio</strong><br/>
    <input type="radio" id="radio-1" value="Option 1">Option1<br/>
    <input type="radio" id="radio-2" value="Option 2">Option2<br/>
    <input type="radio" id="radio-3" value="Option 3">Option3<br/><br/>

    <strong>Checkbox</strong><br/>
    <input type="checkbox" id="check-1" value="checkbox1">Checkbox1<br/>
    <input type="checkbox" id="check-2" value="checkbox2">Checkbox2<br/>
    <input type="checkbox" id="check-3" value="checkbox3">Checkbox3<br/>
</body>
</html>
```

Radio
- Option1
- Option2
- Option3

Checkbox
- Checkbox1
- Checkbox2
- Checkbox3

Example

```
public static void main(String[] args)
{
    System.setProperty("webdriver.gecko.driver", "/opt/WebDriver/bin/geckodriver");

    WebDriver driver = new FirefoxDriver();

    try
    {
        driver.get("file:///Users/michaeli/Desktop/RadioCheckExample.html");
        driver.manage().window().maximize();

        WebElement radio1 = driver.findElement(By.id("radio-1"));
        WebElement radio2 = driver.findElement(By.id("radio-2"));
        radio1.click();
        radio2.click();

        WebElement check1 = driver.findElement(By.id("check-1"));
        check1.click();

        String checkState = check1.isSelected() ? "ON" : "OFF";
        System.out.println("Checkbox is toggled " + checkState);
    }
    finally
    {
        driver.quit();
    }
}
```

Radio

- Option1
- Option2
- Option3

Checkbox

- Checkbox1
- Checkbox2
- Checkbox3

Example

LinksAndDropDownExample.html

```
<ul id="linkTabs">
<li>
    <a href="https://www.heise.de/">Heise</a>
</li>
<li>
    <a href="http://www.dpunkt.de/">dpunkt.verlag</a>
</li>
</ul>

<div>
<select id="colors">
<option value="red">Red</option>
<option value="green">Green</option>
<option value="blue">Blue</option>
<option value="purple">Purple</option>
</select>
</div>
```

- Heise
- dpunkt.verlag

Red

Apple

Example

```
driver.findElement(By.linkText("Heise")).click();
driver.navigate().back();
Thread.sleep(2000);

Select selectByValue = new Select(driver.findElement(By.id("colors")));
selectByValue.selectByValue("green");
Thread.sleep(2000);

Select selectByVisibleText = new Select(driver.findElement(By.id("fruits")));
selectByVisibleText.selectByVisibleText("Lime");
Thread.sleep(2000);
```

- Heise
- dpunkt.verlag

Red

Apple

Selenium in combination with JUnit 5

Selenium and JUnit 5

- **So far we have also gained an idea about Selenium for remote control of websites**
- **BUT: How do we bring it all together?**
- **Let's recap briefly:**

Action		Selenium		JUnit
One time at startup		Driver as system property		`@BeforeAll`
Once before each method		Creation of `WebDriver`		`@BeforeEach`
Once after each method		Close windows, release drivers and resources		`@AfterEach`

Selenium and JUnit 5

```
class SeleniumAsTest
{
    WebDriver driver;

    @BeforeAll
    public static void init()
    {
        System.setProperty("webdriver.gecko.driver", "/opt/WebDriver/bin/geckodriver");
    }

    @BeforeEach
    public void initDriver()
    {
        driver = new FirefoxDriver();
    }

    @AfterEach
    public void closeWindowsAndFreeResources()
    {
        driver.quit();
    }
}
```

Selenium and JUnit 5

```
@Test
public void testRadioAndCheckFunctionality()
{
    // ARRANGE
    driver.get("file:///Users/michaeli/Desktop/RadioCheckExample.html");
    driver.manage().window().maximize();

    // ACT
    WebElement radio1 = driver.findElement(By.id("radio-1"));
    WebElement radio2 = driver.findElement(By.id("radio-2"));
    radio1.click();
    radio2.click();

    WebElement check1 = driver.findElement(By.id("check-1"));
    check1.click();

    // ASSERT
    assertAll(() -> assertFalse(radio1.isSelected(), "Radio 1 should be inactive"),
              () -> assertTrue(radio2.isSelected(), "Radio 1 should be active"),
              () -> assertTrue(check1.isSelected(), "Check should be selected"));
}
```

Selenium and JUnit 5

```
@Test
public void testSelectFunctionality()
{
    // ARRANGE
    driver.get("file:///Users/michaeli/Desktop/LinksAndDropDownExample.html");

    // ACT
    driver.findElement(By.linkText("Heise")).click();
    driver.navigate().back();

    new WebDriverWait(driver, 2).until(presenceOfElementLocated(By.id("colors")));
    Select selectByValue = new Select(driver.findElement(By.id("colors")));
    selectByValue.selectByValue("green");

    Select selectByVisibleText = new Select(driver.findElement(By.id("fruits")));
    selectByVisibleText.selectByVisibleText("Lime");

    // ASSERT
    assertAll(() -> assertEquals("Testing Select Class", driver.getTitle()), // on the right page
              () -> assertEquals("Green", selectByValue.getFirstSelectedOption().getText()),
              () -> assertEquals("Lime", selectByVisibleText.getFirstSelectedOption().getText()));
}
```

Selenium Best Practices

- Page Object Pattern

Page Object Pattern

Page Object

- Is a pattern to increase the maintainability and understandability of tests.
- The Page object is a normal Java class that abstracts the web elements
- Provides methods to, for example, enter values in text fields, click a button, etc.
- Provides good separation between test code and page-specific detail such as locators, etc.
- Let's look at a login page as an example

Page Object Example

```
public class LoginPage
{
    private static final By USERNAME_FIELD = By.id("loginForm:username");
    private static final By PASSWORD_FIELD = By.id("loginForm:password");
    private static final By LOGIN_BUTTON = By.id("loginForm:login");

    private final WebDriver driver;

    public LoginPage(WebDriver driver, URL contextPath)
    {
        this.driver = driver;
        this.driver.get(contextPath + "login.html");
    }

    public void login(String name, String password)
    {

        driver.findElement(USERNAME_FIELD).sendKeys(name);
        driver.findElement(PASSWORD_FIELD).sendKeys(password);
        driver.findElement(LOGIN_BUTTON).click();
    }
}
```

Page Object Demo

Login

Anmelden

```java
public class LandingPage
{
    enum Color { RED, GREEN, BLUE, PURPLE }

    private WebDriver     driver;


    public LandingPage(WebDriver driver)
    {
        this.driver = driver;
        this.driver.get("file:///Users/michaeli/Desktop/LandingPage.html");
    }

    public boolean isLoadedCorrectly()
    {
        return driver.getCurrentUrl().contains("LandingPage");
    }

    public void chooseRadioOption(int nr)
    {
        if (nr >= 1 && nr <= 3)
        {
            driver.findElement(By.id("radio-"+nr)).click();
        }
    }

    public void chooseColor(Color colorToChoose)
    {
        Select select = (Select)driver.findElement(By.id("colors"));
        select.selectByValue(colorToChoose.name().toLowerCase());
    }
```

Landing Page Page O

Hello

Abmelden

Radio

- Option1
- Option2
- Option3

Checkbox

- Checkbox1
- Checkbox2
- Checkbox3

- Heise
- dpunkt.verlag

Red

Apple

Page Object

- In the using tests, there is nothing left of the complexity and low-level details.
- Thus, the test case does not show Selenium and other technical details.
- This allows a large number of UI changes to be executed without (major) impact on the tests.
- You have the following advantages
  - Easier to maintain and more reliable than scattered locator accesses
  - Better readability and comprehensibility of tests
  - Less duplication when accessing elements

Page Object – Simple login test

```
@Test
public void testLoginFunctionality() throws InterruptedException
{
    // ARRANGE
    LoginPage loginPage = new LoginPage(driver);

    // ACT
    LandingPage landingPage = loginPage.login("Peter", "Lustig");
    Thread.sleep(2_000);

    // ASSERT
    assertTrue(landingPage.isLoadedCorrectly());
}
```

Page Object – Simple login test

```
@Test
public void testLoginFunctionality() throws InterruptedException
{
    // ARRANGE
    LoginPage loginPage = new LoginPage(driver);

    // ACT
    LandingPage landingPage = loginPage.login("Peter", "Lustig");
    Thread.sleep(2_000);

    // Old style without page object possible, but rather to be avoided
    WebElement check1 = driver.findElement(By.id("check-1"));
    check1.click();

    // ASSERT
    assertTrue(landingPage.isLoadedCorrectly());
}
```

Page Object – Simple login test

```
@Test
public void testLandingpageFunctionality()
{
    // ARRANGE
    LandingPage landingPage = loginWithCredentials("Peter", "Lustig");

    // ACT
    landingPage.chooseColor(LandingPage.Color.BLUE);
    landingPage.chooseRadioOption(2);

    // ASSERT
    // ...
}

private LandingPage loginWithCredentials(String name, String pwd)
{
    return new LoginPage(driver).login(name, pwd);
}
```

Page Object Advantages / Things to know

✓ Page classes get `WebDriver` in constructor
✓ Page Objects define attributes and methods for accessing web elements
✓ Selenium specifics are encapsulated by Page Objects

✓ Tests do not contain any Selenium code.
✓ Tests use page classes
✓ Tests use `@Test` annotation and simple JUnit assertions
✓ WebDriver is created in `setUp()`
✓ WebDriver is released in `tearDown()`

Other possibilities

Broken Link Checker

Heise dpunkt.verlag Heise dpunkt.verlag Amazon Amazon

```
<!DOCTYPE html>
<html>
<body>
<a href="https://www.heise.de/">Heise</a>
<a href="http://www.dpunkt.de/">dpunkt.verlag</a>
<a href="https://www.heis.de/">Heise</a> <!-- BROKEN -->
<a href="http://www.dpunk.de/">dpunkt.verlag</a> <!-- BROKEN -->
<a href="http://www.amatzon.de/">Amazon</a>  <!-- Wird korrekt auf Amazon.de umgeleitet -->
<a href="http://www.ametzon.de/">Amazon</a>  <!-- BROKEN -->
</body>
</html>
```

Broken Link Checker

```
public static void main(String[] args)
{
    System.setProperty("webdriver.gecko.driver", "/opt/WebDriver/bin/geckodriver");
    final WebDriver driver = new FirefoxDriver();

    try
    {
        driver.get("file:///Users/michaeli/Desktop/SomeBrokenLinksExample.html");

        final List<WebElement> links = getAllLinksFromPage(driver);
        final Iterator<WebElement> it = links.iterator();
        while (it.hasNext())
        {
            final String url = it.next().getAttribute("href");
            final String state = checkValidityOfUrl(url) ? "VALID" : "BROKEN";
            System.out.println(url + " is " + state);
        }
    }
    finally
    {
        driver.quit();
    }
}
```

Broken Link Checker

```
private static boolean checkValidityOfUrl(final String url)
{
    if (url == null || url.isEmpty())
        return false;

    try
    {
        final HttpURLConnection huc = (HttpURLConnection) (new URL(url).openConnection());
        huc.setRequestMethod("HEAD");
        huc.connect();

        final int respCode = huc.getResponseCode();

        return respCode >= 200 && respCode < 400;
    }
    catch (IOException e)
    {
    }
    return false;
}
```

Create Screen Shot

```
public class CaptureScreenshot
{
    public static void main(String[] args) throws IOException
    {
        System.setProperty("webdriver.gecko.driver", "/opt/WebDriver/bin/geckodriver");

        final WebDriver driver = new FirefoxDriver();

        try
        {
            driver.get("http://www.heise.de");

            // capture the screenshot
            final File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
            final Path targetPath = Paths.get("./ScreenShot.jpg");
            Files.move(scrFile.toPath(), targetPath, StandardCopyOption.REPLACE_EXISTING);
        }
        finally
        {
            driver.quit();
        }
    }
}
```

Selenium Exercises

Further Info / Links

- https://www.selenium.dev/downloads/
- https://www.selenium.dev/documentation/en/webdriver/driver_requirements/
- https://www.youtube.com/watch?v=cobEbkTwbwY
- https://www.amazon.de/Improve-Selenium-Code-Automation-Patterns-ebook/dp/B077QFN53F
- http://www.vpl.ca/
- https://www.toptal.com/selenium/test-automation-in-selenium-using-page-object-model-and-page-factory

Thank You