



# Selenium

**Michael Inden**

**Freiberuflicher Consultant, Buchautor und Trainer**

---

# Coding Guidelines – Agenda

---



- **Selenium im Kurzüberblick**
  - **Selenium WebDriver API**
  - **Selenium in Kombination mit JUnit 5**
  - **Selenium Best Practices**
  - **Weitere Möglichkeiten**
-



# Selenium im Kurzüberblick



- Das Open Source Projekt Selenium gilt als Quasi-Standard im Bereich Web-Testautomatisierung.
  - Es erlaubt das Testen für verschiedene Browser und Plattformen
  - Bietet Treiber für viele Programmiersprachen wie Java, C#, Python usw.
  - Überall auf der Welt wird Selenium in Projekten unterschiedlicher Größe für die Testautomatisierung von Webapplikationen eingesetzt.
  - Testautomatisierung erhöht die Qualität Ihrer Software und befreit die Tester und Entwickler von langweiligen und repetitiven Aufgaben.
  - Erinnerung: Testpyramide => möglichst wenige GUI Tests per Hand
  
  - Aktuell ist Selenium 3, veröffentlicht 2016, Selenium 4 steht kurz vor der Veröffentlichung
  - Am einfachsten in Kombination mit Maven
  
  - Selenium ermöglicht Functional Testing und Regression Testing
-

# Selenium IDE

---



- **Selenium IDE ist eine Integrated Development Environment (IDE) und ein Browser-Plugin**
  - **Damit kann man auf einfache Weise Interaktionen im Browser aufnehmen und wieder abspielen und so Testfälle erzeugen, aber nicht mit Java programmieren.**
  - **Es gibt Varianten für Chrome und Firefox**
  - **<https://www.selenium.dev/selenium-ide/>**
-

# Selenium WebDriver

---



- **Selenium WebDriver ist ein Tool zum Testen von Web-Applikationen**
  - **Es ist recht leicht verständlich und gut handhabbar**
  - **Es bietet ein objekt-orientiertes API zum Zugriff auf Elemente von Webseiten**
  - **Kommuniziert direkt mit dem Browser**
  - **Einfache programmatische Automatisierung**
  - **Lässt sich gut in Kombination mit JUnit nutzen**
-



Um Selenium und den WebDriver nutzen zu können, sind ein paar **Vorarbeiten** auszuführen:

1. Es werden die sprachspezifischen Bibliotheken benötigt, in unserem Fall die für Java  
<https://www.selenium.dev/downloads/>
2. Einfacher ist es diese als **Maven-Dependencies** anzugeben
3. Der jeweils gewünschte native Treiber heruntergeladen und installieren
  1. Firefox: <https://github.com/mozilla/geckodriver/releases>
  2. Chrome: <https://chromedriver.chromium.org/downloads>
4. Den Treiber in eigenem Terminal starten (für Chrome nicht nötig)

```
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	<a href="#">Download Alpha</a> <a href="#">Download Changelog</a> <a href="#">API Docs</a>
Java	3.141.59	November 14, 2018	4.0.0-alpha-6	May 29, 2020	<a href="#">Download Alpha</a> <a href="#">Download Changelog</a> <a href="#">API Docs</a>
Python	3.141.0	November 01, 2018	4.0.0a6.post1	May 28, 2020	<a href="#">Download Alpha</a> <a href="#">Download Changelog</a> <a href="#">API Docs</a>
C#	3.14.0	August 02, 2018	4.0.0-alpha05	March 18, 2020	<a href="#">Download Alpha</a> <a href="#">Download Changelog</a> <a href="#">API Docs</a>
JavaScript	3.6.0	October 06, 2017	4.0.0-alpha.7	March 05, 2020	<a href="#">Download Alpha</a> <a href="#">Download Changelog</a> <a href="#">API Docs</a>

# 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>
```

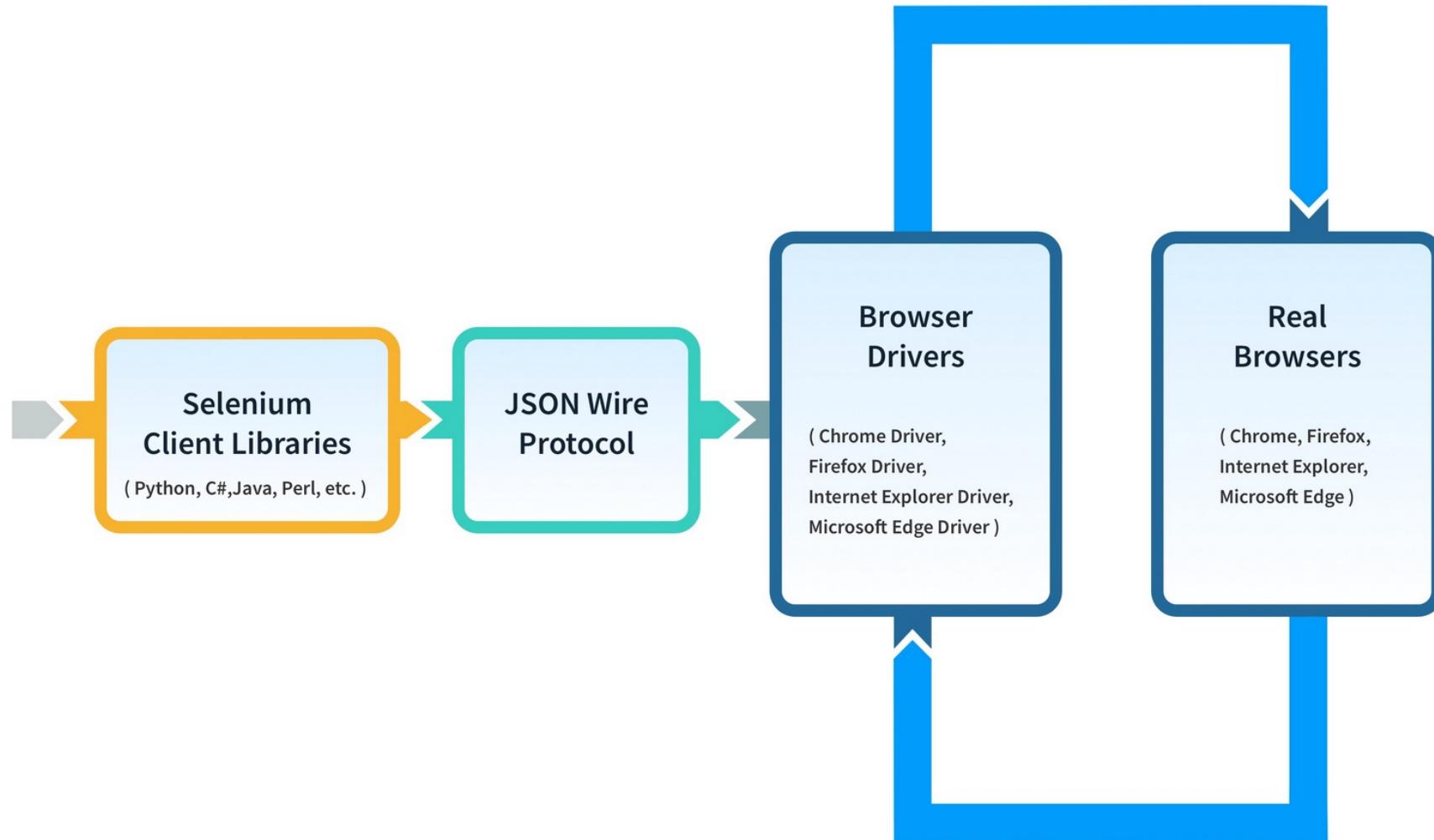


---

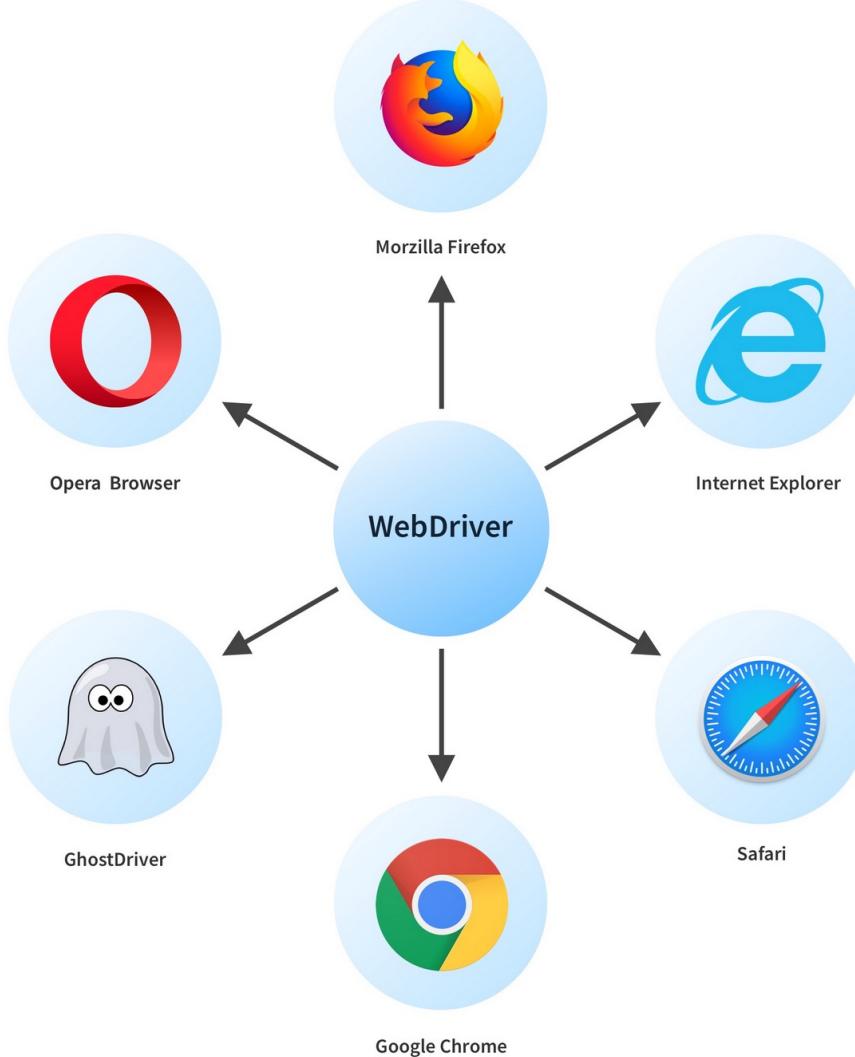
# Selenium WebDriver API

- Grundlagen
  - Auffinden von Webelementen (Locators)
  - Interaktion mit unterschiedlichen Steuerelementen
-

# Selenium Architektur



# Selenium Unterstütze Browser



# Selenium Typischer Ablauf

---



- Nach der Installation eines nativen Treibers ist das Setzen von Properties nötig:

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

- Danach ist eine Abstraktion des jeweiligen Browsers erforderlich:

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

- Nachdem man ein WebDriver-Objekt hat, ist man unabhängig vom Browser
- Zur Automatisierung benötigt Selenium Zugriff auf eine Webseite. Dazu dient ein Aufruf von get():

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

---

# Selenium Wissenswertes – Was passiert ohne Treiber?

---



```
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>

---

## Grundsätzliche 8 Schritte

---



Es gibt folgende Hauptschritte für Testfälle bzw. zu testende Anwendungen (AUT):

- 1. Treiber geeignet als System-Property bereitstellen**
  - 2. Eine WebDriver-Instanz erstellen**
  - 3. Zur gewünschten Webseite navigieren**
  - 4. Das oder die gewünschten HTML-Element(e) auf der Webseite identifizieren**
  - 5. Aktionen auf dem / den HTML-Element(en) ausführen**
  - 6. Auf die Antwort des Browsers warten**
  - 7. Die Ergebnisse prüfen / Testvalidierungen durchführen**
  - 8. Den WebDriver schließen und Ressourcen freigeben**
-

# Selenium Navigation / Browser Handling

---



- `driver.navigate().to(<URL>)`
  - `driver.navigate().refresh()`
  - `driver.navigate().back() / driver.navigate().forward()`
  - `driver.close()` – das aktuelle Fenster schliessen
  - `driver.quit()` – Alle Fenster schliessen
-

# 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: Öffnen neuer Tabs

---



- Selenium unterstützt Öffnen von Tabs noch nicht, kommt in Version 4
- Trick mit JavaScript, aber hacky (Set -> List):

```
/* Tricky: Open new tab in browser with JavaScript */
public static void openNewTab(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));
}
```



# Auffinden von Webelementen (Locators)

# Selenium WebElements

---



- Eine Webseite besteht in der Regel aus einer Vielzahl an Elementen und verschachtelten Strukturen wie div.
  - Folgende gebräuchliche Elemente werden als WebElement modelliert:
    - Text box
    - Button
    - Drop Down
    - Hyperlink
    - Check Box
    - Radio Button
  - Zum Zugriff auf die Elemente einer Webseite müssen diese geeignet identifiziert werden
  - Dazu dienen sogenannte LOCATORS
-

# Selenium WebElements und Locators

---



- Was ist ein Locator? Ein Locator ist eine Art auf Webelemente auf einer Webseite eindeutig zugreifen zu können.
- Normalerweise werden Elemente in Selenium über Ids, Namen, CSS oder den XPath gefunden.
- Praktischerweise existiert ein verständliches API mit `findElement` / `findElements` und `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"));
```
- Allerdings manchmal etwas tricky, da kryptische Kürzel

# Beispiel Google-Suche



```
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();
        }
    }
}
```

# ACHTUNG FRAGILITÄT

---



- Beispiel Google-Suche zeigt die Fragilität der Tests, wenn die Struktur der Webseiten sich (auch nur leicht ändert)
- **Suchbutton**
  - Früher: btnG
  - heute: btnK
- **Ergebnisliste**
  - Früher:

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

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



# **id > name > css > xpath**

Wieso?

- **Per id und name geht es oftmals am einfachsten  
(Trotzdem mitunter problematisch, wenn sich diese ändern)**
- CSS / XPath fast immer leicht kryptisch, manchmal problematisch, da man falsche Dinge referenziert (vielleicht wird eine Tabelle von mehreren gerade nicht dargestellt)
- Schlimmer noch: Wechselnde Bedienelemente bei Datenlage (Tab vs . Tab mit Liste)

# Als Abhilfe: Developer Tools nutzen



Screenshot of a Firefox browser window showing the Facebook login page (<https://www.facebook.com>). The browser interface includes the address bar, toolbar, and developer tools menu.

The developer tools menu on the right side of the screen is open, and a red arrow points to the "Bildschirmfoto aufnehmen" (Screenshot) option in the list.

Facebook Login Form:

- E-Mail-Adresse oder Telefonnummer
- Passwort
- Anmelden (Login button)
- Passwort vergessen? (Forgot password link)
- Neues Konto erstellen (Create new account link)

Text on the page:

Auf Facebook bleibst du mit Menschen in Verbindung und teilst Fotos, Videos und vieles mehr mit ihnen.

Bottom of the page:

- Erstelle eine Seite für einen Star, eine Band oder ein Unternehmen.
- Links: Deutsch, Français (France), English (US), Italiano, Português (Portugal), Shqip, Español, Türkçe, હિન્દી, 中文(简体)
- Right: +
- Footer links: Registrieren, Anmelden, Messenger, Facebook Lite, Watch, Personen, Seiten, Seitenkategorien, Orte, Spiele, Standorte, Marketplace, Facebook Pay, Gruppen, Oculus, Portal, Instagram, Lokales, Spendenaktionen, Services, Wahl-Informationszentrum, Über uns, Werbeanzeige erstellen, Seite erstellen, Entwickler, Karriere, Privatsphäre, Cookies, Datenschutzinfo, Nutzungsbedingungen, Hilfe
- Bottom bar: various application icons (Safari, Mail, Calendar, Photos, etc.)

# Als Abhilfe: Developer Tools nutzen



A screenshot of a Mac desktop showing a web browser window and a design application.

The browser window displays the heise online homepage. A context menu is open over an image of two iPhone 12 phones. The menu items include: Zurück, Vorwärts, Neu laden, Speichern unter..., Drucken..., Streamen..., Auf Deutsch übersetzen, **Untersuchen** (highlighted with a red arrow), and Sprachausgabe.

The design application on the right shows a graphic interface with various tools and panels. It includes sections for "Freigeben", "Kommentare", "Füllereffekt", "Formkontur", and "Designideen". There are also preview windows showing a landscape image and a car advertisement for ŠKODA KAROQ.

The Mac OS Dock at the bottom contains icons for various applications like Finder, Mail, Safari, and others.

# Als Abhilfe: Developer Tools nutzen



A screenshot of a Mac desktop showing a web browser window for heise online. The browser has two tabs: "heise online - IT-News, Nachric" and "iPhone 12 und 12 Pro: Apples E...". The main content area shows two iPhone 12 phones side-by-side. The developer tools' Elements panel is open, highlighting a `div.a-layout` element. A context menu is open over this element, with several options highlighted by red arrows:

- Copy
- Cut element
- Copy element
- Paste element
- Copy outerHTML
- Copy selector
- Copy JS path
- Copy styles
- Copy XPath
- Copy full XPath

The browser's status bar at the bottom shows "div.a-layout 848 x 555.44". The Mac OS X Dock is visible at the very bottom.



# Wartemechanismen



- **Manchmal sind Elemente (noch) nicht sichtbar ... 2 Arten zu Warten:**
  - Implizit – Standardwartezeit für alle Aktionen
  - Explizit – Spezifische Wartezeit für einzelne Aktionen
- **Implicit Wait – Achtung: Legt die Standardwartezeit zwischen JEDER einzelnen Aktion fest, also eine Wartezeit von 10 Sekunden, würde die Testausführung sehr verlangsamen**  
`driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);`
- **Explicit Wait – Sind dafür gedacht, die Ausführung solange anzuhalten, bis eine spezielle Bedingung erfüllt ist**

```
// 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")));
```



## 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 |     }  
5 | }
```



# Aktionen auf Webelementen

# Selenium Aktionen auf Elementen

---



- **Ausführen von Aktionen**
  - Zunächst identifiziert man die WebElemente mit `findElement()` / `findElements()`
  - Man erhält WebElement-Objekte
  - Danach kann man mit Ihnen durch Methodenaufrufe interagieren
- **Wichtige Methoden**
  - `sendKeys()` erlaubt die Texteingabe
  - `clear()` löscht den eingegebenen Text
  - `submit()` sendet eine Form
- ***sendKeys()* ist wohl die am meisten benutzte davon**
- ***submit()* kann auf dem Button oder der Form aufgerufen werden**

# Selenium Aktionen auf Elementen

---



- **Input Box**
  - Text Fields zur Eingabe textueller Infos
  - Password Field zur Eingabe von Passwörtern mit maskierter Darstellung
  - Methode(n): `sendKeys()`, `clear()`
- **Buttons**
  - Button zum Auslösen von Aktion
  - Methode(n): `click()`
- **Submit Buttons**
  - Buttons zum Senden der Form zum Server
  - Methode(n): `click()`, `submit()`
  - Manchmal wirkt `click()` nicht wie `submit()` => siehe Google Suchbeispiel



- **Radio Button**
  - Radio Button zur Auswahl einer Option unter vielen, bei Auswahl einer Option werden die anderen gewählte ausgeschaltet
  - Methode(n): `click()`
- **Check Box**
  - Check Box zum Aktivieren / Deaktivieren einer Auswahl
  - Methode(n): `click()`
- **Drop Down (Select)**
  - Combobox (HTML Select) zur Auswahl aus einer Liste von Möglichkeiten (Multi Select supported)
  - Eigener Typ Select
    - ```
Select selectColors = new Select(driver.findElement(By.id("colors")));
```
    - Methode(n): `selectByValue()`, `selectByVisibleText()`, `selectByValue()`, `selectByIndex()`, `deselectXyz()`, `getAllSelectedOptions()`, ...

# Selenium Links

---



- Bisher haben wir eher klassische Bedienelemente betrachtet ... Was fehlt die LINKs!
- Konkrete bekannte Elemente untersuchen:

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

- Wir können die Links klicken und werden zur der entsprechenden Seite redirected:

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

- Vorab unbekannte Webelemente untersuchen, als Beispiel: Alle Links auf einer Seite finden:

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

# Selenium Statusabfragen

---



Man kann abfragen, ob WebElemente (buttons, drop boxes, checkboxes, radio buttons, labels etc.) sichtbar oder aktiviert sind:

- `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();
```

# Beispiel

---



## 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

# Beispiel



```
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

# Beispiel

---



## 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

## Beispiel

---



```
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](#)





# Selenium in Kombination mit JUnit 5



- Gestern haben wir JUnit 5 ausführlich kennengelernt
- Bisher haben wir auch eine Idee zu Selenium zum Fernsteuern von Webseiten gewonnen
- ABER: Wie bringen wir das Ganze zueinander?
- Rekapitulieren wir kurz:

Aktion	Selenium	JUnit
Einmalig zum Start	Driver als System-Property	@BeforeAll
Einmalig vor jeder Methode	Erzeugung WebDriver	@BeforeEach
Einmalig nach jeder Methode	Schließen der Fenster, Driver und Ressourcen freigeben	@AfterEach

# Selenium und 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 und 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 und 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
    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

---



- Ist ein Muster, um die Wartbarkeit und Verständlichkeit von Tests zu erhöhen
  - Beim Page-Objekt handelt es sich um eine normale Java-Klasse, die die Webelemente abstrahiert
  - Es werden Methoden angeboten, um etwa Werte in Textfeldern einzugeben, einen Button zu clicken usw.
  - Bietet eine gute Separation zwischen Testcode und Seiten-spezifischen Detail wie Locators usw.
  - Schauen wir auf eine Login-Seite als Beispiel
-

# Page Object Beispiel



```
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

```

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	<input type="button" value="▼"/>
Apple	<input type="button" value="▼"/>

# Page Object

---



- In den verwendenden Tests ist von der Komplexität und den Low-Level-Details nichts mehr zu sehen.
  - Der Testfall zeigt somit keine Selenium und andere technische Details
  - Dadurch lassen sich eine Vielzahl an UI-Änderungen ohne (größere) Auswirkungen auf die Tests ausführen
  - Man hat folgende Vorteile
    - Leichter wartbar und zuverlässiger als verstreute Locator-Zugriffe
    - Bessere Lesbarkeit und Verständlichkeit der Tests
    - Weniger Duplikat beim Zugriff auf Elemente
-

# Page Object – Einfacher 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 – Einfacher 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);

    // Alter Stil ohne Pageobject möglich, aber eher zu vermeiden
    WebElement check1 = driver.findElement(By.id("check-1"));
    check1.click();

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

# Page Object – Einfacher 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 Vorteile / Wissenswertes

---



- ✓ **Page Classes erhalten WebDriver im Konstruktor**
  - ✓ **Page Objects definieren Attribute und Methoden zum Zugriff auf Web-Elemente**
  - ✓ **Selenium-Spezifika werden durch Page Objects gekapselt**
  
  - ✓ **Tests enthalten keinen Selenium-Code**
  - ✓ **Tests nutzen Page Classes**
  - ✓ **Tests nutzen @Test-Annotation und einfache JUnit Assertions**
  - ✓ **WebDriver wird in “setUp()” erstellt**
  - ✓ **WebDriver wird in “tearDown()” freigegeben**
-



# Weitere Möglichkeiten



# 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.amazon.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;
}
```

# Screen Shot erstellen



```
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();
        }
    }
}
```

# Übungen Selenium

---



---

## Infos / Links

---



- <https://www.selenium.dev/downloads/>
  - <https://www.selenium.dev/documentation/en/webdriver/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