

# Testowanie Akceptacyjne z wykorzystaniem Selenium WebDriver

Artur Panek



# AGENDA



- Selenium WebDriver – wprowadzenie i utworzenie projektu
- Nawigacja oraz inne podstawowe operacje dostarczane przez WebDriver
- WebElement – szukanie elementów, podstawowe operacje i właściwości
- Firefox + Firebug + FirePath → lokatory z XPath i CSS selectors
- Typowe operacje na różnych kontrolkach
- PageObject pattern i PageFactory



# Selenium WebDriver - wprowadzenie

<http://www.seleniumhq.org/projects/webdriver/>

<http://www.seleniumhq.org/download/>

Language	Client Version	Release Date			
Java	2.48.2	2015-10-09	<a href="#">Download</a>	<a href="#">Change log</a>	<a href="#">Javadoc</a>
C#	2.48.0	2015-10-07	<a href="#">Download</a>	<a href="#">Change log</a>	<a href="#">API docs</a>
Ruby	2.48.0	2015-10-07	<a href="#">Download</a>	<a href="#">Change log</a>	<a href="#">API docs</a>
Python	2.48.0	2015-10-07	<a href="#">Download</a>	<a href="#">Change log</a>	<a href="#">API docs</a>
Javascript (Node)	2.47.0	2015-09-15	<a href="#">Download</a>	<a href="#">Change log</a>	<a href="#">API docs</a>

## Workflow of Selenium Webdriver



© <http://www.helloselenium.com>

1) Utwórz projekt maven'owy w IntelliJ oraz dodaj stosowne zależności.

2) Dodaj klasę TestLogin do projektu, która testuje nazwę zakładki dla strony:  
<https://owa.pgs-soft.com>

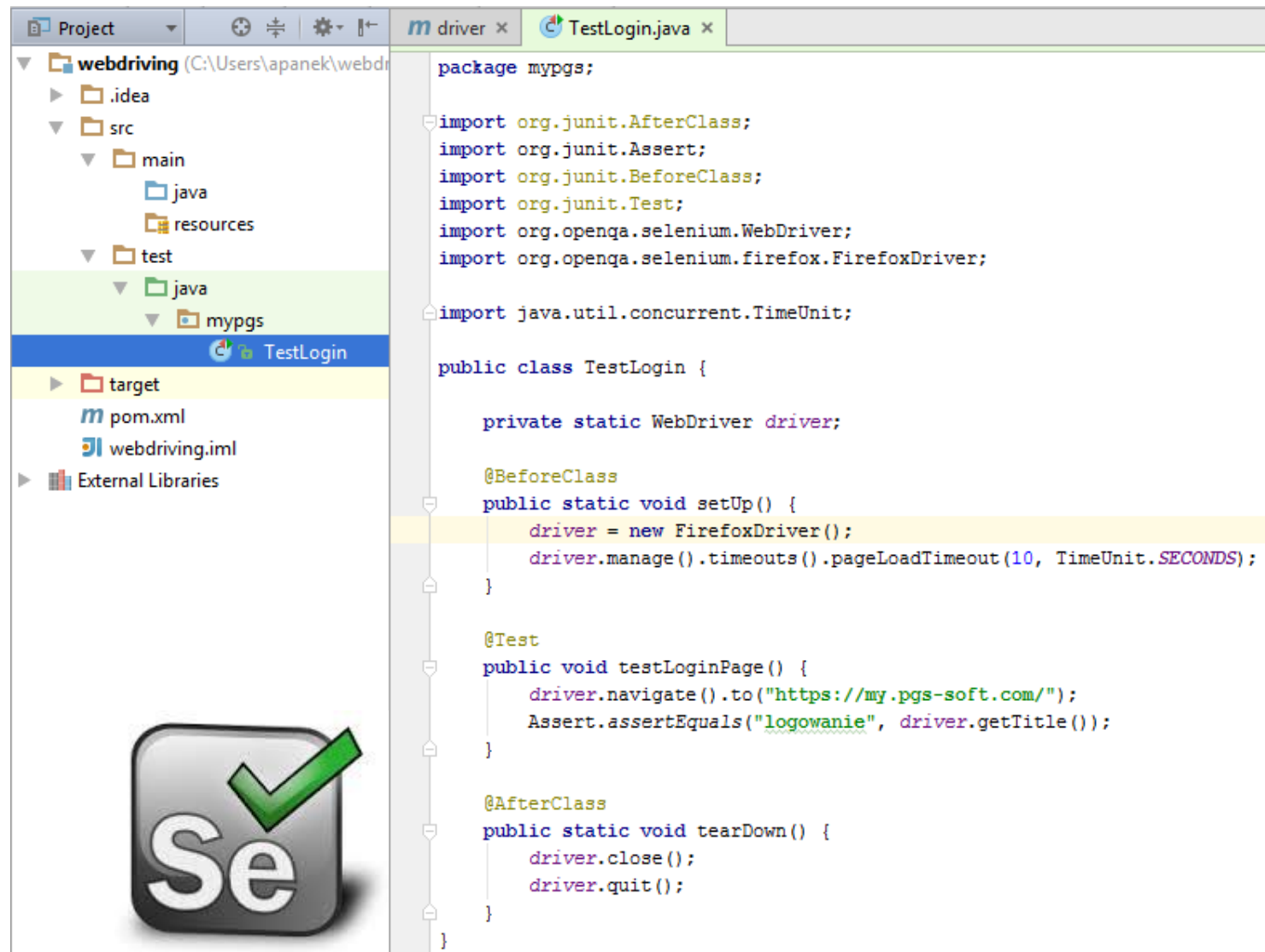
# WebDriver – pierwszy projekt

```
<properties>
  <junit.version>4.12</junit.version>
  <selenium.version>2.48.2</selenium.version>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
  </dependency>

  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-support</artifactId>
    <version>${selenium.version}</version>
  </dependency>

  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>${selenium.version}</version>
  </dependency>
</dependencies>
```



The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the directory structure of the 'webdriver' project, including 'src/main/java' and 'src/test/java'. The 'TestLogin' class is highlighted in the 'src/test/java' directory. On the right, the TestLogin.java file is open, showing the following code:

```
package mypgs;

import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

import java.util.concurrent.TimeUnit;

public class TestLogin {

    private static WebDriver driver;

    @BeforeClass
    public static void setUp() {
        driver = new FirefoxDriver();
        driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS);
    }

    @Test
    public void testLoginPage() {
        driver.navigate().to("https://my.pgs-soft.com/");
        Assert.assertEquals("logowanie", driver.getTitle());
    }

    @AfterClass
    public static void tearDown() {
        driver.close();
        driver.quit();
    }
}
```





# WebDriver – podstawowe operacje

Rozpoczynanie i kończenie pracy z driver'em:

```
driver = new FirefoxDriver();  
driver.close();  
driver.quit();
```

Nawigowanie po stronach w przeglądarce:

```
driver.navigate().to("https://my.pgs-soft.com/");  
driver.navigate().back();  
driver.navigate().forward();  
driver.navigate().refresh();
```

Poszukiwanie elementów na stronie:

```
driver.findElement();  
driver.findElements();
```

Ustawianie time out'ów, usuwanie i dodawanie ciasteczek, zarządzanie wymiarami okna przeglądarki:

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

```
driver.manage().deleteAllCookies();  
driver.manage().getCookies();  
driver.manage().addCookie();
```

```
driver.manage().window().maximize();  
driver.manage().window().getPosition();  
driver.manage().window().setPosition();  
driver.manage().window().setSize();  
driver.manage().window().getSize();
```



# WebDriver – profilowanie przeglądarki

Profilowanie przeglądarki pozwala na określenie jej zachowania w momencie otwierania plików danego typu. Dzięki temu możemy zdecydować, które pliki mają być automatycznie zapisywane lokalnie na dysku:  
*(niestety dla chrome wygląda to zupełnie inaczej)*

```
@BeforeClass
public static void setUp() {

    FirefoxProfile firefoxProfile = new FirefoxProfile();
    firefoxProfile.setPreference("browser.download.folderList", 2);
    firefoxProfile.setPreference("browser.download.manager.showWhenStarting", false);
    firefoxProfile.setPreference("browser.download.dir", System.getProperty("user.dir") + "\\\" + "downloads");
    firefoxProfile.setPreference("browser.helperApps.neverAsk.saveToDisk", "application/vnd.ms-excel,application/pdf");
    firefoxProfile.setPreference("pdfjs.disabled", true);
    firefoxProfile.setPreference("plugin.scan.Acrobat", "99.0");
    firefoxProfile.setPreference("plugin.scan.plid.all", false);

    driver = new FirefoxDriver(firefoxProfile);

    driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS);
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    driver.manage().window().maximize();
}
```



# WebDriver – profilowanie przeglądarki

W przypadku przeglądarki chrome musimy pobrać dla swojej wersji systemu WebDriver ze strony <https://sites.google.com/a/chromium.org/chromedriver/downloads> i ustawić w systemowych properties gdzie jest plik wykonywalny:

```
@BeforeClass
public static void setUp() {
    HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
    chromePrefs.put("profile.default_content_settings.popups", 0);
    chromePrefs.put("download.default_directory", System.getProperty("user.dir") + "\\\" + "downloads");
    chromePrefs.put("download.prompt_for_download", false);
    chromePrefs.put("download.directory_upgrade", true);
    chromePrefs.put("plugins.plugins_disabled", new String[]{"Chrome PDF Viewer"});

    ChromeOptions options = new ChromeOptions();
    options.setExperimentalOption("prefs", chromePrefs);
    options.addArguments("--disable-extensions");
    options.addArguments("--disable-print-preview");

    DesiredCapabilities capabilities = DesiredCapabilities.chrome();
    capabilities.setCapability(CapabilityType.ACCEPT_SSL_CERTS, true);
    capabilities.setCapability(ChromeOptions.CAPABILITY, options);

    System.setProperty("webdriver.chrome.driver", System.getProperty("user.dir") + "\\chrome\\chromedriver.exe");

    driver = new ChromeDriver(capabilities);

    driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS);
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    driver.manage().window().maximize();
}
```





# WebElement, By

Poza klasą WebDriver'a kolejnym ważnym elementem jest klasa WebElement. WebElement może być rozumiany jako konkretny element strony internetowej. Dodatkowo dany WebElement może zawierać w sobie kolejne WebElement'y.

The screenshot shows the Google Polska search page. A Selenium IDE element picker is overlaid on the search input field. The picker shows the following HTML structure:

```
<div id="sb_ifc0" class="sbib_b" dir="ltr">
  <div id="gs_lc0" style="position: relative;">
    <input id="lst_ib" class="gsfi lst-d-f" type="text" aria-label="Szukaj" value="" title="Szukaj" autocomplete="off" name="q" maxlength="2048" aria-haspopup="false" role="combobox" aria-autocomplete="both" dir="ltr" spellcheck="false" style="border: medium none; padding: 0px; margin: 0px; height: auto; position: absolute; z-index: 6; left: 0px; outline: medium none;">
```

The input field is highlighted with a blue border, and its attributes are displayed in the right-hand pane of the picker.





# WebElement, By

Poszukiwanie elementów na stronie:

```
driver.findElement();  
driver.findElements();
```

W poszukiwaniu konkretnych elementów na stronie pomaga nam klasa „By” poprzez jej metody statyczne:

```
driver.findElement(By.id("userNameBox"));
```

	<code>id (String id)</code>	By
	<code>className (String className)</code>	By
	class	
	<code>cssSelector (String selector)</code>	By
	<code>linkText (String linkText)</code>	By
	<code>name (String name)</code>	By
	<code>partialLinkText (String linkText)</code>	By
	<code>tagName (String name)</code>	By
	<code>xpath (String xpathExpression)</code>	By

Mając konkretny WebElement możemy na nim wykonać dostępne akcje:

```
@Test  
public void testLoginPage() {  
    driver.navigate().to("https://my.pgs-soft.com/");  
    Assert.assertEquals("Logowanie", driver.getTitle());  
  
    WebElement userNameInput = driver.findElement(By.id("userNameBox"));  
    userNameInput.sendKeys("login");  
    System.out.println(userNameInput.getText());  
}
```



# WebElement – podstawowe operacje

Metody dostępne dla WebElement'u:

```
userNameInput.sendKeys("login");
```

System.out	WebElement
findElement (By by)	WebElement
getText ()	String
sendKeys (CharSequence... charSequences)	void
click ()	void
clear ()	void
findElements (By by)	List<WebElement>
getAttribute (String s)	String
getCssValue (String s)	String
getLocation ()	Point
getSize ()	Dimension
getTagName ()	String
isDisplayed ()	boolean
isEnabled ()	boolean
isSelected ()	boolean
submit ()	void

W pewnych sytuacjach pobieranie tekstu za pomocą metody `getText()` nie daje oczekiwanych rezultatów. Wówczas należy posłużyć się wywołaniem:

```
//System.out.println(userNameInput.getText());  
System.out.println(userNameInput.getAttribute("value"));
```



# Ćwiczenie 1

Dokończ test `testLoginPage()` tak, aby wpisał dowolny login oraz hasło i podjął próbę zalogowania się. Następnie powinien zweryfikować pojawienie się komunikatu mówiącym o wprowadzeniu błędnych danych:

```
@Test
public void testLoginPage() {
    driver.navigate().to("https://owa.pgs-soft.com/");
    WebElement usernameInput = driver.findElement(By.id("username"));
    usernameInput.sendKeys("login");
}
```

W celu sprawdzenia czy podany komunikat istnieje wykorzystaj odpowiednio w asercji metodę:

```
public boolean isElementPresent(By by) {
    try {
        driver.findElement(by);
        return true;
    } catch (NoSuchElementException e) {
        return false;
    }
}
```



# Ćwiczenie 1 - rozwiązanie

```
@Test
public void testLoginPage() {
    driver.navigate().to("https://owa.pgs-soft.com/");
    WebElement usernameInput = driver.findElement(By.id("username"));
    usernameInput.sendKeys("login");

    WebElement passwordInput = driver.findElement(By.id("password"));
    passwordInput.sendKeys("hasło");

    WebElement signInButton = driver.findElement(By.className("signinbutton"));
    signInButton.click();

    Assert.assertTrue(isElementPresent(By.id("signInErrorDiv")));
}
```



# XPath, CSS selectors

W pewnych sytuacjach nie mamy możliwości skorzystania z takich właściwości elementów na stronie jak: „id”, „name”, „tagName”, „className” czy „linkText” lub „partialLinkText”.

W takich sytuacjach możemy sobie poradzić wykorzystując XPath oraz CSS selectors. W celu uproszczenia konstruowania odpowiednich lokatorów naszych elementów i weryfikowania ich poprawności można posłużyć się dodatkiem do przeglądarki: Firefox + Firebug + FirePath



**Firebug 2.0.13**

NIE WYMAGA PONOWNEGO URUCHOMIENIA

Autor: Joe Hewitt, Jan Odvarko, robcee, Firebug Working Group

Firebug dodaje do Firefoksa bogactwo narzędzi programistycznych. Można edytować, analizować kod oraz monitorować CSS, HTML i JavaScript bezpośrednio na dowolnej stronie internetowej...

Firebug 1.4 działa z Firefoksem 3.0 i nowszymi wersjami.

+ Zainstaluj



**FirePath 0.9.7.1.1-signed**

Autor: Pierre Tholence

FirePath is a Firebug extension that adds a development tool to edit, inspect and generate XPath 1.0 expressions, CSS 3 selectors and JQuery selectors (Sizzle selector engine).

+ Zainstaluj



# XPath



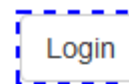
```
//*[@id='LoginForm_save']  
//button[@name='LoginForm[save]']  
//button[text()='Login']  
//div[@class='form-group']/button  
//button[contains(text(), 'ogin')]  
//button[starts-with(@id, 'Lo')]
```

<- Dostęp do przycisku „Login” jest możliwy na wiele różnych sposobów

*Należy zwracać uwagę na długie i nie optymalne ścieżki, które są generowane automatycznie*



# CSS selectors



```
error ^ v Highlight All Match Case 1 of 16 matches
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight CSS: (X) #LoginForm_save
<document>
  <html>
    <head>
    <body>
      <nav class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="col-md-12">
          <div class="col-md-offset-4 col-md-4">
            <form action="/task_6/login_check" method="post" name="LoginForm">
              <div id="LoginForm">
                <div class="form-group">
                <div class="form-group">
                  <input id="LoginForm_target_path" class="form-control" type="hidden" value="task_6" name="LoginForm[_target_
                <div class="form-group">
                  <button id="LoginForm_save" class="btn-default btn" name="LoginForm[save]" type="submit">Login</button>
                </div>
              </div>
            </form>
          </div>
        </div>
      </div>
    </body>
  </html>
</document>
```

```
div.form-group > button
div button#LoginForm_save
button[name='LoginForm[save]']
button[type='submit']
.form-group > button.btn-default
.form-group:nth-child(4) > button
```

<- Dostęp do przycisku „Login” jest możliwy na wiele różnych sposobów

Więcej informacji na temat CSS selectors na stronach:

[http://www.w3schools.com/cssref/css\\_selectors.asp](http://www.w3schools.com/cssref/css_selectors.asp)

<http://www.w3schools.com/cssref/trysel.asp>





# Checkbox i RadioButton

```
@Test
public void testCheckbox() {
    driver.navigate().to("http://codeseven.github.io/toastr/demo.html");

    WebElement closeCheckbox = driver.findElement(By.cssSelector("#closeButton"));
    if(!closeCheckbox.isSelected())
        closeCheckbox.click();

    Assert.assertTrue("Checkbox should be selected", closeCheckbox.isSelected());
}
```

```
@Test
public void testRadioButton() {
    driver.navigate().to("http://codeseven.github.io/toastr/demo.html");

    List<WebElement> toastTypeRadioButtons = driver.findElements(By.xpath("//*[@id='toastTypeGroup']//input[@type='radio']"));
    toastTypeRadioButtons.get(3).click();

    Assert.assertTrue("Radio button should be selected", toastTypeRadioButtons.get(3).isSelected());
}
```



# RadioButton – ćwiczenie 2

Zmień poniższy test tak, aby weryfikował działanie wszystkich możliwych przełączeń przycisków radiowych:

```
@Test
public void testRadioButton() {
    driver.navigate().to("http://codeseven.github.io/toastr/demo.html");

    List<WebElement> toastTypeRadioButtons = driver.findElements(By.xpath("//*[@id='toastTypeGroup']//input[@type='radio']"));
    toastTypeRadioButtons.get(3).click();

    Assert.assertTrue("Radio button should be selected", toastTypeRadioButtons.get(3).isSelected());
}
```



# RadioButton – ćwiczenie 2 - rozwiązanie

```
@Test
public void testRadioButton() {
    driver.navigate().to("http://codeseven.github.io/toastr/demo.html");

    List<WebElement> toastTypeRadioButtons = driver.findElements(By.xpath("//*[@id='toastTypeGroup']//input[@type='radio']"));
    for (WebElement radioOption : toastTypeRadioButtons) {
        radioOption.click();
        Assert.assertTrue("Radio button should be selected", radioOption.isSelected());
    }
}
```



# Dropdown Lookup

```
@Test
public void testDropdown() {
    driver.navigate().to("https://testingcup.pgs-soft.com/");
    driver.findElement(By.linkText("Zadanie 8")).click();

    WebElement typyKartyWebElement = driver.findElement(By.id("task8_form_cardType"));
    Select typyKarty = new Select(typyKartyWebElement);
    typyKarty.selectByVisibleText("Discover");

    Assert.assertEquals("Discover", typyKarty.getFirstSelectedOption().getText());
}
```



# PageObject i PageFactory

Podział stron danego serwisu na poszczególne podstrony – klasy zawierające elementy na danej stronie oraz funkcje operujące na tych elementach.

Dzięki wprowadzeniu PageFactory w selenium mamy możliwość definiowania elementów w jednym miejscu/klasie i odwołując się do nich uzyskujemy efekt taki jak w przypadku wywołania *findElement()*

Dodatkowa adnotacja `@FindBy` pozwala na analogiczne do `findElement` określanie w jaki sposób dany `WebElement` ma być poszukiwany.



# PageObject i PageFactory

```
package mypgs;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;

public class LoginLocators {

    @FindBy(how = How.ID, using = "username")
    public WebElement usernameInput;

    @FindBy(how = How.ID, using = "password")
    public WebElement passwordInput;

    @FindBy(how = How.CLASS_NAME, using = "signinbutton")
    public WebElement signInButton;

    @FindBy(how = How.ID, using = "signInErrorDiv")
    public WebElement signInError;
}
```

```
package mypgs;

import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {

    private LoginLocators locators;

    public LoginPage(WebDriver driver) {
        locators = new LoginLocators();
        PageFactory.initElements(driver, locators);
    }

    public void loginAs(String username, String password) {
        locators.usernameInput.sendKeys(username);
        locators.passwordInput.sendKeys(password);
        locators.signInButton.click();
    }

    public boolean isError() {
        try {
            locators.signInError.isDisplayed();
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

```
@Test
public void testLoginPage() {
    driver.navigate().to("https://owa.pgs-soft.com/");

    LoginPage loginPage = new LoginPage(driver);
    loginPage.loginAs("user", "haslo");

    Assert.assertTrue(loginPage.isError());
}
```





Dzięki! 😊

