Lesson 14

WEB TESTING

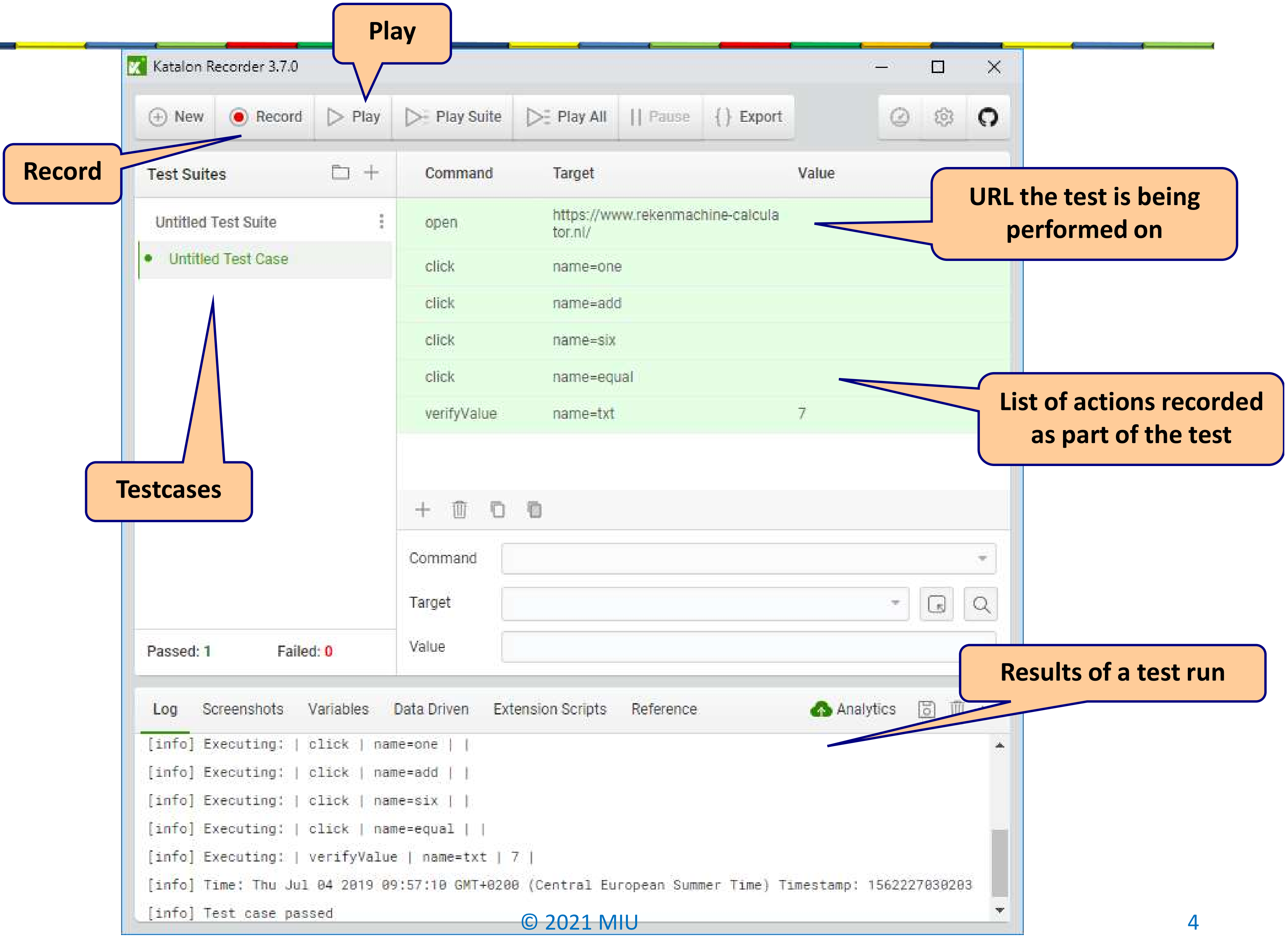
Web testing strategies

- Java coding
  - Selenium webdriver
- Record/replay
  - Katalon
  - Selenium IDE

Scripting vs. Record/Replay

- Record/replay
  - Easy to record test script
  - Test scripts are fragile and hard to maintain
  - Hard to do more complex testing (conditional logic, loops, etc)
  - Hard to reuse test scripts
  - Hard to do data driven tests
- Writing tests in code
  - Takes more time to write
  - Test scripts are easier to maintain
  - Easier to do more complex testing (conditional logic, loops, etc)
  - Easier to reuse test scripts

Katalon



Command	Target	Value
open	https://www.rekenmachine-calculator.nl/	
click	name=one	
click	name=add	
click	name=six	
click	name=equal	
verifyValue	name=txt	7

```
[info] Executing: | click | name=one | |
[info] Executing: | click | name=add | |
[info] Executing: | click | name=six | |
[info] Executing: | click | name=equal | |
[info] Executing: | verifyValue | name=txt | 7 |
[info] Time: Thu Jul 04 2019 09:57:10 GMT+0200 (Central European Summer Time) Timestamp: 1562227030203
[info] Test case passed
```

SELENIUM WEBDRIVER

Selenium webdriver

- API to write web page tests
- One API, many drivers
  - Firefox
  - Chrome
  - Internet Explorer
  - Edge

Webdriver example

```java
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

import static org.junit.Assert.assertThat;
import static org.hamcrest.CoreMatchers.*;

public class CalculatorTest {
  private WebDriver driver;

  @Test
  public void testGoogle() {
     System.setProperty("webdriver.chrome.driver", "C:\\temp\\chromedriver.exe");
     driver = new ChromeDriver();



     driver.navigate().to("http://google.com/");
     WebElement queryField=driver.findElement(By.name("q"));
     queryField.sendKeys("dogs");
     queryField.submit();
     assertThat(driver.getCurrentUrl(),containsString("dogs"));

     driver.quit();
  }
}
```

Place to find the Chrome WebDriver

Create a Chrome WebDriver

Open a URL

Find the field with name=q

Enter the text 'dog' and submit the form

Check if the URL is correct

Shut down the browser

Page navigation

```
driver.get("http://www.bol.com");
```

```
driver.navigate().to("http://www.bol.com");
```

Work the same

Identify page elements

- findElement..By pattern

```
@Test
public void verifySearch() {
  driver = new FirefoxDriver();
  driver.get("http://www.google.com");
  WebElement queryField=driver.findElement(By.name("q"));
  queryField.sendKeys("dogs");
  queryField.submit();
  assertThat(driver.getCurrentUrl(),containsString("dogs"));
  driver.quit();
}
```

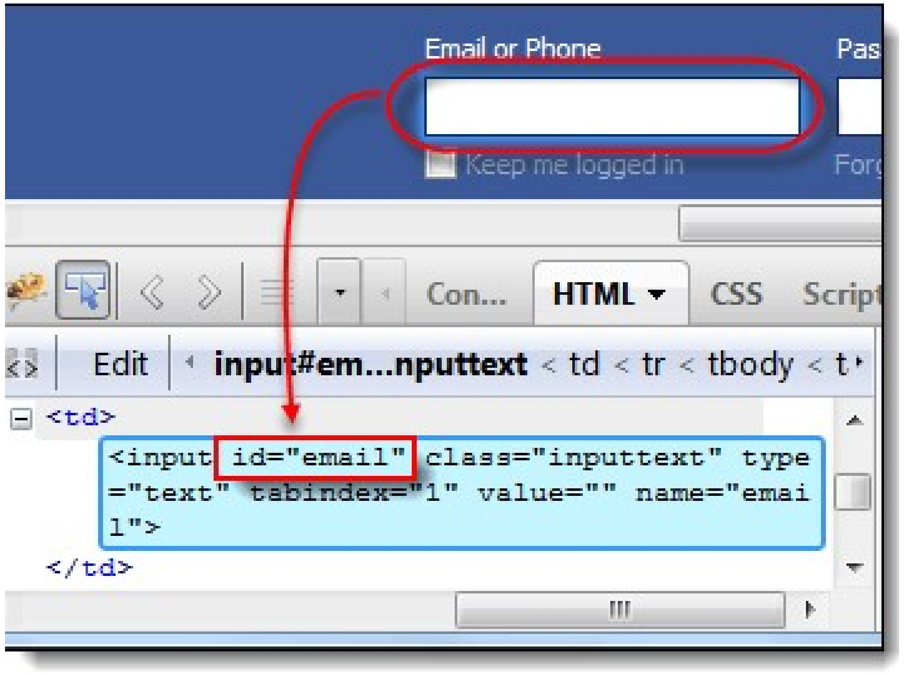
Find element with name ‘q’

ELEMENT LOCATORS

Finding elements

- By ID:
  - driver.findElement(By.id("element id"))
- By CLASS:
  - driver.findElement(By.className("element class"))
- By NAME:
  - driver.findElement(By.name("element name"))
- By TAGNAME:
  - driver.findElement(By.tagName("element html tag name"))
- By CSS Selector:
  - driver.findElement(By.cssSelector("css selector"))
- By Link:
  - driver.findElement(By.link("link text"))
- By XPath:
  - driver.findElement(By.xpath("xpath expression"))

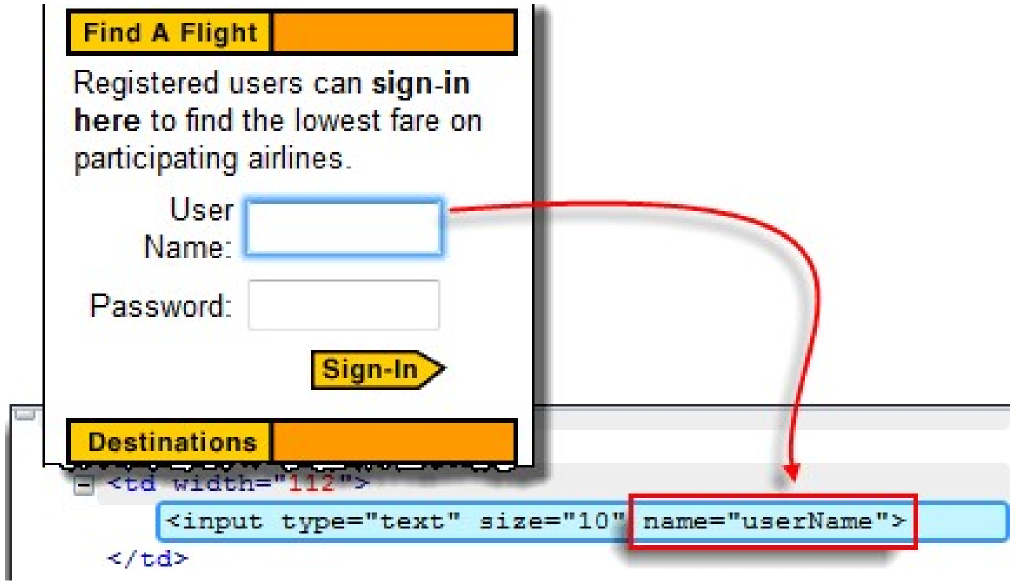
Locate by id



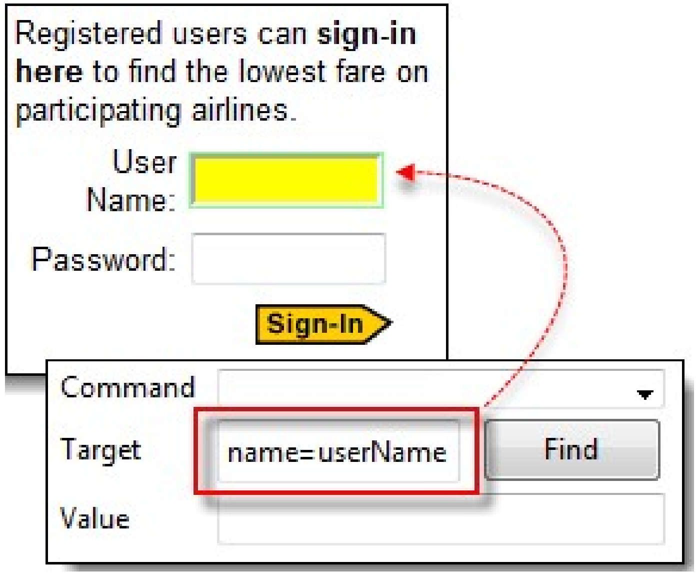
```
<td>
    <input id="email" class="inputtext" type
    ="text" tabindex="1" value="" name="emai
    l">
</td>
```

```
WebElement field = driver.findElement(By.id("email"));
```

Locate by name


Find A Flight
Registered users can sign-in here to find the lowest fare on participating airlines.
User Name:
Password:
Sign-In
Destinations

```
<td width="112">
    <input type="text" size="10" name="userName">
</td>
```


Registered users can sign-in here to find the lowest fare on participating airlines.
User Name:
Password:
Sign-In
Command
Target
Find
Value

Target: name=userName

```
WebElement field = driver.findElement(By.name("userName"));
```

Finding elements example

```html
<div>
<h2>Newsletter</h2>
   Subscribe to our weekly Newsletter and stay tuned.

<form action="" method="post" name="subscribe"><label for="name">Name: </label>
      <input class="name" id="name" type="text" placeholder="Enter name..." />
      <label for="email">Email: </label>
      <input class="email" id="email" type="text" placeholder="your@email.com" />
      <input class="btn btn-large" type="submit" value="Subscribe" />
   </form>

   <a title="first link" href="#link1">First Link</a>
   <a title="second link" href="#link2">Second Link</a>
</div>
```

```java
WebElement nameInputField = driver.findElement(By.id("name"));
WebElement nameInputField2 = driver.findElement(By.className("name"));
WebElement emailInputField = driver.findElement(By.id("email"));
```

findElements

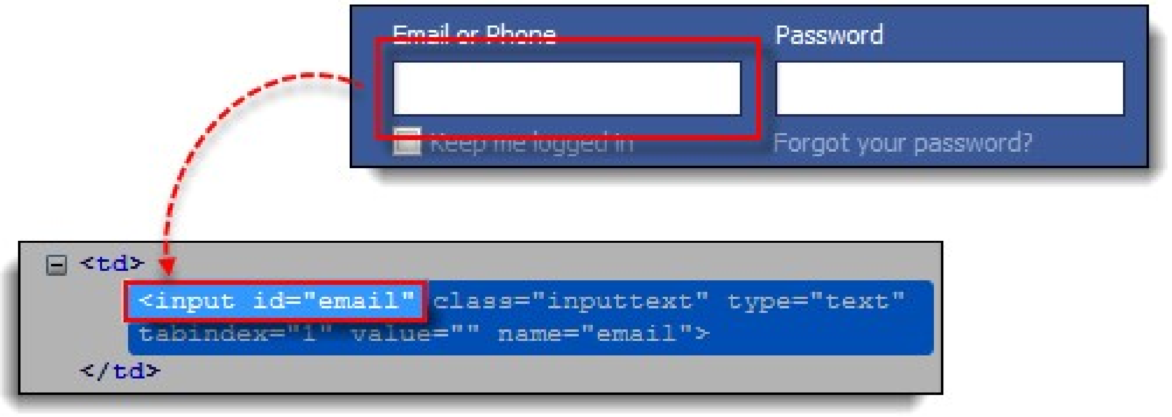
- findElement()
  - 0 matches: throws exception
  - 1 match: returns WebElement instance
  - 2+ matches: returns first element in the DOM
- findElements()
  - 0 matches: returns empty list
  - 1 match: returns list with one WebElement instance
  - 2+ matches: returns list with all matching instances

Finding elements example

```
<div>
<h2>Newsletter</h2>
   Subscribe to our weekly Newsletter and stay tuned.

<form action="" method="post" name="subscribe"><label for="name">Name: </label>
      <input class="name" id="name" type="text" placeholder="Enter name..." />
      <label for="email">Email: </label>
      <input class="email" id="email" type="text" placeholder="your@email.com" />
      <input class="btn btn-large" type="submit" value="Subscribe" />
   </form>

   <a title="first link" href="#link1">First Link</a>
   <a title="second link" href="#link2">Second Link</a>
</div>
```

```
List<WebElement> links = driver.findElements(By.tagName("a"));
assertEquals(2, links.size());
for(WebElement link : links)
  System.out.print(link.getAttribute("href"));
}
```

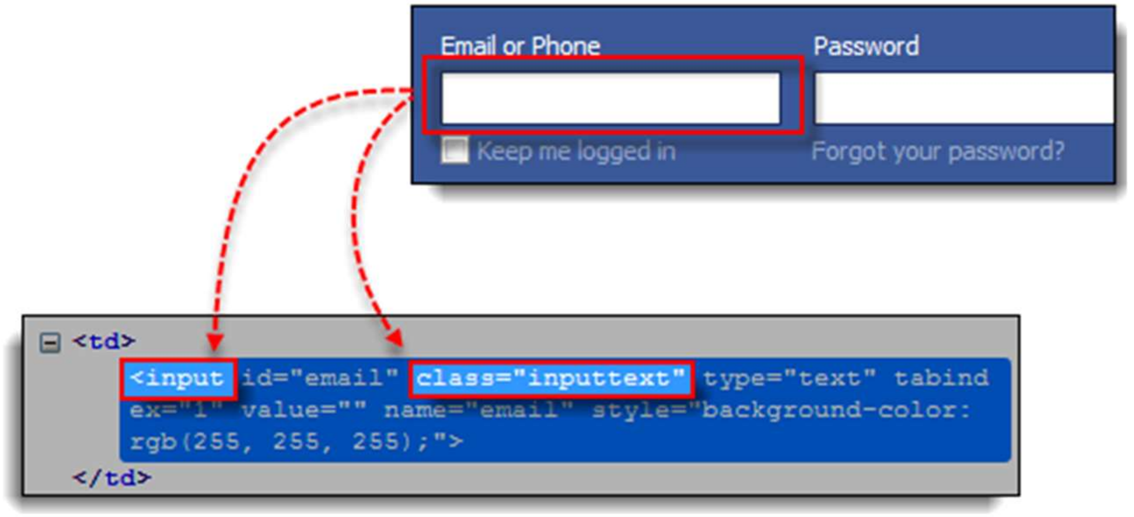
Locating by CSS Selector - Tag and ID



```
<td>
    <input id="email" class="inputtext" type="text"
    tabindex="1" value="" name="email">
</td>
```

```
//find by tag and id
WebElement fistNameField = driver.findElement(By.cssSelector("input#email"));
```

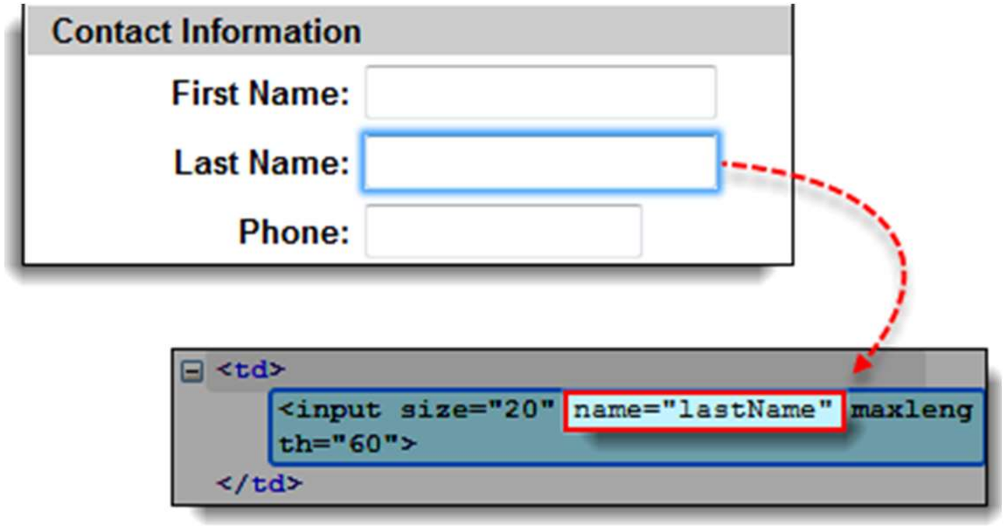
Locating by CSS Selector - Tag and Class



```
<td>
    <input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">
</td>
```

```
//find by tag and class
WebElement fistNameField = driver.findElement(By.cssSelector("input.inputtext"));
```

Locating by CSS Selector - Tag and Attribute


Contact Information

First Name:

Last Name:

Phone:

```
<td>
    <input size="20" name="lastName" maxleng
    th="60">
</td>
```

```
//find by tag and attribute
WebElement fistNameField = driver.findElement(By.cssSelector("input[name=lastName]"));
```

CSS selectors

```
<input type="text" id="firstname" name="first_name" class="myForm">
```

```
//find by tag
WebElement fistNameField = driver.findElement(By.cssSelector("input"));

//find by name
WebElement fistNameField = driver.findElement(By.cssSelector("name=first_name"));

//find by id
WebElement fistNameField = driver.findElement(By.cssSelector("#firstname"));

//find by class
WebElement fistNameField = driver.findElement(By.cssSelector(".myform"));

//find by tag and id
WebElement fistNameField = driver.findElement(By.cssSelector("input#firstname"));

//find by tag and class
WebElement fistNameField = driver.findElement(By.cssSelector("input.myform"));
```

Locating by XPath

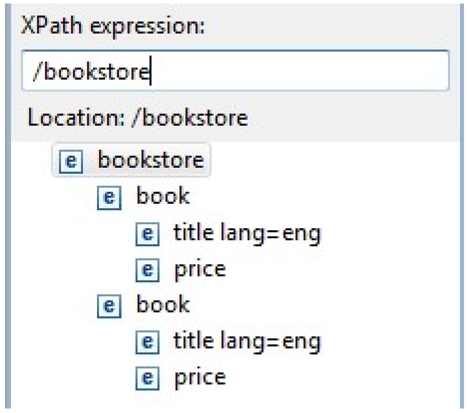
- XPath is the language used when locating XML
  - Advantage: It can access almost any element, even those without class, name, or id attributes.
  - Disadvantage: complex
- Tools can automatically generate XPath locators

XPath expressions

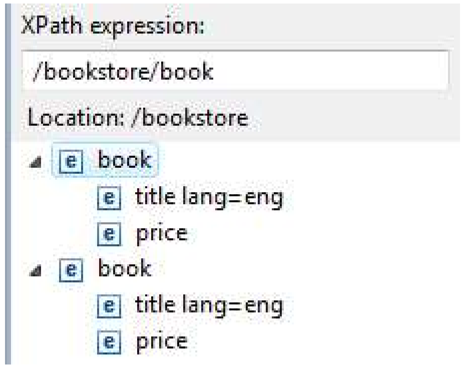
XPath expression	Result
nodename	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

XPath examples

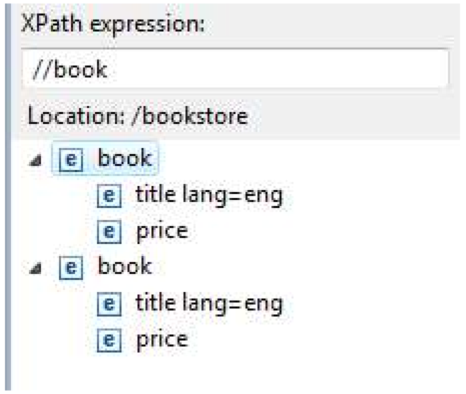
```
<bookstore>
  <book>
    <title lang="eng">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="eng">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```



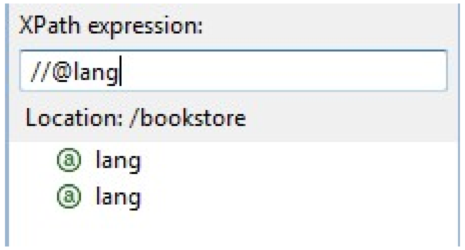
```
XPath expression:
/bookstore
Location: /bookstore
e bookstore
  e book
    e title lang=eng
    e price
  e book
    e title lang=eng
    e price
```



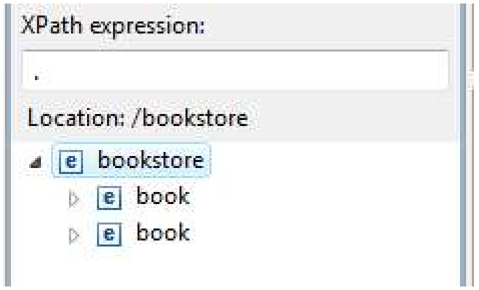
```
XPath expression:
/bookstore/book
Location: /bookstore
◢ e book
    e title lang=eng
    e price
◢ e book
    e title lang=eng
    e price
```



```
XPath expression:
//book
Location: /bookstore
◢ e book
    e title lang=eng
    e price
◢ e book
    e title lang=eng
    e price
```



```
XPath expression:
//@lang
Location: /bookstore
@ lang
@ lang
```



```
XPath expression:
.
Location: /bookstore
◢ e bookstore
  ▷ e book
  ▷ e book
```

Predicates

XPath expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element.
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang='eng']	Selects all the title elements that have an attribute named lang with a value of 'eng'
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

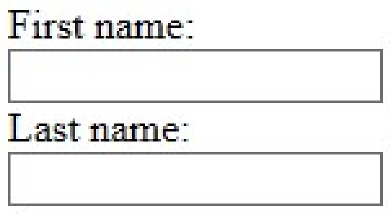
Selecting elements best practice

- Preferred selector order :
  - id > name > css > xpath
- IDs are the safest locator option and should always be your first choice
  - They are unique
  - Id remains the same if the location of the element changes
  - Do not use dynamically assigned id’s
- CSS are the way to go in conjunction of id and name
  - CSS is faster as XPath
- XPath is your last choice
  - Slow
  - Can be extremely brittle
  - Can become complex

HANDLING UI CONTROLS

Input fields

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```



```
WebElement firstname = driver.findElement(By.name("firstname"));
firstname.sendKeys("Frank");
WebElement lastname = driver.findElement(By.name("lastname"));
lastname.sendKeys("Brown");

assertThat(firstname.getAttribute("value"), is("Frank"));
assertThat(lastname.getAttribute("value"), is("Brown"));
firstname.clear();
assertThat(firstname.getAttribute("value"), is(""));
```

Enter text

Get the text

Clear the text

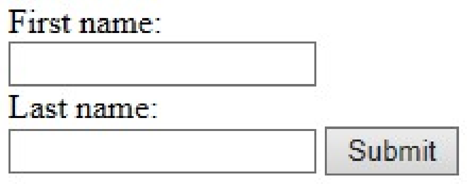
text fields

```
<b id="text">20.2</b>
```

```
WebElement text = driver.findElement(By.id("text"));

assertThat(text.getText(), is("20.2"));
```

Buttons

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
  <input id="submitbutton" type="submit" value="Submit">
</form>
```



```
WebElement firstname = driver.findElement(By.name("firstname"));
firstname.sendKeys("Frank");
WebElement lastname = driver.findElement(By.name("lastname"));
lastname.sendKeys("Brown");
WebElement button = driver.findElement(By.id("submitbutton"));

button.click();

button.submit();
```

Click a button

Submit a form

Dropdown lists and list boxes

```
<select id="mySelect">
  <option value="option1">France</option>
  <option value="option2">Italy</option>
  <option value="option3">Spain</option>
</select>
```

```
import org.openqa.selenium.support.ui.Select;


Select dropdown= new Select(driver.findElement(By.id("mySelect")));

dropdown.selectByVisibleText("Italy");

WebElement selection = dropdown.getFirstSelectedOption();
assertThat(selection.getText(), is("Italy"));
```

Get the dropdown element

Select an item

Get the selected item

Dropdown lists and list boxes

```
<select id="mySelect">
  <option value="option1">France</option>
  <option value="option2">Italy</option>
  <option value="option3">Spain</option>
</select>
```

```
dropdown.selectByVisibleText("Italy");
dropdown.selectByIndex(2);
dropdown.selectByValue("option2");

dropdown.deselectAll();
dropdown.deselectByVisibleText("Italy");
dropdown.deselectByIndex(2);
dropdown.deselectByValue("option2");

List<WebElement> options = dropdown.getOptions();
for (WebElement option : options) {
  System.out.println(option.getText());
}
```

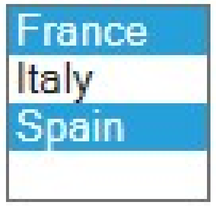
Selecting an element

Deselecting an element

Get a list of all options

Multiple select dropdown and list boxes

```
<select id="mySelect" multiple="true" >
  <option value="option1">France</option>
  <option value="option2">Italy</option>
  <option value="option3">Spain</option>
</select>
```



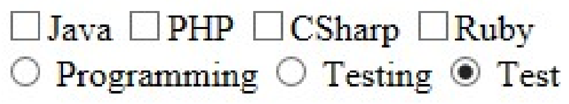
```
Select dropdown = new Select(driver.findElement(By.id("mySelect")));

dropdown.selectByVisibleText("France");
dropdown.selectByIndex(2);

List<WebElement> options = dropdown.getAllSelectedOptions();
for (WebElement option : options) {
  System.out.println(option.getText());
}
```

Select the elements

Get a list of all selected options

Radio buttons and check boxes

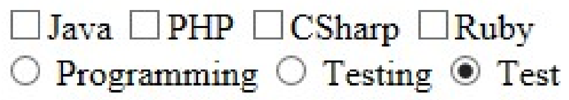
```
<input type="checkbox" name="option-1" value="Java">Java
<input type="checkbox" name="option-2" value="PHP">PHP
<input type="checkbox" name="option-3" value="CSharp">CSharp
<input type="checkbox" name="option-4" value="Ruby">Ruby
<br>
<input type="radio" name="group-1" value="Programming"> Programming
<input type="radio" name="group-1" value="Testing"> Testing
<input type="radio" name="group-1" value="Test Automation" checked> Test
```



```
WebElement  languageCheckbox = driver.findElement (By.name("option-1"));
languageCheckbox.click();
assertThat(languageCheckbox.getAttribute("value"), is("Java"));
assertTrue(languageCheckbox.isSelected());
assertTrue(languageCheckbox.isDisplayed());
assertTrue(languageCheckbox.isEnabled());

WebElement  activityRadioBtn = driver.findElement (By.xpath("//input[@value='Testing']"));
activityRadioBtn.click();
assertThat(activityRadioBtn.getAttribute("value"), is("Testing"));
assertTrue(activityRadioBtn.isSelected());
```

Radio buttons and check boxes

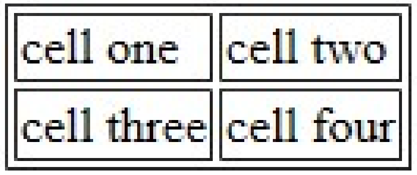
```
<input type="checkbox" name="option-1" value="Java">Java
<input type="checkbox" name="option-2" value="PHP">PHP
<input type="checkbox" name="option-3" value="CSharp">CSharp
<input type="checkbox" name="option-4" value="Ruby">Ruby
<br>
<input type="radio" name="group-1" value="Programming"> Programming
<input type="radio" name="group-1" value="Testing"> Testing
<input type="radio" name="group-1" value="Test Automation" checked> Test
```



```
List<WebElement> list = driver.findElements(By.tagName("input"));
for (int i = 0; i < list.size(); i++) {
  // Checking the check box
  if (list.get(i).getAttribute("type").trim().equalsIgnoreCase("checkbox")) {
    // Show the checkboxes
    System.out.println("CheckBox = " + i + " " + list.get(i).getAttribute("value").trim());
  }
  if (list.get(i).getAttribute("type").trim().equalsIgnoreCase("radio")) {
    // Show the radio buttons.
    System.out.println("Radio  = " + i + " " + list.get(i).getAttribute("value").trim());
  }
}
```

Tables

```
<table border="1">
  <tbody>
    <tr>
      <td>cell one</td>
      <td>cell two</td>
   </tr>
   <tr>
      <td>cell three</td>
      <td>cell four</td>
   </tr>
  </tbody>
</table>
```



cell one	cell two
cell three	cell four

```
WebElement tablefield = driver.findElement(By.xpath("//table/tbody/tr[1]/td[1]"));
assertThat(tablefield.getText(), is("cell one"));
tablefield = driver.findElement(By.xpath("//table/tbody/tr[2]/td[2]"));
assertThat(tablefield.getText(), is("cell four"));
```

Browser commands

```
driver.navigate().refresh();
driver.navigate().forward();
driver.navigate().back();
```

Move forward or backward in the browser's history

```
driver.manage().window().maximize();
```

Maximize the browser window

```
Dimension d = new Dimension(420,600);
driver.manage().window().setSize(d);
```

Set the size of the browser window

PAGE OBJECT PATTERN

Webdriver example

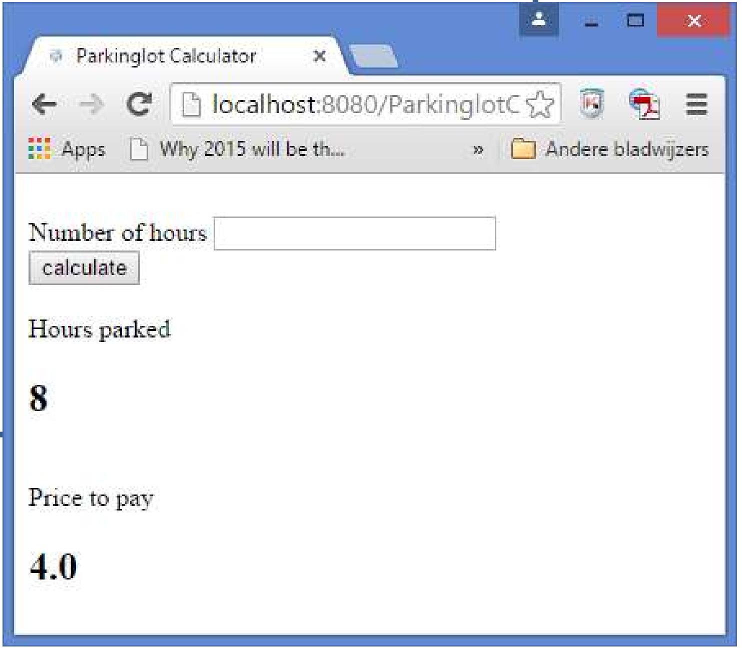
```java
public class CalculatorTest {
  private WebDriver driver;

  @Before
  public void createWebDriver() {
    System.setProperty("webdriver.chrome.driver",
                       "C:\\cucumberTraining\\drivers\\chromedriver.exe");
    driver = new ChromeDriver();
  }

  @Test
  public void verifyTitle() {
    driver.navigate().to("http://www.rekenmachine-calculator.nl/");

    WebElement button = driver.findElement(By.name("one"));
    button.click();
    button = driver.findElement(By.name("add"));
    button.click();
    button = driver.findElement(By.name("four"));
    button.click();
    assertThat(driver.findElement(By.name("txt")).getAttribute("value"),  is("1+4"));
    button = driver.findElement(By.name("equal"));
    button.click();
    assertThat(driver.findElement(By.name("txt")).getAttribute("value"),  is("5"));

    driver.close();
  }
}
```

Testing logic and page specific HTML details both together in the test class

If the page changes, most of the testcode has to change

Webdriver example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Parkinglot Calculator</title>
</head>
<body>
  <form name="hours_form" method="post" action="ParkingCalculatorServlet">
    <br>
    Number of hours  <input type="TEXT" name="hours">
    <br />
    <input id="submitbtn" type="submit" value="calculate" >
  </form>
  <br />
   Hours parked <h2>${hours }</h2>
  <br />
   Price to pay <h2>${price }</h2>
</body>
</html>
```



Hours parked

8

Price to pay

4.0

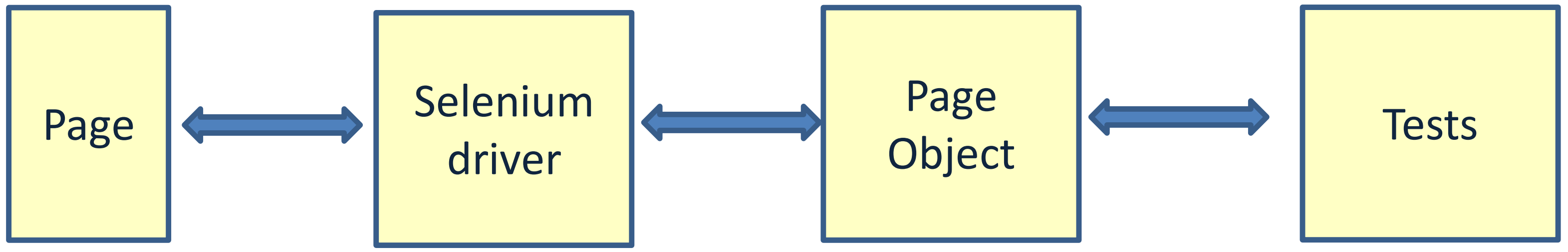
Webdriver example

Testing logic and page specific HTML details both together in the test class

If the page changes, most of the testcode has to change

```
public class ParkingLotCalculatorTest {

  private WebDriver driver;

  @Before
  public void setUp() throws Exception {
    driver = new FirefoxDriver();
    driver.get("http://localhost:8080/ParkinglotCalculatorWeb/parkinglotcalculator.jsp");
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
  }

  @Test
  public void verifySearch() {
    WebElement queryField = driver.findElement(By.xpath("/html/body/form/input[1]"));
    queryField.sendKeys("8");
    driver.findElement(By.xpath("/html/body/form/input[2]")).click();
    assertEquals("4.0", driver.findElement(By.xpath("/html/body/h2[2]")).getText());
    driver.quit();
  }
}
```

Page Object pattern



- The page object
  - Exposes methods that the user can see and do
    - addToCart(), getPrize()
- Hide the HTML details from the tests
- If the page changes, only the page object needs to change

Page Object (1/2)

```
public class CalculatorPage {
  protected WebDriver driver;

  public CalculatorPage(WebDriver driver) {
    this.driver = driver;
    PageFactory.initElements(driver, this);
  }

  @FindBy(name = "one")
  private WebElement oneButton;
  @FindBy(name = "four")
  private WebElement fourButton;
  @FindBy(name = "add")
  private WebElement addButton;
  @FindBy(name = "equal")
  private WebElement equalButton;
  @FindBy(name = "txt")
  private WebElement resultField;

...
```

Replaces @FindBy by findElement() functionality

@FindBy

Page Object (2/2)

```
public void open() {
  driver.get("http://www.rekenmachine-calculator.nl/");
}
public void close() {
  driver.close();
}
public String clickOne() {
  oneButton.click();
  return resultField.getAttribute("value");
}
public String clickFour() {
  fourButton.click();
  return resultField.getAttribute("value");
}
public String clickEqual() {
  equalButton.click();
  return resultField.getAttribute("value");
}
public String clickAdd() {
  addButton.click();
  return resultField.getAttribute("value");
}
public String getResult() {
  return resultField.getAttribute("value");
}
public void verifyCalculatorResult(String string) {
  assertThat(getResult(), is(string));
}
```

Put URL in the page object

The test

```
public class CalculatorTestWithPageObject {
  private static CalculatorPage page;

  @BeforeClass
  public static void openTheBrowser() {
    System.setProperty("webdriver.chrome.driver",
          "C:\\cucumberTraining\\drivers\\chromedriver.exe");

    WebDriver driver = new ChromeDriver();
    page = new CalculatorPage(driver);

    page.open();
  }

  @AfterClass
  public static void closeTheBrowser() {
    page.close();
  }

  @Test
  public void oneAndFour() {
    page.clickOne();
    page.clickAdd();
    page.clickFour();
    page.verifyCalculatorResult("1+4");
    page.clickEqual();
    page.verifyCalculatorResult("5");
  }
}
```

@FindBy

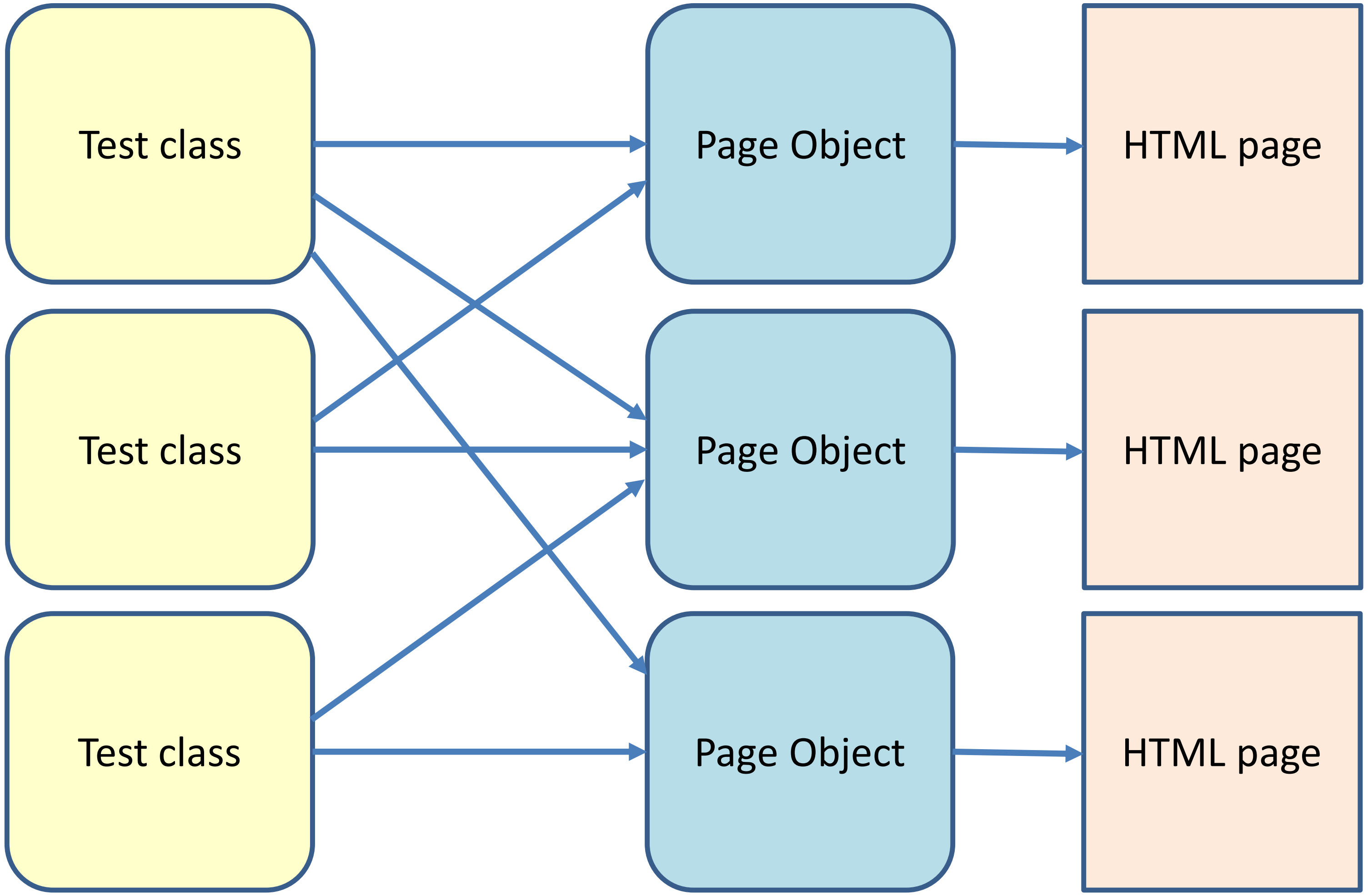
```
@FindBy(how = How.ID, using = "username")
private WebElement userName;

@FindBy(id="username")
private WebElement userName;
```

These do the same

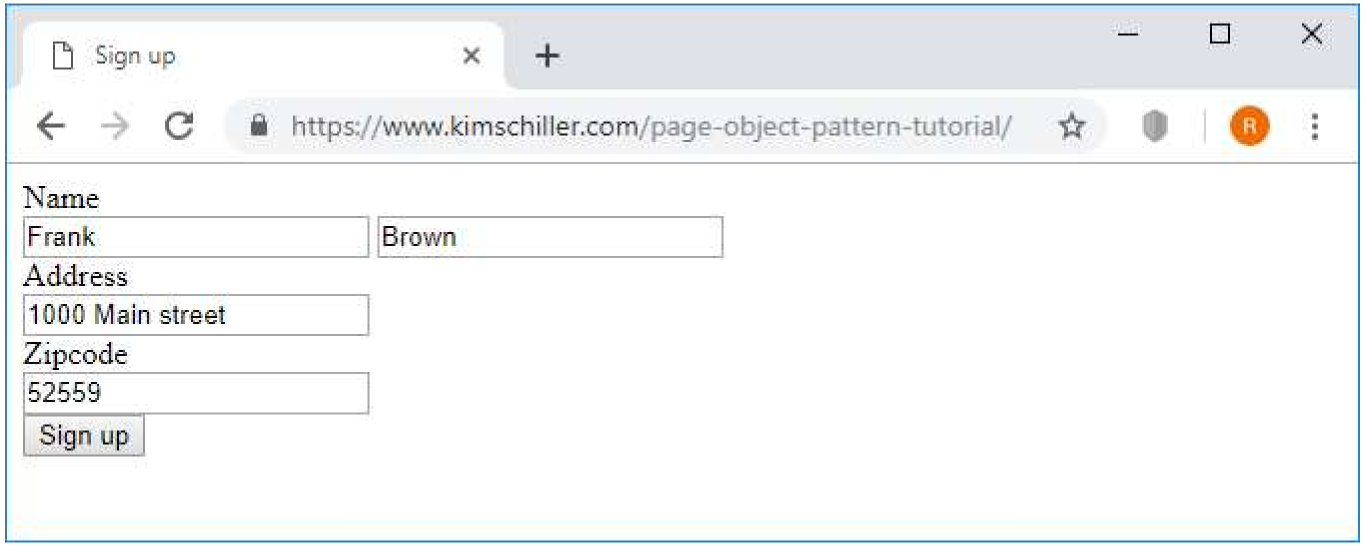
- Supported findBy strategies:
  - id
  - name
  - className
  - css
  - xpath
  - tagName
  - linkText
  - partialLinkText

PAGE OBJECT WITH NAVIGATION

Page Object pattern



Example


Sign up

https://www.kimschiller.com/page-object-pattern-tutorial/

Name

Frank | Brown

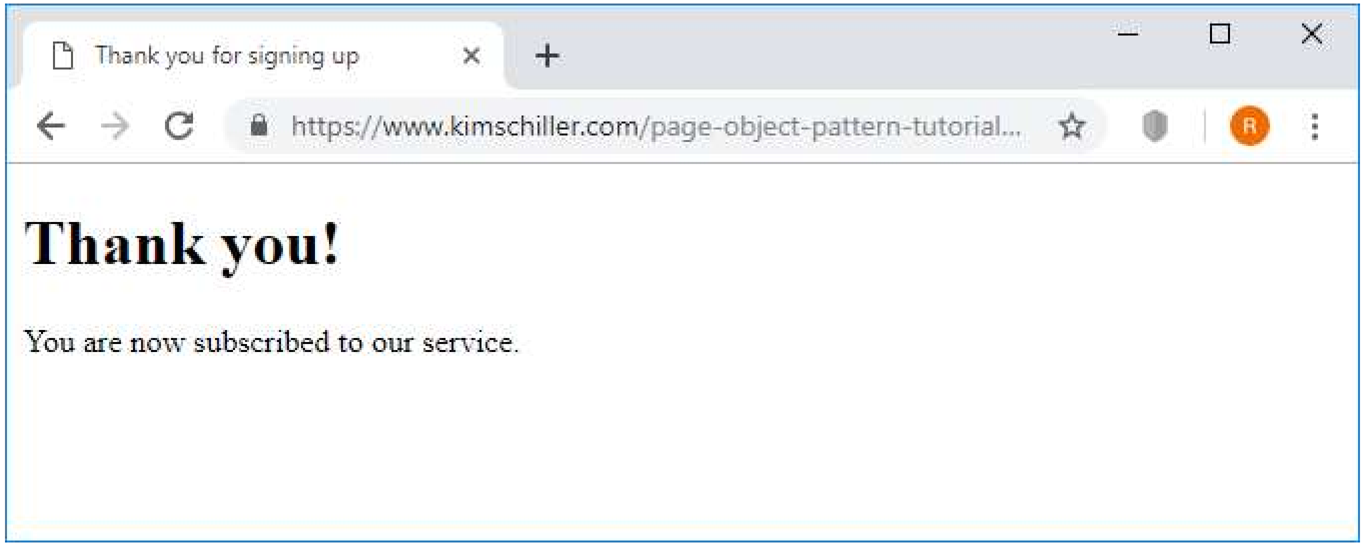
Address

1000 Main street

Zipcode

52559

Sign up


Thank you for signing up

https://www.kimschiller.com/page-object-pattern-tutorial...

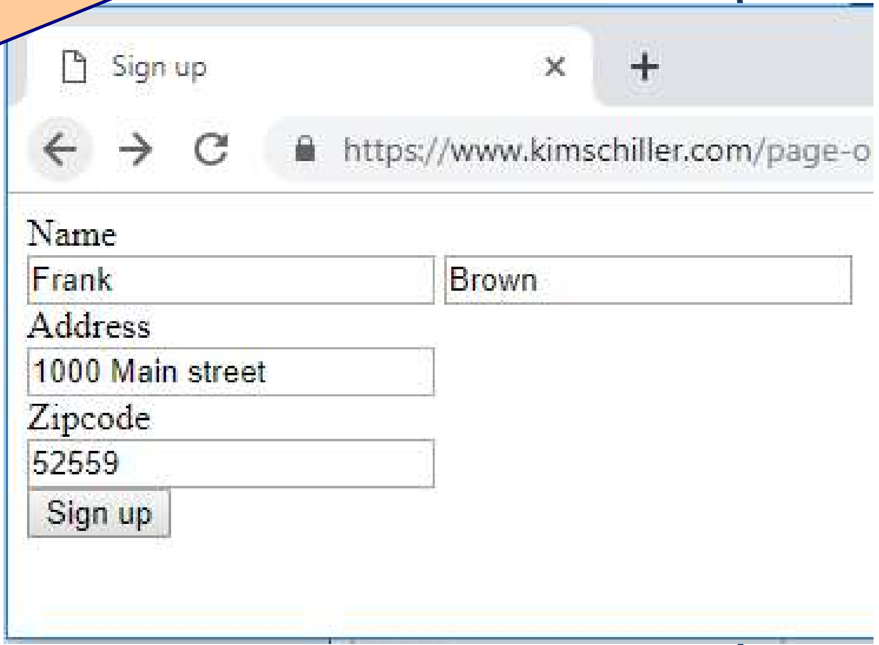
Thank you!

You are now subscribed to our service.

LoginPage

```
public class LoginPage {
  protected WebDriver driver;

  public LoginPage(WebDriver driver) {
    this.driver = driver;
    PageFactory.initElements(driver, this);
  }

  @FindBy(id = "firstname")
  private WebElement firstName;
  @FindBy(id = "lastname")
  private WebElement lastName;
  @FindBy(id = "address")
  private WebElement address;
  @FindBy(id = "zipcode")
  private WebElement zipCode;
  @FindBy(id = "signup")
  private WebElement submitButton;

  public void open() {
   driver.get ("https://www.kimschiller.com/page-object-pattern-tutorial/");
  }

  public void close() {
    driver.close();
  }
```

Initialize the LoginPage



Name

Frank | Brown

Address

1000 Main street

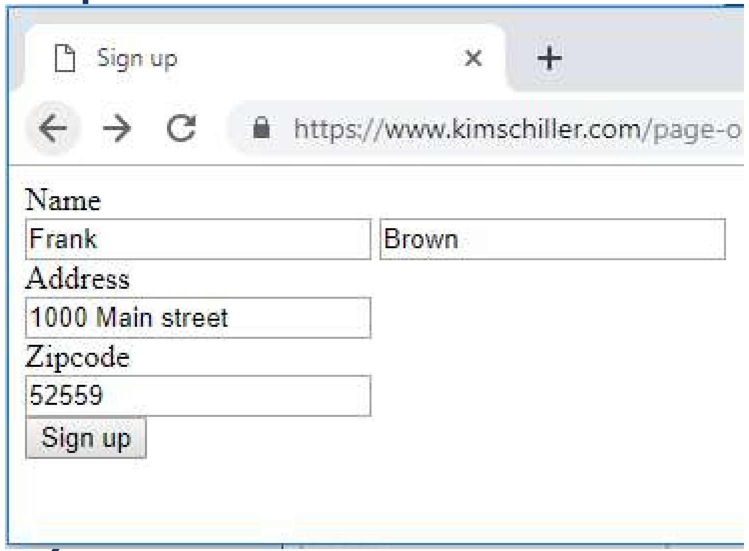
Zipcode

52559

Sign up

LoginPage

```
  public void enterName(String firstName, String lastName) {
    this.firstName.clear();
    this.firstName.sendKeys(firstName);

    this.lastName.clear();
    this.lastName.sendKeys(lastName);
  }

  public void enterAddress(String address, String zipCode) {
    this.address.clear();
    this.address.sendKeys(address);

    this.zipCode.clear();
    this.zipCode.sendKeys(zipCode);
  }

  public WelcomePage submit() {
    submitButton.click();
    return new WelcomePage(driver);
  }
}
```



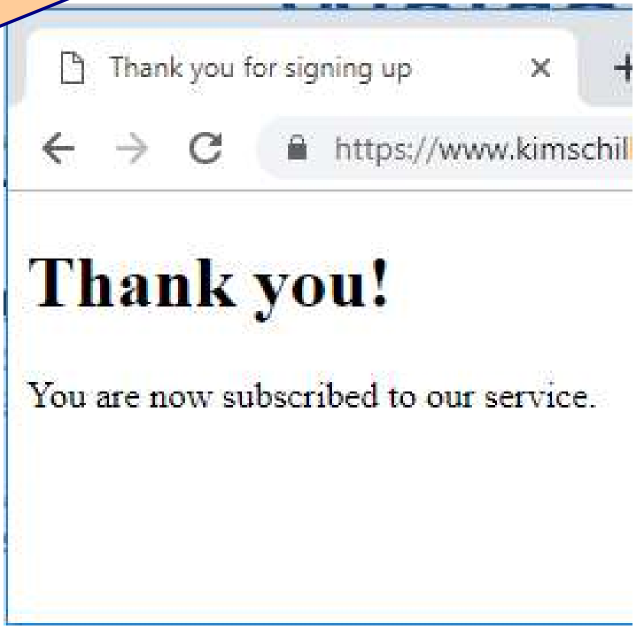
Name
Frank | Brown
Address
1000 Main street
Zipcode
52559

Return the WelcomePage

WelcomePage

```
public class WelcomePage {
  protected WebDriver driver;

  public WelcomePage(WebDriver driver) {
    this.driver = driver;
    PageFactory.initElements(driver, this);
  }

  @FindBy(tagName = "h1")
  private WebElement header;

  public String getHeader(){
    return header.getText();
  }

  public void verifyHeader(String header) {
    assertThat(getHeader(), is(header));
  }

  public void close() {
    driver.close();
  }
}
```

Initialize the WelcomePage



Thank you!

You are now subscribed to our service.

The test

```java
public class LoginTest {
  private static LoginPage loginPage;
  private static WelcomePage welcomePage;

  @BeforeClass
  public static void openTheBrowser() {
    System.setProperty("webdriver.chrome.driver",
                       "C:\\cucumberTraining\\drivers\\chromedriver.exe");
    // create chrome instance
    WebDriver driver = new ChromeDriver();
    loginPage = new LoginPage(driver);

    loginPage.open();
  }

  @AfterClass
  public static void closeTheBrowser() {
    loginPage.close();
    if (welcomePage != null)
      welcomePage.close();
  }

  @Test
  public void signUp() {
    loginPage.enterName("Frank", "Brown");
    loginPage.enterAddress("1000 Mainstreet", "52559");

    WelcomePage welcomePage = loginPage.submit();
    welcomePage.verifyHeader("Thank you!");
  }
}
```

Create an initialized LoginPage

Return an initialized WelcomePage

HEADLESS BROWSER

Headless browser

- Browser without an UI
- Advantages
  - Faster
  - You can run it on a system without browser
- Disadvantage
  - Difficult to debug your tests
- Headless browsers
  - HtmlUnit
  - PhantomJS
  - Headless Chrome

Chrome headless driver

```java
public class CalculatorTest {
  private WebDriver driver;

  @Before
  public void createWebDriver() {
    // set path to chromedriver.exe
    System.setProperty("webdriver.chrome.driver", "C:\\cucumberTraining\\drivers\\chromedriver.exe");
    // create chrome instance
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--headless");
    driver = new ChromeDriver(options);
}
```

Headless Chrome driver

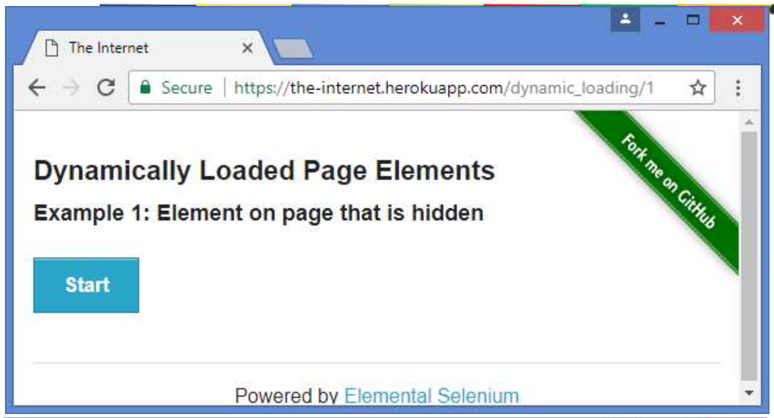
WAITING FOR AN ELEMENT TO BE PRESENT

Why waits?

- When testing a web application, you often have to wait for
  - An element to become visible on the page
  - A page to load
  - …

  Before you can proceed to your next action
- Selenium Offers 3 wait types
  - Implicit Waits
  - Explicit Waits
  - Fluent Waits

Dynamic webpage

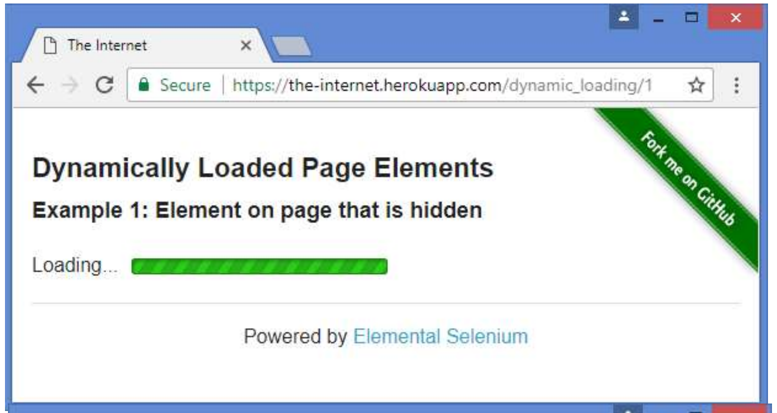

The Internet

Secure | https://the-internet.herokuapp.com/dynamic_loading/1

Dynamically Loaded Page Elements

Example 1: Element on page that is hidden

Powered by Elemental Selenium


The Internet

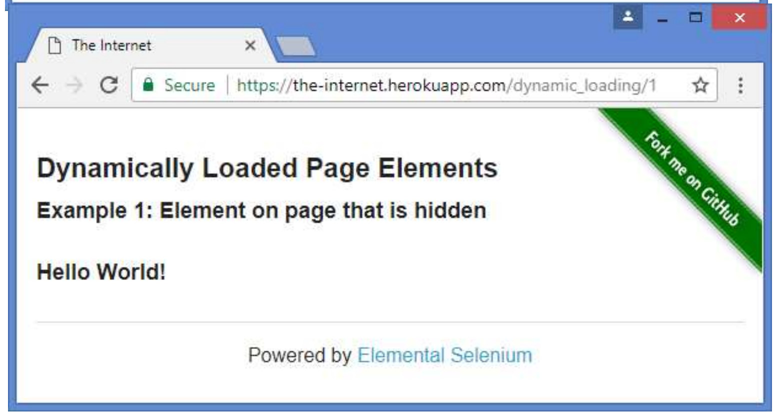
Secure | https://the-internet.herokuapp.com/dynamic_loading/1

Dynamically Loaded Page Elements

Example 1: Element on page that is hidden

Loading...

Powered by Elemental Selenium


The Internet

Secure | https://the-internet.herokuapp.com/dynamic_loading/1

Dynamically Loaded Page Elements

Example 1: Element on page that is hidden

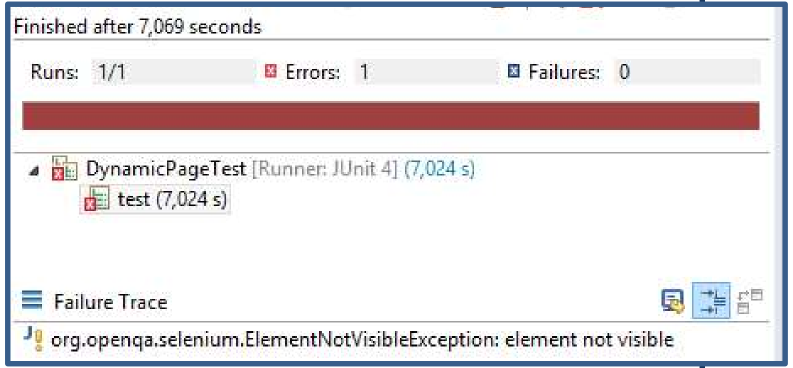
Hello World!

Powered by Elemental Selenium

```
      <script>
  $(function(){
    $('#start button').click(function(){
      $('#start').hide();
      $('#start').before("<div id='loading'>Loading... <img src='/img/ajax-loader.gif'></div>");
      setTimeout(function() {
        $('#loading').hide();
        $('#finish').show();
      } , 5000 );
    });
  });
</script>

<div class='example'>
  <h3>Dynamically Loaded Page Elements</h3>
  <h4>Example 1: Element on page that is hidden</h4>

  </br>

  <div id='start'>
    <button>Start</button>
  </div>

  <div id='finish' style='display:none'>
    <h4>Hello World!</h4>
  </div>

</div>
```

Without waiting

```
public class DynamicPageTest2 {
  WebDriver driver;

  @Before
  public void setUp() throws Exception {
    System.setProperty("webdriver.chrome.driver",
                       "C:\\cucumberTraining\\drivers\\chromedriver.exe");
    driver = new ChromeDriver();
  }

  @After
  public void tearDown() throws Exception {
    driver.quit();
  }

  @Test
  public void test() throws InterruptedException {

    driver.navigate().to("https://the-internet.herokuapp.com/dynamic_loading/1");
    //click the start button
    driver.findElement(By.tagName("button")).click();
    //find the element that has the text Hello World
    WebElement text = driver.findElement(By.xpath(".//*[contains(text(),'Hello
World!')]"));
    //click on the text
    text.click();
  }
}
```



```
Finished after 7,069 seconds
Runs: 1/1    Errors: 1    Failures: 0
DynamicPageTest [Runner: JUnit 4] (7,024 s)
  test (7,024 s)
Failure Trace
org.openqa.selenium.ElementNotVisibleException: element not visible
```

This element is not available yet

Tread.sleep()

```
public class DynamicPageTest2 {
  WebDriver driver;

  @Before
  public void setUp() throws Exception {
    System.setProperty("webdriver.chrome.driver",
                       "C:\\cucumberTraining\\drivers\\chromedriver.exe");
    driver = new ChromeDriver();
  }

  @After
  public void tearDown() throws Exception {
    driver.quit();
  }

  @Test
  public void test() throws InterruptedException {

    driver.navigate().to("https://the-internet.herokuapp.com/dynamic_loading/1");
    //click the start button
    driver.findElement(By.tagName("button")).click();
    Thread.sleep(10000);
    //find the element that has the text Hello World
    WebElement text = driver.findElement(By.xpath(".//*[contains(text(),'Hello
World!')]"));
    //click on the text
    text.click();
  }
}
```

Never use Tread.sleep()

Thread.sleep(10000) will ALWAYS wait 10 seconds

Implicit wait

- An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available.
- The default setting is 0.
- Once set, the implicit wait is set for the life of the WebDriver object instance.

```
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
```

Implicit wait

```java
public class DynamicPageTest2 {
  WebDriver driver;

  @Before
  public void setUp() throws Exception {
    System.setProperty("webdriver.chrome.driver",
                       "C:\\cucumberTraining\\drivers\\chromedriver.exe");
    driver = new ChromeDriver();
  }

  @After
  public void tearDown() throws Exception {
    driver.quit();
  }

  @Test
  public void test(){
    driver.manage().timeouts().implicitlyWait(30000, TimeUnit.MILLISECONDS);
    driver.navigate().to("https://the-internet.herokuapp.com/dynamic_loading/1");
    //click the start button
    driver.findElement(By.tagName("button")).click();
    //find the element that has the text Hello World
    WebElement text = driver.findElement(By.xpath(".//*[contains(text(),'Hello 
World!')]"));
    //click on the text
    text.click();
  }
}
```

Implicit wait

ExplicitWait

- Wait a certain maximum time until a condition becomes true.

Wait a maximum of 10 seconds

```
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement searchBoxElement = wait.until(
          ExpectedConditions.
            elementToBeClickable
               (By.id("search-box")));
```

Wait till the element with id="search-box" is clickable

- Poll the application every 500 ms
- Explicit waits happen on a per transaction basis

ExplicitWait

```java
public class DynamicPageTest2 {
  WebDriver driver;

  @Before
  public void setUp() throws Exception {
    System.setProperty("webdriver.chrome.driver",
                       "C:\\cucumberTraining\\drivers\\chromedriver.exe");
    driver = new ChromeDriver();
  }

  @After
  public void tearDown() throws Exception {
    driver.quit();
  }

  @Test
  public void test() {
    driver.navigate().to("https://the-internet.herokuapp.com/dynamic_loading/1");
    //click the start button
    driver.findElement(By.tagName("button")).click();

    By textLocator = By.xpath(".//*[contains(text(),'Hello World!')]");
    WebDriverWait wait = new WebDriverWait(driver, 10);
    WebElement text = wait.until(
              ExpectedConditions.visibilityOfElementLocated(textLocator));
    //click on the text
    text.click();
  }
}
```

Wait until the element becomes visible with a timeout of 10 seconds

ExpectedConditions

1. **static ExpectedCondition < WebElement > elementToBeClickable(By locator)**
   This condition is used to instruct a command to wait until the element is clickable by the locator.

2. **static ExpectedCondition < Boolean > elementToBeSelected(By locator)**
   This condition is used to instruct a command to wait until the element is selected by the locator.

3. **static ExpectedCondition < WebElement > presenceOfElementLocated(By locator)**
   This condition is used to instruct a command to wait until the element becomes visible or present.

4. **static ExpectedCondition < Boolean > titleContains(String title)**
   This condition is used to instruct a command to check if the title of the web element or the webpage contains the specific String or the group of characters.

5. **static ExpectedCondition < Boolean > titleIs(String title)**
   This condition is used to instruct a command to check whether the title is the String or the group of characters.

6. **static ExpectedCondition < Boolean > urlToBe(String url)**
   This condition is used to instruct a command to check if the URL of the webpage matches the expected URL.

7. **static ExpectedCondition < WebElement > visibilityOfElementLocated(By locator)**
   This condition is used to instruct a command to wait until the element becomes visible.

…

FluentWait

- Defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition.
- You can configure the wait to ignore specific types of exceptions whilst waiting
  - such as **NoSuchElementExceptions**

```
// Waiting 30 seconds for an element to be present on the page, checking
// for its presence once every 5 seconds.
 Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
   .withTimeout(30, TimeUnit.SECONDS)
   .pollingEvery(5, TimeUnit.SECONDS)
   .ignoring(NoSuchElementException.class);

 WebElement foo = wait.until(ExpectedConditions.
           elementToBeClickable
              (By.id("foo")));
```

FluentWait

```
@Test
public void test() {
  driver.navigate().to("https://the-internet.herokuapp.com/dynamic_loading/1");
  //click the start button
  driver.findElement(By.tagName("button")).click();

  By textLocator = By.xpath(".//*[contains(text(),'Hello World!')]");
  Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
          .withTimeout(20, TimeUnit.SECONDS)
          .pollingEvery(1, TimeUnit.SECONDS)
          .ignoring(NoSuchElementException.class);

  WebElement text = wait.until(
       ExpectedConditions.visibilityOfElementLocated(textLocator));
  //click on the text
  text.click();
}
```

FluentWait

Selenium synchronization

- Implicit wait
  - Works for the whole browser session
  - Only checks the presence of element, not if the element is visible or any other condition
- Explicit wait
  - Wait a certain maximum time until a condition becomes true.
  - Polling interval is 500 ms
  - Works on a per transaction basis
- Fluent wait
  - Like explicit wait, but gives finer control over
    - Polling interval
    - Exceptions to ignore

Wait best practice

- Do not use Thread.sleep()
- Do not use implicit wait
    - Use explicit wait (or fluent wait)
- Do not mix implicit and explicit wait