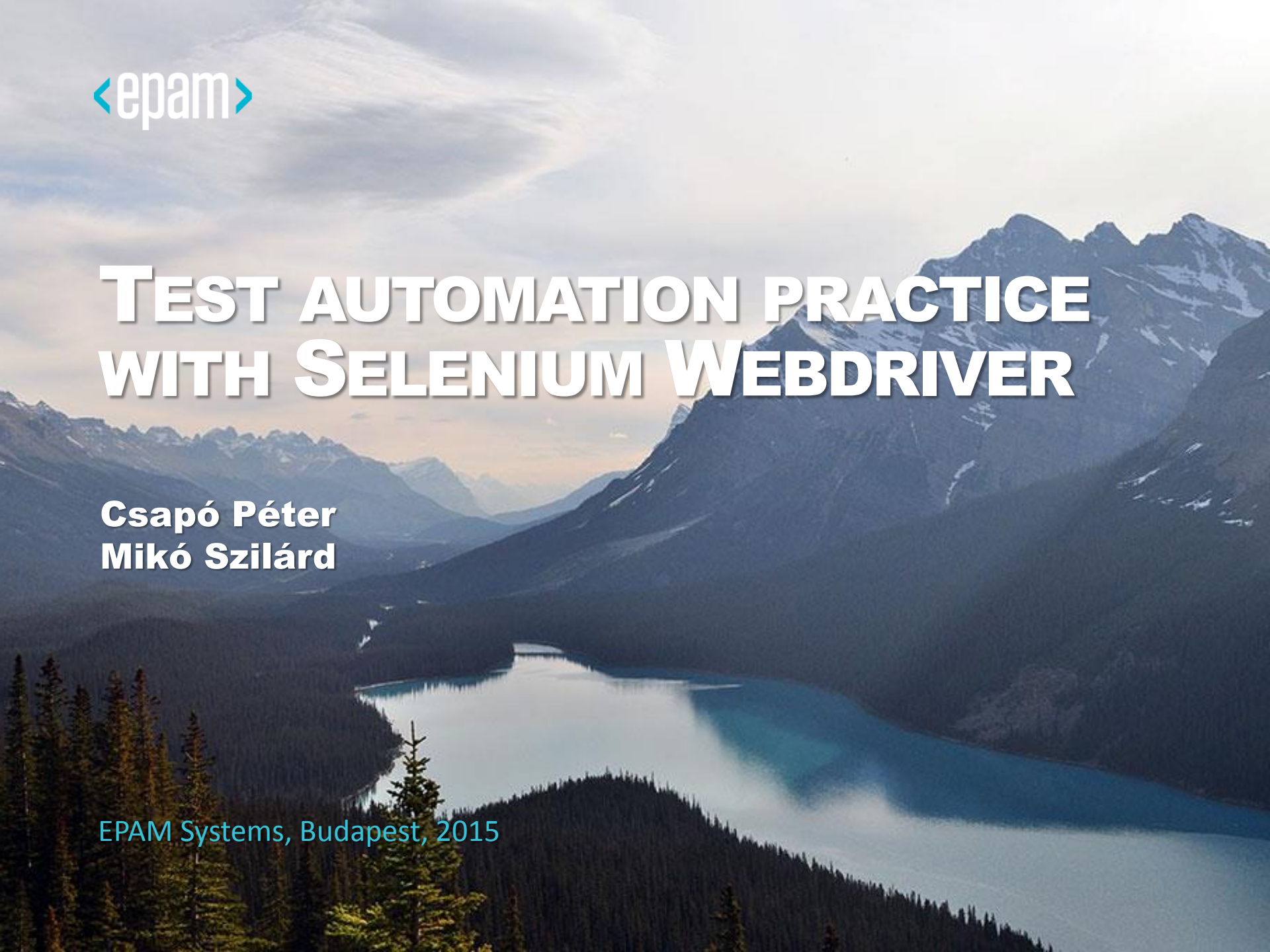


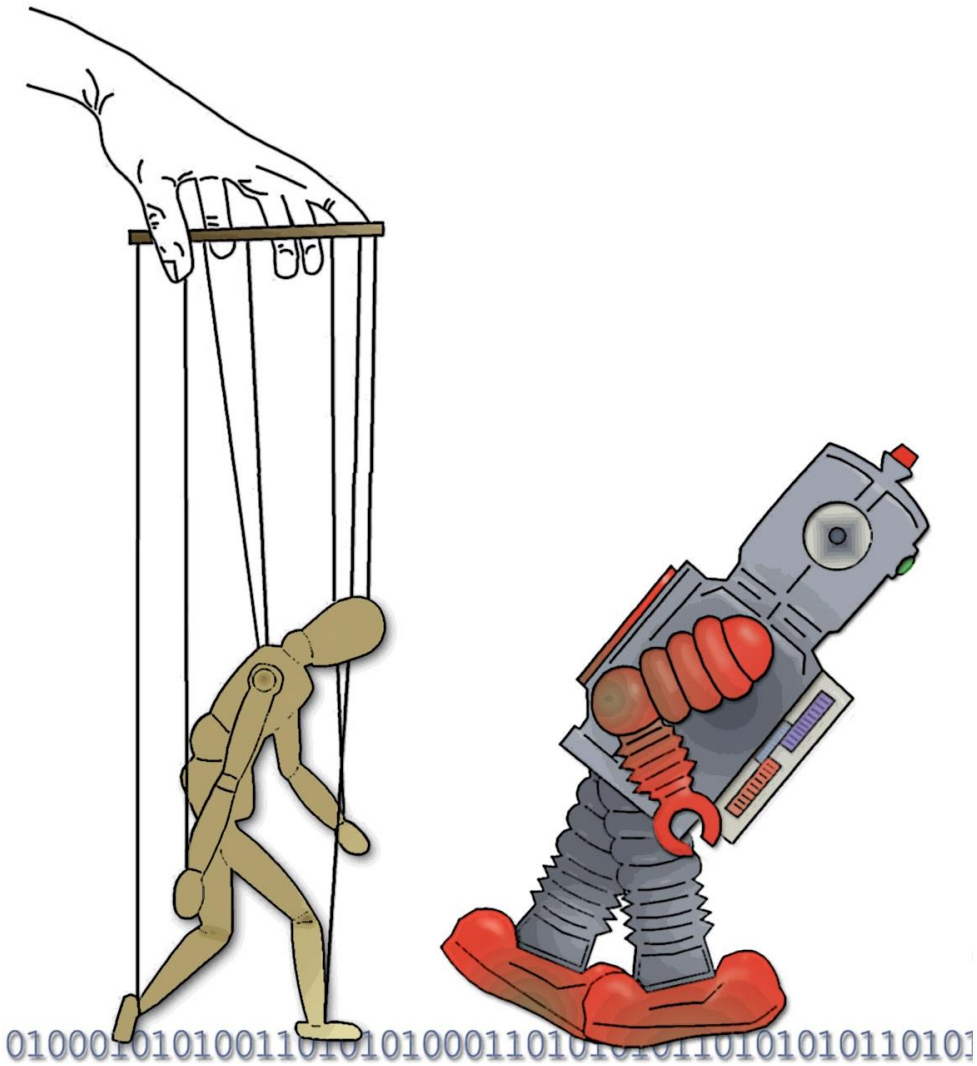
Test automation practice with Selenium Webdriver

Csapó Péter
Mikó Szilárd

EPAM Systems, Budapest, 2015

Basics Agenda

1. Assertion
2. Navigation
3. Interrogation
4. Manipulation



Basics 1

ASSERTION

Assertion

Fail method

```
fail(error_message)
```

Conditional assert

```
assertTrue(error_message, boolean_condition)
```

Equality assert

```
assertEquals(error_message, expected, actual)
```

Provide meaningful messages in assertions!

Assertion

Identity assert

```
assertSame(error_message, expected_object, actual_object)
```

Custom assert

```
assertThat(actual_object, Matcher<object> matcher)
```

String assert

```
assertThat("myString", containsString("ring"))
```

Click here for more JUnit assertions

Benefits of assertThat

Basics 2

NAVIGATION

Navigation

Loading a web page in current browser window

- driver.get(java.lang.String)
- driver.navigate().to(java.lang.String)
- driver.navigate().to(java.net.URL)

Navigation

Move back & forward

- Driver.Navigate().Back()
- Driver.Navigate().Forward()

Refresh page

- Driver.Navigate().Refresh()

Basics 3

INTERROGATION

Interrogation

Window Title

- driver.getTitle()

Current URL

- driver.getCurrentUrl()

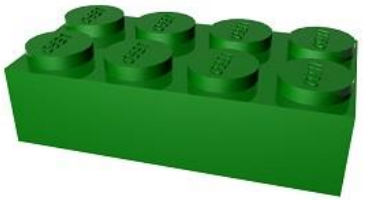
Page Source

- driver.getPageSource()

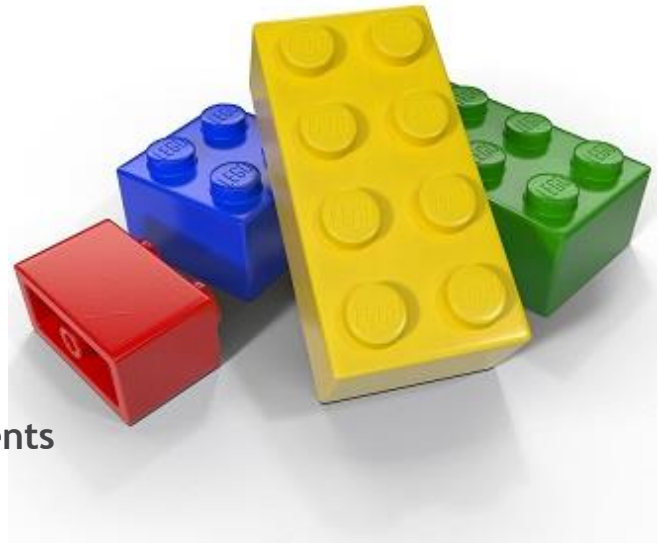
Interrogation

Locating web elements

- driver.findElement(org.openqa.selenium.By)
  - 0 match -> throws exception
  - 1 match -> returns a WebElement instance
  - 2+ matches -> returns only the first match from web page



- driver.findElements(org.openqa.selenium.By)
  - 0 match -> returns an empty list
  - 1 match -> returns a list with one WebElement
  - 2+ matches -> returns list with all matching WebElements



Interrogation

By class

- Supports various locator strategies
- By locating mechanisms
  - Id
  - ClassName
  - LinkText
  - PartialLinkText
  - Name
  - TagName
  - CssSelector
  - XPath

Interrogation

Inspecting elements in web browsers

- Firefox
  - Firebug add-on (Right click -> Inspect element / F12)
  - Firefinder add-on (Try out your CSS & Xpath expressions)
- Chrome
  - Built-in (Right click -> Inspect element / F12)
- IE
  - Built-in (Tools -> Developer Tools / F12)

Interrogation

Id

- driver.findElement(By.id(”some_id”));
- Ideal solution, however…
  - Ids don’t always exist
  - Their uniqueness is not enforced
  - Used by developers as well

ClassName

- driver.findElement(By.className(”some_class_name”));

Interrogation

Linktext

- driver.findElement(By.linkText(”Sign in”));
- driver.findElement(By.partiallinkText(”Sign”));

Name

- <input id=”modCustLoginPassword” name=”password”>
- driver.findElement(By.name(”password”));

Interrogation

tagName

- <label>Email address</label>
- driver.findElement(By.tagName("label"));

Support classes

- Return all that matches each of the locators in sequence
  - driver.findElements(new ByChained(by1, by2))
- Return all that matches any of the locators in sequence
  - driver.findElements(new ByAll(by1, by2))

Interrogation

CssSelector

- Absolute path
  - driver.findElement(By.cssSelector(”html>body>div>p>input”));
- Relative path
  - driver.findElement(By.cssSelector(”input”));
- Attribute selection
  - driver.findElement(By.cssSelector(”button[name]”));
  - driver.findElement(By.cssSelector(”button[name=,cancel’]”));
  - driver.findElement(By.cssSelector(”img:not[alt]”));

Interrogation

CssSelector

- Id selection
  - driver.findElement(By.cssSelector(”#save”));
- Class selection
  - driver.findElement(By.cssSelector(”.login”));
- Combined selection
  - driver.findElement(By.cssSelector(”button#save”));
  - driver.findElement(By.cssSelector(”input.login”));

Interrogation

CssSelector

- First matching child of the specified tag
  - driver.findElement(By.cssSelector(”div#students:first-child”));
- Nth matching child of the specified tag
  - driver.findElement(By.cssSelector(”#loginForm:nth-child(3)”));
- First matching enabled tag
  - driver.findElement(By.cssSelector(”button:enabled”));

Interrogation

XPath

- Absolute path
  - driver.findElement(By.xpath(”html/body/p/input”));
- Relative path
  - driver.findElement(By.xpath(”//input”));
- Attribute selection
  - driver.findElement(By.xpath(”//input[@id=’username’]”));
  - driver.findElement(By.xpath(”//*[@id=’myId’]”))

Interrogation

Element interrogation

- element.getText();
- element.getAttribute();
- element.getTagName();
- element.isDisplayed();
- element.isEnabled();
- element.isSelected(); -> checkbox is selected or not
- selectElement.isMultiple(); -> multi select listbox or not
- selectElement.getOptions(); -> listbox select options

Basics 4

MANIPULATION

Manipulation

Click

- element.click()
  - Button
  - Link
  - Checkbox
  - Combobox

Submit

- form.submit()
  - Form

Manipulation

Shift + Click

- Actions(driver).keyDown(Keys.SHIFT).click(element).

  keyUp(Keys.SHIFT).build().perform();

Special Actions

- Actions(driver).moveToElement(element).build().perform();
- Actions(driver).contextClick().build().perform();
- Actions(driver).doubleClick().build().perform();
- Actions(driver).clickAndHold().build().perform();
- Actions(driver).release().build().perform();

Manipulation

Type text

- element.sendKeys(”string”)
  - Input field

Clear text

- element.clear()

Manipulation

Listbox Selection

- new Select(element).selectByIndex(elementCount)

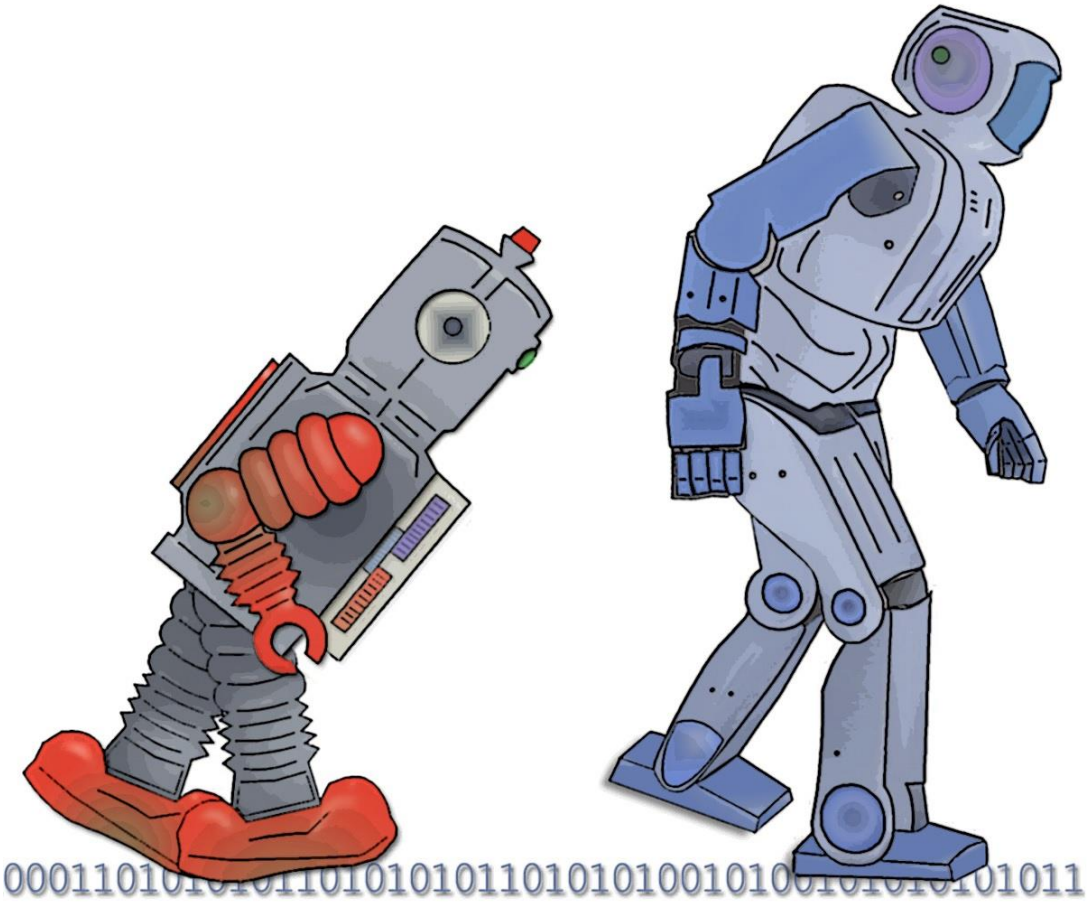
Listbox Manipulating Commands

- select[ByIndex, ByVisibleText, ByValue]
- deselect[ByIndex, ByVisibleText, ByValue]
- deselectAll()

Questions

Advanced Agenda

1. Synchronization
2. Window Handling
3. Screenshots
4. Browser Profile
5. Cookies



Advanced 1

SYNCHRONIZATION

Synchronization

Page Load Timeout

- Sets the amount of time to wait for a page load to complete
- Global setting of the Webdriver object
- Negative value means indefinite wait time

Example

- driver.manage().timeouts(). pageLoadTimeout(30, TimeUnit.SECONDS);

Synchronization

Implicit Wait

- Specifies the waiting time for element not immediately visible
- Global setting of the Webdriver object
- 0 by default

Example

- driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

Synchronization

Explicit Wait

- Waiting for a certain condition
- Poor alternative
  - Thread.sleep(1000);
- Recommended
  - WebDriverWait class

Example

- WebDriverWait wait = new WebDriverWait(driver, TIME_OUT);
- wait.until(ExpectedConditions.method);

Synchronization

ExpectedConditions class

- presenceOfElementLocated(By locator)
- textToBePresentInElement(WebElement element, java.lang.String text)
- titleContains(java.lang.String title)
- visibilityOf(WebElement element)
- invisibilityOfElementLocated(By locator)
- elementToBeSelected(WebElement element)
- elementToBeClickable(By locator)

Click here for more ExpectedConditions

Advanced 2

WINDOW HANDLING

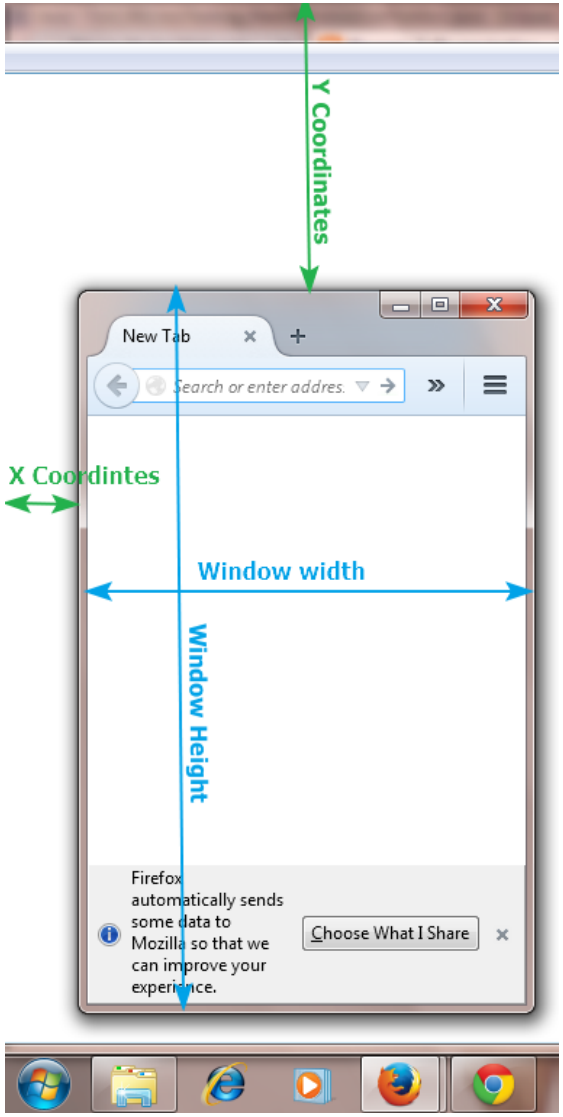
Window Handling

Size

- driver.manage().window().getSize() .getHeight(); .getWidth();
- driver.manage().window().setSize(Dimension d);
- driver.manage().window().maximize();

Position

- driver.manage().window().getPosition() .getX(); .getY();
- driver.manage().window().setPosition(Point p);



Window Handling

Handles

- String windowHandle = driver.getWindowHandle();
- Iterator<String> windowIterator = browser.getWindowHandles();

Switch To

- driver.switchTo().window(windowHandle);

Advanced 3

SCREENSHOTS

Screenshots

Advantages

- Keep track of changing UI
- Store pages with error

Example

- File screenshot =

```
((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
```

- FileUtils.copyFile(screenshot, new File(fileSource));

Advanced 4

BROWSER PROFILE

Browser Profile

Introduction

- c:\Users\[user]\AppData\Roaming\Mozilla\Firefox\Profiles\
- Unlimited number of profiles
- Stores many user attributes
  - Passwords
  - Bookmarks
  - Browser history
  - Settings
  - Etc.

Browser Profile

Usages

- Set preferred language
- Change User Agent
- Set trusted sites
- Disable confirmation dialog
- Enable Firefox extensions, e.g. Firebug and Firefinder
- Enable native events for drag-and-drop

Browser Profile

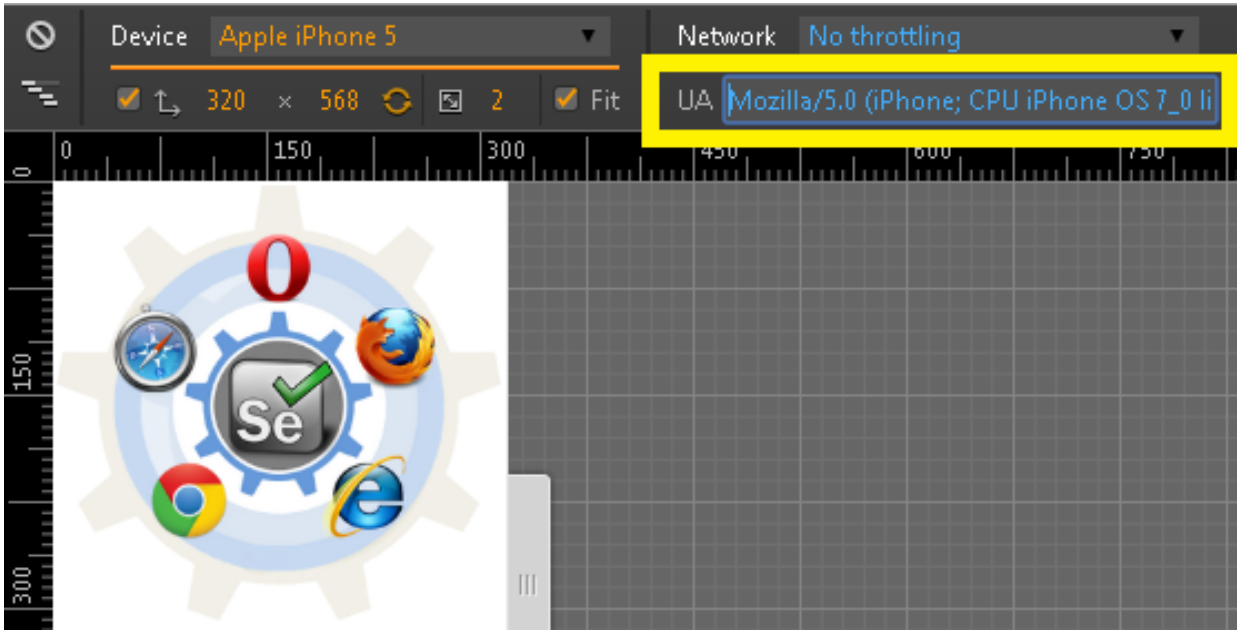
Set preferred language

- var profile= new FirefoxProfile();
- profile.setPreference("intl.accept_languages", "de");
- IWebDriver driver = newFirefoxDriver(profile);
- Search for preference keys on Firefox by about:config
- Get all special pages about:about

Browser Profile

Changing user agent

- var profile= new FirefoxProfile();
- profile.setPreference(USERAGENT_OVERRIDE, "Mozilla/5.0(iPad; U; CPU iPhone OS 3_2 like Mac OS X; en-us) AppleWebKit/531.21.10 (KHTML, like Gecko) Version/4.0.4 Mobile/7B314 Safari/531.21.10");



Browser Profile

Enable Extension

- var profile= new FirefoxProfile();
- profile.addExtension(new File(PATH_TO_FIREBUG));
- profile.setPreference("extensions.firebug.currentVersion", "2.0.12");

Advanced 5

COOKIES

Cookies

Introduction

- Useful for testing the login feature
- Getting or setting session IDs
- Cookie attributes
  - Name
  - Value
  - Domain
  - Path
  - Expiry
  - Secure
  - Http only

Cookies

Interrogation

- Get all cookies from the current session
  - driver.manage().getCookies();
- Get cookie with a given name
  - driver.manage().getCookieNamed(cookieToTest);

Cookies

Manipulation

- Delete all cookies from the current session
  - driver.manage().deleteAllCookies()
- Delete a specific cookie
  - driver.manage().deleteCookie(TestCookie);
- Delete cookie with a given name
  - driver.manage().deleteCookieNamed(cookieToTest);

Cookies

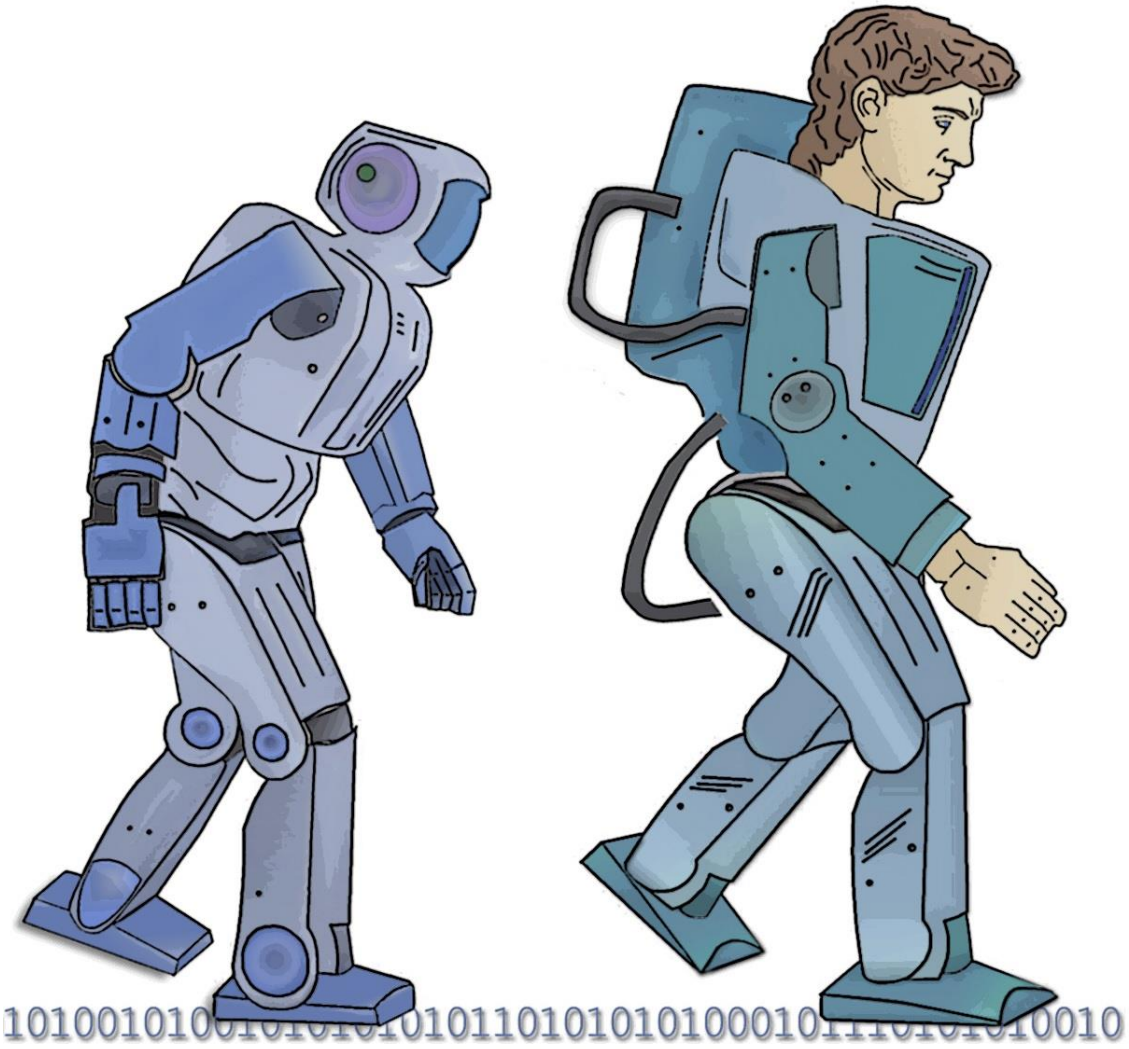
Manipulation

- Add a specific cookie
  - Cookie cookie = new Cookie("mycookie", "123456");
  - driver.manage().addCookie(cookie);
- Domain attribute is the current document by default

Questions

Test Design Agenda

1. Data Driven Testing
2. Page Object Model



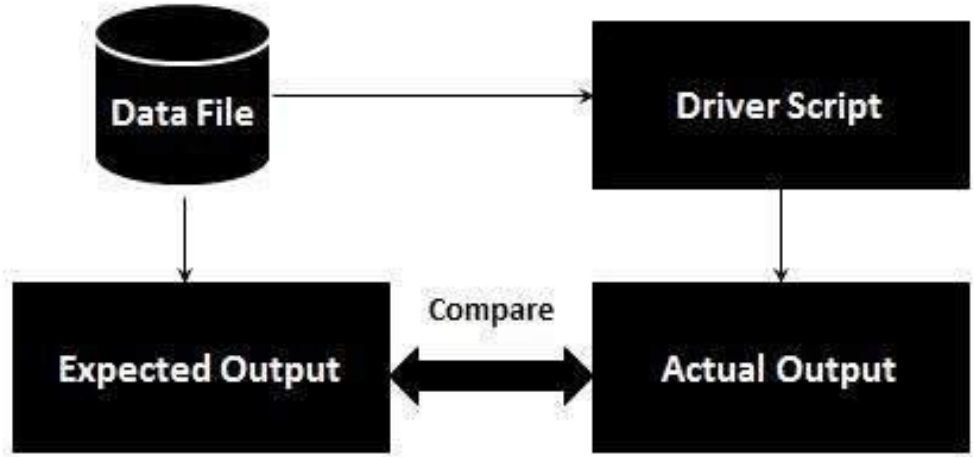
Test Design 1

DATA DRIVEN TESTING

Data Driven Testing

Concept

- Use pre-stored data as input and expected output
- Run your script to get actual output and compare them
- Continue testing with the next set of data



Data Driven Testing

Possible data sources

- Database
- XML file
- Property file
- Etc.

Pros of Data Driven Testing

- Repeatability and reusability
- Separation of test code and data
- Reduction in number of tests

Data Driven Testing

Where to use

- Testing different localizations of a site
  - <testData lang="en" phone="(+36-1) 444 44 99" />
  - <testData lang="hu" phone="06 (40) 49 49 49" />

Data Driven Testing

How to use

- Use JUnitParamsRunner class
  - @RunWith(JUnitParamsRunner.class)
  - public class Testclass { … }
- Add test parameters
  - @Parameters(method = "testData")
  - public void testCase(String param1, String param2) { … }

Test Design 2

PAGE OBJECT MODEL

Page Object Model

Agenda

- New Approach
- @FindBy annotation
- PageFactory class
- Page Flow
- Best practices

Page Object Model

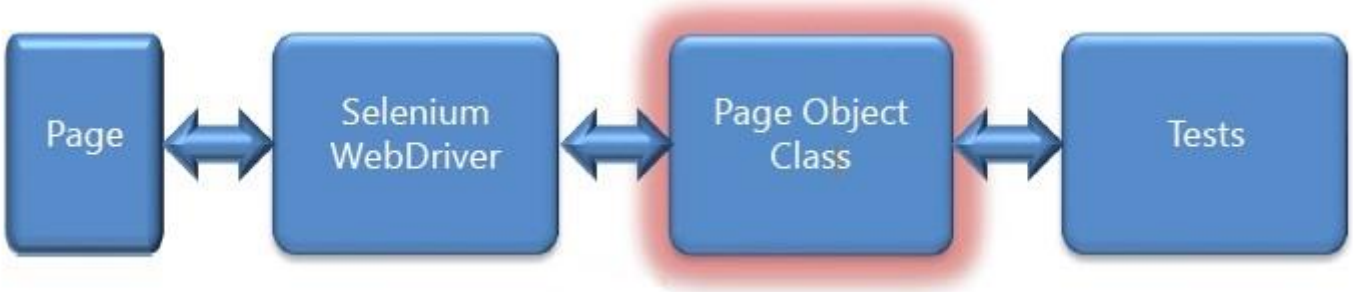
New Approach

- Every UI change could cause a lot of test maintenance work
- We have to keep maintenance changes as low as possible
- We minimize the change sensitive automation code

Page Object Model

How to do it

- We accept that the application and its elements are bound to change
- We create a new layer which contains change prone data
- This way we can adapt to changes with minimal refactoring cost



Page Object Model

Rules

- What describes one page should be in one class
- Use class methods to interact with your page
- This class represents your page in the test code
- Divide complex pages into smaller components called widgets
- Create widgets with the same approach as pages

Page Object Model

@FindBy Annotation

- We mark elements with the @FindBy annotation
- FindBy directs Webdriver to locate an element

1. @FindBy(how = How.ID, using = "i")

   public WebElement routeFrom;

2. @FindBy(id = "i")

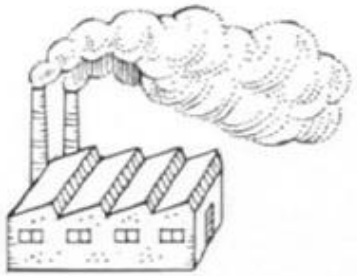
   public WebElement routeFrom;

3. public WebElement i;

Page Object Model

PageFactory Class

- You can instantiate page/widget WebElements using the PageFactory class
- Use static method initElements
- WebElements are evaluated lazily



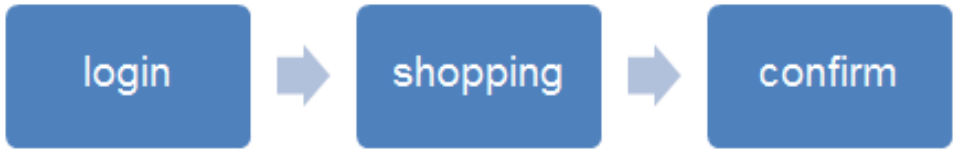
Example

- PageFactory.initElements(WebDriver driver, java.lang.Class PageObjectClass);
- initElements returns PageObjectClass

Page Object Model

Page Flow

- We want to describe the logical relationship between pages
- Manipulation methods should reflect these relationships
- We should return a page object class after each method
- Return itself if we stay at the same page (e.g. typing)
- Return next page if we navigate (e.g. submit)



Page Object Model

Best practices

- Create base pages to represent common parts
  - E.g. same header, footer, sidebar
- Reuse common widgets in each affected pages
- Use your pages to initiate and confirm the success of navigation
- Put your verification methods inside the page object as well

Questions

Cucumber agenda

1. Understanding Cucumber
2. Syntax
3. Best practices

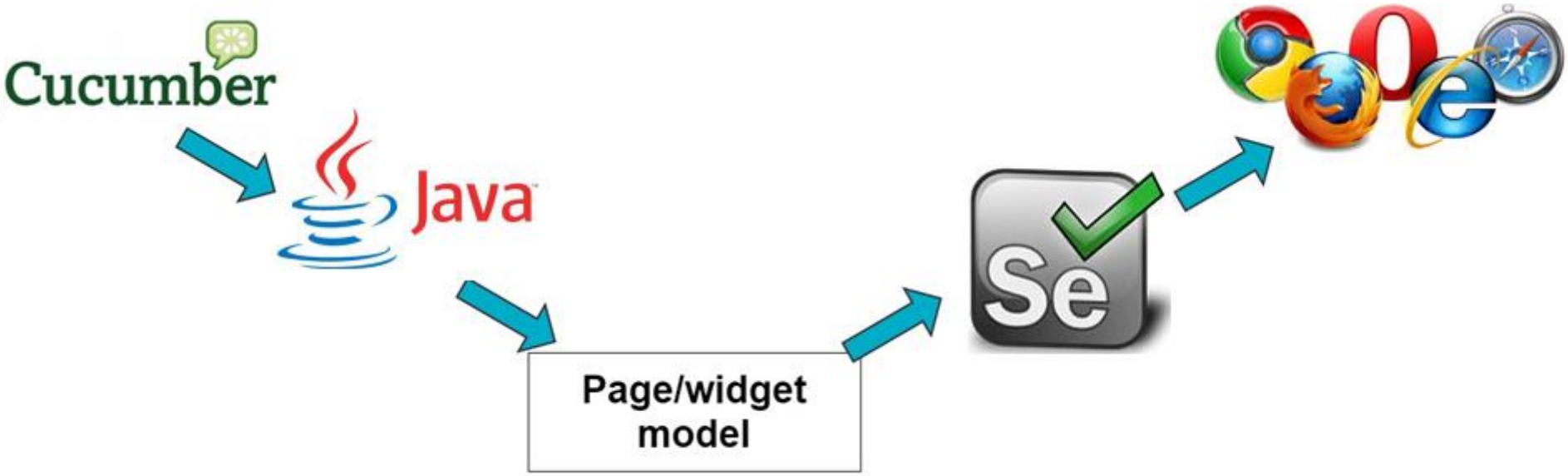
Cucumber 1

UNDERSTANDING CUCUMBER

BDD Quick Recap

- BDD is TDD done right
- Encourages communication between business, QA and dev teams
- Driven by business value
- Extends TDD by using natural language understandable for non technical people (Given When Then)
- Test cases can even be created by product owners, business analysts, TPMs.
- Cucumber is the most well known BDD framework.

Used technologies



Pros and Cons

Advantages

- Provides a form of documentation (feature file)
- Focus on functionality, operation and behavior
- Test cases are understandable for non-tech stakeholders – common language with business
- New tests are easy to create by reusing exiting steps
- Plays nicely with TDD, BDD

Disadvantages

- More layers for testers
- More time consuming

Cucumber 2

SYNTAX

Feature Files

Feature: A feature would describe the current test script which has to be executed.

Scenario: Scenario describes the steps and expected outcome for a particular test case.
Scenario Outline: Same scenario can be executed for multiple sets of data using scenario outline. The data is provided by a tabular structure separated by (| |).

Given: It specifies the context of the text to be executed.
And: use this keyword when multiple steps are of the same type are required
When: "When" specifies the test action that has to performed
Then: The expected outcome of the test can be represented by "Then"
But: another way to start your step

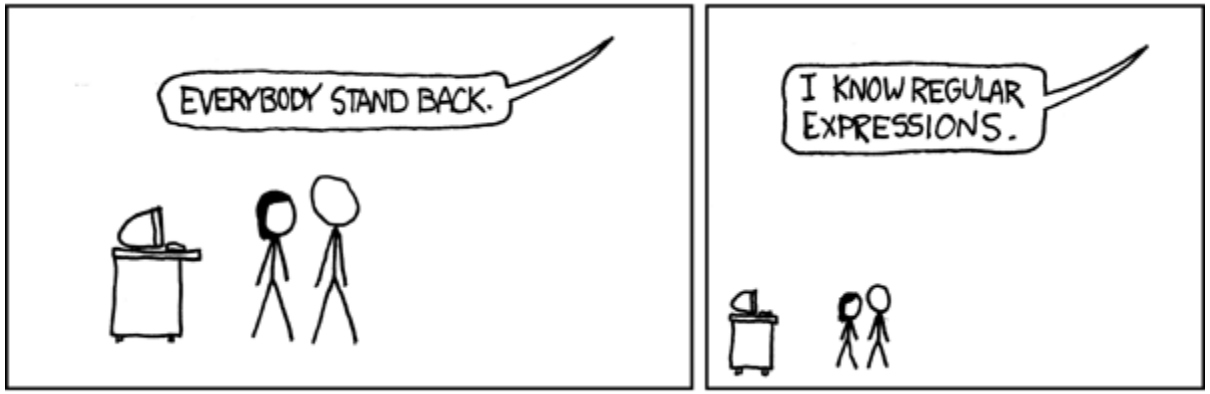
Step Definitions

In the feature file:

```
Given the following animals: cow, horse, sheep
```

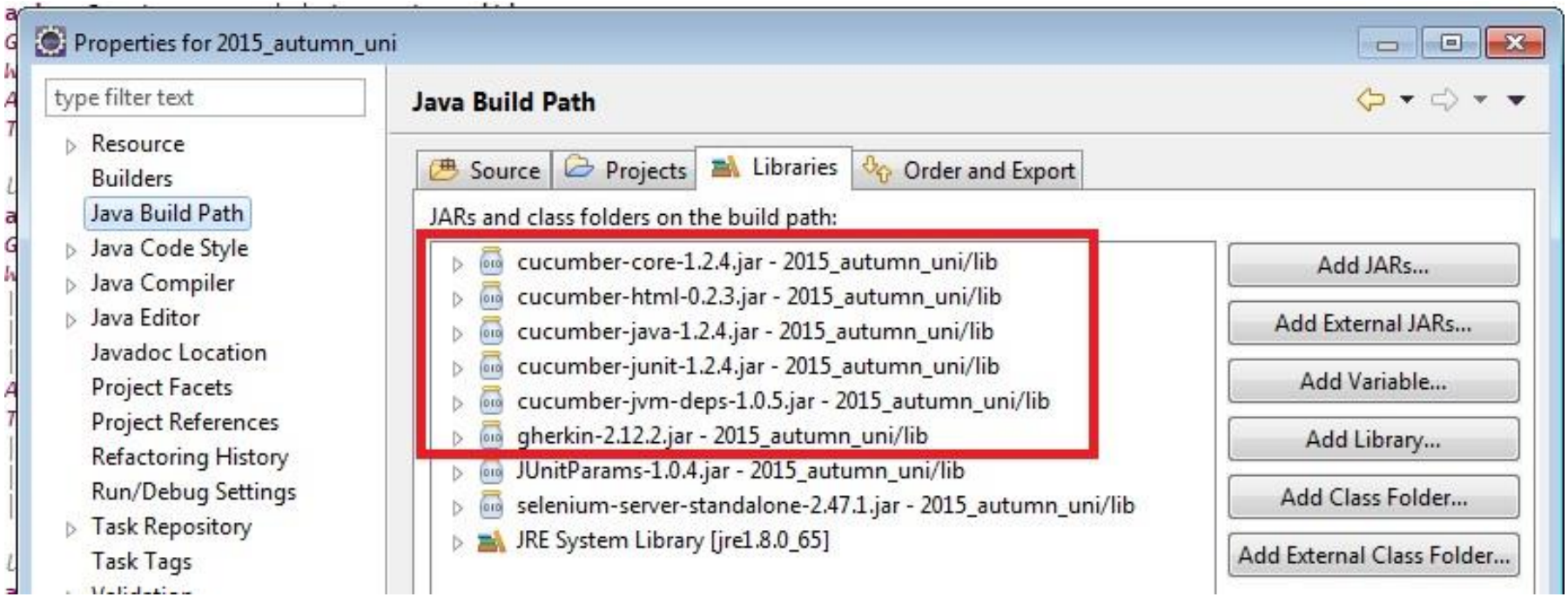
Translates to the following code:

```
@Given("the following animals: (.*)")
public void the_following_animals(List<String> animals) {
    //do something terrible
}
```

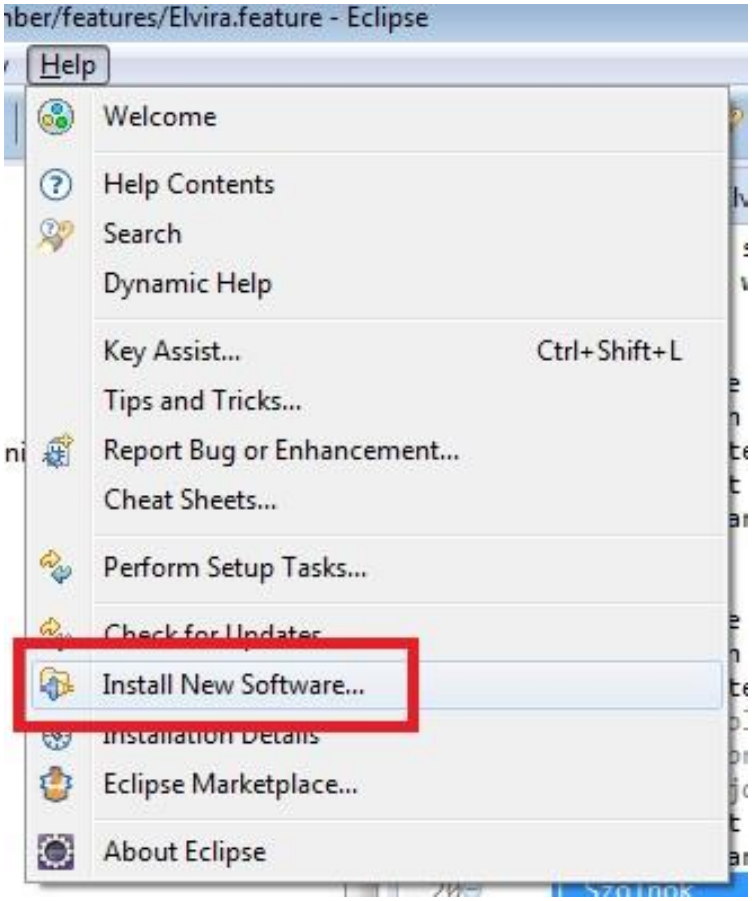


Cucumber in Eclipse

Add JARs to your project



Cucumber in Eclipse



Help -> Install New Software

Work with: Cucumber

Basic scenario

```
@basic
Scenario: Create a search between two cities
    Given I open Elvira page
    When I create a search from
        "Székesfehérvár" to "Sopron"
    And I submit the search from
    Then the search result title should
        contain "Székesfehérvár" and "Sopron"
```

Tables scenario

```
Scenario: Create a search between two cities using tables
    Given I open Elvira page
    When I create a search with the following parameters
    | from | Szolnok         |
    | to   | Debrecen        |
    | via  | Hajdúszoboszló |
    And I submit the search from
    Then the search result title should contain the following city names
    | Szolnok         |
    | Debrecen        |
    | Hajdúszoboszló |
```

Advanced tables scenario

```
Scenario Outline: Create an advanced search between two cities
    Given I open Elvira page
    When I create a search with the following parameters
    | from | <fromCity> |
    | to   | <toCity>   |
    | via  | <viaCity>  |
    And I submit the search from
    Then the search result title should contain the following city names
    | <fromCity> |
    | <toCity>   |
    | <viaCity>  |
@tag1
Examples:
    | fromCity | toCity   | viaCity         |
    | Szolnok  | Debrecen | Hajdúszoboszló |
@tag2
Examples:
    | fromCity | toCity   | viacity |
    | Budapest | Sopron   | Tata    |
```

Cucumber 3

BEST PRACTICES

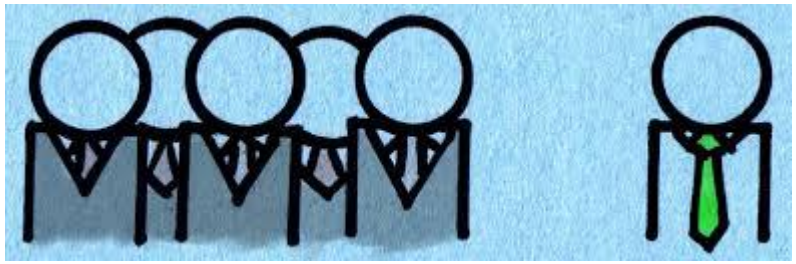
Cucumber tips 1

1. Keep a feature file feature specific

2. Use tags (dependencies, test levels, environments)

Cucumber tips 2

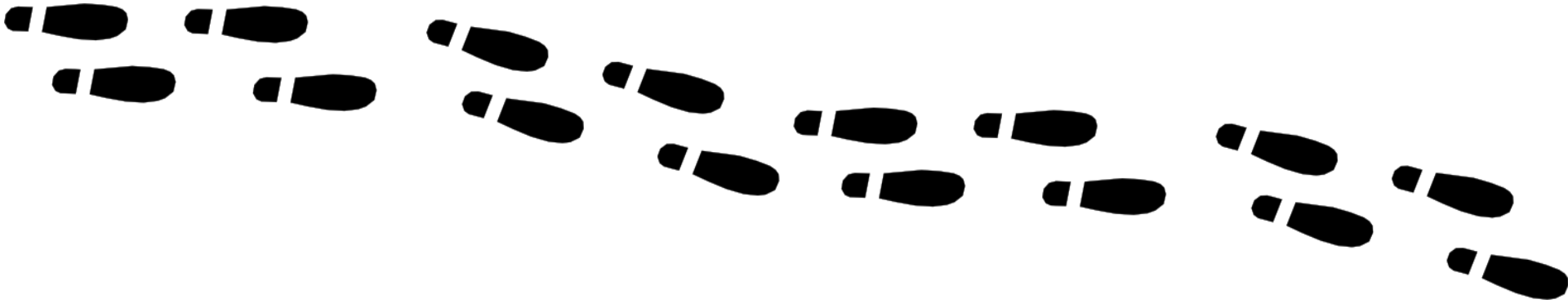
1. Create independent and deterministic scenarios



2. Don’t forget non-happy paths

Cucumber tips 3

1. Follow the one step one thing rule
2. Use nested steps sparsely (calling steps from steps)

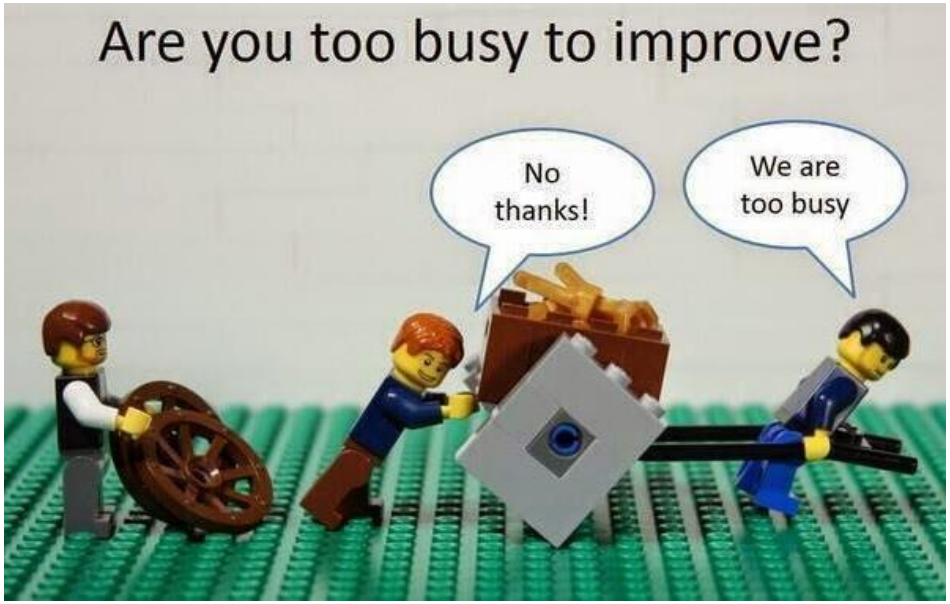


Cucumber tips 4

1. Use an object (table) for multiple test data values
   - Helps adding, removing test data
2. Use global containers for data used by multiple scenarios
   - Helps fast changes (e.g. passwords, ids)

Cucumber tips 5

1. Refactor and Reuse Step Definitions
2. Look for opportunities to generalize



Questions