CS1632, LECTURE 15: AUTOMATED WEB TESTING WITH JUNIT AND SELENIUM

Bill Laboon

The Story So Far…

- We know:
  - *Test plans*
  - *Junit*
  - *Selenium*
- Time to put them together!

Keep In Mind

- The same basic concepts of testing apply.
- We're just going to be working with text (aka HTML and associated code) at a higher level of abstraction.
- Except in rare instances, don’t directly check the HTML!

Steps for Truly Automated Web Testing

- Install Junit (you should have already done this)
- Install Selenium WebDriver (comes with Selenium install)
- Add appropriate jars
  - *selenium-java-2.x.x.jar*
  - *selenium-server-standalone-2.x.x.jar*
- Write some code!

Same concept as other JUnit tests

- Write tests
- Run tests
- Green is good (pass), red is bad (fail)

Behind the scenes…

JUnit test -->
Selenium library -->
Translates to Selenese -->
Sends to WebDriver -->
Executes on web browser -->
Returns results (if any)

import org.openqa.selenium.*;

1:1-ish Correspondence

Most of the Java commands are going to line up with Selenese commands, which means that if you were here last lecture, the only really new thing should be the syntax and some name changes.

Setting up the driver

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


// HtmlUnitDriver is lightweight and faster
// FirefoxDriver is most supported
// Other drivers exist!
// I recommend you use one of these two
```

Navigating to a web page

```
// Note that this modifies the state of driver!
// It does not return a new web page, it makes
// driver point to a new page

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

Testing the Current Page

- *driver* now contains a reference to google.com
- Internally, all of the elements of google.com's homepage have been stored as WebElements.

WebElements Can Be Anything, Even Collections of Other WebElements

- WebElement (page)
  - *WebElement (div)*
    - WebElement (text)
    - WebElement (button)
  - *WebElement (div)*
    - WebElement (div)
      - *WebElement (pulldown)*
      - *WebElement (text)*
    - WebElement (div)
      - *WebElement (image)*

Get References

- You can use driver.findElement() to get a reference to a given WebElement
- element = driver.findElement(***)

Using By Command

- There are lots of different ways to select elements on a web page
- There are different By commands to do so
- `e = driver.findElement(By.name("input_1"));`
  - *Returns this element: <input name="input_1" type="text"/>*

Finding By ID

- Finding by ID is most common
- There should be one item per page with a given ID (that is, it should be unique)
  - *Note: nothing enforces this!*
- `e = driver.findElement(By.id("gbqfq"));`

Other Ways To Select Elements

- By.cssSelector
- By.linkText
- By.partialLinkText
- By.tagName
- By.xpath

Now that you have a reference to an element, you can do things

- `e.sendKeys("wocka");`
- Sends keystrokes “w”, “o”, “c”, “k”, and “a” to the element *e*

Clicking

```
// Find a checkbox and click on it

WebElement checkbox =
    driver.findElement(By.tagName("option"));

checkbox.click();
```

Click on Submit Button (shortcut)

```
// Will find the specified submission button for
// an element e and click it
e.submit();
```

You Can Control The Whole Browser

- Switch between windows
  - `driver.switchTo().window("other_window");`
- Switch between tabs (frames)
  - `driver.switchTo().frame("other frame");`

Go Forward or Back

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

Add / Delete Cookies

```
Cookie cookie = new Cookie("Bill", "yay");
driver.manage().addCookie(cookie);
// Do stuff...
driver.deleteAllCookies();
```

Don’t Forget Our Old Friend Wait!

```
// Done by instantiating a “wait object”
WebDriverWait wait =
    new WebDriverWait(driver, 30);
// Or implicitly wait (probably better)
driver.manage().timeouts().implictlyWait(3
0, TimeUnit.SECONDS);
```

Stopping the Driver

```
// Good clean-up practice
driver.quit();
```

So we can do things, how do we test?

- Answer: asserts!
  - *assertTrue*
  - *assertEquals*
  - *assertNotNull*
  - *etc.*

General Idea

```
// Get the foo element, check that its text is “foo!”
WebElement fooElement = driver.findElement(By.id("foo"));

assertEquals(fooElement.getText(), "foo!");
```

Check the page's title exists…

```
assertNotNull(driver.getTitle());
```

Checking that an element [does/does not] exist

- If a WebElement is not found, instead of returning null, a NoSuchElementException is thrown. Simple workaround: fail the test if the exception is thrown.
- Alternatively, you can do a check before each access, but it's often easier to just catch an exception if you're accessing numerous WebElements.

Example

```
try {
    WebElement e =
        driver.findElement(By.name("foo"));
    assertEquals(e.getText(), "meow");
} catch (NoSuchElementException ex) {
    fail();
}
```

Can also use to ensure that an element does NOT exist!

```
try {
   WebElement e = driver.findElement(By.name(“BAD"));
   fail();
} catch (NoSuchElementException ex) {
   // Good, no asserts, so will always pass
}
```

Debugging

```
System.out.println(driver.getPageSource());
```

Reminder: Do Not Do This

```
String expectedText = "<html><body>” +
   “...” + “</body></html>“;
String actualText = driver.getPageSource();
assertEquals(expectedText, actualText);
```

Further Resources

- Full JavaDocs:
  - http://seleniumhq.github.io/selenium/docs/api/java/
- WebElement JavaDocs:
  - http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html
- WebDriver API Instructions:
  - http://www.seleniumhq.org/docs/03_webdriver.jsp

A Word on Cucumber

- Provides a way to “execute” scenarios directly with the magic of regular expressions
- However, difficult to set up, and really not necessary to get the idea, so I don’t teach it anymore
- But if you’re interested in web testing, you should check it out!
- https://cucumber.io/