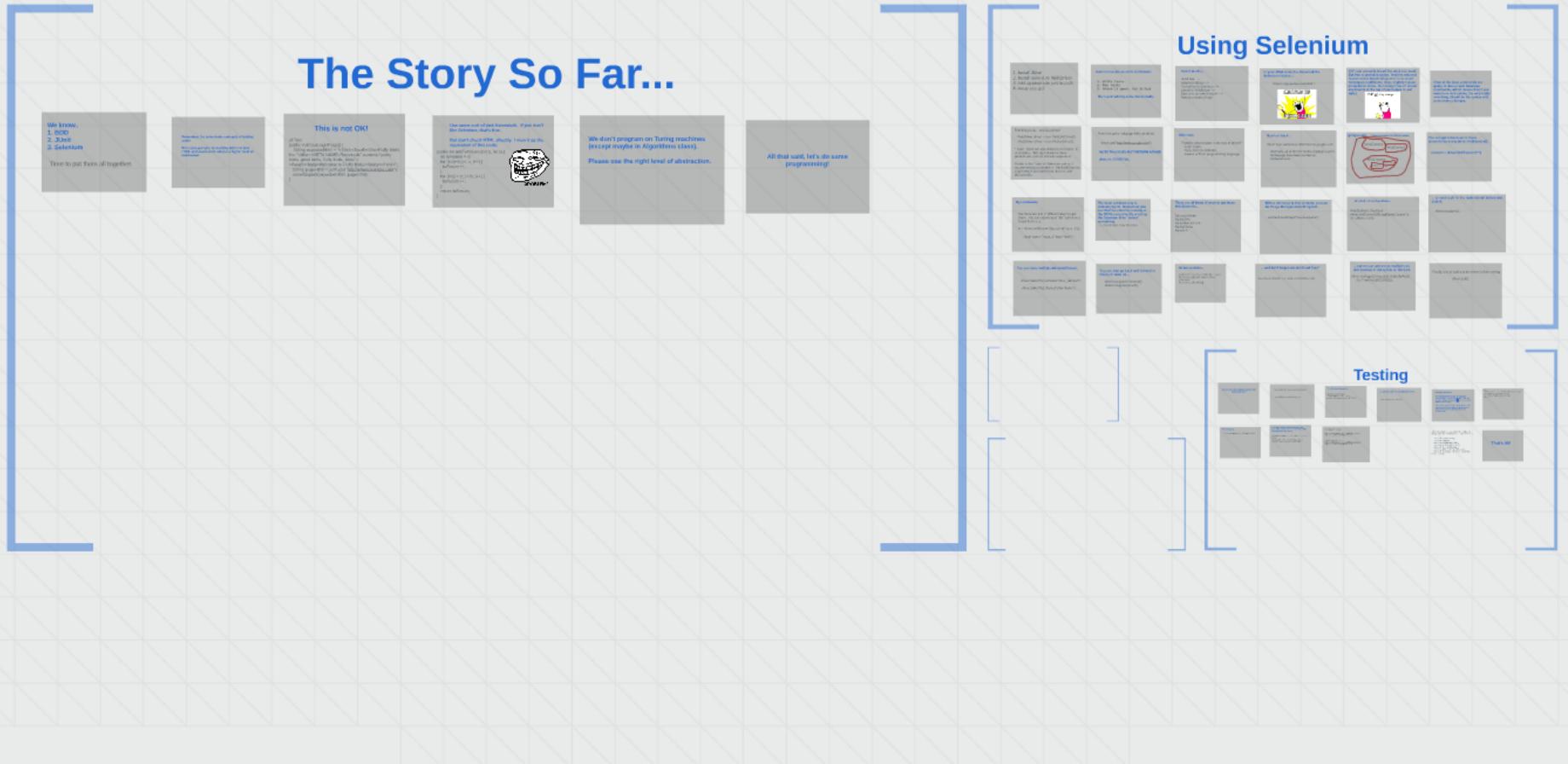


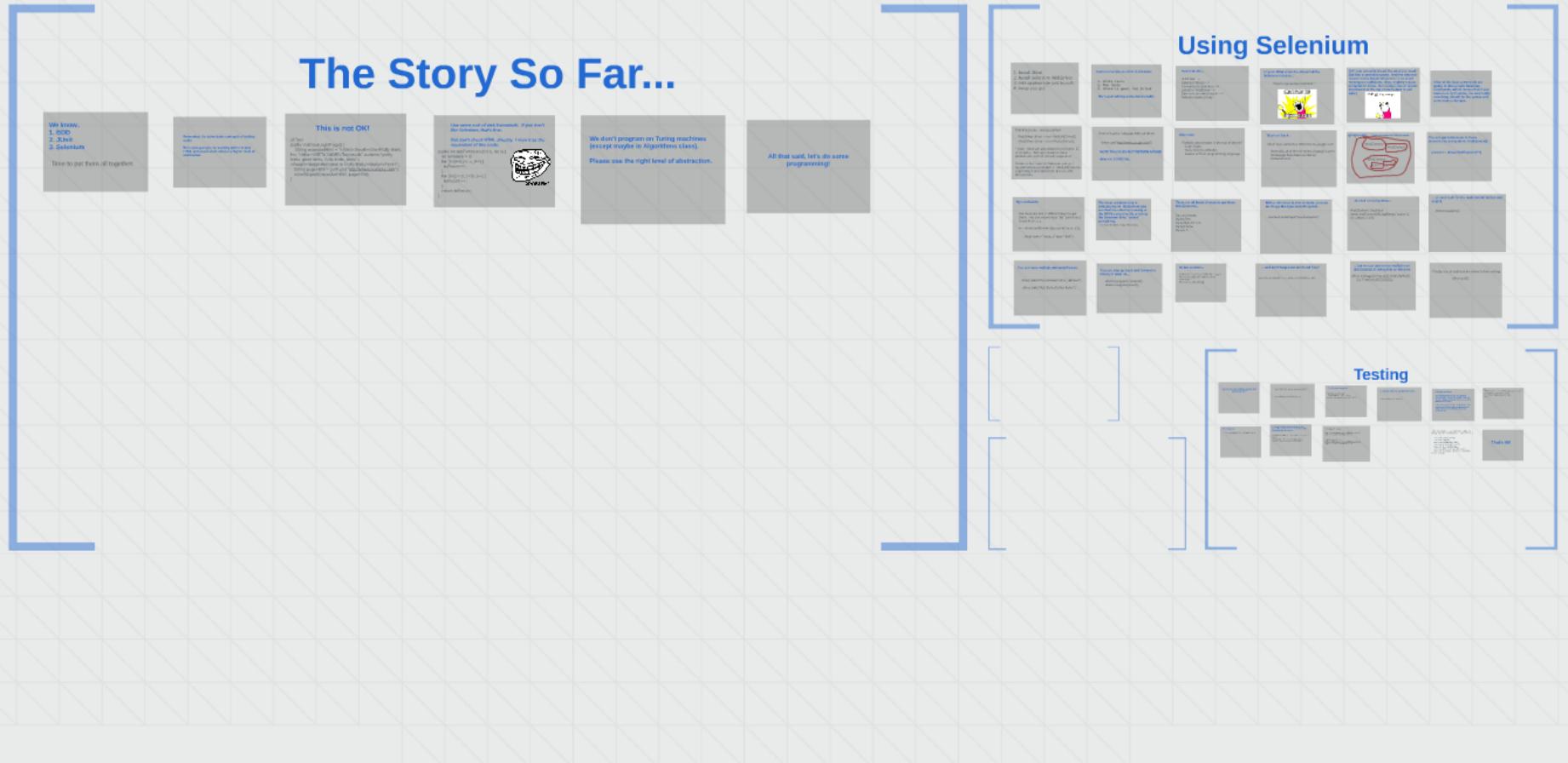
Lecture 16 - Testing the Web with JUnit, Selenium (and maybe Cucumber)

The Story So Far...



Lecture 16 - Testing the Web with JUnit, Selenium (and maybe Cucumber)

The Story So Far...



JUnit, Selenium (and maybe more)

The Story So Far...

We know..
1. BDD
2. JUnit
3. Selenium

Time to put them all together.

Remember, the same basic concepts of testing apply.
We're just going to be working with their (fake) HTML, and associated code at a higher level of abstraction.

This is not OK!

```
class Test {
    public void testLoginPage() {
        String expectedHTML = "<html><head><title>Fluffy Birds</title><meta name='keywords' content='pretty birds'></head><body>Welcome to Fluffy Birds</body></html>";
        String pageHTML = getFrom("http://www.example.com");
        assertEquals(expectedHTML, pageHTML);
    }
}
```

Use some sort of web framework. If you don't like Selenium, that's fine.

But don't check HTML directly. I view it as the equivalent of this code:

```
public int addTwoNums(int a, int b) {
    int toReturn = 0;
    for (int i=0; i < a; i++) {
        toReturn++;
    }
    for (int j = 0; j < b; j++) {
        toReturn++;
    }
    return toReturn;
}
```



We don't program on Turing machines
(except maybe in Algorithms class).

Please use the right level of abstraction.

All that said, let's do some programming!

We know..

- 1. BDD**
- 2. JUnit**
- 3. Selenium**

Time to put them all together.



Prezi

Remember, 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.

This is not OK!

```
@Test  
public void testLoginPage() {  
    String expectedHtml = "<html><head><title>Fluffy Birds,  
Inc."</title><META NAME="keywords" content="pretty  
birds, good birds, fluffy birds, birds">  
</head><body>Welcome to Fluffy Birds!</body></html>";  
    String pageHtml = getPage("http://www.example.com");  
    assertEquals(expectedHtml, pageHtml);  
}
```

Use some sort of web framework. If you don't like Selenium, that's fine.

But don't check HTML directly. I view it as the equivalent of this code:

```
public int addTwoNums(int a, int b) {  
    int toReturn = 0;  
    for (int j=0; j < a; j++) {  
        toReturn++;  
    }  
    for (int j = 0; j < b; j++) {  
        toReturn++;  
    }  
    return toReturn;  
}
```



PROBLEM?

**We don't program on Turing machines
(except maybe in Algorithms class).**

Please use the right level of abstraction.

**All that said, let's do some
programming!**

Using Selenium

1. Install JUnit
2. Install Selenium WebDriver
3. Add appropriate jars to path
4. Away you go!

Same Concepts as other JUnit tests:

1. Write tests
2. Run tests
3. Green is good, red is bad

We're just adding some functionality

First thing to do - set up a driver!

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

* Note - there are also drivers for Chrome, IE, and Opera. Testing in Safari is more problematic and not officially supported.

Firefox is the "core" of Selenium and so I recommend you test with it. HtmlUnitDriver is a lightweight and fast driver, but it is a bit idiosyncratic;

By commands

But there are lots of different ways to get them. You can use various "By" commands to get them, e.g.,

```
e = driver.findElement(By.name("input_1"));  
  
<input name="input_1" type="text"/>
```

The most common way is probably by ID. Remember you can find IDs either by looking at the HTML code directly or using the Selenium ID to "select" something.

```
e = driver.findElement(By.id("goku"));
```

How it works...

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

We're just adding some functionality

In your JUnit class file, import all the Selenium classes...

```
import org.openqa.selenium.*;
```



(OK, you can only import the ones you need. But this is probably easier. And the only real reason not to import whatever.* is to avoid namespace collisions. Also, slightly reduce compilation times. But ninety lines of import statements at the top of each class is just ugly.)



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

You can have multiple windows/frames...

```
driver.switchTo().window("other_window");  
driver.switchTo().frame("other frame");
```

You can also go back and forward in history if need be...

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

Or set cookies...

```
Cookie cookie = new Cookie("bill", "yw");  
driver.manage().addCookie(cookie);  
if (do stuff)  
driver.deleteAllCookies();
```

There are all kinds of ways to get these WebElements...

```
By.cssSelector  
By.linkText  
By.partialLinkText  
By.tagName  
By.xpath
```

With a reference to that element, you can do things like type something in it.

```
element.sendKeys("wooka wocka");
```

... and don't forget our old friend 'wait'

```
WebDriverWait wait = new WebDriverWait(driver, 30);
```

... but we can just set an implicit wait time instead of doing this all the time.

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

Finally, you should quit the driver before ending.

```
driver.quit();
```

Testing

1. Install JUnit
2. Install Selenium WebDriver
3. Add appropriate jars to path
4. Away you go!

Same Concepts as other JUnit tests:

- 1. Write tests**
- 2. Run tests**
- 3. Green is good, red is bad**

We're just adding some functionality

How it works...

JUnit test -->

Selenium library -->

Translates to Selenese -->

Sends to WebDriver -->

Executes on web browser -->

Returns results (if any)

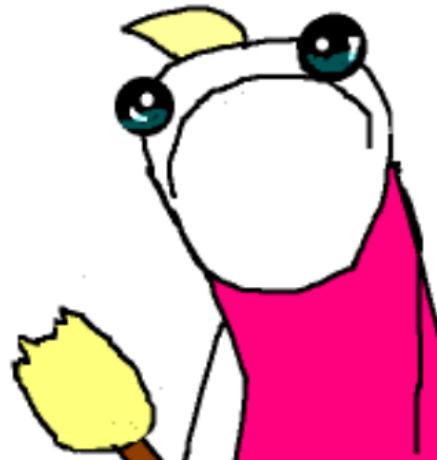
In your JUnit class file, import all the Selenium classes...

```
import org.openqa.selenium.*;
```



(OK, you can only import the ones you need. But this is probably easier. And the only real reason not to import whatever.* is to avoid namespace collisions. Also, slightly reduce compilation times. But ninety lines of import statements at the top of each class is just ugly.)

Clean all the things?



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.

First thing to do - set up a driver!

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

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

* Note - there are also drivers for Chrome, IE, and Opera. Testing in Safari is more problematic and not officially supported.

Firefox is the "core" of Selenium and so I recommend you test with it. HtmlUnitDriver is a lightweight and fast driver, but it is a bit idiosyncratic.



Now let's get a webpage with our driver.

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

NOTE THIS DOES NOT RETURN A PAGE

driver is STATEFUL.

Side note:

"Mutable shared state is the root of all evil."

-José Valim,
Rails Core Contributor,
creator of Elixir programming language

Back on track...

driver now contains a reference to google.com

Internally, all of the elements of google.com's homepage have been stored as WebElements

google.com

These correspond to DOM elements.

WebElement

WebElement

WebElement

WebElement

WebElement

You can get references to these elements by using `driver.findElement()`

`element = driver.findElement(***)`

By commands

But there are lots of different ways to get them. You can use various "By" commands to get them, e.g.,

```
e = driver.findElement(By.name("input_1"));
```

```
<input name="input_1" type="text"/>
```

The most common way is probably by ID. Remember you can find IDs either by looking at the HTML code directly or using the Selenium ID to "select" something.

```
e = driver.findElement(By.id("gbqfq"));
```

There are all kinds of ways to get these WebElements..

By.cssSelector

By.linkText

By.partialLinkText

By.tagName

By.xpath

With a reference to that element, you can do things like type something in it..

```
element.sendKeys("wocka wocka");
```

or click on a checkbox..

WebElement checkbox

```
driver.findElement(By.tagName("option"));  
checkbox.click();
```

... or even look for the main submit button and click it.

```
element.submit();
```

You can have multiple windows/frames..

```
driver.switchTo().window("other_window");
```

```
driver.switchTo().frame("other frame");
```

You can also go back and forward in history if need be...

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

Or set cookies...

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

... and don't forget our old friend 'wait'

```
WebDriverWait wait = new WebDriverWait(driver, 30);
```

... but we can just set an implicit wait time instead of doing this all the time.

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

Finally, you should quit the driver before ending.

```
driver.quit();
```

Testing

So we can now navigate around, but how do we test?

We can just use our old pals, the asserts!
assertTrue(), assertEquals(), etc.

The Simplest Thing to Do

```
WebElement fooElement = driver.findElement(By.id("foo"));
assertEquals(fooElement.getText(), "foo");
```

Or, say, check that the page does have a title...

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

A little bit of trickiness...

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.

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

For Debugging

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

Once again, please, please, pretty please with a cherry on top.. DO NOT DO THIS FOR YOUR ACTUAL TESTS.

```
String expectedText = "<html><head>...</head></html>";
String actualText = driver.getPageSource();
assertEquals(expectedText, actualText);
```

The driver API is here

<http://selenium.googlecode.com/svn/trunk/driver/doc/api/org/openqa/selenium/WebDriver.html>

Selenium provides a good way of checking for things. Use in combination with JUnit assertions.

```
selenium.isTextPresent()
selenium.getAlert()
selenium.isSelected(locator)
selenium.isSelected(locator)
selenium.isEditable(locator)
selenium.isElementPresent(locator)
selenium.isVisible() - To verify whether the object is visible
```

That's All!

**So we can now navigate around, but
how do we test?**



We can just use our old pals, the asserts!

`assertTrue()`, `assertEquals()`, etc.

The Simplest Thing to Do

```
WebElement fooElement =  
    driver.findElement(By.id("foo"));  
assertEquals(fooElement.getText(), "foo!");
```

Or, say, check that the page does have a title..

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

A little bit of trickiness..

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.

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

For Debugging

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

**Once again, please, please, please, pretty
please with a cherry on top.. DO NOT DO THIS
FOR YOUR ACTUAL TESTS.**

```
String expectedText = "<html><head>... </head></html>"
```

```
String actualText = driver.getPageSource();  
assertEquals(expectedText, actualText);
```

The driver API is here:

[http://selenium.googlecode.com/git/docs/api/java/
org/openqa/selenium/WebDriver.html](http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/WebDriver.html)

Web Element API here:

[http://selenium.googlecode.com/git/docs/api/java/
org/openqa/selenium/WebElement.html](http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/WebElement.html)

That's All!

Lecture 16 - Testing the Web with JUnit, Selenium (and maybe Cucumber)

The Story So Far...

We know:
1. Test
2. XUnit
3. Selenium

Now what? We have tests and a test runner. What's next?

This is not OK!

I am scared of stock frameworks. If you don't think I'm being paranoid, then you're probably right. I'm not. I'm just being realistic. I'm not afraid of frameworks, but I am afraid of frameworks that are not well designed.



Low-level test of stock frameworks. If you don't think I'm being paranoid, then you're probably right. I'm not. I'm just being realistic. I'm not afraid of frameworks, but I am afraid of frameworks that are not well designed.

We don't program on Turing machines (except maybe in Algorithms class). Please use the right level of abstraction.

All that said, let's do some programming!

