Test Automation Lecture 14 – Design Patterns & Using the Page Object Model

PRAGMATIC IT Learning & Outsourcing Center

Lector: Milen Strahinski
Skype: strahinski
E-mail: milen.strahinski@pragmatic.bg
Facebook: http://www.facebook.com/LamerMan
LinkedIn: http://www.linkedin.com/pub/milen-strahinski/a/553/615

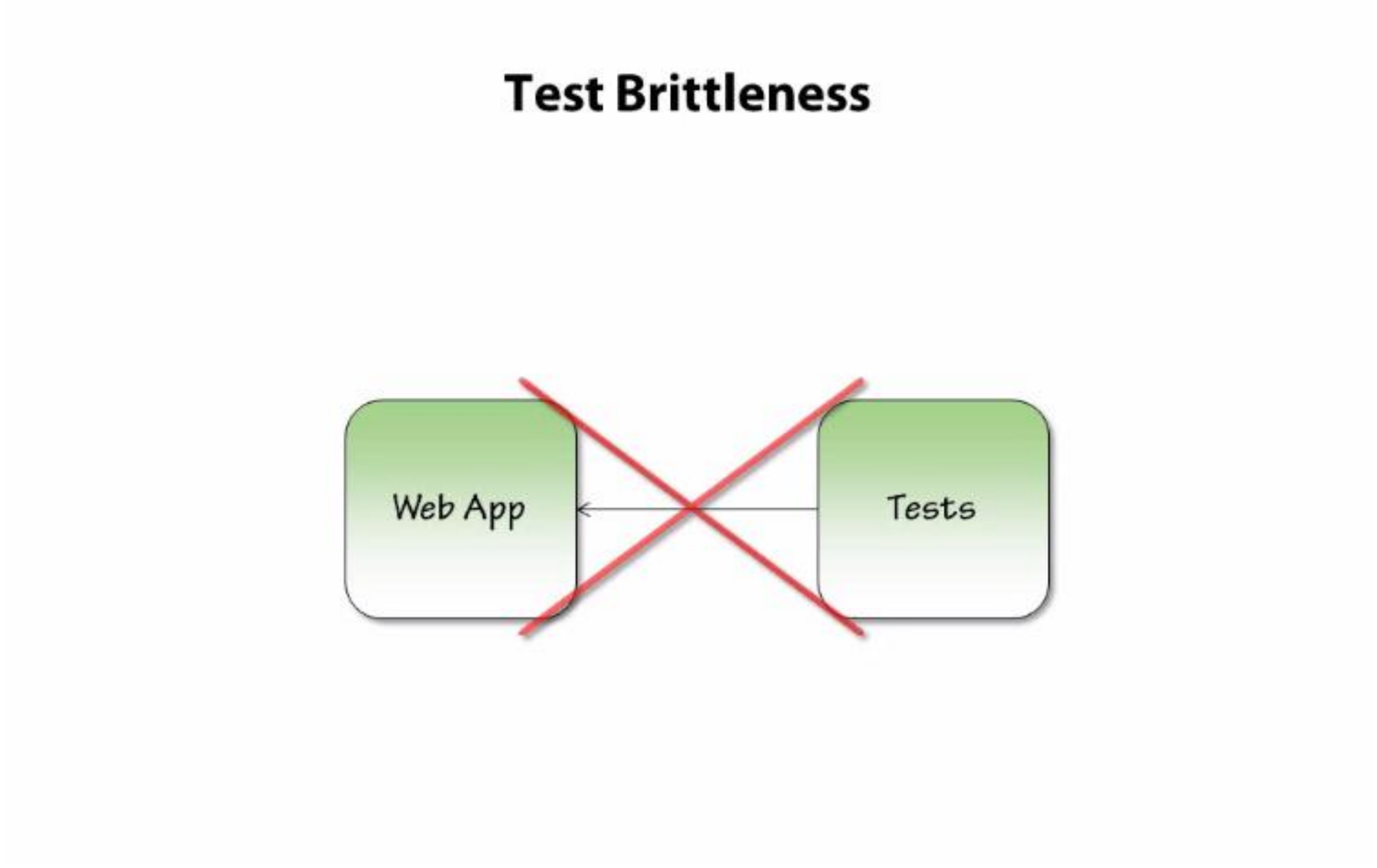
www.pragmatic.bg

2013 – 2014

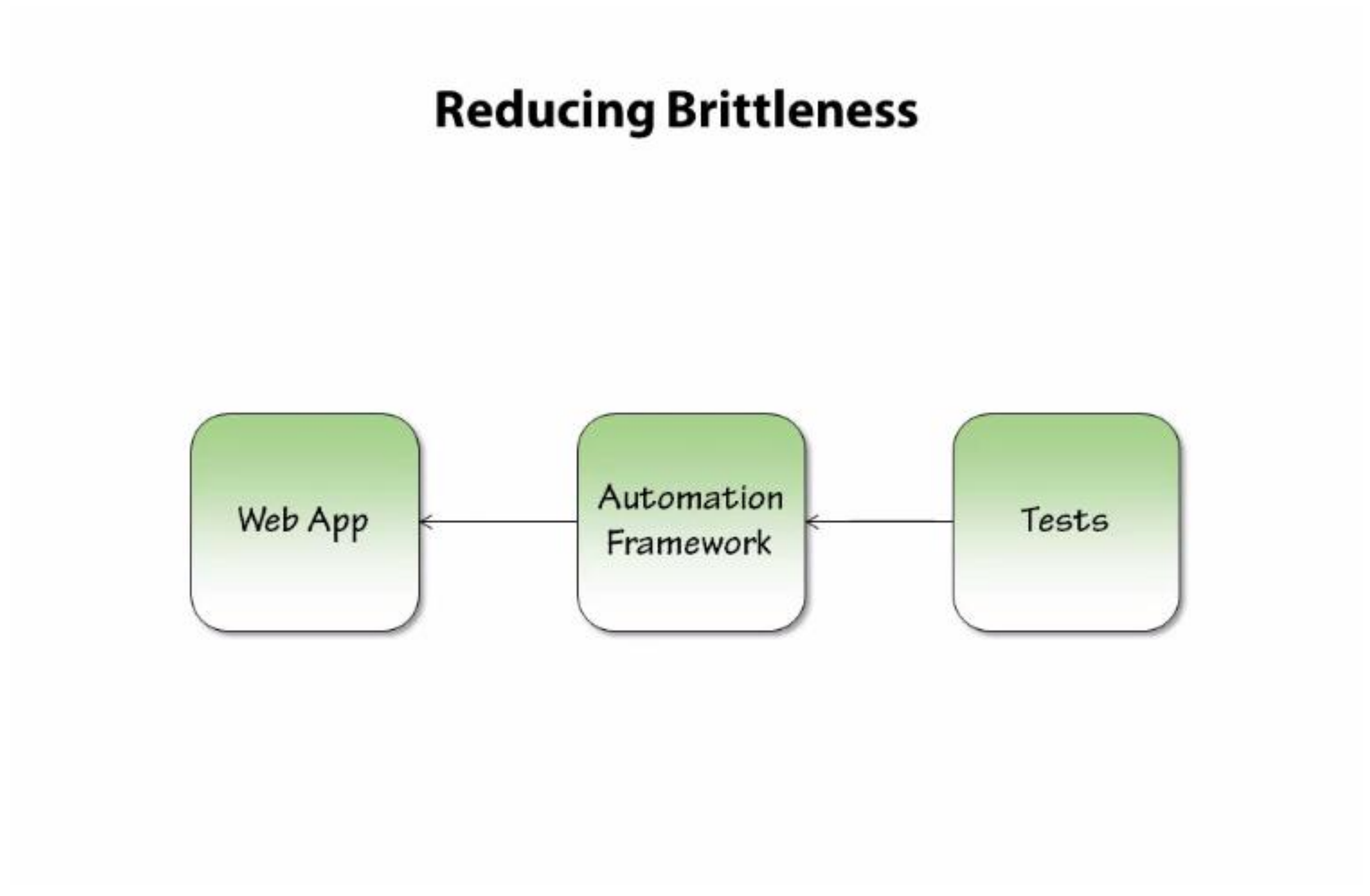
Summary - overall

- Page Object design
- Implementing nested Page Object instances
- Using PageFactory in Page Objects
- Using LoadableComponents
- Some practice example

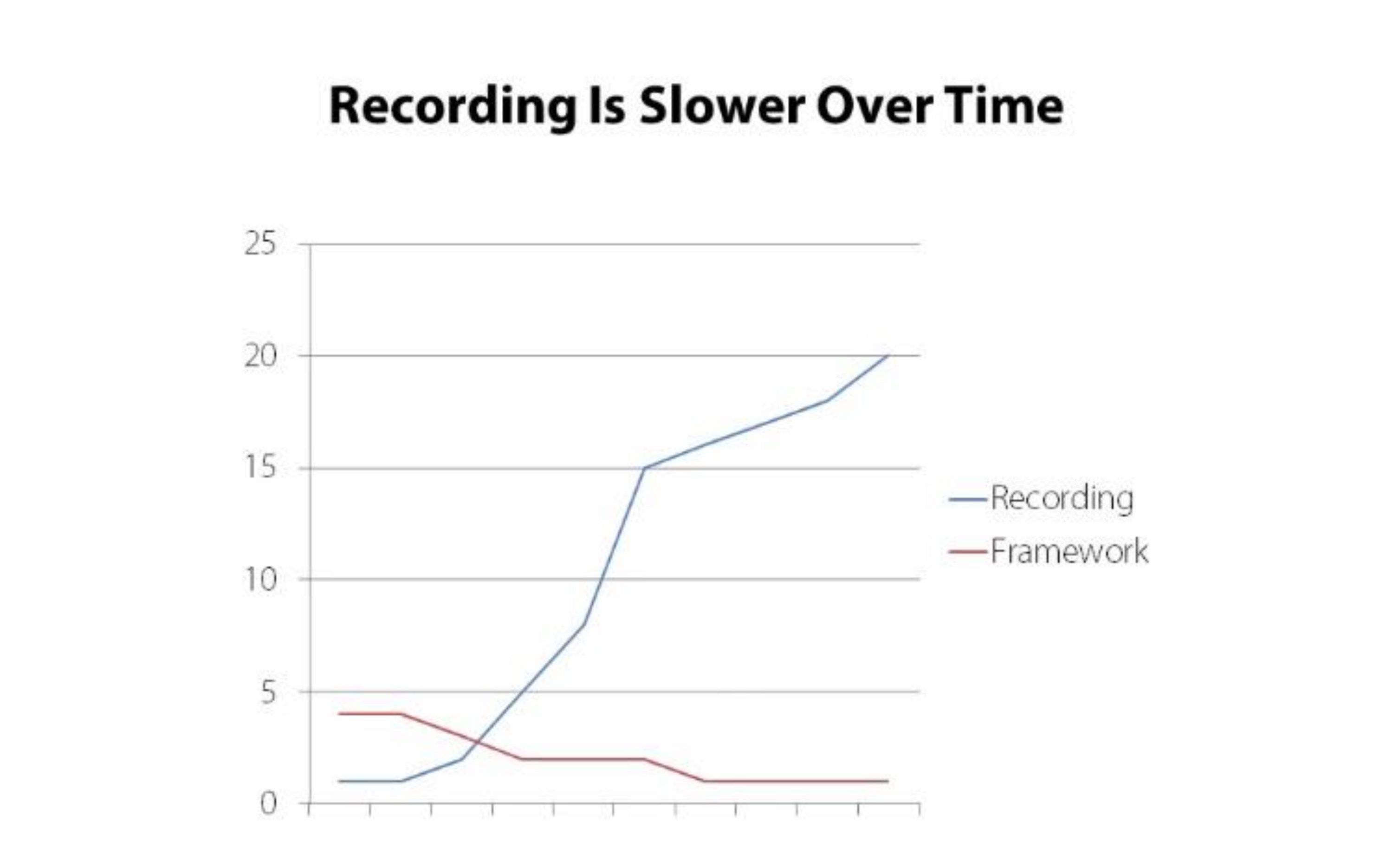
Test Brittleness



Reducing Brittleness



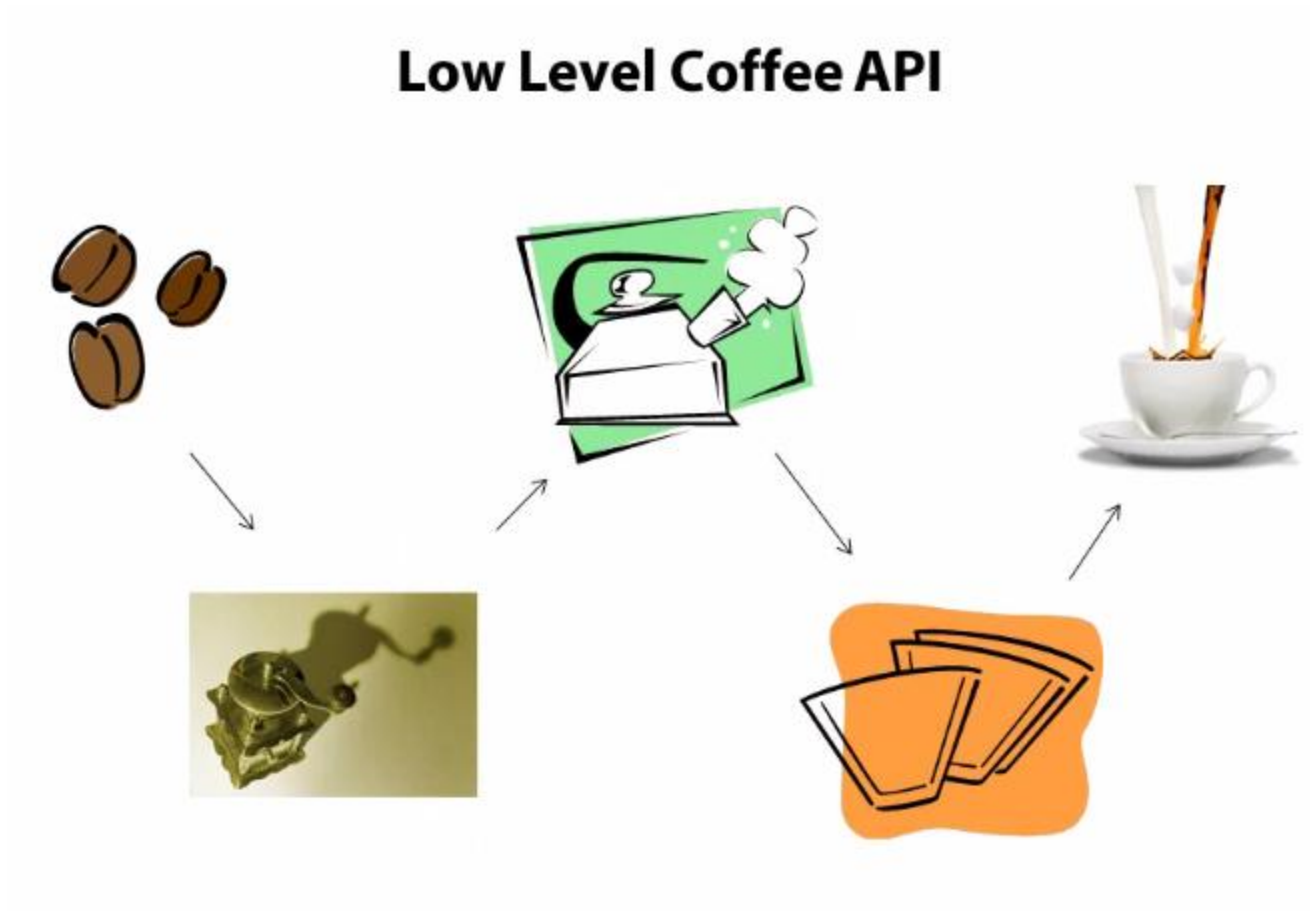
Why not record?



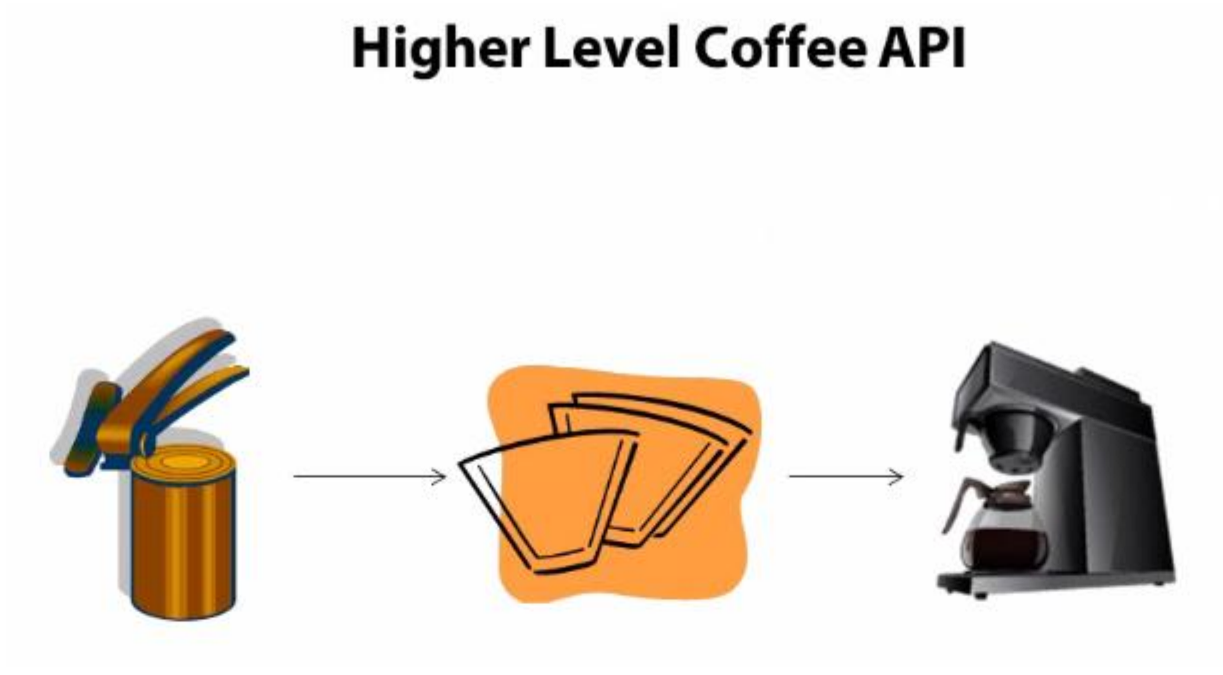
Architecture – Coffee API

Making Coffee

Architecture – Coffee API

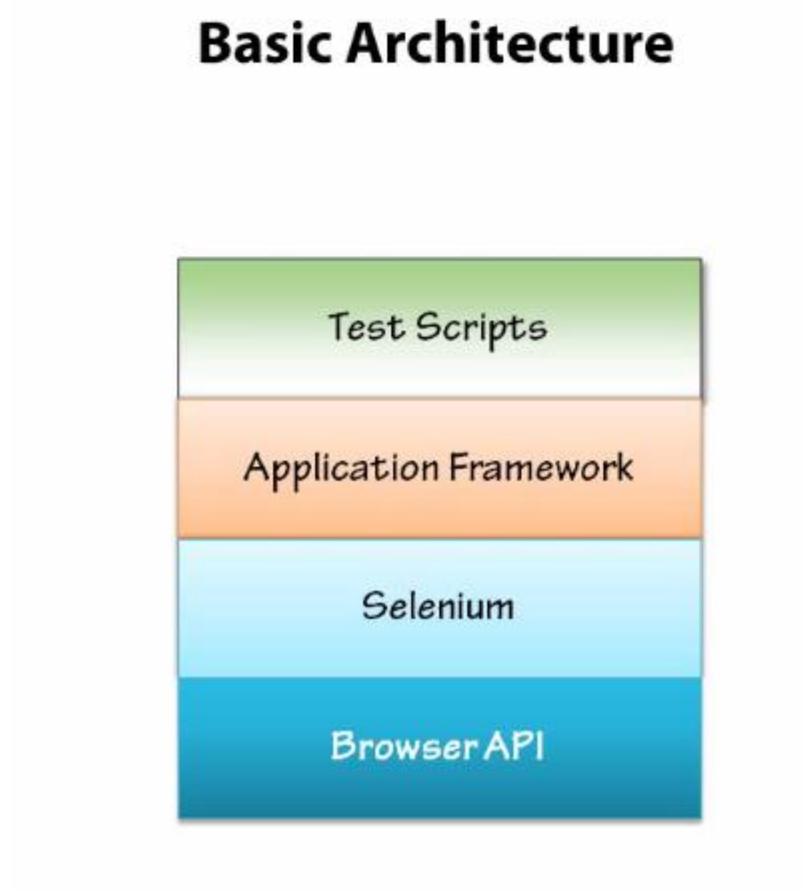


Architecture – Coffee API

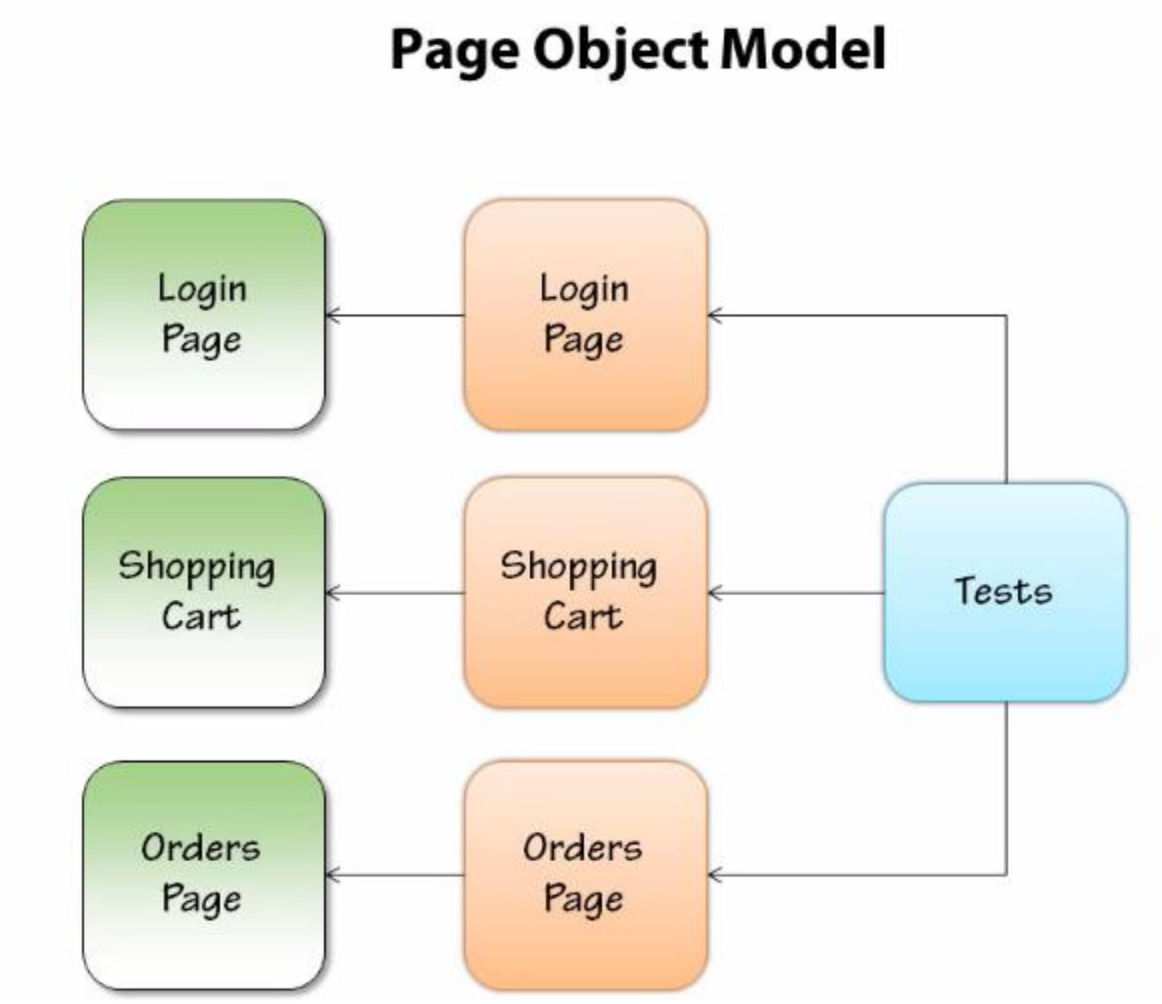


Architecture – Coffee API

Basic Architecture



PageObject model

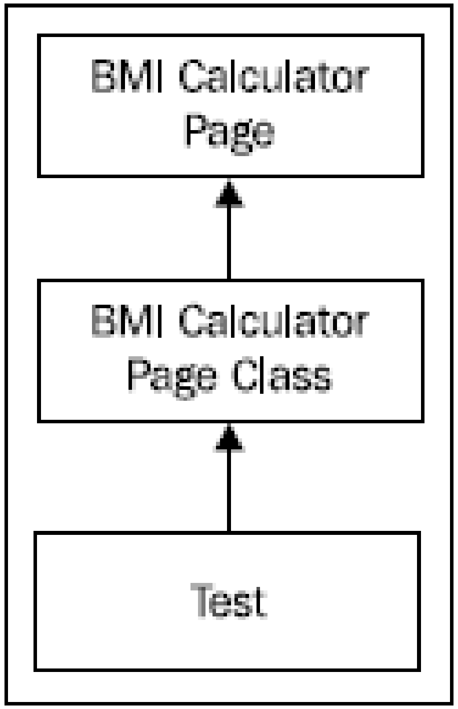


Page Objects design (part 1)

- **Page Objects:** This is a technique where we split the test logic out into separate classes. This allows us to create a Java class for each of the pages that we use on the page. For implementing the Page Object model in tests, we need to map and create a Page Object class for each page being tested.

Page Objects design (part 2)

- For example, to test the BMI Calculator application(http://dl.dropbox.com/u/55228056/bmicalculator.html), a BMI Calculator page class will be defined, which will expose the internals of the BMI Calculator page to the test, as shown in following diagram. This is done by using the PageFactory class of Selenium WebDriver API.



Page Factory in Page Objects (part 1)

- **PageFactory:** This allows us to decorate our WebElement variables in our Page objects so that we remove a lot of the look up code.

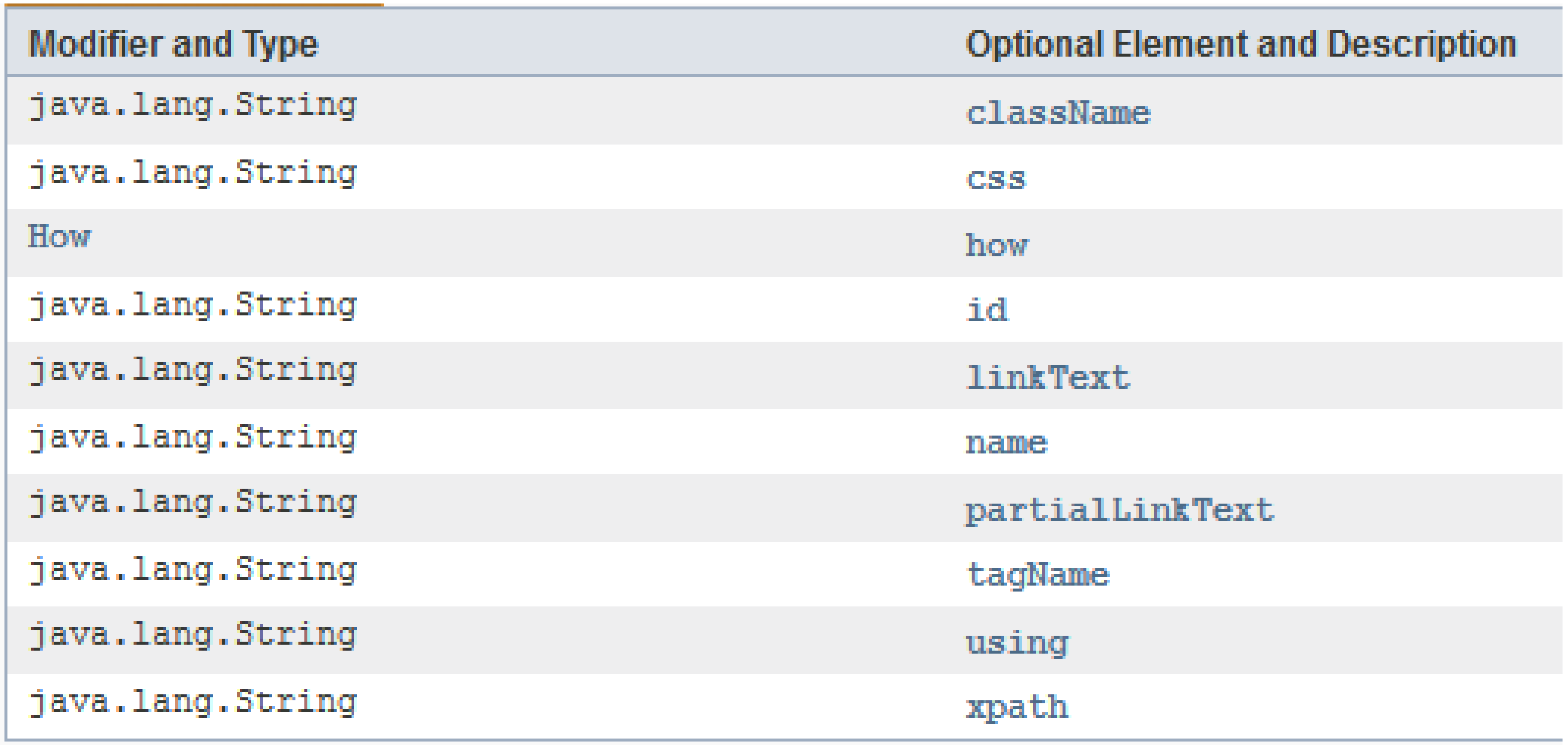
- The elements get initialized when we call PageFactory.initElements(); in our tests or anything else that may use that code.

```
public BmiCalcPage(WebDriver driver) {
    PageFactory.initElements(driver, this);
}
```

Annotations - @FindBy

```
@FindBy(id = "foobar") WebElement foobar;
@FindBy(how = How.ID, using = "foobar") WebElement foobar;

@FindBy(tagName = "a") List links;
@FindBy(how = How.TAG_NAME, using = "a") List links;
```



Modifier and Type	Optional Element and Description
java.lang.String	className
java.lang.String	css
How	how
java.lang.String	id
java.lang.String	linkText
java.lang.String	name
java.lang.String	partialLinkText
java.lang.String	tagName
java.lang.String	using
java.lang.String	xpath

Page Factory in Page Objects (part 2)

- Using the Page Object model and the PageFactory class, the BMI Calculator page's elements are exposed through the BmiCalcPage class to the test instead of the test directly accessing the internals of the page.

```
//Create instance of BmiCalcPage and pass the driver
BmiCalcPage bmiCalcPage = new BmiCalcPage(driver);

//Enter Height & Weight
bmiCalcPage.heightCMS.sendKeys("181");
bmiCalcPage.weightKg.sendKeys("80");

//Click on Calculate button
bmiCalcPage.Calculate.click();
```

Annotations - @CacheLookup

- One downside to using the @FindBy annotation is that every time we call a method on the WebElement object, the driver will go and find it on the current page again. This is useful in applications where elements are dynamically loaded or AJAX-heavy applications.
- However, in applications where we know that the element is always going to be there and stay the same without any change, we can cache it. Tests work faster with cached elements when these elements are used repeatedly. @CacheLookUp annotation along with the @FindBy annotation:

```
@FindBy(id = "heightCMS")
@CacheLookup
public WebElement heightField;
```

DO NOT REALLY USE THE ONES THAT FOLLOW ☺

- In the next slides I’ll demonstrate you some ways of already implemented frameworks in some companies which I can “hate” easily ☺ ☺ ☺ ☺
- After those demonstrations I’ll show you how I truly believe you should do your Selenium Framework

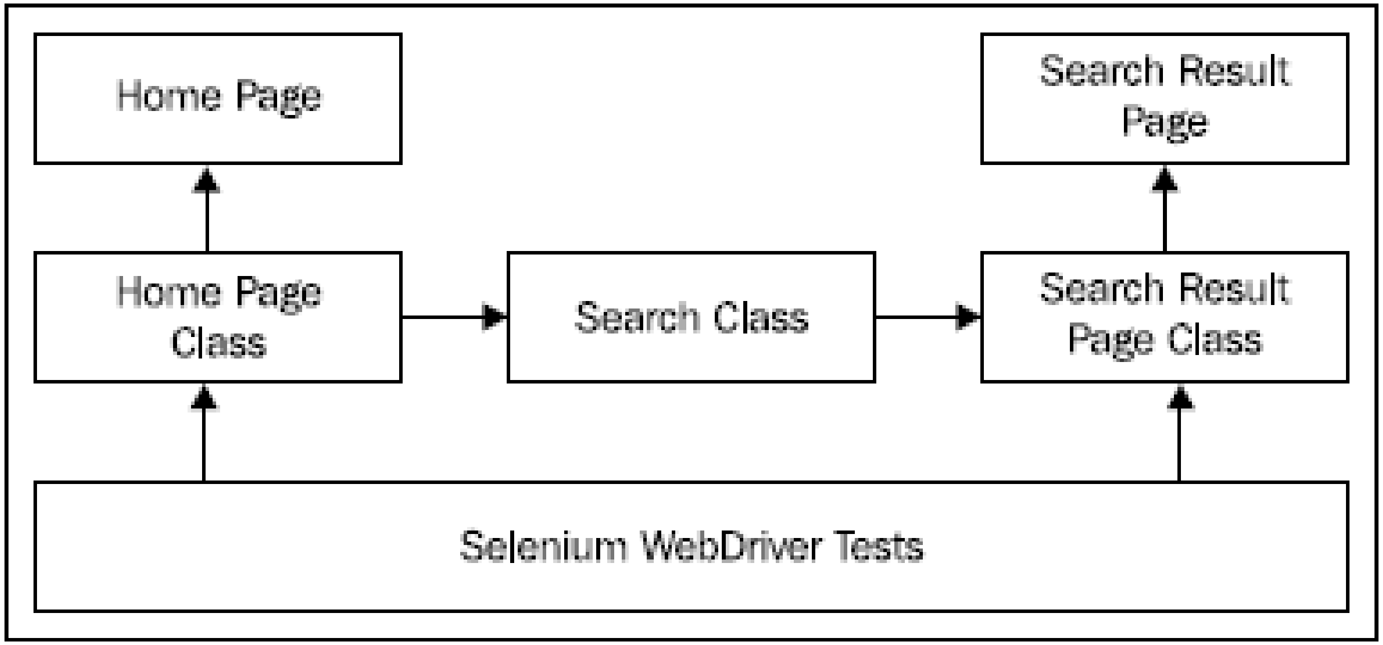
Page Objects design (part 3)

- Lets take a look and explain our simple example of the BMI Calculator application:

Archive in the Code Examples: Lecture14-BMIPageObjectExample.rar

Nested Page Objects

- Lets take a look and explain the archive in code examples: Lecture14-UsingPageObjectModel-NestedPageObjects-MagentoDemoSite.rar



LoadableComponent (part 1)

- We can implement the objects of the Page Object model using the LoadableComponent class of Selenium WebDriver. This helps in building a robust Page Object that provides a standard way to ensure that the page is loaded and that the page load issues are easy to debug.

LoadableComponent (part 2)

- The base class has the following methods on the interface:

```
get()
isLoaded()
load()
```

```
//This is what actually happens behind the scene
public T get() {
    try {
      isLoaded();
      return (T) this;
    } catch (Error e) {
      load();
    }

    isLoaded();
    return (T) this;
}
```

LoadableComponent (part 3)

- Instead of the usual public class PageObject, we change it:

```
public class PageObject extends LoadableComponent<PageObject>
```

- We will have to add overrides for the load() and isLoaded() method . The load method will load the page for us and the isLoaded() method can allow us to check if the page has been loaded correctly.

```
@Override
protected void load() {
  selenium.get("http://the-site-you-test.com");
}

@Override
protected void isLoaded() {
  String url = selenium.getCurrentUrl();
  If (url != "http://the-site-you-test.com"){
    throw new Exception("The wrong page has loaded");
  }
}
```

LoadableComponent (part 3)

- Lets take a look again and explain the LoadableComponent part in the archive in code examples:Lecture14-UsingPageObjectModel-NestedPageObjects-MagentoDemoSite.rar

LoadableComponent (part 4)

- Always call "get()" instead of "load()" directly. This code calls "isLoaded()" first. If there are no errors thrown, then it returns as quickly as possible to the user. This is the case if the page is already loaded. If not, it attempts to load the page, and then queries again to see if that load was a success.

Questions?

Practice

Lets try to build our own test framework by doing it in a "correct"(if you would ask me) way.