Test Automation(Selenium) Lecture 5 – Design Patterns & Using the Page Object Model

PRAGMATIC IT Learning & Outsourcing Center

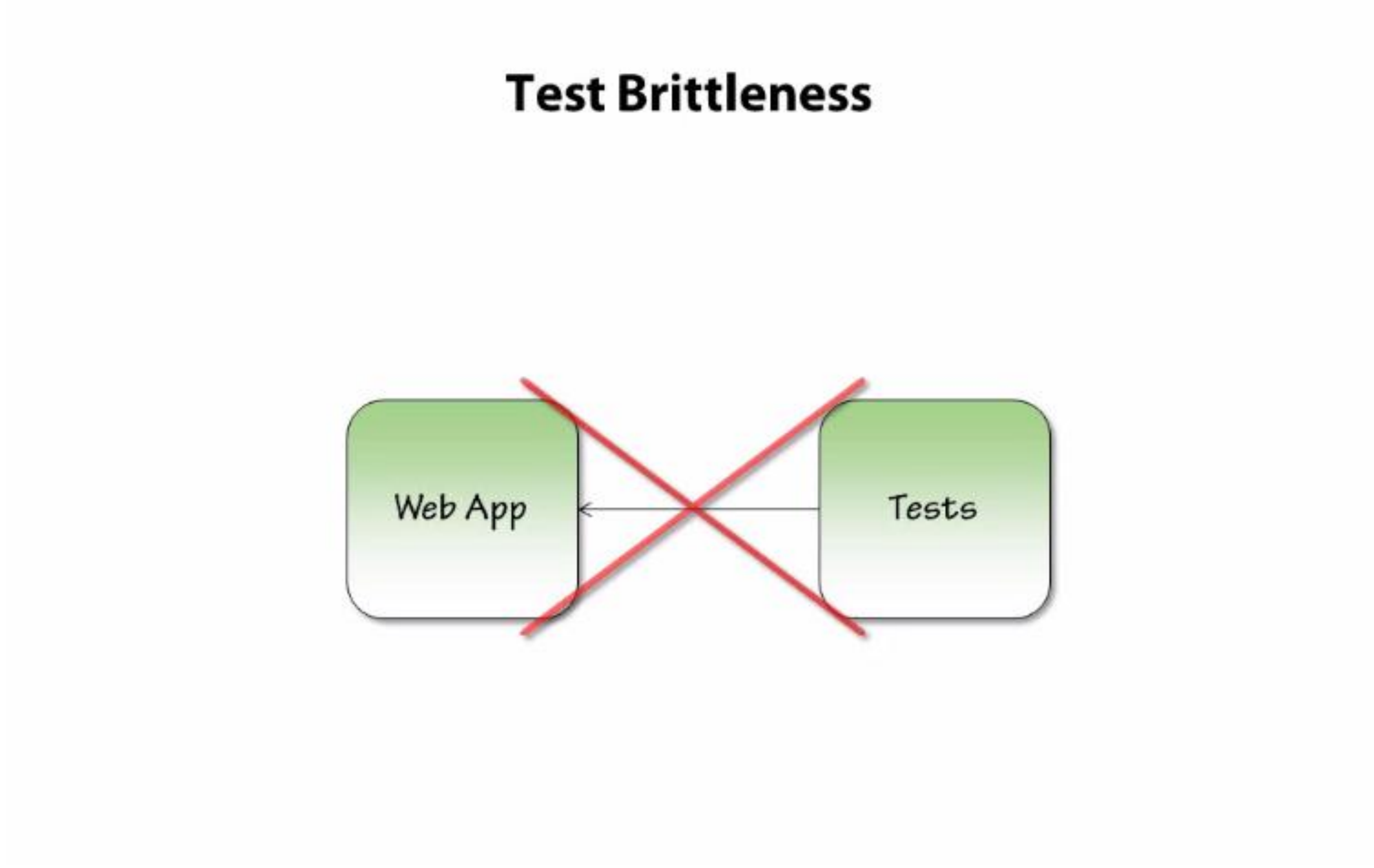
Lector: Georgi Tsvetanov
Skype: georgi.pragmatic
E-mail: george.tsvetanov@hotmail.com
Facebook: https://www.facebook.com/georgi.tsvetanov.18

www.pragmatic.bg

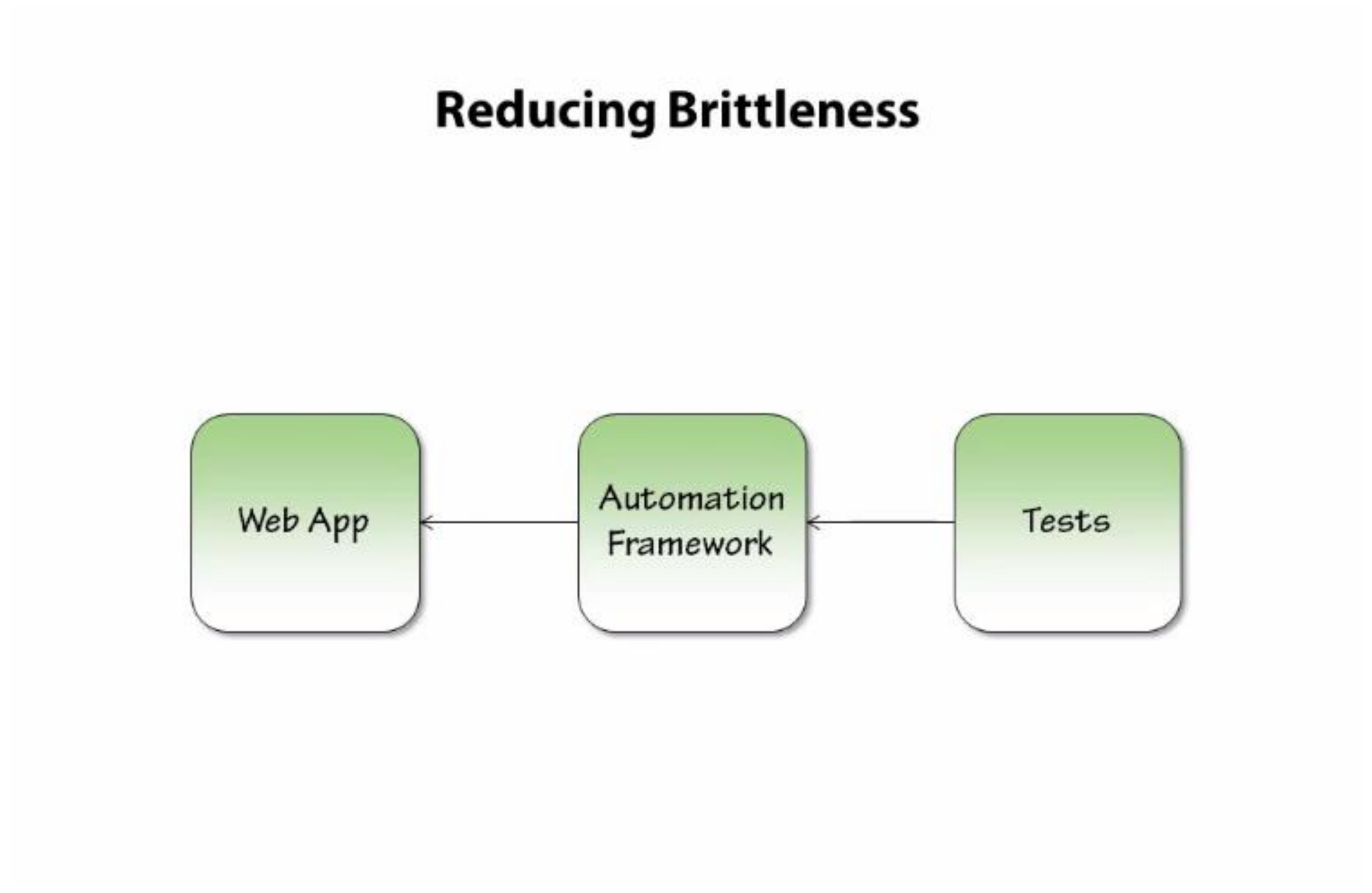
Summary - overall

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

Test Brittleness



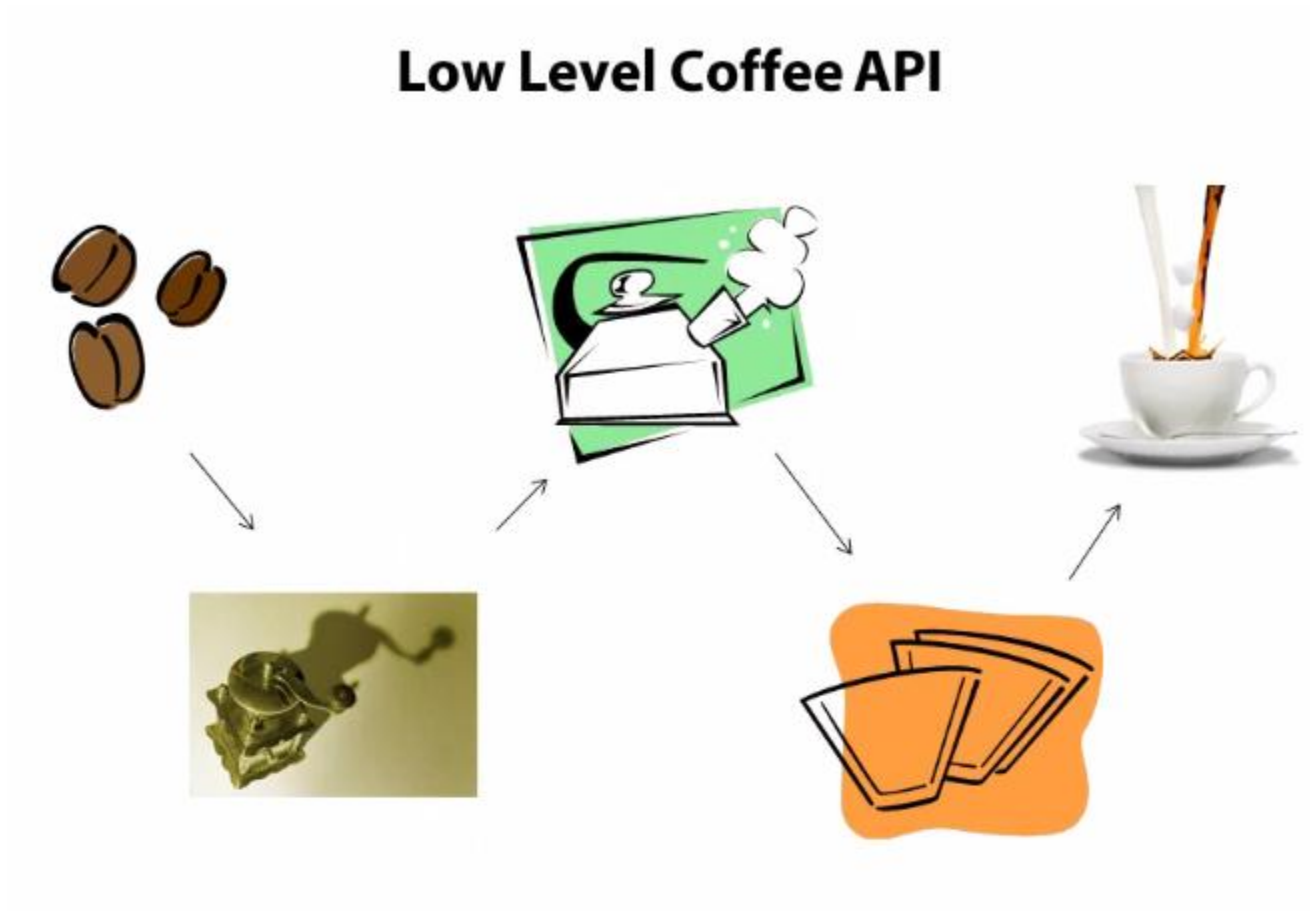
Reducing Brittleness



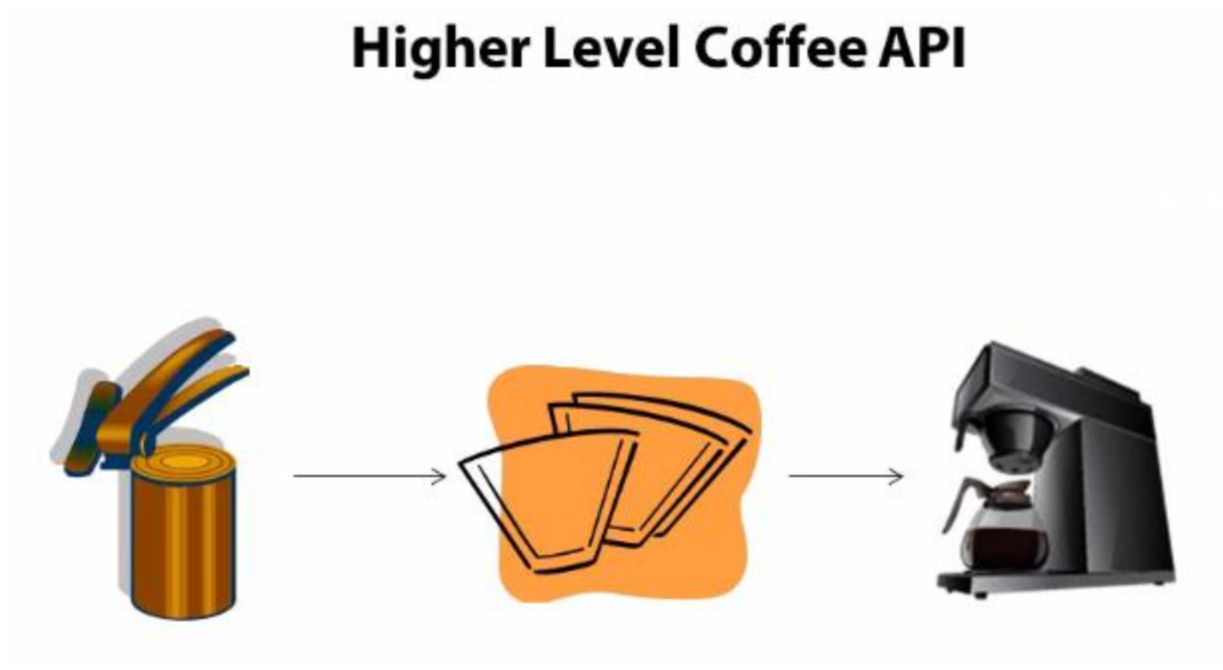
Architecture – Coffee API

Making Coffee

Architecture – Coffee API



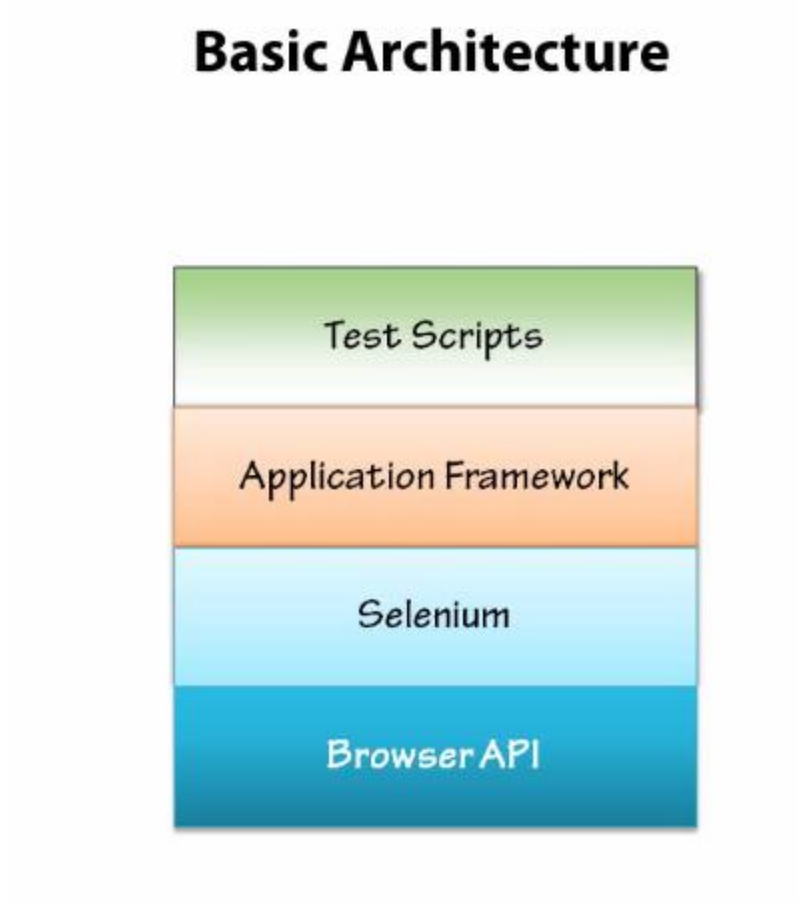
Architecture – Coffee API



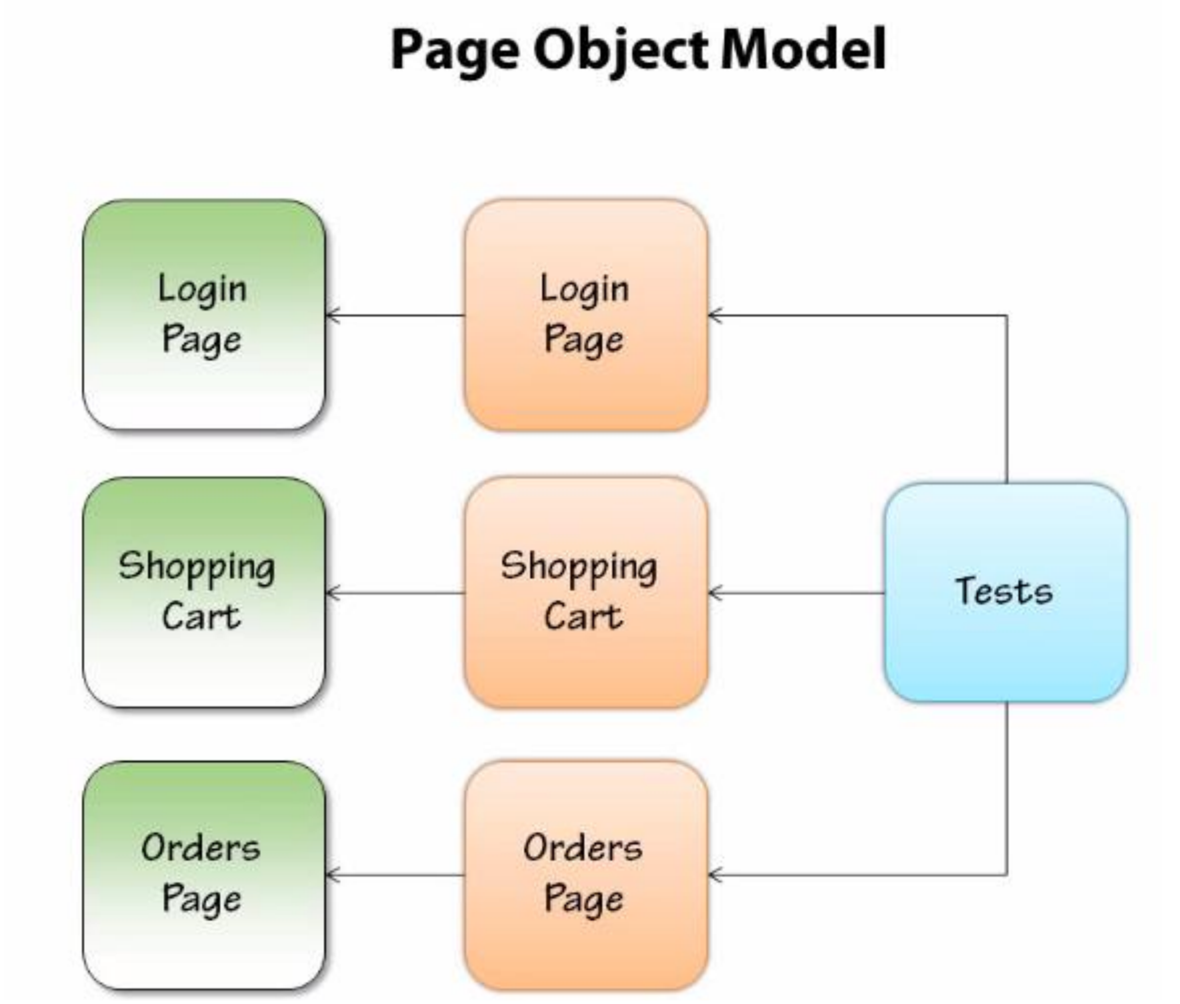
Architecture – Coffee API

Highest Level Coffee API

Basic Architecture



PageObject model

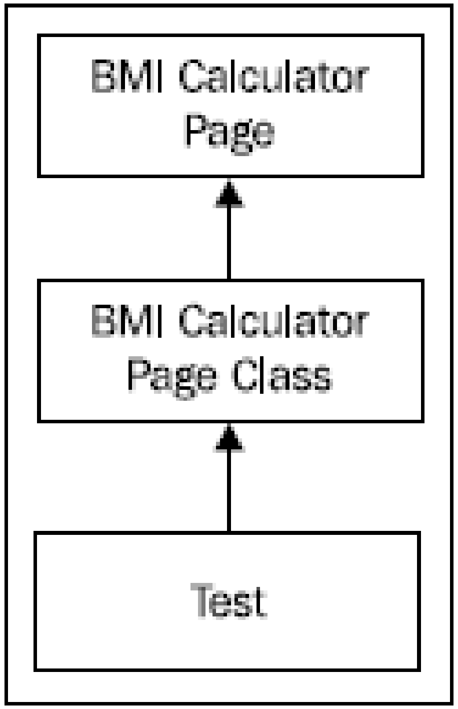


Page Objects design (part 1)

- **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 in the project. 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://pragmatic.bg/automation/lecture15/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 Objects design (part 3)

- Lets take a look and explain our simple example of the BMI Calculator application, which is also explained in the next slides:

Archive in the Code Examples: Lecture14-BMIPageObjectExample.rar

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.

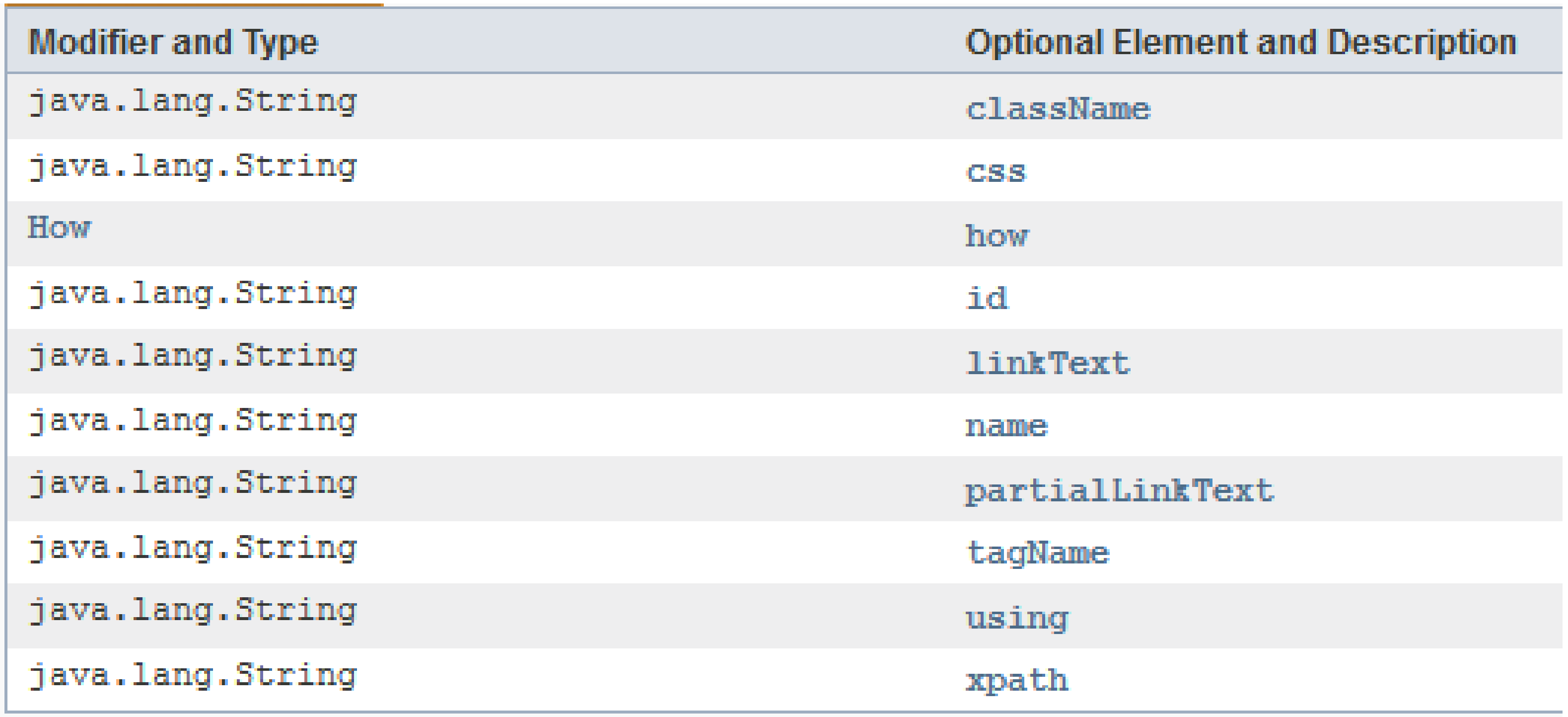
- The elements get initialized when we call PageFactory.initElements(); in our tests or anything else that may use that code.

```
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<WebElement> links;
@FindBy(how = How.TAG_NAME, using = "a") List<WebElement>
  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
BmiCalcPage bmiCalcPage = new BmiCalcPage();

//Enter Height & Weight
bmiCalcPage.calculateBmi("181", "80");
```

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;
```

Questions?

Live implementation

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

We will take a look on both - a thread safe and a static(more readable) implementations, it’s up to you what you prefer in your case.