A mocking framework for Java

http://www.mockito.org/

Richard Paul - http://www.rapaul.com/

State vs Interaction testing

- *State testing* asserts properties on an object
  - e.g. assertEquals(4, item.getCount());

- *Interaction testing* verifies the interactions between objects.
  - e.g. Did my controller correctly call my service.

- Mockito provides a framework for interactions testing.

Test Doubles

Terminology defined by Gerard Meszaros.
See: http://tinyurl.com/testdoubles

- Test Stub
  - Hand coded object used for testing.
- Mock object *(e.g. EasyMock)*
  - Expectations configured before calling the method under test.
- Test Spy *(e.g. Mockito)*
  - Verification occurs after the method under test has been called.

Mockito Example

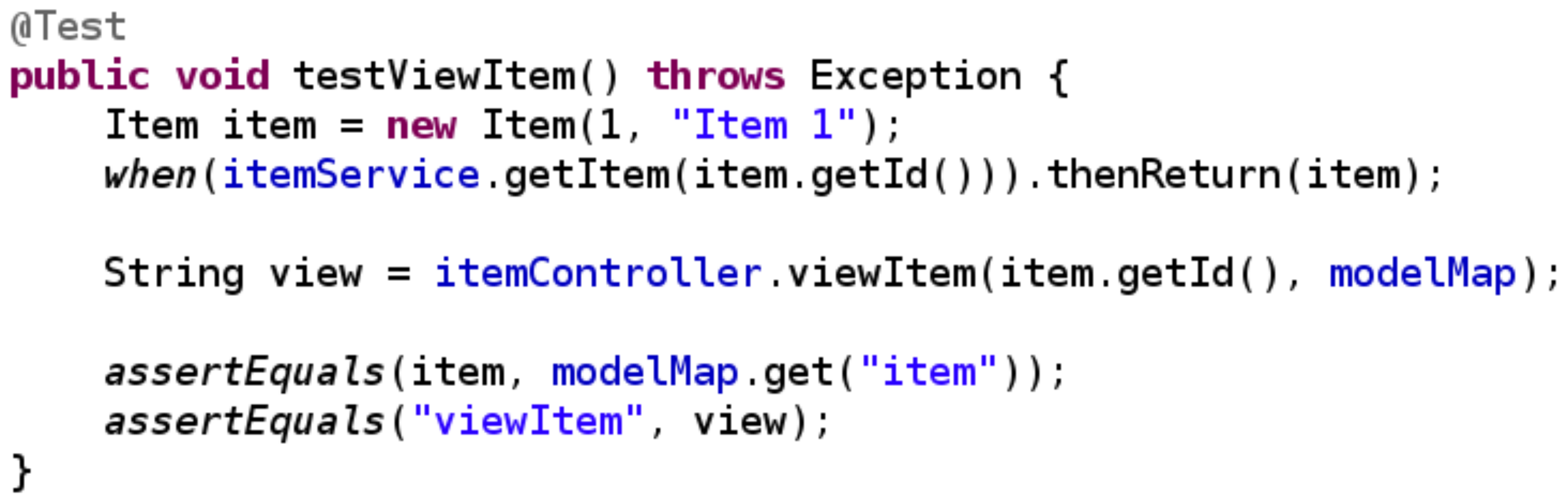
Simple MVC example:

- Item - simple POJO (has a name and ID).
- ItemController - handles requests from the browser.
- ItemService - service the ItemController delegates to.
- ItemNotFoundException

Following slides detail testing the ItemController using Mockito.

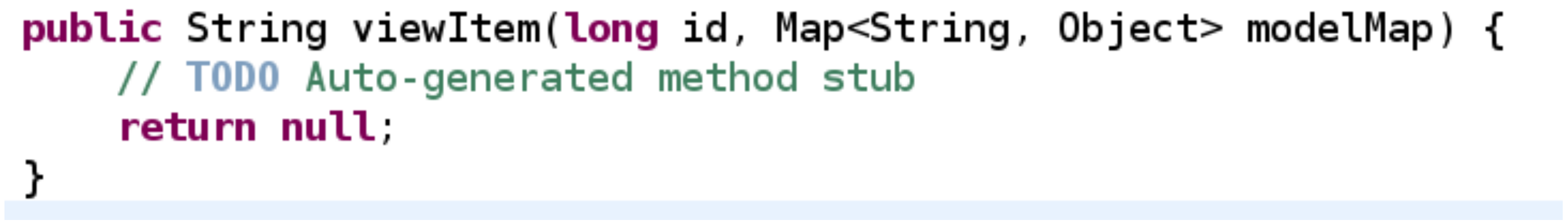
Note: the controller is implemented using a Springframework 2.5 annotation style, with annotations omitted for clarity.

View Item - using *when*



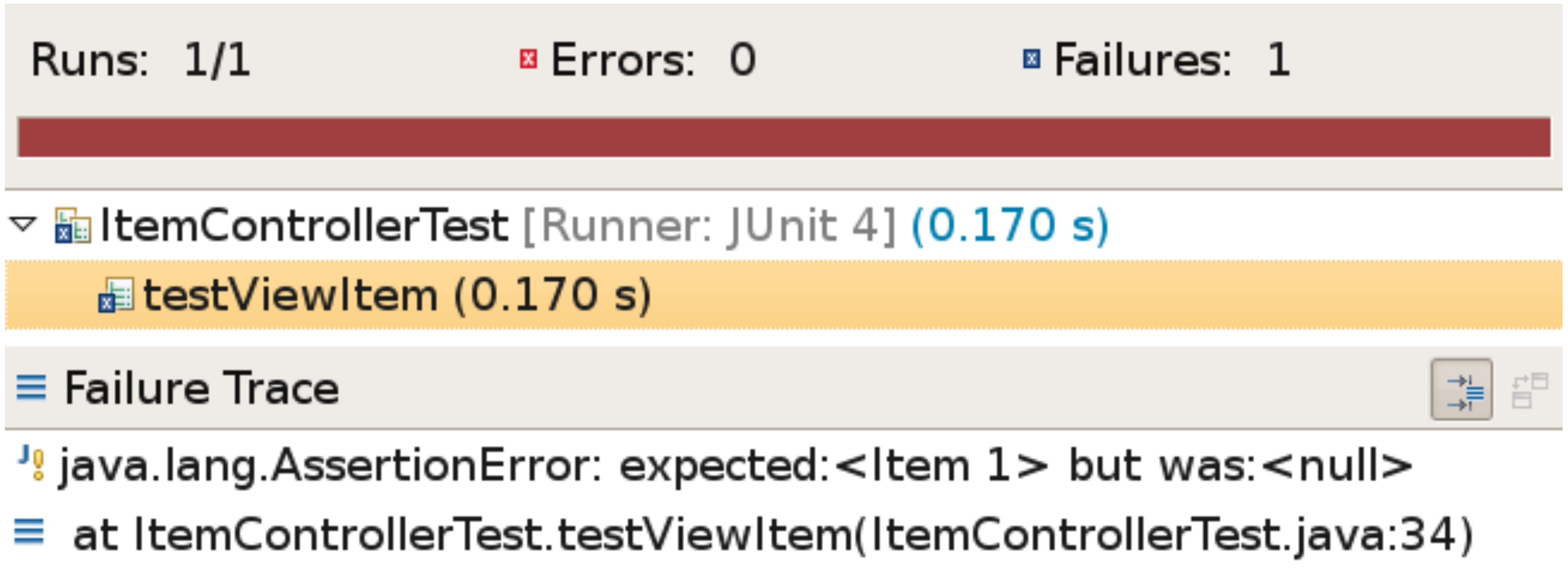
```java
@Test
public void testViewItem() throws Exception {
    Item item = new Item(1, "Item 1");
    when(itemService.getItem(item.getId())).thenReturn(item);

    String view = itemController.viewItem(item.getId(), modelMap);

    assertEquals(item, modelMap.get("item"));
    assertEquals("viewItem", view);
}
```

Eclipse then generates the viewItem method signature in ItemController.java



```java
public String viewItem(long id, Map<String, Object> modelMap) {
    // TODO Auto-generated method stub
    return null;
}
```

Test Failure


Runs: 1/1 Errors: 0 Failures: 1

ItemControllerTest [Runner: JUnit 4] (0.170 s)

testViewItem (0.170 s)

Failure Trace

```
java.lang.AssertionError: expected:<Item 1> but was:<null>
at ItemControllerTest.testViewItem(ItemControllerTest.java:34)
```

Then implement viewItem to pass the test

```java
public String viewItem(long id, Map<String, Object> modelMap) {
    Item item = itemService.getItem(id);
    modelMap.put("item", item);
    return "viewItem";
}
```

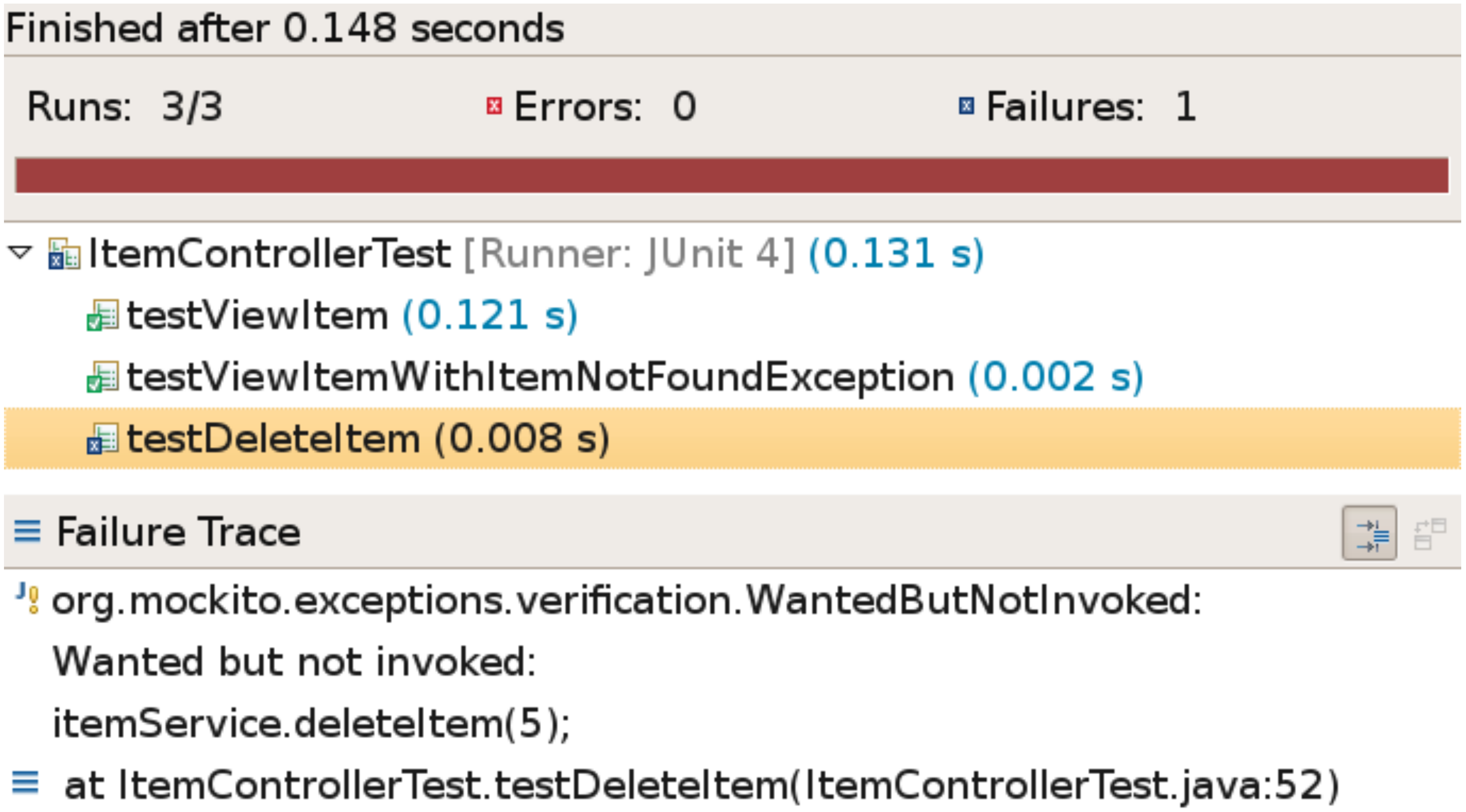
Delete Item - using *verify*

```java
@Test
public void testDeleteItem() throws Exception {
    String view = itemController.deleteItem(5);

    verify(itemService).deleteItem(5);
    assertEquals("itemList", view);
}
```

Eclipse then generates the deleteItem method signature in ItemController.java

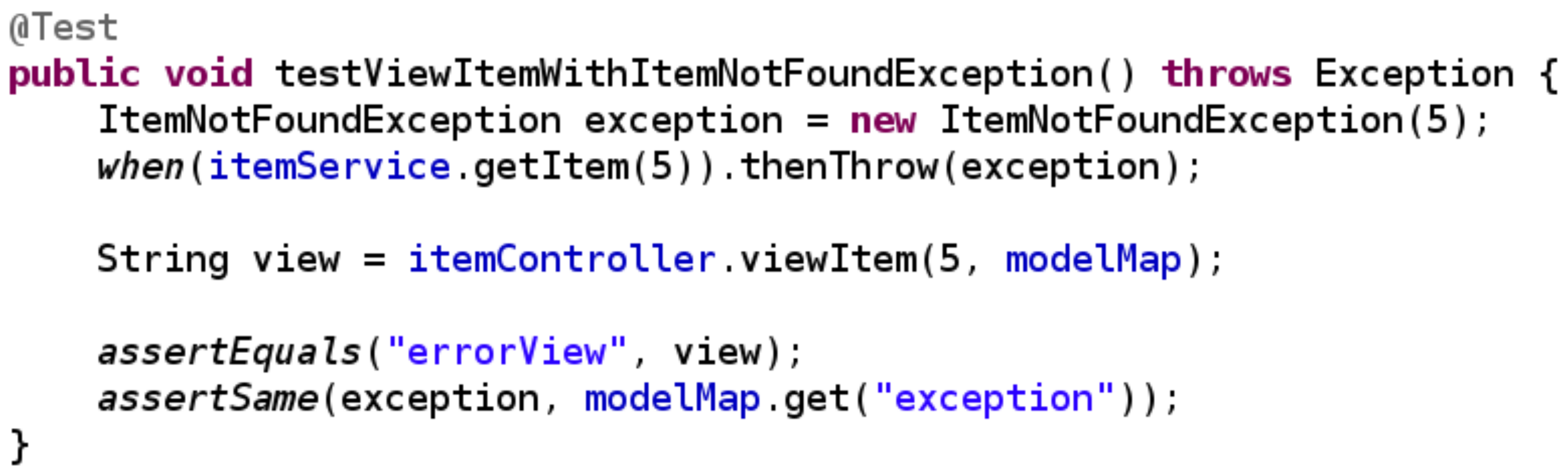
```java
public String deleteItem(long id) {
    // TODO Auto-generated method stub
    return null;
}
```

Wanted, but not invoked



```
Finished after 0.148 seconds

Runs:   3/3          Errors:   0          Failures:   1

ItemControllerTest [Runner: JUnit 4] (0.131 s)
    testViewItem (0.121 s)
    testViewItemWithItemNotFoundException (0.002 s)
    testDeleteItem (0.008 s)

Failure Trace
org.mockito.exceptions.verification.WantedButNotInvoked:
Wanted but not invoked:
itemService.deleteItem(5);
 at ItemControllerTest.testDeleteItem(ItemControllerTest.java:52)
```

Exceptional Cases - using *thenThrow*



```
@Test
public void testViewItemWithItemNotFoundException() throws Exception {
    ItemNotFoundException exception = new ItemNotFoundException(5);
    when(itemService.getItem(5)).thenThrow(exception);

    String view = itemController.viewItem(5, modelMap);

    assertEquals("errorView", view);
    assertSame(exception, modelMap.get("exception"));
}
```

Allows simple testing of exceptional cases with similar syntax to *thenReturn* .

Initialising mocks in Mockito

```
import static org.junit.Assert.*;
import static org.mockito.Mockito.*;
...
```

Allows access to when, verify etc without the Mockito.prefix

```
@RunWith(MockitoJUnitRunner.class)
public class ItemControllerTest {

    private ItemController itemController;
    @Mock private ItemService itemService;
    private Map<String, Object> modelMap;

    @Before
    public void setUp() {
        itemController = new ItemController(itemService);
        modelMap = new HashMap<String, Object>();
    }

    ...
```

Default Values

Mockito provides default values (null, 0, empty list, etc) for method calls that have not been defined. This allows you to define only the interactions that are relevant to your test.

See slides by Szczepan Faber

- http://tinyurl.com/5cxg64
- Slides 8-31

Why ignore interactions not directly related to a test case? (slide 29)

- "because aggressive validation makes the tests brittle"
- "because I have to fix tests even when the code is not broken"
    - "... can increase noise"
    - "or lead to overspecification"

More features...

- Verify how many times a method is called. e.g.
  - verify(itemService, never()).deleteItem(3);
  - verify(itemService, atLeastOnce()).deleteItem(3);
  - verify(itemService, times(3)).deleteItem(3);
- Verify the order calls are made using InOrder
- Chain subsequent calls to return different values.

  -
    ```
    when(itemService.getItem(3))
            .thenReturn(item1)
            .thenReturn(item2);
    ```
- Argument matchers
  - when(service.get(eq(1), anyFloat()).thenReturn("x");
- Can mock concrete classes (uses cglib)

Comments, queries, suggestions or theories?

Slides + code examples will be up on
http://www.rapaul.com