Writing Good Tests

Richard Warburton
JAVA CHAMPION, AUTHOR AND PROGRAMMER

@richardwarburto www.monotonic.co.uk

Why Test Quality Matters

Well … aren’t they just tests?

Tests have a cost

- Maintenance
- Readability
- Coupling

Balance cost and benefits

Tests are code - maintain them with care

What Makes a Test Good?

Well-named

Behaviour not implementation

Don’t repeat yourself

Failure diagnostics

Why Name?

Executable documentation

Maintenance

Readability

Naming Anti-pattern: Numbers

```
@Test
public void test1() {
```

```
@Test
public void espresso_beans() {


…
@Test
public void restock() {
```

Naming Anti-pattern

Describing the thing that you're testing but not the property under test

Good Naming

```
@Test
public void brewingEspressoConsumesBeans() {

@Test
public void shouldBlockTransactionBeyondCreditLimit() {
```

Naming Guidelines

Use domain terminology

Natural language

Be descriptive

Test Code

Behaviour not Implementation

Behaviour Anti-pattern

```
assertEquals(5, object.internalState);

assertEquals(5, object.getInternalState());
```

Behaviour Anti-pattern

```
// Exposing the field via a getter just for
// testing is cheating!
assertEquals(5, object.getInternalState());
```

Two Approaches

Implementation

Exposing state results in brittle tests

Behaviour

Assert on results

Implementation can still be refactored

Duplication

Duplication Anti-pattern

```
Cafe cafe = new Cafe();
cafe.restockBeans(7);
```

Magic Number Anti-pattern

```
cafe.restockBeans(7);
```

Magic Number Fixed

```
cafe.restockBeans(ESPRESSO_BEANS);
```

Diagnostics

```
List<Coffee> order = cafe.brewOrder(...);
assertTrue(order.size() == 1);
```

```
java.lang.AssertionError
```

Size of List

Only use of assertTrue with actual boolean values

Exposing the Value

```
List<Coffee> order = cafe.brewOrder(...);
assertEquals(1, order.size());
```

```
Expected :1
Actual :0
```

```
List<Coffee> order = cafe.brewOrder(...);
assertEquals("Wrong quantity of coffee", 1, order.size());
```

```
Wrong quantity of coffee
Expected :1
Actual :0
```

Exposing a Reason

Giving messages to asserts helps diagnostics

Before & After: Common Code

JUnit Practices

Junit helps

Features align wih good practices

Reduced duplication

Before and after

@BeforeEach

@AfterEach

@BeforeAll

@AfterAll

- Common code run in before and after blocks
- Each variants run before/after each test method
- All variants run before/after all the test methods in a single class

Hamcrest Matchers

Matcher

A simple and general blob of logic used in assertions.

Example: A map contains *this* key

Compositional

A Matcher can combine other Matchers

Example: A number is 5 or 6.

Summary

Summary

Problems

- Naming
- Behaviour not implementation
- DRY
- Diagnostics

Solutions

- Discipline when writing tests
- Junit and Hamcrest features