Effective Testing

COMSM0086

Dr Simon Lock & Dr Sion Hannuna

Testing

In other units you learn the theory of testing
In this unit you get the chance to apply it in practice

We've already encountered the JUnit test framework
You should already have a feel for the way it works
But it would be good to think about HOW to use it

Levels of Abstraction

Design your tests to target all levels:

Element testing: Exercise each & every line of code
For example, make sure all branches are executed

Unit Testing: Fully test operation of each method
Test different combinations of various parameters

Integration Testing: Test the high-level features
Ensure all user goals can be achieved using system

Equivalence Partitions

We don't need to test EVERY SINGLE input value

We can cluster input values together by similarity

Just pick representative values from each cluster

Drastically reduces the number of test cases

(Without impacting the test coverage)

Example Unit Test Scenario

Let's imagine we are writing a `Month` class:

```
Month currentMonth = new Month(MARCH,2024);
```

We want to Unit test the `getDayOfWeek` method:

```
String dayOfWeek = currentMonth.getDayOfWeek(14);
```

You could test method with ALL possible integers

```
for(int i=Integer.MIN_VALUE; i<=Integer.MAX_VALUE; i++)
```

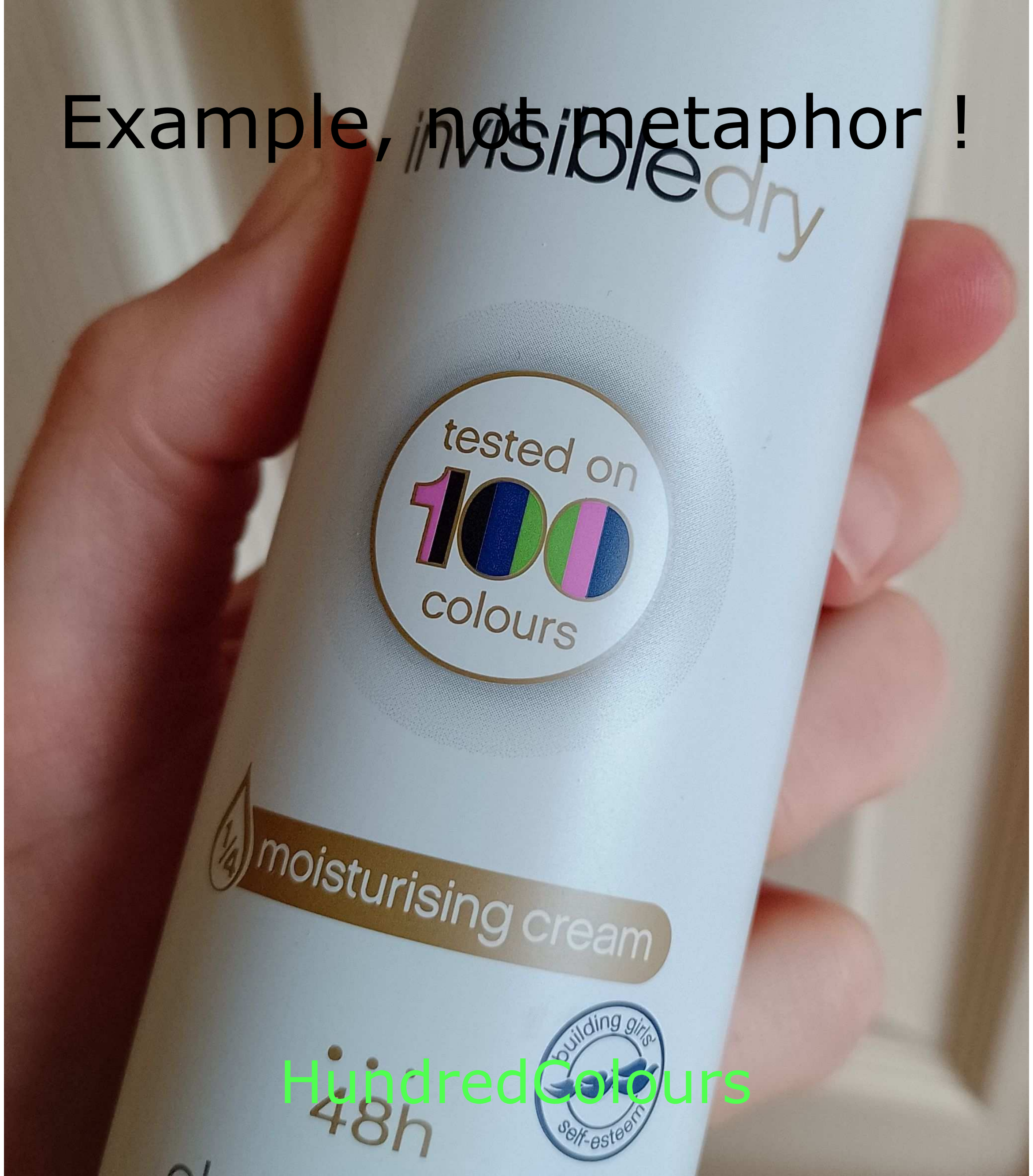
But would take a long time and is unnecessary
Instead we pick a set of representative numbers...

Suitable Equivalence Partitions

- Negative numbers: check invalid inputs are trapped
- Zero: a "special" edge case, in an EP on its own
- One: a "special" edge case, in an EP on its own
- Numbers from 2 to 28: should all work the same
- Borderlines: 29,30,31 may work (depends on month)
- Big numbers: >31 check invalid inputs are trapped

```
-2    0    1    5    31    32    50
```

Example, not metaphor !



HundredColours

My Test Cases

- Both extremes: Black and White
- A selection from the across greyscale range
- Primary light colours (Red, Green, Blue)
- Primary print colours (Cyan, Yellow, Magenta)
- A typical dark colour (low brightness)
- A typical light colour (high brightness)
- A typical dull colour (low saturation)
- A typical vibrant colour (high saturation)
- Different fabrics/material !
- Different lighting (sun, halogen, redhead, LED, UV)

But I digress, let's return to software

Top Ten Tips for Writing Test Cases

1. Keep tests short & simple - test just one thing
   So it's easier to keep track of what's been tested

2. It's fine to have more that one assertion in a test
   It's good to check feature from different angles

3. OK to have more than one test case in a method
   Clustering for same feature limits method count

4. Don't waste time with brute force coverage
   Use equivalence partitions to target test cases

5. Often need to perform a number of "setup" steps
   Which is where @BeforeEach is useful

Top Ten Tips for Writing Test Cases

6. Give test methods detailed and descriptive names
   So it is clear which test has failed

7. Include useful and informative failure comments
   So you know what failed and why

8. Be sure to cover all core feature (obviously)
   But also spend time checking the edge cases

9. Ensure software DOESN'T do what it SHOULDN'T
   As well as it DOES do what it SHOULD

10. Use sub-functions to perform common tasks
    "Divide and conquer" as with normal code

Failure Comments

All assertions take a "failure comment" parameter
A message which gets printed out if assertion fails

These can be extremely useful (if used well)
Try to write specific and detailed messages

Include parameters passed in to provide more detail

```
"Interpreter failed on " + incommingCommand
```

There is a lot of information printed during testing
Careful for-thought will make debugging a lot easier

Failure Comment Examples

Bad Failure Comments

```
error claiming cell
wrong exception occurred
incorrect return value
```

Good Failure Comments

```
cell owner still null after attempting to claim cell A2
expected an IOException, but got a NullPointer instead
expected 100 to be returned, but 10 returned instead
```

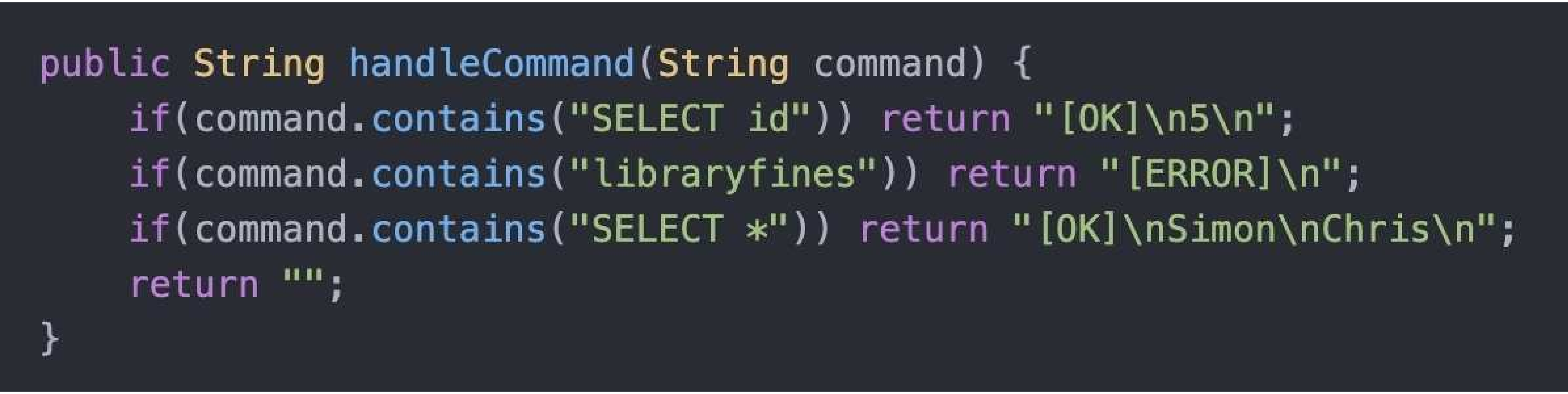
Note: Failure report includes the test method name
So be sure to give it a clear and descriptive name

Each Year Someone Always Asks...

Why can't we just have the marking tests ?

Well, lets consider the example test script:

ExampleDBTests



```
public String handleCommand(String command) {
    if(command.contains("SELECT id")) return "[OK]\n5\n";
    if(command.contains("libraryfines")) return "[ERROR]\n";
    if(command.contains("SELECT *")) return "[OK]\nSimon\nChris\n";
    return "";
}
```

Questions ?