Refreshing “Stuff” about Unit Testing

Burak Turhan

burak.turhan@oulu.fi

Unit Testing

- Validation
- Regression
- Sandboxing
- Experimentation
- Automation
- Fast feedback

Unit Testing Principles

Keep it simple: keep it safe!

Take baby steps

A test should be readable and meaningful

Tests are self-documenting

Again: Make your test readable!

Avoid magic numbers!

A test should not be overprotective

Remove redundant assertions

A test should have only one reason to fail!

Avoid split logic and behavior

100% Test coverage is NOT the goal

Focus on behaviour, NOT implementation

Tests should be maintainable!

Avoid duplication

Yes…Tests should be maintainable!

Do not use conditionals in your tests

Tests should not fail at random!

Isolate and handle the source of non-determinism

Never failing tests do not make sense!

Do not write tests without assertions

Tests should span best of both worlds

Test happy paths / Test sad paths

Test logic in production code?

Do not propagate your test logic into production

Tests should provide fast feedback

Re-think your test…

Test fixtures should be minimal

Remember?: Keep it simple

Unfinished tests should fail

Assert failures when leaving the office

Tests could be run in any order

Do not rely on test execution order

Tests should not depend on other tests

Tests should be isolated

Avoid tests that depend on expensive resources

Use Test Doubles: Stub, Mock, Fake etc.

Refactor (test code)

Test code IS code

Four-phase Test Design

Clear enough?

Keep it simple: keep it safe!

Take babysteps

A test should be readable and meaningful

Tests are self-documenting

Again: Make your test readable!

Avoid magic numbers!

A test should not be overprotective

Remove redundant assertions

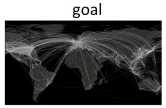
Test fixtures should be minimal

Remember?: Keep it simple

A test should have only one reason to fail!

Avoid split logic and behavior

100% Test coverage is NOT the goal



Focus on behaviour, NOT implementation

Tests should be maintainable!

Avoid duplication

Yes…Tests should be maintainable!

Do not use conditionals in your tests

Tests should not fail at random!

Isolate and handle the source of non-determinism

Never failing tests do not make sense!

Do not write tests without assertions

Tests should span best of both worlds

Happy tests, sad tests

Test logic in production code?

Do not propagate your test logic into production

Tests should provide fast feedback

Re-think your test…

Unfinished tests should fail

Assert failures when leaving the office

Tests could be run in any order

Do not rely on test execution order

Tests should not depend on other tests

Tests should be isolated

Tests might depend on external resources

Use Test Doubles: Stub, Mock, Fake etc.

Refactor (test code)

Refactor (test code)

Four-phase Test Design

Setup
Exercise
Verify
Teardown

Clear enough?

When writing tests

- Check a few representative cases (No overkill)
- Check boundary cases
- Check exceptional cases
- Focus on behaviour

Development Environment

Assert statements in JUnit

- fail()
- assertEquals()
- assertTrue()
- assertFalse()
- assertNotNull()
- assertNull()
- assertArrayEquals()
- assertNotSame()
- assertSame()

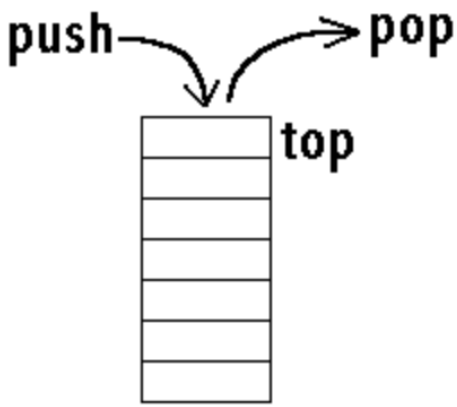
- All assertions have an optional first parameter of an “Assert Message” to display
- All assertions comparing two entities have the expected value first and the actual value after that

Example

```
public class PlayerTest {

    private Player player1;
    private Player player2;

    @Before
    public void setUp() {
        player1 = new Player("Steffi Graf");
        player2 = new Player("Arancha Sanchez");

    }

    @Test
    public void canCreateNewPlayers(){
        assertNotNull(player1);
        assertNotNull(player2);
        assertEquals("Steffi Graf", player1.getName());
        assertEquals("Arancha Sanchez", player2.getName());
    }
```

Example: Stack ADT

A stack is a LIFO sequence. Addition and removal takes place **only** at one end, called the top.



Operations

- `push(x)`: add an item on the top
- `pop`: remove the item at the top
- `peek`: return the item at the top (without removing it)
- `size`: return the number of items in the stack
- `isEmpty`: return whether the stack has no items