Alcor Training Walking

Andy Nyffenegger

A look back

- in 2013 I did the RomanNumber Kata on my commute to work
- Starting from scratch each time
- 6 times dec -> roman
- 28 times roman -> dec
- 2 times both ways

- In 2016 I started again
- 6 times roman -> dec

2013

- Work in advance:
  - Defining a mapping table with all the values upfront
- Big Steps:
  - Very few tests with numbers (99, 666, 999)
- Tests written last
- All test cases in one test, sometimes kind of parameterized
- Very little refactoring (as I remember)
- no use of IDE refactoring capabilities (Eclipse)
- Most were „operational“, but still had flaws (IXI)
- Code mostly hard to read

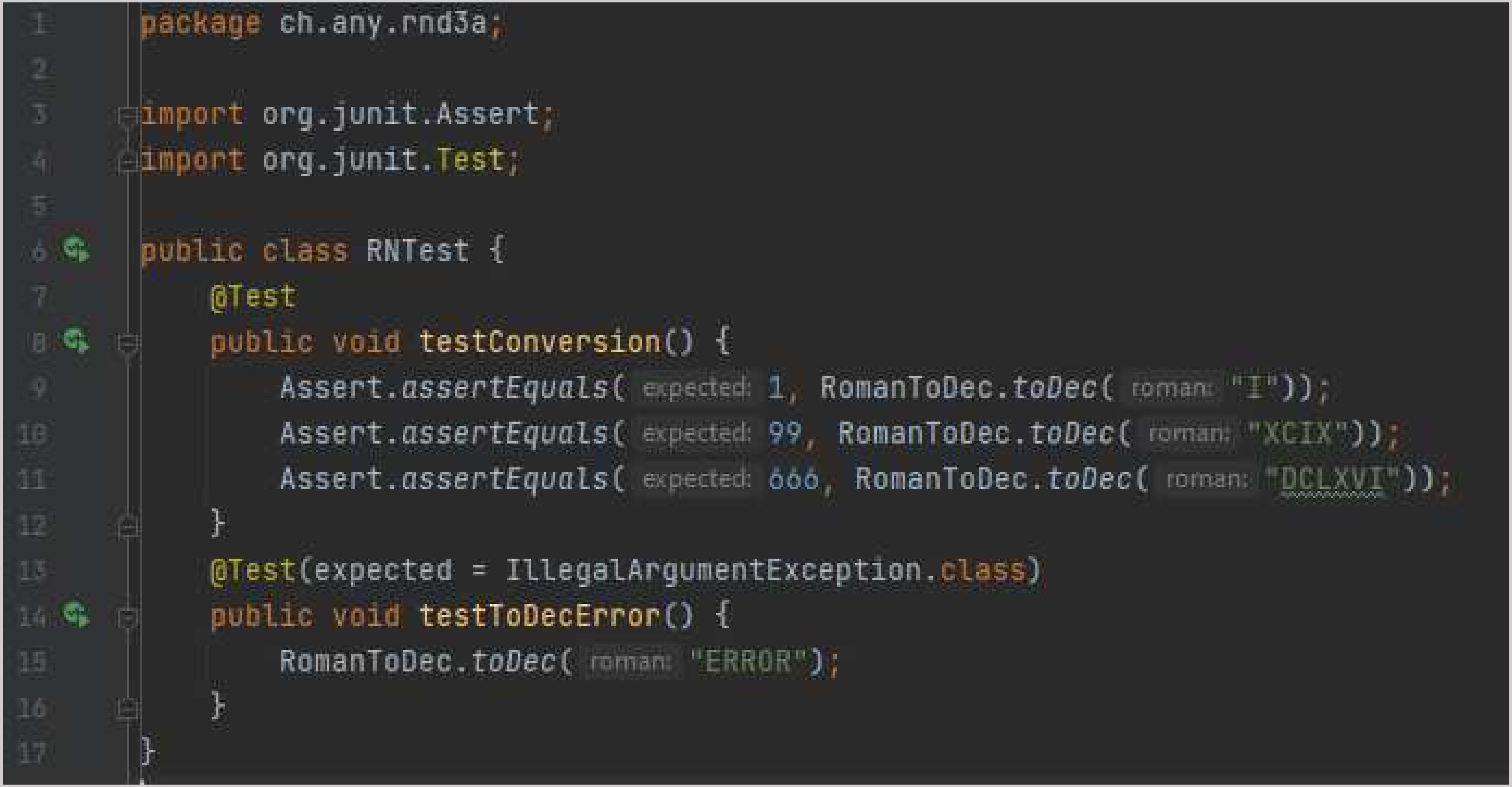
2016

- After first TDD training (1/2 day)
- No upfront coding
- Test first
- All single Tests (very repetitive)
- None of the implementations was finished (Trains got faster?)
- No refactoring
- No use of IDE capabilities

2021: Statistics?

- very time consuming
- only the first pass through a third of the code took several hours
- what focus?
- the code is about the same throughout
- Not very interesting for the audience

2021 let‘s have a closer look: Test first

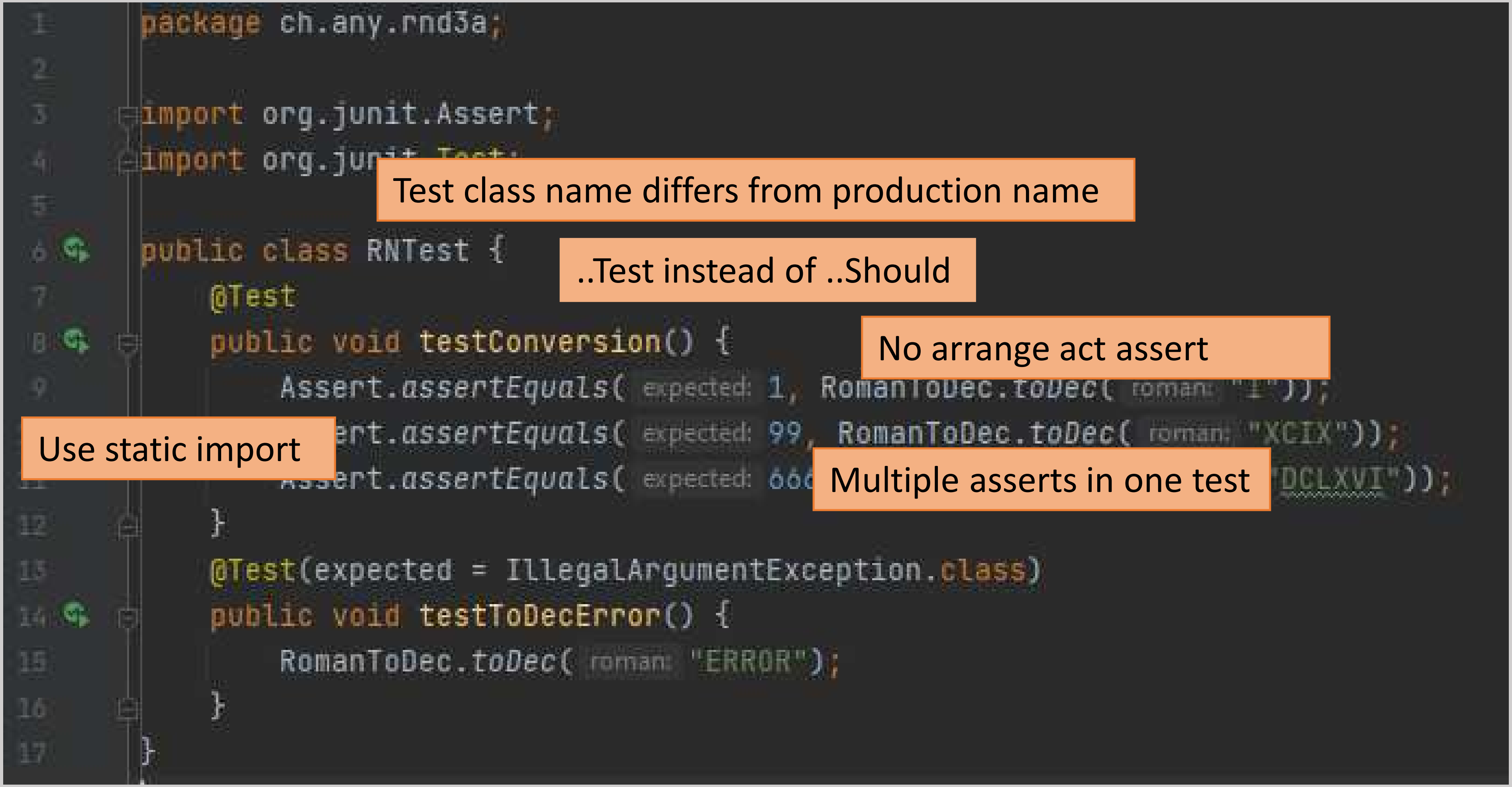


```
package ch.any.rnd3a;

import org.junit.Assert;
import org.junit.Test;

public class RNTest {
    @Test
    public void testConversion() {
        Assert.assertEquals( expected: 1, RomanToDec.toDec( roman: "I"));
        Assert.assertEquals( expected: 99, RomanToDec.toDec( roman: "XCIX"));
        Assert.assertEquals( expected: 666, RomanToDec.toDec( roman: "DCLXVI"));
    }
    @Test(expected = IllegalArgumentException.class)
    public void testToDecError() {
        RomanToDec.toDec( roman: "ERROR");
    }
}
```

2021 let‘s have a closer look: Test first



```
package ch.any.rnd3a;

import org.junit.Assert;
import org.junit.Test;

public class RNTest {
    @Test
    public void testConversion() {
        Assert.assertEquals( expected: 1, RomanToDec.toDec( roman: "I"));
           ert.assertEquals( expected: 99, RomanToDec.toDec( roman: "XCIX"));
        Assert.assertEquals( expected: 66                              "DCLXVI"));
    }
    @Test(expected = IllegalArgumentException.class)
    public void testToDecError() {
        RomanToDec.toDec( roman: "ERROR");
    }
}
```

Test class name differs from production name

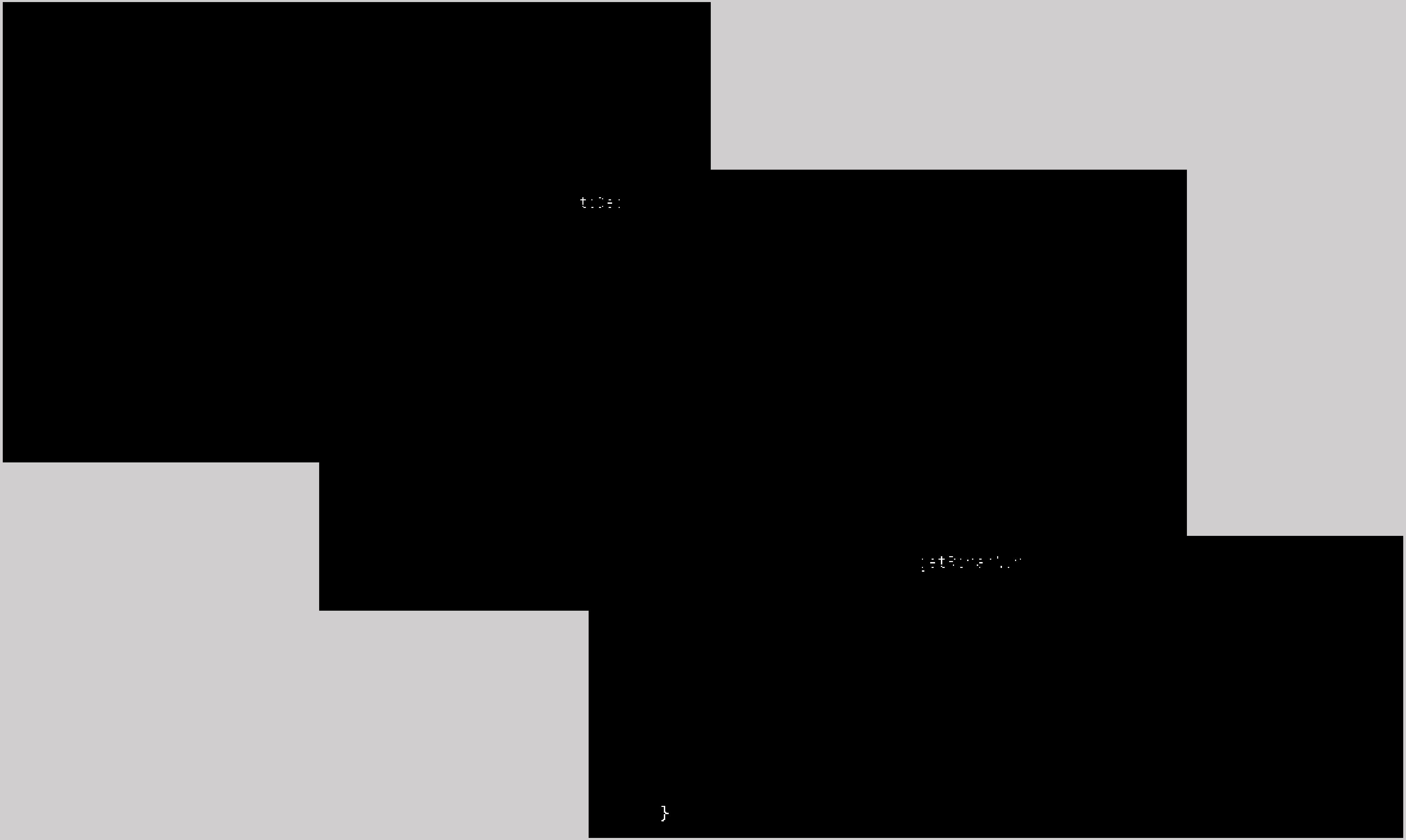
..Test instead of ..Should

No arrange act assert

Use static import

Multiple asserts in one test

2021 closer look: Implementation



```
[illegible]
[illegible]
    }
```

2021 All Object Calistenics rules violated?

[redacted]

☑ No abbreviations

☑ First class collections (wrap all collections)

☑ Wrap all primitives and strings

☑ Only one level of indentation per method

☑ Keep all entities small

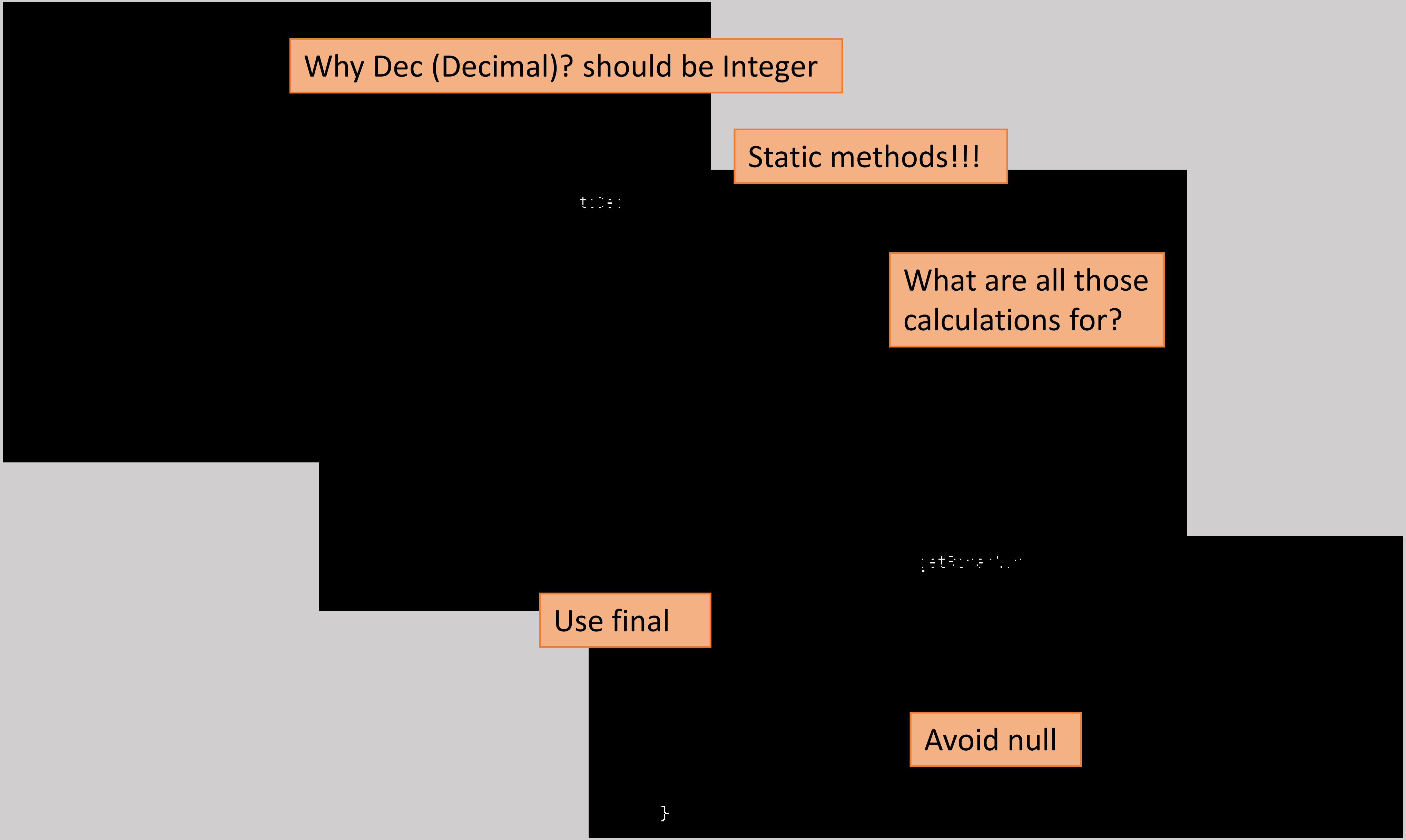
☑ Don’t use the ELSE keyword

☒ No public getters/setters/properties

☒ Only one dot per line

☒ No classes with more than two instance variables

2021 What else is wrong?



Why Dec (Decimal)? should be Integer

Static methods!!!

What are all those calculations for?

Use final

Avoid null

2021 The solution

Refactoring!

2021 The solution: Refactoring

Have a go at it, I will gladly provide the code.

Remark

When doing a kata like this:

- don‘t follow a fix time restriction
- finish the code properly
- have a fixed set of rules
- skip on rules rather then code quality