AARHUS UNIVERSITET

Software Engineering and Architecture

Test-Driven Development

What is it?

- ***Test-Driven Development***

- *It is a* ***systematic programming technique***
  - Focus on
    - **Continued Speed**
    - **Reliability**

- It is not a *testing technique, despite the name*
  - Tests are *means*, not *an end in itself*
  - But they are sooo nice to have afterwards…

AARHUS UNIVERSITET

Systematic Programming

- TDD replaces *tacit knowledge* with a formulated process
  - *The mantra*
    - ***Clean code that works***
  - *The values*
    - The **four values** that abstractly define what we want
  - *The rhythm*
    - The **five steps** in each highly focused and fast-paced iteration.
  - *Testing principles*
    - The **testing principles** that defines a term for a specific action to make in each step
    - Form: Name - Problem – Solution

AARHUS UNIVERSITET

The Mantra

Clean code that works

- *To ensure that our software is reliable all the time*
  - *"Clean code **that works**"*
- *To ensure fast development progress*
- *To ensure that we dare restructure our software and its architecture*
  - *"**Clean code** that works"*
  - *Clean = understandable and changeable*

AARHUS UNIVERSITET

The Values

- ***Keep focus***
  - Make one thing only, at a time!
  - *Often*
    - *Fixing this, requires fixing that, hey this could be smarter, fixing it, ohh – I could move that to a library, hum hum, …*

- ***Take small steps***
  - Taking small steps allow you to backtrack easily when the ground becomes slippery
  - *Often*
    - *I can do it by introducing these two classes, hum hum, no no, I need a third, wait…*

AARHUS UNIVERSITET

The Values

- ***Speed***
  - You are what you do! Deliver every 14 days!!!
  - *Often*
    - *After two years, half the system is delivered, but works quite in another way than the user anticipate/wants…*
  - Speed, not by being sloppy but by making less functionality of superior quality!
- ***Simplicity***
  - Maximize the amount of work *not done!*
  - *Often*
    - *I can make a wonderful recursive solution parameterized for situations X, Y and Z (that will never ever occur in practice)*

AARHUS UNIVERSITET

The Iteration Skeleton

- Each TDD iteration follows the Rhythm

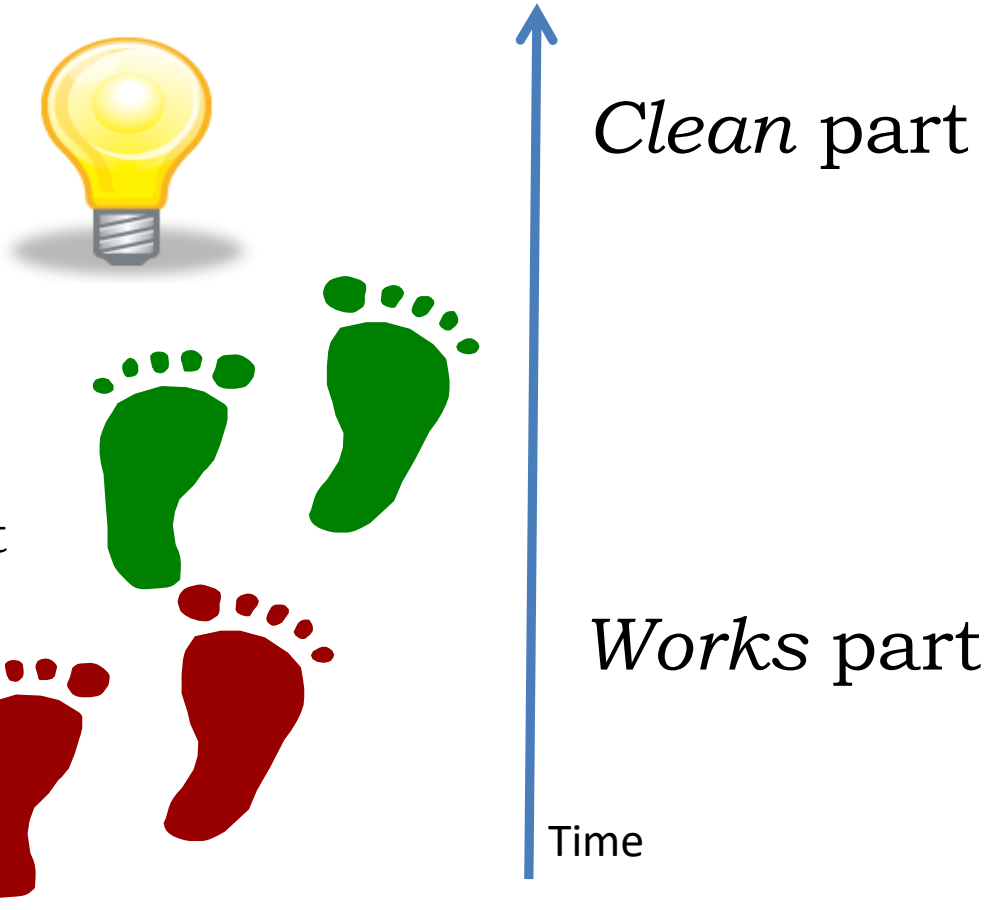
The TDD Rhythm:

1. Quickly add a test
2. Run all tests and see the new one fail
3. Make a little change
4. Run all tests and see them all succeed
5. Refactor to remove duplication

- (6. All tests pass again after refactoring!)

The Rhythm: Red-Green-Refactor

- The Rhythm



The Three Starter Principles

TDD Principle: **Automated Test**

How do you test your software? Write an automated test.

TDD Principle: **Test First**

When should you write your tests? Before you write the code that is to be tested.

TDD Principle: **Test List**

What should you test? Before you begin, write a list of all the tests you know you will have to write. Add to it as you find new potential tests.

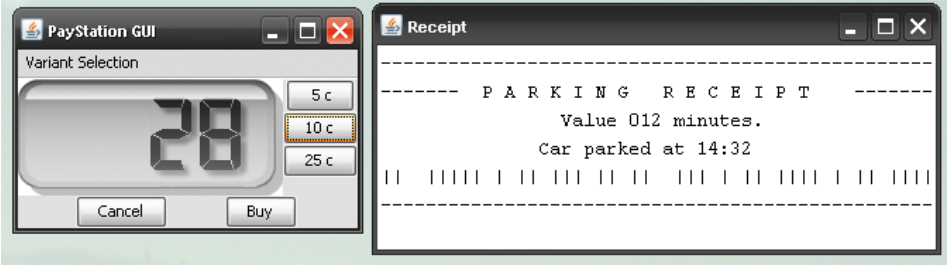
AARHUS UNIVERSITET

Enough…

Learning by Doing…

You are all employed today

- Welcome to *PayStation Ltd.*
- We will develop the main software to
  - [Demo]



AARHUS UNIVERSITET

Case: Pay Station

- Welcome to *PayStation Ltd.*
- Customer: AlphaTown

- Requirements
  - accept coins for payment
    - 5, 10, 25 cents
  - show time bought on display
  - print parking time receipts
  - US: 2 minutes cost 5 cent
  - handle buy and cancel

Stories

Story 1: Buy a parking ticket. A car driver walks to the pay station to buy parking time. He enters several valid coins (5, 10, and 25 cents) as payment. For each payment of 5 cents he receives 2 minutes parking time. On the pay station's display he can see how much parking time he has bought so far. Once he is satisfied with the amount of time, he presses the button marked "Buy". He receives a printed receipt, stating the number of minutes parking time he has bought. The display is cleared to prepare for another transaction.

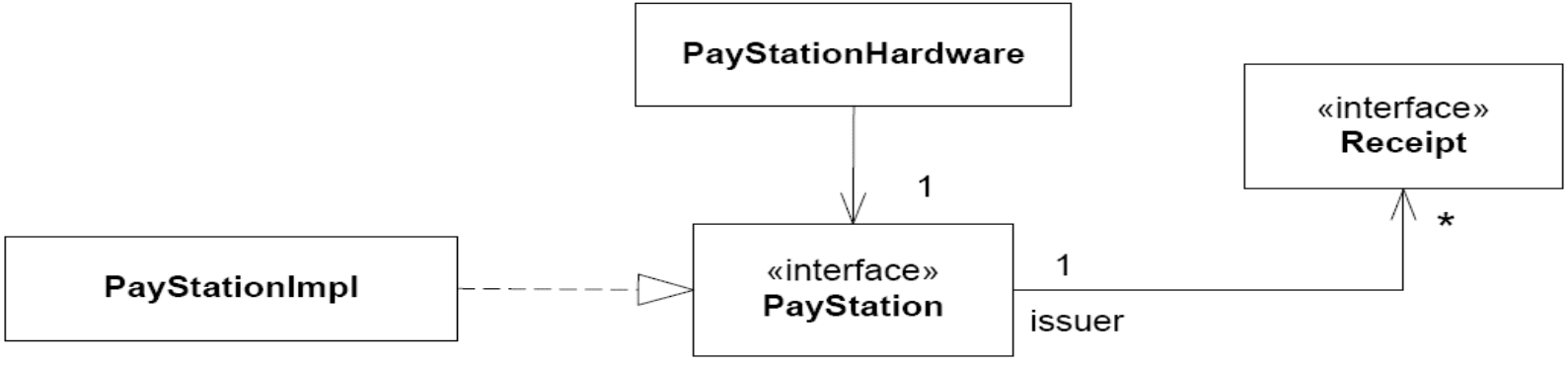
Story 2: Cancel a transaction. A driver has entered several coins but realize that the accumulated parking time shown in the display exceeds what she needs. She presses the button marked "Cancel" and her coins are returned. The display is cleared to prepare for another transaction.

Story 3: Reject illegal coin. A driver has entered 50 cents total and the display reads "20". By mistake, he enters a 1 euro coin which is not a recognized coin. The pay station rejects the coin and the display is not updated.

AARHUS UNIVERSITET

Design: Static View

- For our purpose the design is given…
  - Central *interface* **PayStation**



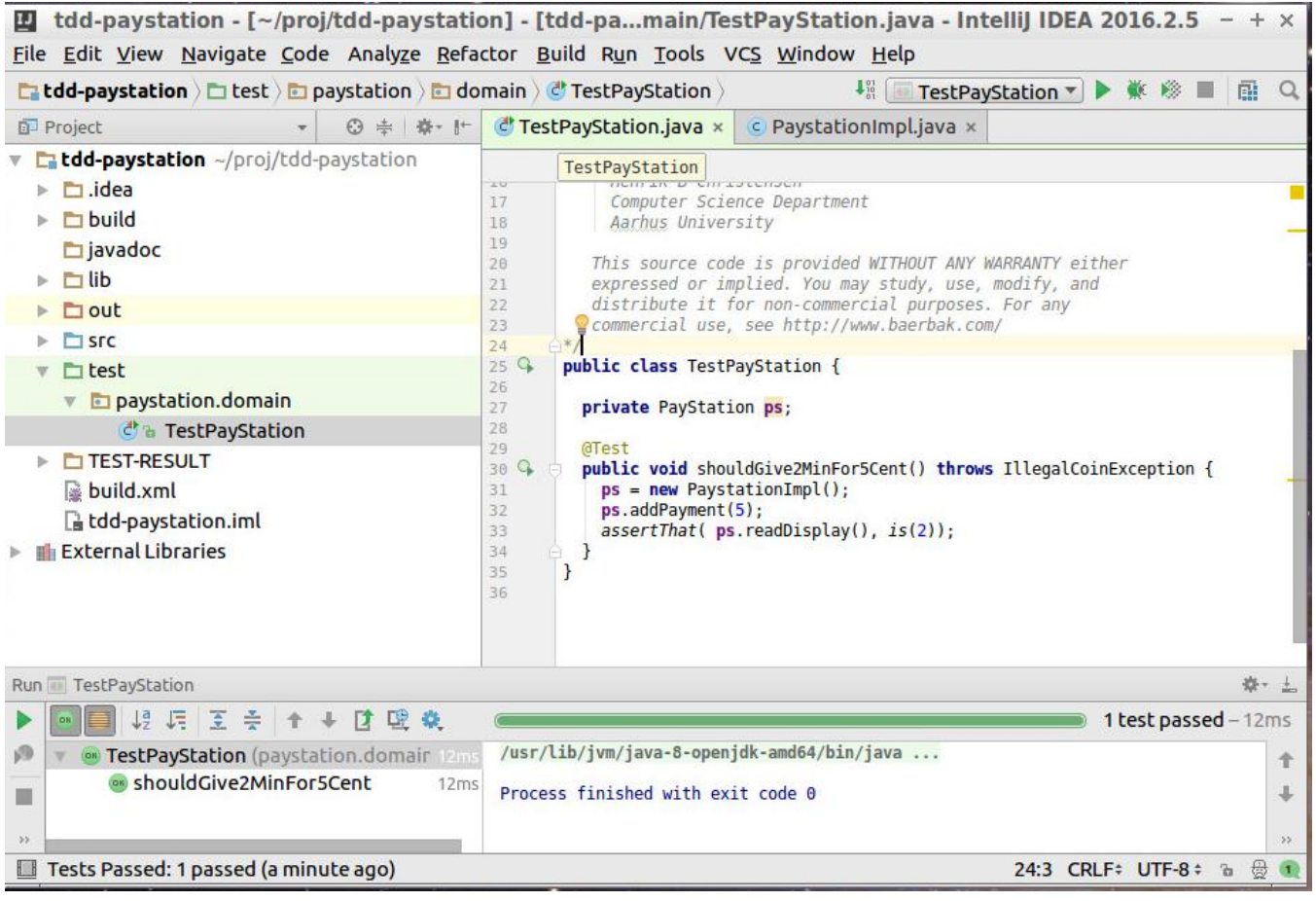
My Testlist

- From the book

* accept legal coin
* 5 cents should give 2 minutes parking time
* reject illegal coin
* readDisplay
* buy produces valid receipt
* cancel resets pay station

AARHUS UNIVERSITET

And on…

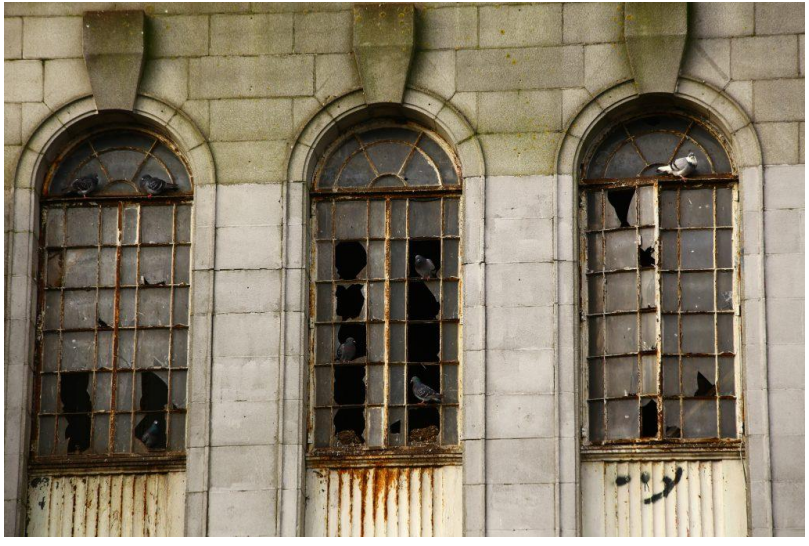


AARHUS UNIVERSITET

Central Principles Overview

Refactoring

- *Fix your broken windows*
  - *Clean up **now*** or your code *will* deteriorate quickly!



Definition: Refactoring

Refactoring is the process of modifying and restructuring the source code to improve its maintainability and flexibility without affecting the system's external behavior when executing.

TDD Principle: **One Step Test**

Which test should you pick next from the test list? Pick a test that will teach you something and that you are confident you can implement.

TDD Principle: **Fake It ('Til You Make It)**

What is your first implementation once you have a broken test? Return a constant. Once you have your tests running, gradually transform it.

TDD Principle: **Triangulation**

How do you most conservatively drive abstraction with tests? Abstract only when you have two or more examples.

TDD Principle: **Obvious Implementation**

How do you implement simple operations? Just implement them.

TDD Principle: **Isolated Test**

How should the running of tests affect one another? Not at all.

TDD Principle: **Evident Tests**

How do we avoid writing defective tests? By keeping the testing code evident, readable, and as simple as possible.

TDD Principle: **Evident Data**

How do you represent the intent of the data? Include expected and actual results in the test itself, and make their relationship apparent. You are writing tests for the reader, not just for the computer.

TDD Principle: **Representative Data**

What data do you use for your tests? Select a small set of data where each element represents a conceptual aspect or a special computational processing.

TDD Principle: **Assert First**

When should you write the asserts? Try writing them first.

TDD Principle: **Break**

What do you do when you feel tired or stuck? Take a break.

TDD Principle: **Do Over**

What do you do when you are feeling lost? Throw away the code and start over.

AARHUS UNIVERSITET

Outlook

Tests as Specifications

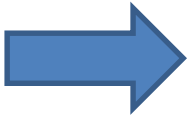
AARHUS UNIVERSITET

Insight

- An early insight was that *the tests* are actually a *specification of the requested behaviour*
  - *Aka* a **requirements specification!**
- What if costumers could write them themselves !?!?

- Gave rise to much experimentation
  - Behaviour-Driven development

As a [X]
I want [Y]
so that [Z]



Given some initial context (the givens),
When an event occurs,
then ensure some outcomes.

Spock

- A lot of frameworks were developed
  - Many are... Well... IMO somewhat cumbersome to use
- Spock I find is pretty easy to get going, though
  - Only problem, you code in *groovy*, not Java...

```
def "should handle multiple coins"() {
  given: "I have a new paystation instance"
  ps = new PayStationImpl()

  when: "I enter valid coins"
  ps.addPayment( coinValue: 5)
  ps.addPayment( coinValue: 10)
  ps.addPayment( coinValue: 25)
  ps.addPayment( coinValue: 25)

  then: "the display prints the right value"
  ps.readDisplay() == (2*25+5+10)/5*2
}
```

```
def "should handle all three coin types"() {
  given: "I have a new paystation instance"
  ps = new PayStationImpl()

  expect: "that I can add a coin and see the right time"
  ps.addPayment(coin)
  ps.readDisplay() == calculatedParking

  where: "I try all the three types of coins"
  coin            || calculatedParking
  5               || 2
  10              || 4
  25              || 10
}
```