Program

General cut-up

- Day 1: Mainly on writing clean and testable code.
- Day 2: Mainly on writing better test code.

Sources and reading material for after the training:

- Book: Clean Code
- Book: Implementation Patterns
- Book: Design Patterns
- Web: https://sourcemaking.com/
- Web: https://www.cs.helsinki.fi/u/luontola/tdd-2009/ext/ObjectCalisthenics.pdf
- Paper: http://www.literateprogramming.com/mccabe.pdf

Module 0 - Introduction and Clean code (~30m).

0.1 General introduction

0.2 Why?

- Reference to 'There will be code'
- Code will be read more times than written.
- Readability. Affects the 'cost of change'.
- Getting stuck affects your state of mind.

0.3 Clean code by R.C. Martin.

- Readability
- Isolation of functionality
- No duplication (DRY)
- Less code.

Module 1. Object Oriented Programming (~3h)

1.1 Brief intro into Object Oriented programming (45m - 1h)

- Differences with imperative / functional
- Classes, Methods, Fields;
- Public, private, protected;

1.2 Class Design (30m)

- Hide implementation.
- Encapsulation and Cohesion.

1.3 Inheritance (30m)

- Abstract

1.4 Interfaces (30m)

- "All problems in computer science can be solved by another level of indirection" - David Wheeler (https://en.wikipedia.org/wiki/Indirection) (held).

Module 2 - General Software Development (~2h).

2.1 Attitude (10m - 15m)

- Group effort.
  - Creating software is non-reproduce-able process. Therefor, everything is always arguable. Just keep using your head.
- Be strict on the little things. It is a good and easy start
  - For example code formatting.
- Boy scout rule.
  - Always leave the campsite cleaner then you how you found it.

2.2 Clean code (1h).

The book `Clean Code`, divided by chapter

1. Meaningful names

- Theory
- Example by code. Include how it helps in testing.
- Example of naming convention (controller, service, repository/dao, client)
- Hands-on by rewriting a 'cryptic' Java class - make the unit tests run again.

2. Functions
3. Comments
4. Formatting
5. Object & data structures
6. Error handling
7. Boundaries
8. Unit tests
9. Classes

2.3 DRY (45m).

- Theory
- Example by code. Include how it helps in testing.
- Hands-on by rewriting one JUnit test class with some duplication in there.

2.4 Primitive Obsession (1h)

- https://www.cs.helsinki.fi/u/luontola/tdd-2009/ext/ObjectCalisthenics.pdf (subject)

Module 3. SOLID (~2h)

3.1 Single Responsibility Principle (30m)

- Helps in the readability and makes code easier to understand.
- Hands-on: Refactor a Java class. Cut it up in methods.
  - For example, a class that determines if the thermostat should heating should turn on based on a heating plan, the current date and time and the current temperature.
- Evaluate / present solutions

3.2 Open-closed principle (30m)

3.3 Liskov Substitution Principle (20m)

- Explain the Liskov Substitution Principle
- Show an example where the liskov substitution principle does not hold and the consequences of this.

3.4 Interface segregation principle (30m)

- What is it and why is this important?
- Hands-on: refactor a code example that contains one big interface and split up to multiple ones.

3.5 Dependency Inversion (30m):

- Show problems for testing code that is not using DI.
- Hands-on:
  - Write unit tests for a code example that does not use dependency injection
  - See that it "fails"
  - Refactor the code to use dependency injection and introduce mocking using Mockito
  - Rewrite the test.

Module 4. Patterns (~2h)

For each of the patterns:

- What is the problem
- How does the pattern solve the problem

Module 4.1. Adapter pattern

Module 4.2. Factory pattern

Module 5. Measuring / improving code quality (~2h)

5.1 Cyclomatic Complexity

- What is cyclomatic complexity and how can it be helpful
- Reference for the number of tests.
- Hands-on:
    - Determine the complexity of a few methods (by hand?)

---

6 Testing (~4h)

6.1 Building a simple REST API

6.2 What makes a good unit test

- Arrange act assert
- Testing one thing
- Test every path (code coverage)
- Independence

6.3 Integration test an API

- How to setup
- Create your own integration test

6.4 Continuous integration

- Setting up a CI pipeline for automation (very briefly)

6.5 (Optional) Acceptance testing using Selenium

- Create a simple acceptance test using selenium