# Clean Code Fundamentals

Functions

## Pre-work

- Video: https://cleancoders.com/episode/clean-code-episode-3
- Exam: https://cleancoders.com/episode/clean-code-episode-3/exam

## Timetable

| Activity | Time |
|---|---|
| Warmup | 5 min |
| Exercise 1 | 20 min |
| Exercise 2 | 20 min |
| Exercise 3 | 20 min |
| Wrap up | 5 min |

## Warmup

- What are some “landmarks” you look for when you’re reading code?
    - Type in the meeting chat

## Exercise 1

- Prompt
  - How to safely refactor code without breaking it? Discuss possible strategies.
  - What to do if code is not covered by tests?
- Time limit: 10 minutes

## Safe refactoring

- **Refactoring** is a process of
  - Restructuring existing code
  - Without changing its external behavior
- Safe refactoring
  - Put the system under a test
  - Run tests often
  - Understand test coverage to avoid blind spots

## Common approach to working with legacy code

- Create a “characterization test” that captures the current behavior
- Restructure the code to enable testing of a specific part of the code
- Write a test for wanted behavior that fails
- Implement the behavior to make the test pass

## Characterization test

- This test has many names
  - “Characterization test”
  - “Golden Master”
  - “Snapshot test”
- Characterization test checks general behavior
  - Uses fixed seed for program inputs
  - Checks that the output is the same as the previous run

## Exercise 2

- Prompt
  - What code behavior do you find suspicious and why?
  - What “code smells” do you find useful and why?
- Time limit: 10 minutes

## Code smells catalog

- Bloaters
  - Long method
  - Long parameter list
  - Data clumps
  - Primitive obsession
  - Long class
- Object-Orientation Abusers
  - Switch statements
  - Refused bequest
  - Alternative classes with different interfaces
  - Temporary field
- Change Preventers
  - Divergent change
  - Shotgun surgery
  - Parallel inheritance hierarchies
- Dispensables
  - Lazy class
  - Data class
  - Comments
  - Duplicate code
  - Dead code
  - Speculative generality
- Couplers
  - Feature envy
  - Inappropriate intimacy
  - Incomplete library class
  - Message chains
  - Middle man

## “Feature envy” code smell

- Definition
  - A method accesses the data of another object more than its own data
- Possible reason
  - After fields move to a data class/structure
- Treatment
  - Move operations on data to the class as well

## Exercise 3

- Prompt
  - How to define if a function is doing “one thing”?
- Time limit: 10 minutes

## Where classes hide

- Classes hide in long functions with many local variables
- Functions that fill the screen are likely doing more than one thing
- Functions crossing levels of abstraction

## “Extract class” refactoring

- Create characterizations test – run often
- Extract function body to a new class’s `invoke` method
- Extract local variables to fields
- Extract methods or new classes
- Repeat until you can’t extract anymore

## Wrap up

- Functions should be small
- Functions should do one thing
- Functions should have one level of abstraction
- Functions should have descriptive names

## What is next?

- Expect an e-mail with instructions for upcoming coding dojo

## Final words

*Always leave the code better than you found it.*
*– The Software Craftsmanship Rule*