Refactoring

- What is it
- Main refactors
- Exercise
- Guidelines
- Exercise

“Refactor, not because you know the abstraction, but because you want to find it.”

Martin Fowler

What is it?

What it isn't: Change all the code while having breaking tests, changing the design without any safety net, spend nights and days doing it alone, solving a merge conflict and then put everything together

The idea behind refactoring is to change the design without changing the behavior of your code. We should refactor both production and test code, because having solid and well-designed tests is as important as production code.

Do we know when to refactor?

- **DRY**: When we find duplication in our code.
- **Calisthenics:** When we break any object calisthenics rule

Main refactors

- **Rename**: Change the name of classes, methods, variables, etc.
- **Extract:** Extract a class (or methods or variables ...), creating a new abstraction.
- **Inline:** The inverse of extract. Inline a method (or variable), deconstructing an abstraction.
- **Move:** Move a class (or methods or variables...) to some other place in the codebase.
- **Safe delete:** Delete code and its usages in the code base.

Exercise: Refactoring golf

Refactoring Golf is a game designed to stretch your refactoring muscles and get you to explore your IDE to see what's really possible using shortcuts and automation.

Your goal is to refactor the Hole-X code to look like the Hole X+1 code. You must aim to do it in as few "strokes" as possible.

Your pairing partner should carefully score you as follows:

- Zero points for code formatting (e.g., deleting whitespace or optimizing imports).
- 1 point for every change made to the code using a shortcut or automated IDE feature (e.g., an automated refactoring, code template, or Find/Replace)
- 2 points for every manual edit. Note that a single "edit" could cover multiple lines of code.
- Double points for every change made while the code cannot pass the tests after the previous change.

Allow yourselves a maximum of 2 attempts at each round to determine your best score.

Refactoring guidelines

- **Stay in green:** tests should be testing behavior, so there is no reason why we should break any tests during refactoring. If you break any, undo, go back to green and start over
- **Commit as often as possible:** Make use of source control to safely and quickly revert to a known good point, provided we took small steps and committed frequently
- **Refactor readability before design:** Prefer small improvements in code readability in order to proceed with later design changes.

Refactoring guidelines: Readability

- **Format:** Format consistently and don't force the reader to waste time due to inconsistent formatting. Keep an eye on code conventions.
- **Rename:** Rename bad names, variables, arguments, methods, etc. Make abbreviations explicit.
- **Remove:** Delete unnecessary comments, delete dead code
- **Extract:** Constants from magic numbers and strings. Conditionals
- **Reorder:** Refine scope for improper variable scoping, and make sure variables are declared close to where they are used.

Refactoring guidelines: Design

- Extract private methods from deep conditionals.
- Extract smaller private methods from long methods, and encapsulate cryptic code in private methods.
- Return from methods as soon as possible.
- Encapsulate where we find missing encapsulation.
- Remove duplication

Parallel change (expand, migrate, contract)

This is a useful technique to implement breaking changes safely while staying in green. It consists of 3 steps

- **Expand:** Introduce new functionality by adding new code instead of changing existing one. Start from the tests, following the same TDD practices you've learned. It's allowed to duplicate tests
- **Migrate:** Deprecate old code and allow clients to migrate to new expanded code, or change client code to point to new code
- **Contract:** Once all client code is migrated to new code, remove old functionality by removing deprecated code and its tests.

Exercise: Tennis refactoring

Imagine you work for a consultancy company, and one of your colleagues has been doing some work for the Tennis Society. The contract is for 10 hours billable work, and your colleague has spent 8.5 hours working on it. Unfortunately he has now fallen ill. He says he has completed the work, and the tests all pass. Your boss has asked you to take over from him. She wants you to spend an hour or so on the code so she can bill the client for the full 10 hours. She instructs you to tidy up the code a little and perhaps make some notes so you can give your colleague some feedback on his chosen design. You should also prepare to talk to your boss about the value of this refactoring work, over and above the extra billable hours.

This exercise was created by Emily Bache, who you can find in many videos online going through different refactoring exercises