Refactoring
Lecture 3:
OO Refactorings
DAT159/H18
Volker Stolz

Supported by the bilateral SIU/CAPES project “Modern Refactoring” 2017/18
Move Method

• Sometimes things are just in the wrong place. (Smell: Feature Envy)
• Copy method to new class, delete method?
• Obviously compilation errors, we need to fix all invocations.
• Question: How do we find all of them?
• Also: API change!
Move Method, Formally

Move Method $m_a$ to $m_b$:

[1] $\exists (M, U_{T_1}, U_{T_2}) = \text{matching}(m_a, b, m_b, b) \land m_a \in M^- \land m_b \in M^+ \land m_a.c \neq m_b.c \land |M| > |U_{T_1}| \land |M| > \sum \text{import type}(r_{d_b}, m_a.c)$

$\text{subtype}(m_b, c, m_a, c) \Rightarrow \text{Push Down Method}$

$\text{subtype}(m_a, c, m_b, c) \Rightarrow \text{Pull Up Method}$

Tsantalis et al., “Accurate and efficient refactoring detection in commit history”. ICSE 2018
Pull Up Field

- Moving a field upwards in the inheritance hierarchy
- How far “up”? Visibility?
- Q: Can you come up with a "cooking recipe" for this refactoring in Java? Will the source code still compile at each intermediate stage?

![Diagram showing the refactoring process from Salesman and Engineer extending Employee name to be pulled up to Employee]
Pull Up Field: Mechanics [Fowler]

- Inspect all uses of the candidate fields to ensure they are used in the same way.
- If the fields do not have the same name, rename.
- Compile & test (Why — what did we do?),
- Create a new field in superclass (check visibility).
- Delete the subclass fields.
- Compile & test.

Manual/Subjective!
Pull Up Method

• Compare with “Pull Up Field”.

• Usually we don’t *shadow* fields, but in object orientation, we make frequent reuse of methods: `@override` important for specialisation.

• Before we start worrying, let’s look at a normal example:

![Diagram showing pull up method example]
Push Down Field

- A field is used only by some subclasses.
- Eliminates kind of “dead code” (many classes see it, but don’t need it)
Extract Class

- You have one class doing work that should be done by two.
- Create a new class and move the relevant fields and methods from the old class into the new class.
Be aware of your surroundings!

- Moving (deleting) public/protected members changes the API
- Q: When does this become a problem (even though your unit-tests are green)?
- Libraries!
Replace Type Code With Polymorphism

You have a conditional that chooses different behaviour depending on the type of an object.

double getSpeed() {
    switch (_type) {
        case EUROPEAN:
            return getBaseSpeed();
        case AFRICAN:
            return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
        case NORWEGIAN_BLUE:
            return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException("Should be unreachable");
}
Anything Else?

• Most common refactoring? Fields/Methods/Classes/Packages (and we didn’t even talk about it!)

• Refactoring dialogs have plenty of options…but nobody cares!