Refactoring
Lecture 3:
OO Refactorings

DAT159/H18

Volker Stolz

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

Høgskulen på Vestlandet

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!

```
class Project {
        Person manager;
        Person[] participants;
        void shouldntReallyBeHere() {
                System.out.println(manager.id);
                System.out.println(manager.name);
        }
}
```

```
class Person {
        boolean participate(Project p) {
          for(int i=0; i<p.participants.length; i++) {
                if (p.participants[i].id == id)
                        return(true);
          }
          return(false);
        }
}
```

Move Method, Formally

input : e – target expression of type E
: m – an instance method in a class C
output: m moved to E, updated method call
1 declare a method n in E with same signature and –
2 replace invocation of m with e.n
3 $V \leftarrow$ all explicit –
4 if V

- Examine all features used by the source method that are defined on the source class. Consider whether they also should be moved
  - ⇒ *If a feature is used only by the – might as well –*

Tsantalis et al., "Accurate and efficient refactoring detection in commit history". ICSE 2018

[L

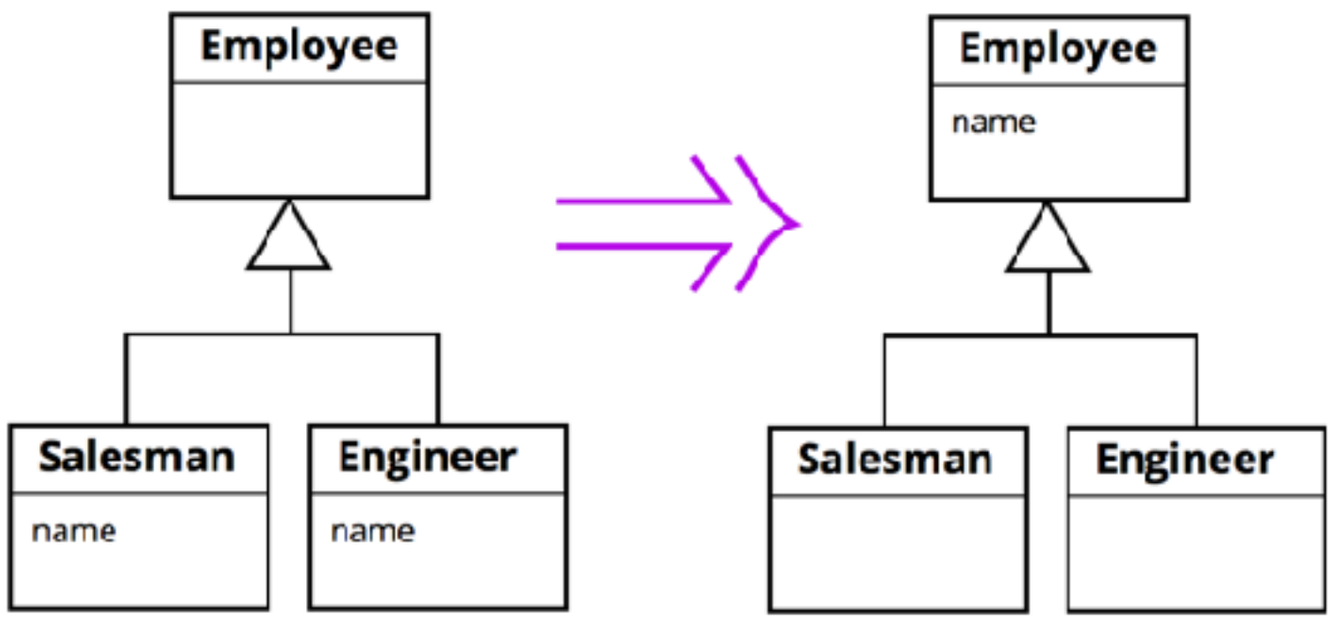
Move Method m_a to m_b:

$$\exists\,(M, U_{T_1}, U_{T_2}) = \texttt{matching}(m_a.b, m_b.b) \mid m_a \in M^- \wedge m_b \in M^+ \wedge m_a.c \neq m_b.c \wedge |M| > |U_{T_1}| \wedge |M| > |U_{T_2}| \wedge (td_a, td_{a'}) \in TD^= \wedge m_a \in td_a \wedge (td_b, td_{b'}) \in TD^= \wedge m_b \in td_{b'} \wedge (\texttt{importsType}(td_{a'}, m_b.c) \vee \texttt{importsType}(td_b, m_a.c)) \wedge \boxed{\texttt{subType}(m_a.c, m_b.c)} \Rightarrow \textsc{Pull Up Method} \quad \boxed{\texttt{subType}(m_b.c, m_a.c)} \Rightarrow \textsc{Push Down Method}$$

- *… can store … but you can also make it … refactored enough to remove it.*
- … the source method into a delegating method.
- Compile and test.
- Decide whether to remove the source method or retain it as a delegating method.
  - ⇒ *Leaving the source as a delegating method is easier if you have many references.*

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?



Pull Up Field: Mechanics [Fowler]

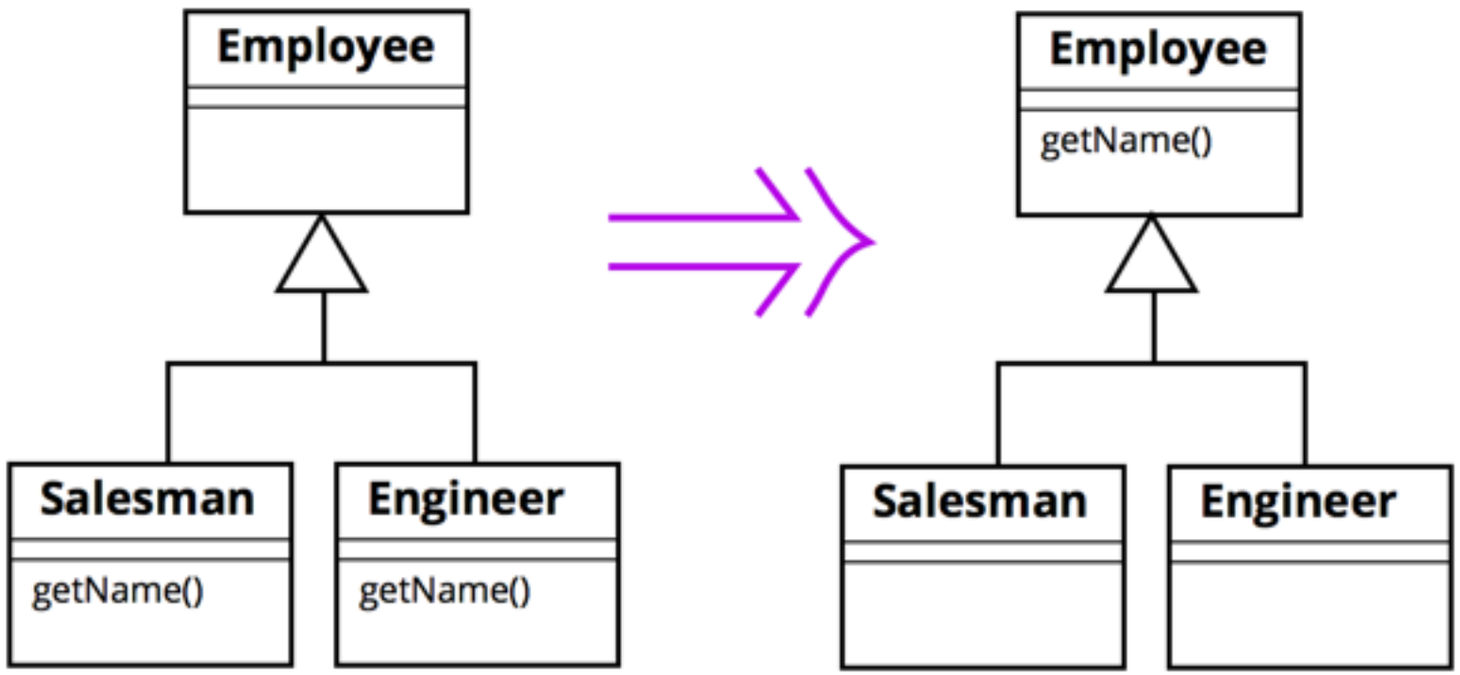
- Inspect all uses of the candidate fields to ensure they are used in the same way.
- If the fields do not have the same name, rename.
- Compile & test (*Why — what did we do?*),
- Create a new field in superclass (check visibility).
- Delete the subclass fields.
- Compile & test.

Manual/ Subjective!

Pull Up Method

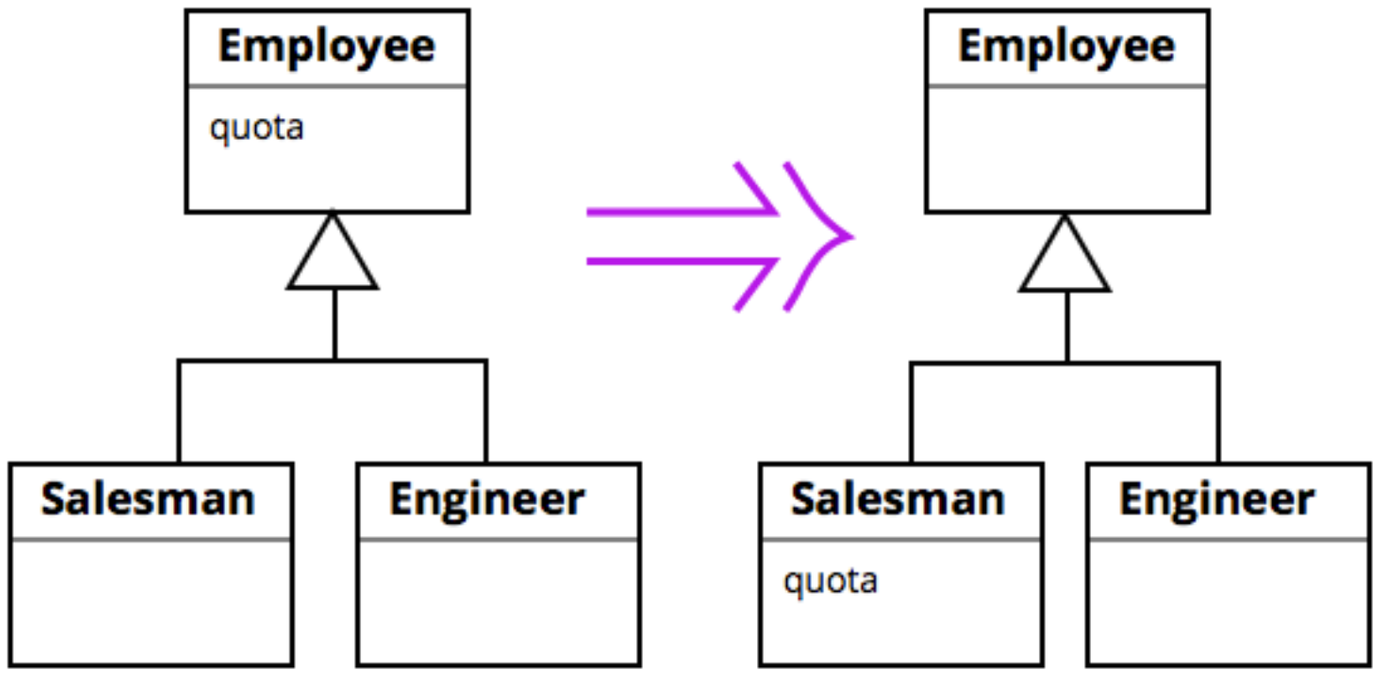
- Compare with “Pull Up Field”.
- Usually we don’t *shadow* fields, but in object orientation, we make frequent reuse of methods: `@override` important for specialisation.

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



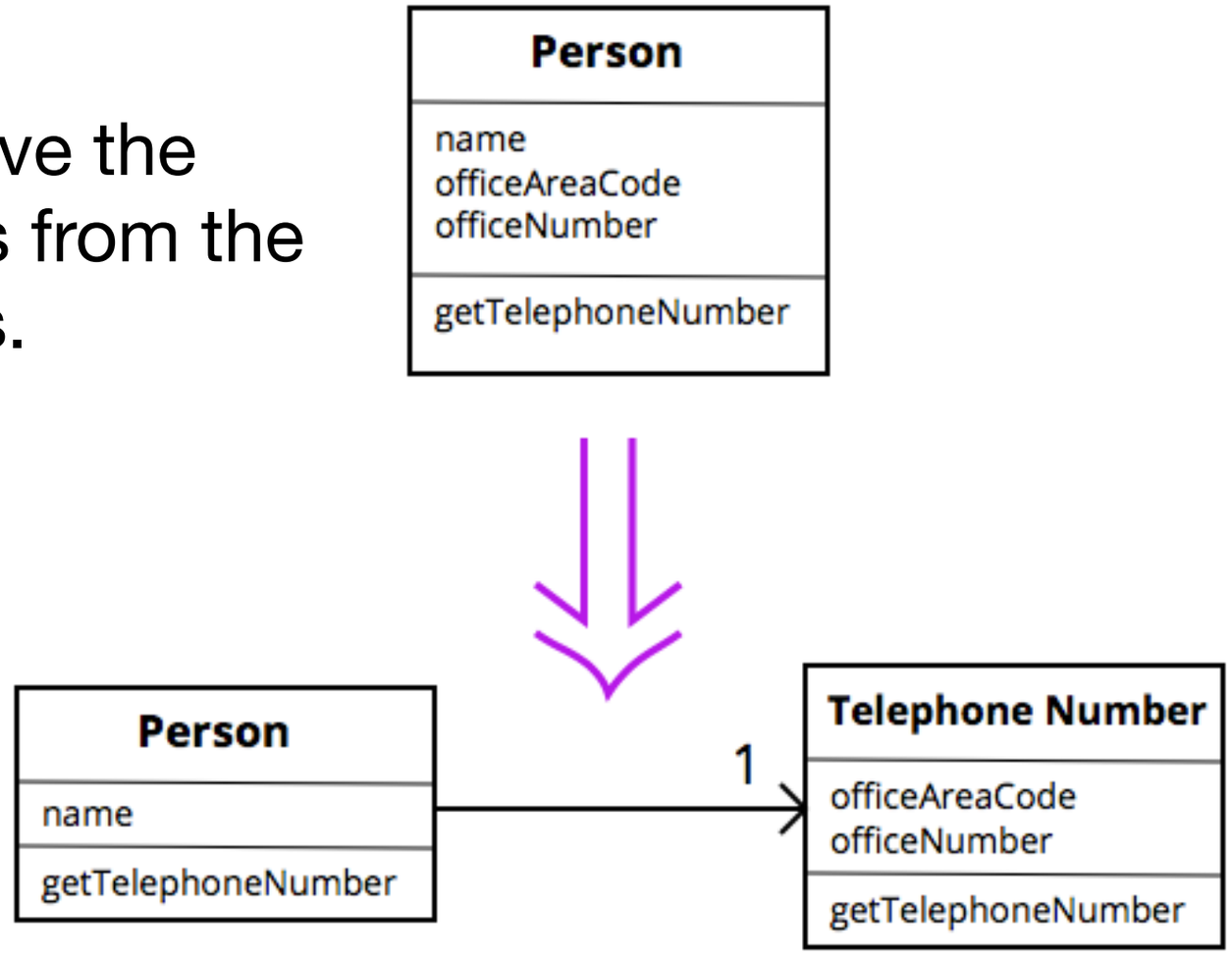
Push Down Field

- 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.



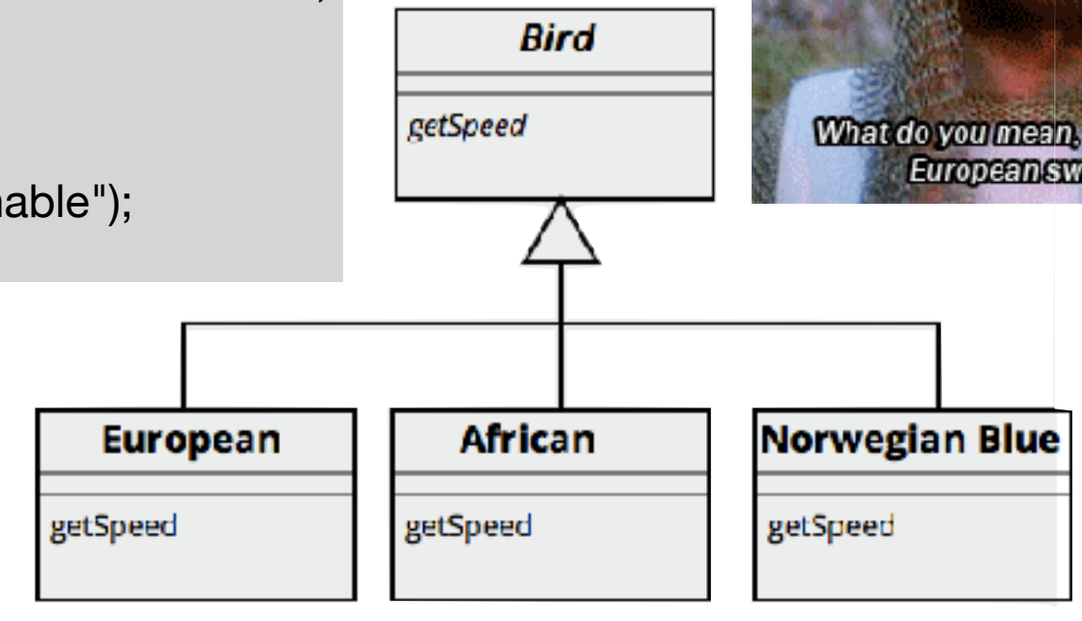
Be aware of your surroundings!

- 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.

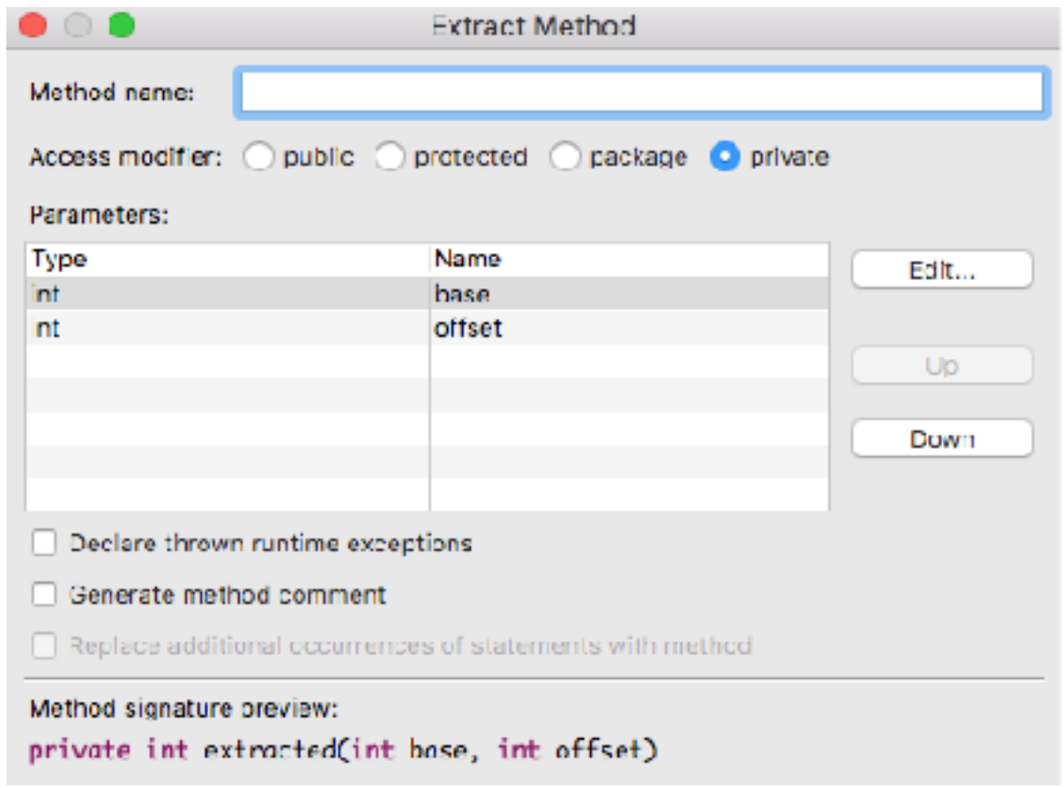
```
double getSpeed() {
  switch (_type) {
    case EUROPEAN:
      return getBaseSpeed();
    case AFRICAN:
      return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
    case NORWEGIAN_BLUE:
      return (_isNailed) ? 0 : getBaseSpeed(_voltage);
  }
  throw new RuntimeException ("Should be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



Murphy-Hill, Parnin, Black:
How We Refactor, and How We Know It.
IEEE Trans. Software Eng. 38(1): 5-18 (2012)