Refactoring: Improving the Design of Existing Code

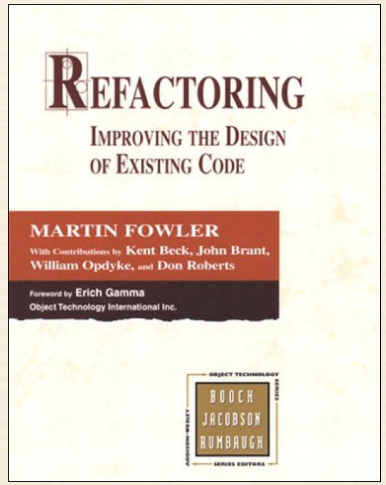
One of the best references on software refactoring, with illustrative examples in Java:

Refactoring: Improving the Design of Existing Code. Martin Fowler. Addison Wesley, 2000. ISBN: 0201485672

See also www.refactoring.com Overview of this presentation

A. Refactoring basics

B. Categories of refactoring



Categories of refactorings (according to [Fowler2000])

Small refactorings

(de)composing methods [9]

moving features between objects [8] organizing data [16]

dealing with generalisation [12]

simplifying method calls [15]

Big refactorings

Tease apart inheritance

Extract hierarchy

Convert procedural design to objects Separate domain from presentation

Big refactorings

Require a large amount oftime (>1 month)

Require a degree ofagreement among the development team

No instant satisfaction, no visible progress

Big Refactorings

1. Tease apartinheritance
2. Extract hierarchy
3. Convert procedural design to objects
4. Separate domain from presentation

Big refactorings:
1. Tease apart inheritance

Problem

A tangled inheritance hierarchy that is doing 2 jobs at once

Solution

Create 2 separate hierarchies and use delegation to invoke one from the other

Big refactorings: 1. Tease apart inheritance

Approach

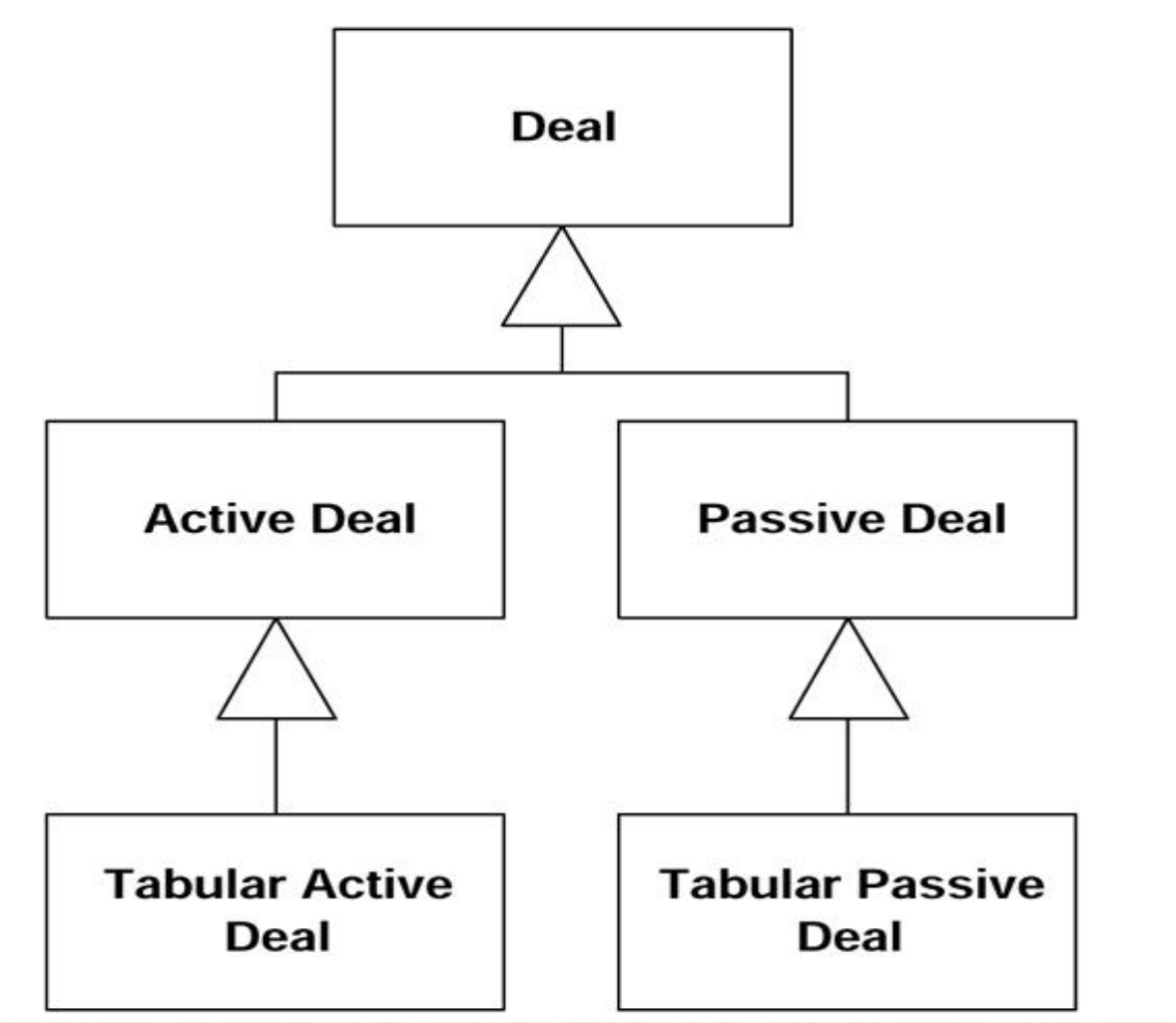
Identify the different jobs done by the hierarchy. Extract least important job into a separate hierarchy.

Use extract class to create common parent of new hierarchy. Create appropriate subclasses.

Use move method to move part of the behaviour from the old hierarchy to the new one.

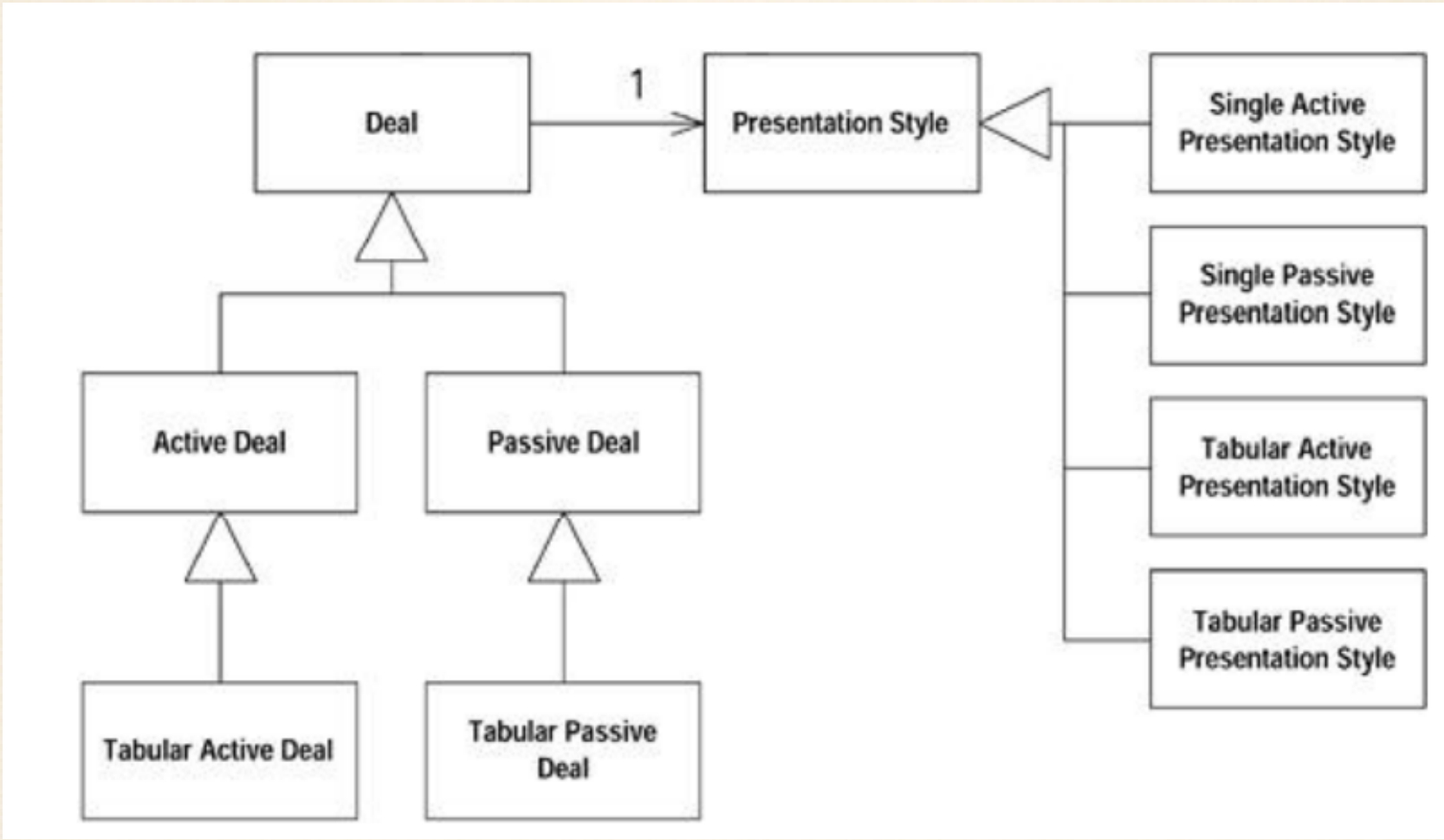
Big refactorings:

1. Tease apart inheritance



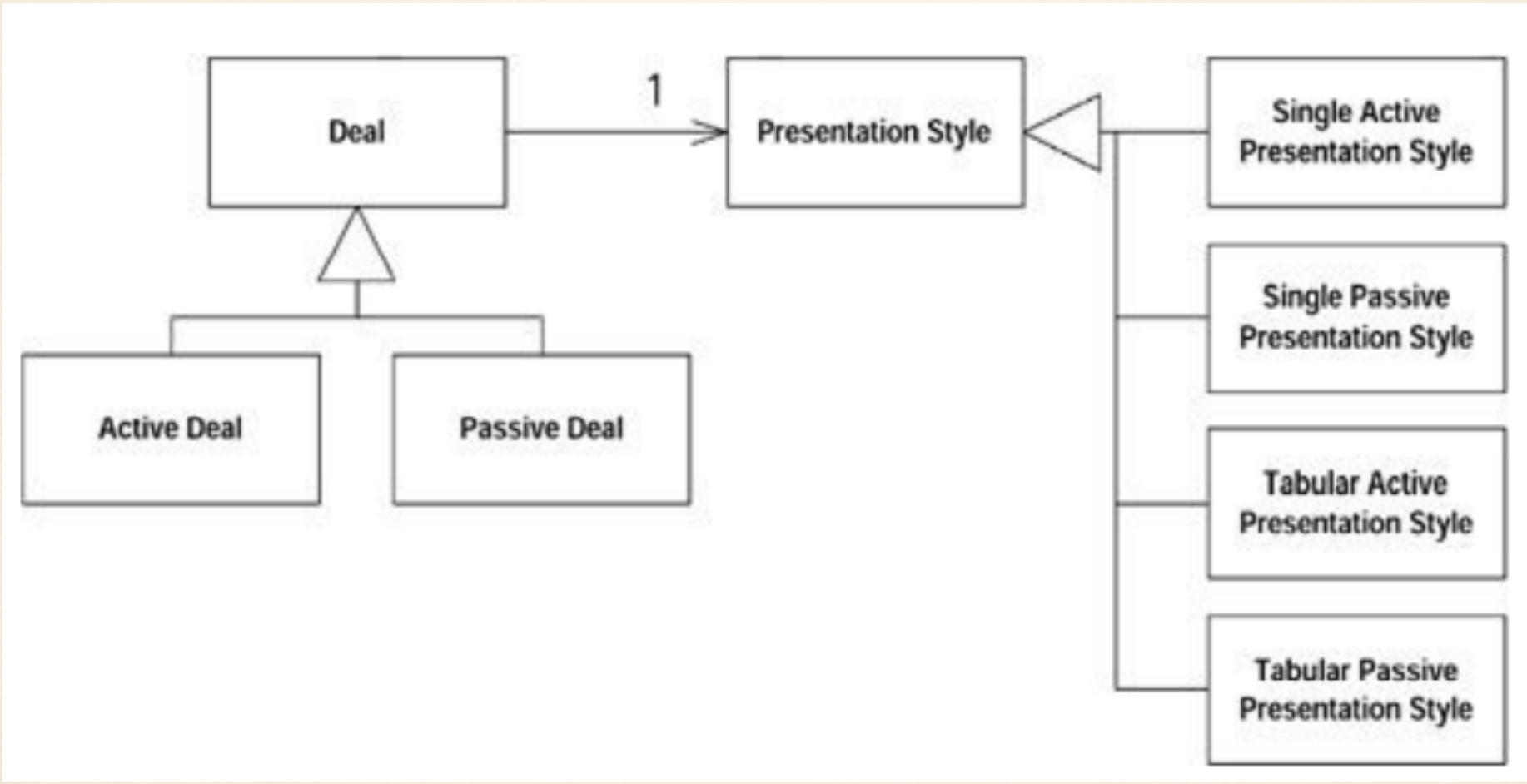
Big refactorings:

1. Tease apart inheritance



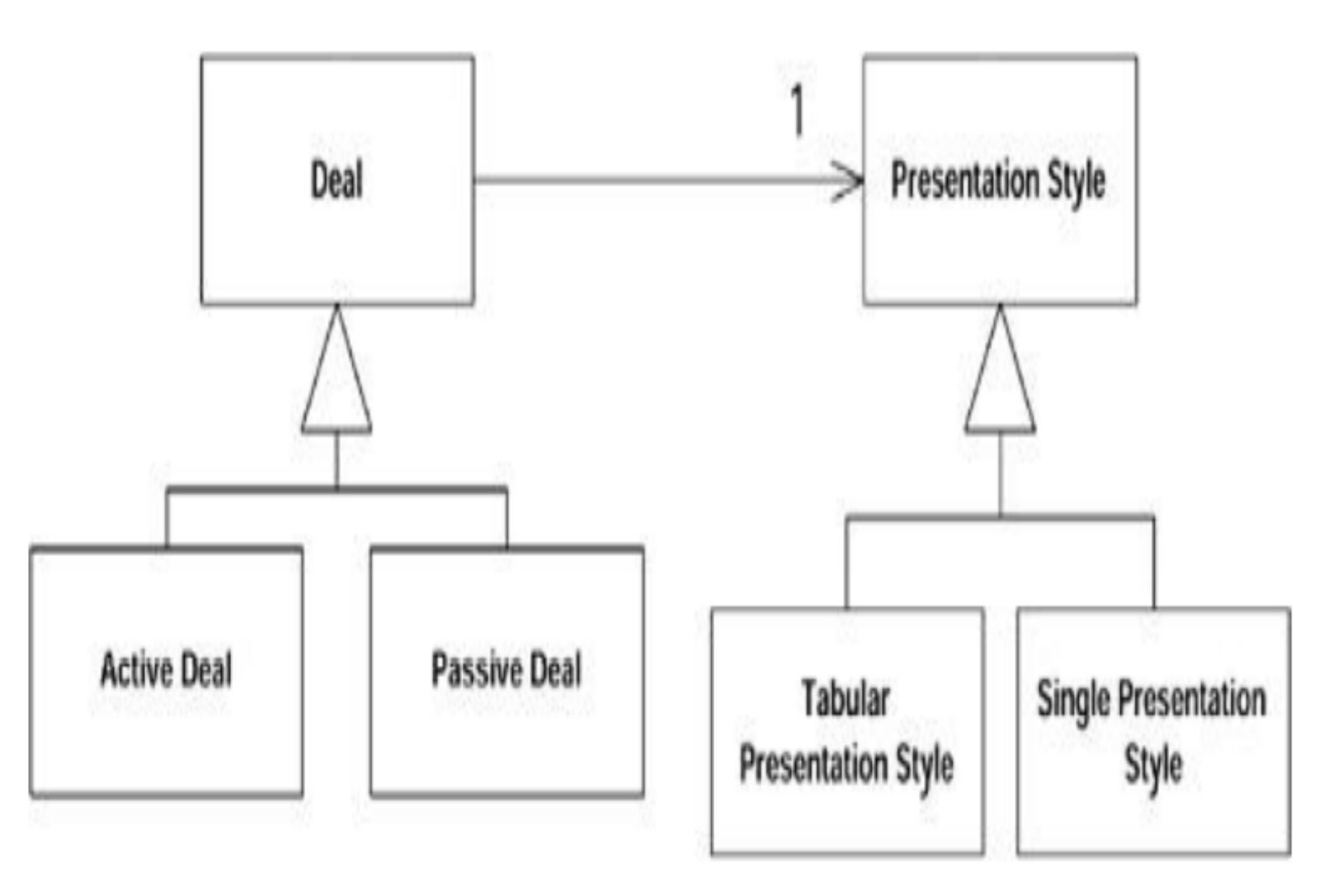
Big refactorings:

1. Tease apart inheritance



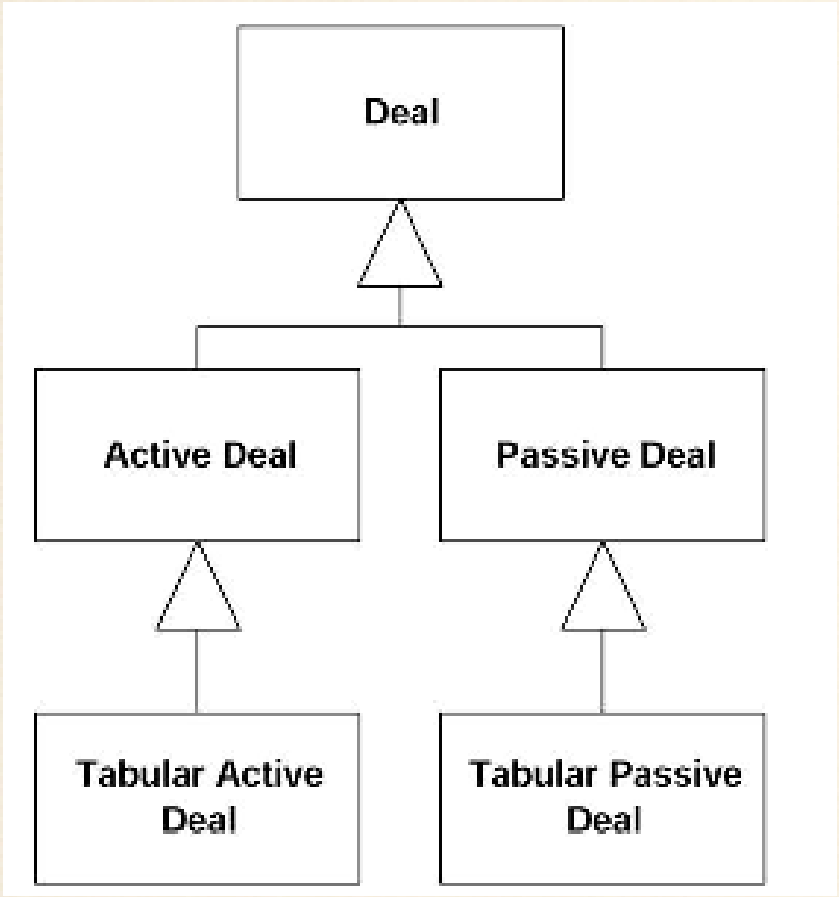
Big refactorings:

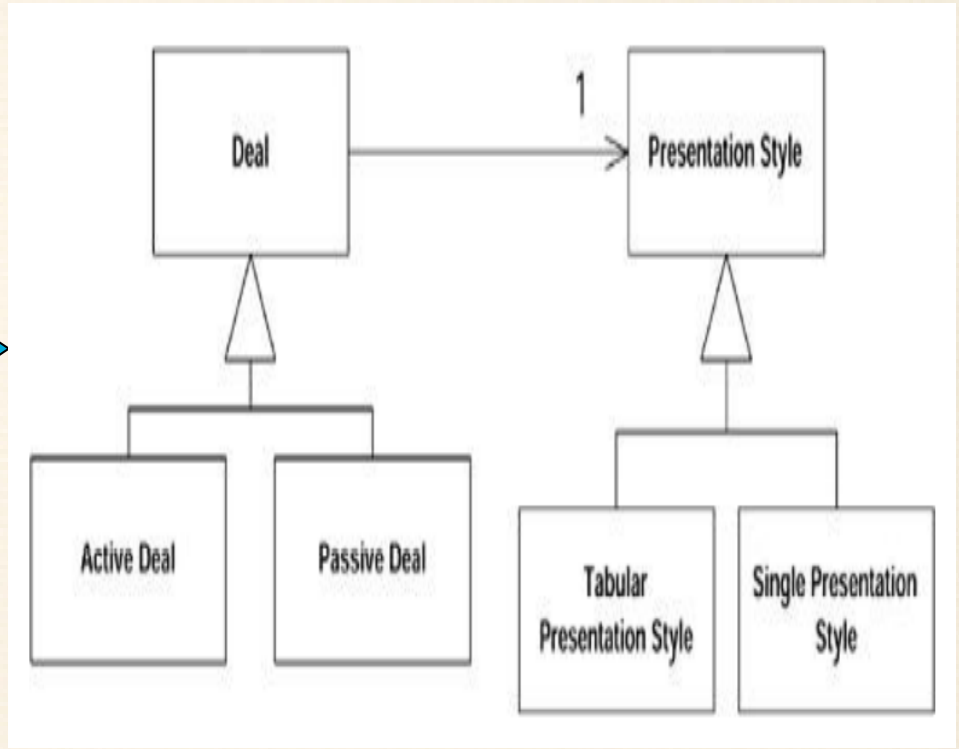
1. Tease apart inheritance



Big refactorings:

1. Tease apart inheritance





Big refactorings: 2. Extract hierarchy

Problem

An overly-complex class that is doing too much work, at least in part through many conditional statements.

Solution

Turn class into a hierarchy where each subclass represents a special case.

Big refactorings: 2. Extract hierarchy

Approach

Create a subclass for each special case.

Use one of the following refactorings to return the appropriate subclass for each variation:

- replace constructor with factory method
- replace type code with subclasses
- replace type code with state/strategy

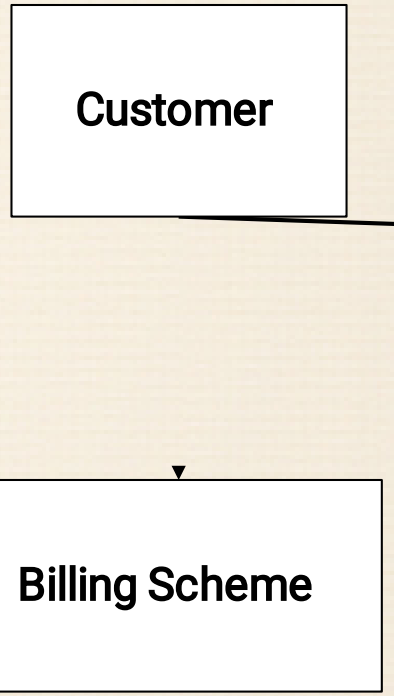
Take methods with conditional logic and apply: replace conditional with polymorphism

Big refactorings: 2. Extract hierarchy (example)

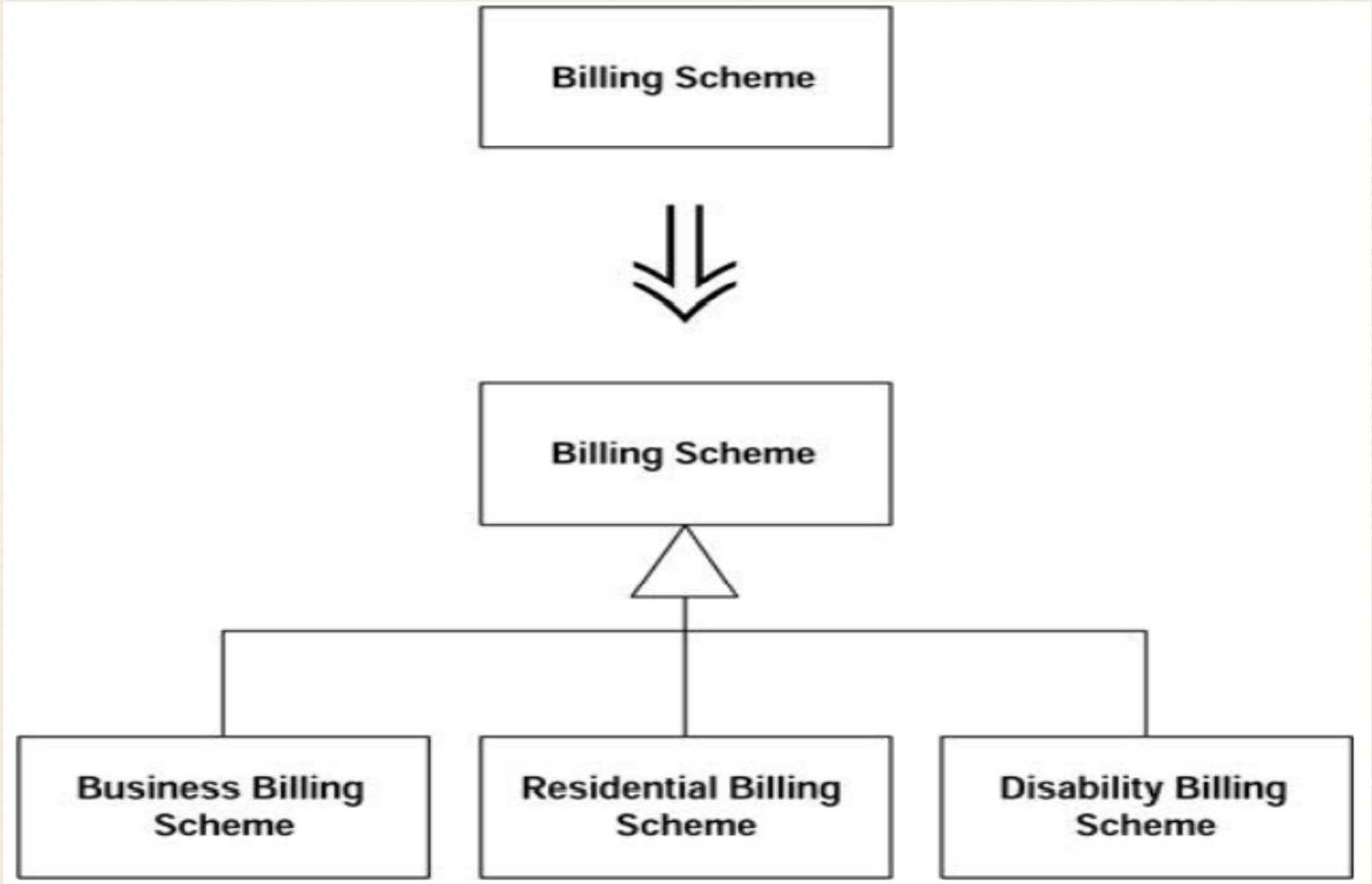
Calculating electricity bills.

Lots of conditional logic needed to cover many different cases:

- different charges for summer/ winter
- different tax rates
- different billing plans for personal / business / government / …
- reduced rates for persons with disabilities or social security



Big refactorings: 2. Extract hierarchy (example)



Big refactorings:
3. Convert procedural design into objects

Problem

You have code written in a procedural style.

Solution

Turn the data records into objects, break up the behaviour, and move the behaviour to the objects.

Smaller refactorings used

extract method, move method, …

Big refactorings: 4. Separate domain from presentation

Goal

Change a two-tier design (user interface/database) into a a three-tier one (UI/business logic/database).

Solution

Separate domain logic into separate domain classes.

Smaller refactorings used

extract method, move method/field, duplicate observed data, …