Software Engineering

COEN 174

Ron Danielson

Software Implementation

Chapter 9

Objectives

- Describe characteristics of good implementations
- Know best practices to achieve good implementations
- Understand refactoring

Design and Implementation

- Classic approach is full detailed design before implementation
  - Typically produces better organized, more cohesive design
  - Same team generates whole design
- Modern approaches sometimes leave detailed design to implementation
  - Especially for smaller projects
  - Usually faster implementation
  - Design done by many persons

Good Implementation Characteristics

- Readability
- Maintainability
- Performance
- Traceability
- Correctness
- Completeness

Style and Coding Guidelines

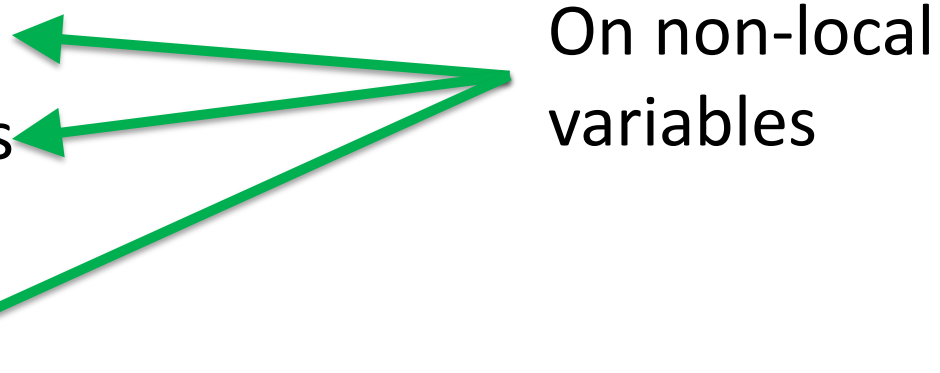
- Imposed by organization to support consistency
- Include
  - Language
  - Naming conventions
  - Indentation
  - Comment styles
  - Techniques for associating error messages with code location
  - Limits on function size, number of parameters, loop nesting
  - Banned language features

Comments

- Pushed in introductory programming classes
  - May make code harder to read
  - Often don't add much
    - Repeat code
    - Explain code
    - Marker for developers
    - Summarize code
    - Describe code
    - Reference external things
  - Need to be maintained as code evolves

Comments (cont.)

- Documenting methods
  - Intent
  - Preconditions
  - Postconditions
  - Return
  - Invariants
  - Exceptions
  - Known issues



Defensive Programming

- Anticipating potential errors and write code to deal with them
- Data
  - Wait for a legal value
  - Replace with default value if specified
  - Use the previous value
  - Log the bad value
  - Throw an exception
  - Abort

Defensive Programming (cont.)

- Exception handling in general
  - Stack exception handlers by scope
  - Handle what you can and rethrow to outer scope
  - Make “reasonable” expectations about ability to handle exception this code is throwing
  - Continue if you can

Defensive Programming (cont.)

- Check for common security attacks
  - Buffer overflow
- Enforce intentions
  - Make sure your code is not vulnerable to someone misusing it
  - Use qualifiers (const, final, …)
  - Make constants, variables, etc. as local as possible
  - Make attributes protected
  - Make methods private
  - Use classes to limit possible parameter values

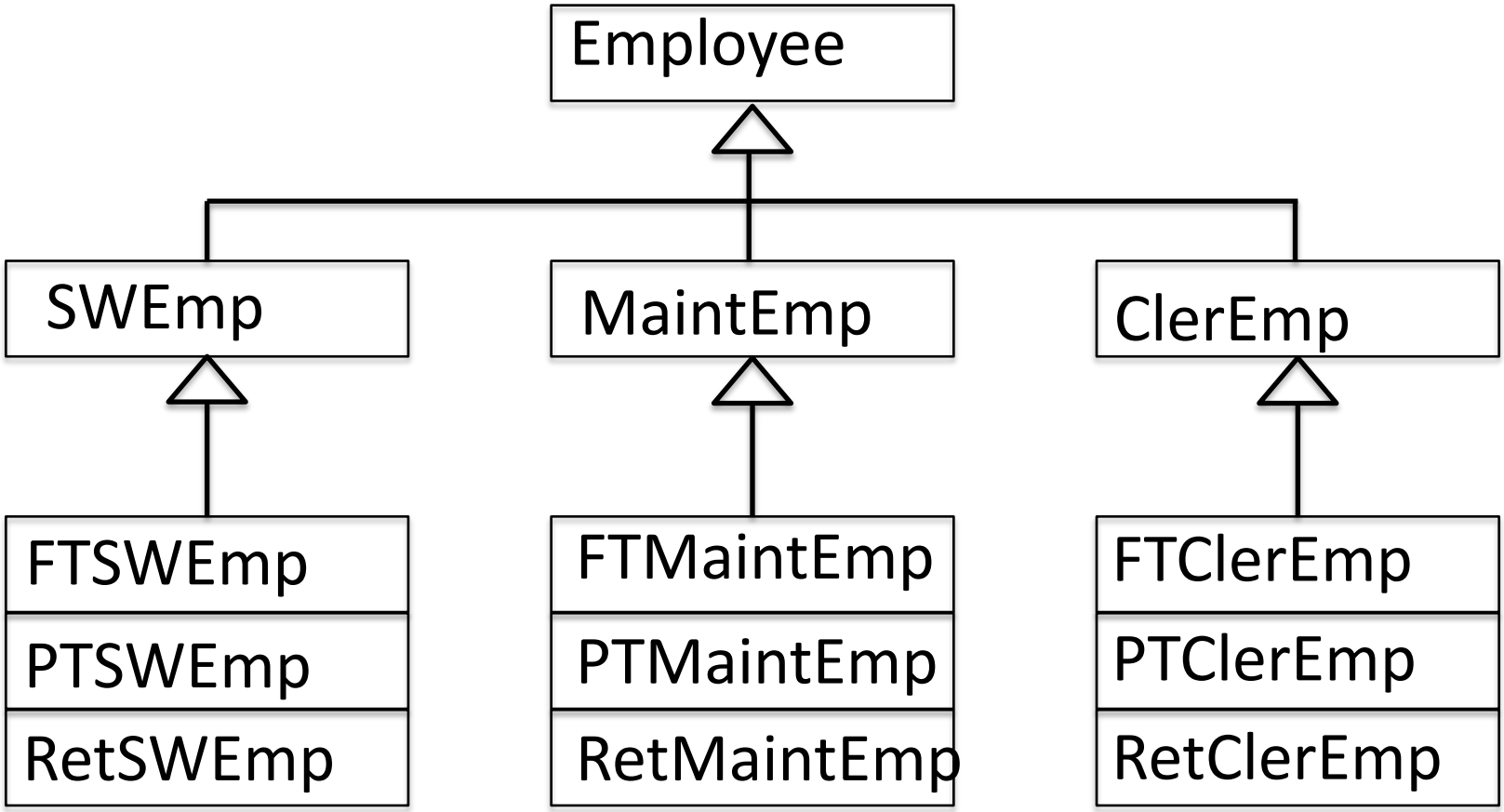
Refactoring

- Improving code (to make easier to understand or faster to modify) without altering behavior (Fowler, 1999)
- Symptoms
  - Duplicated code
  - Long method(s)
  - Large class(es)
  - Feature envy
  - Inappropriate intimacy

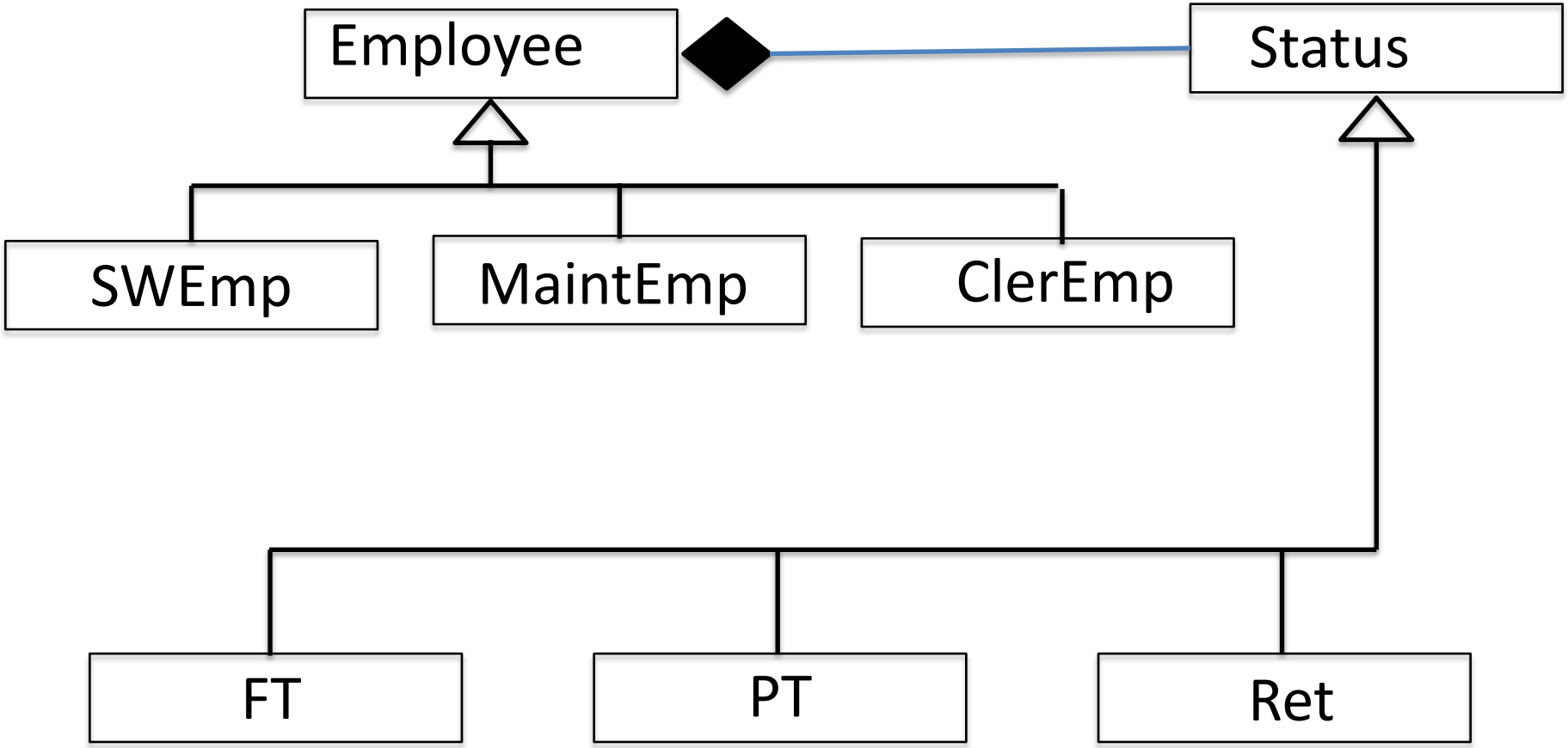
Refactoring (cont.)

- Big refactorings
  - Class-level, architectural impact
- Composing methods
  - Create, remove, or combine methods
- Move features between objects
- Reorganize data
- Deal with generalization
  - Exploit inheritance to simplify code

Big Refractoring



Big Refactoring (cont.)



Tease Apart Inheritance

Composing Methods

- Create, delete or combine methods to deal with evolving application
  - Extract method: turn block of code into method
  - Inline method: opposite
  - Replace temp with query: recalculate value each time needed
  - Introduce explaining variable: opposite

Move Features Between Objects

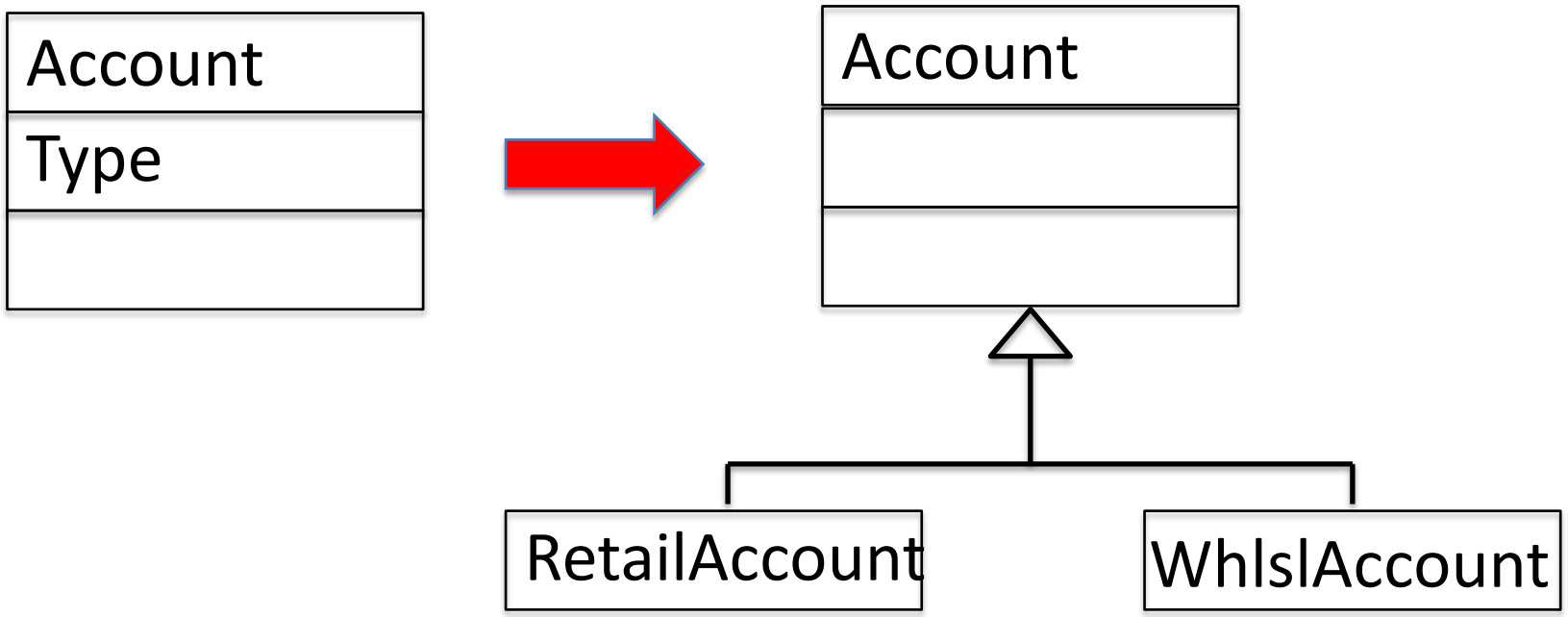
- Modify placement of class features
  - Move method: from one class to another
  - Move field: attribute to another class
  - Extract class: create new class from some attributes and methods that already exist
  - Inline class: opposite

Reorganize Data

- Change location of data
  - Self encapsulate field: convert direct access to attribute to only access using accessor method
  - Replace data value with object: when data value becomes too complex
  - Change bidirectional association to unidirectional: bidirectional interferes with reuse

Reorganize Data

– Replace type code with subclass

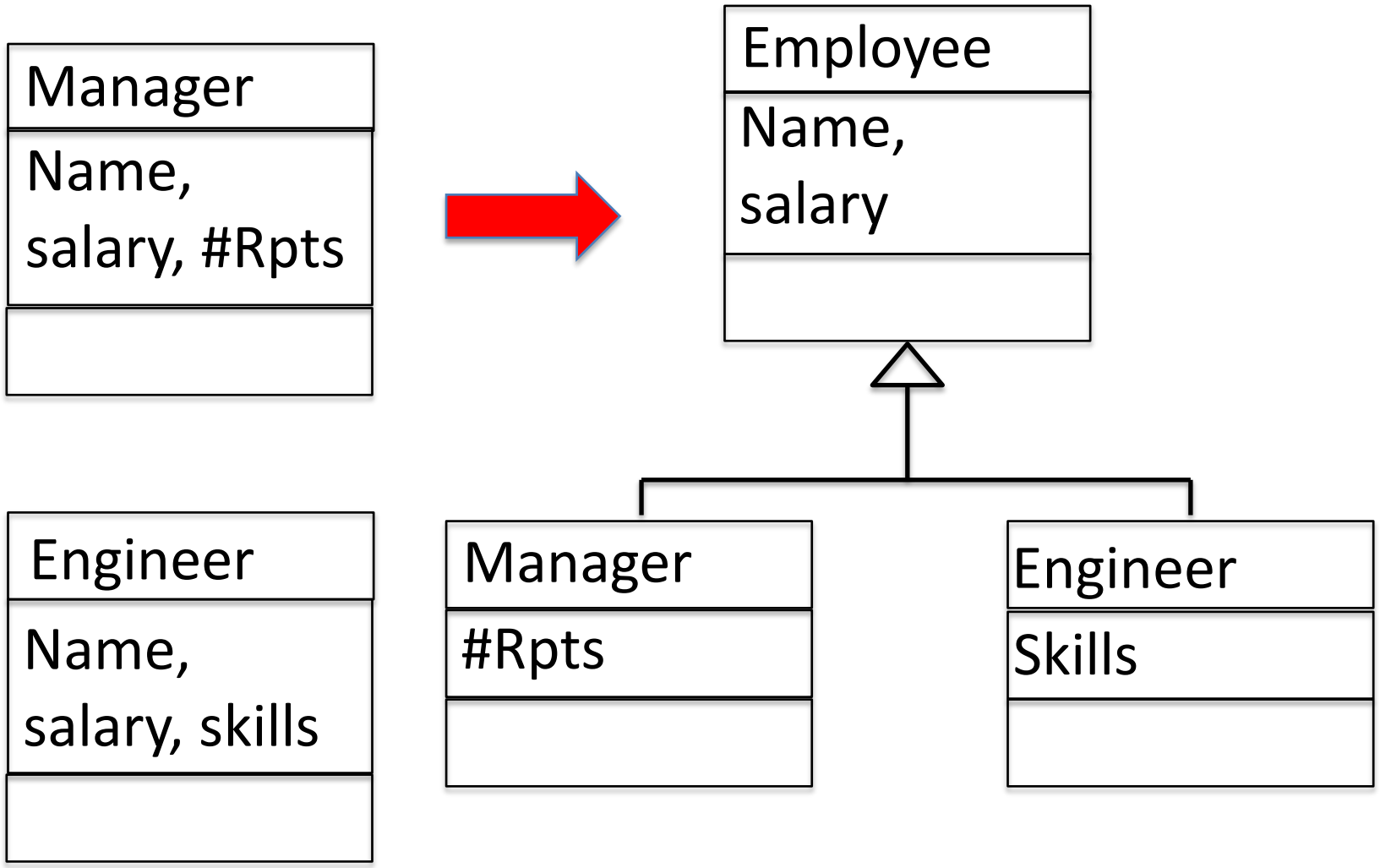


Generalize

- Pull up field/method: attribute or method to base class
- Pull down field/method: opposite
- Collapse hierarchy: lower inheritance tree

Generalize

– Extract superclass



Implementation Metrics

- Categories
  - Sufficiency: degree to which requirements are satisfied
  - Robustness: recovery from errors
  - Flexibility: easy to adapt
  - Reusability
  - Efficiency: meets performance requirements
  - Reliability
  - Scalability: degree to which can be expanded
  - Security

Implementation Metrics (cont.)

- Sufficiency
  - % of detailed requirements that are implemented
  - % of methods specified in the design that are implemented (if design is detailed)
- Robustness
  - For each method, consider preconditions
    - Are they complete?
    - How does method respond if each precondition is violated?
    - Rate on scale 0, 0.5, 1
  - Average method robustness to get class robustness

Implementation Metrics (cont.)

- Flexibility
  - Evaluate each class based on
    - Complete documentation
      - % of comment lines
      - % of commented lines
    - Use of named constants
      - % of numeric values with names
    - Is code hidden where possible?
      - % of private vs. public methods and attributes
      - Standard deviation of class size
      - Standard deviation of method size

Implementation Metrics (cont.)

- Flexibility (cont.)
  - Evaluate each class based on
    - Have common code segments been collected in one place?
      - % of code paragraphs repeated (sample)
    - Is there limited dependency on global variables?
      - % of public attributes, protected attributes
    - Is the code written at a general level?
      - % of generic classes
    - Are names understandable?
      - % of confusing names

Implementation Metrics (cont.)

- Reusability
  - Match classes to a real-world concept
  - % of classes that clearly match an understandable concept
- Efficiency
  - For speed, required time/actual time
  - For space, same

Implementation Metrics (cont.)

- Reliability
  - Calculate MTBF by actually running system for significant time
  - Failure is system needs to be restarted
  - MTBF = (total time)/(number of failures)
- Scalability
  - Really hard to measure
  - Difficult to evaluate through inspections
  - Expensive to run tests to measure things like maximum volume of transactions