COMP 3550

5.5 — REFACTORING CODE: COMMON SMELLS & FIXES

Week 5: Design Principles & Refactoring

WHAT IS REFACTORING?

- We’ve seen the word before but let’s officially and properly define it

    - what: change the implementation, not the functionality
    - why: readability, extendability, flexibility, more
    - when: remember tdd? → red, green, refactor
        - while that loop can still work, if you aren’t doing TDD it’s
        - a green → refactor → green loop

CODE SMELLS YOU MIGHT SEE

Long Methods

- What it is:
  - A method that tries to do too much, making it hard to understand, test, and maintain.
- Why it’s a problem:
  - Difficult to read or understand quickly
  - Harder to reuse logic
  - Prone to bugs when modified
- How to fix:
  - Break into smaller, focused methods that do one thing ("Extract Method" refactoring).

CODE SMELLS YOU MIGHT SEE

Data Clumps

- What it is:
  - A group of variables (often parameters) that are frequently found together, suggesting they should be part of their own object.
- Why it’s a problem:
  - Leads to duplication and inconsistencies
  - Makes method signatures bloated and confusing
- How to fix:
  - Encapsulate the related data in a class or record.

CODE SMELLS YOU MIGHT SEE

Shotgun Surgery

- What it is:
  - When making a small change forces you to edit many different classes or methods all over your codebase.
- Why it’s a problem:
  - Increases the risk of bugs
  - Makes refactoring painful and error-prone
- How to fix:
  - Improve cohesion; group related behavior and data in one place to localize changes.

CODE SMELLS YOU MIGHT SEE

Feature Envy

- What it is:
  - A method in one class that seems overly interested in the data or behavior of another class.
- Why it’s a problem:
  - Violates encapsulation
  - Indicates the method might belong in the other class
- How to fix:
  - Move the method to the class it’s most interested in (“Move Method” refactoring).

CODE SMELLS YOU MIGHT SEE

Switch Statements on Type

- What it is:
  - Code with switch or if-else chains based on type codes (enums, strings, integers) to decide behaviour
- Why it’s a problem:
  - Violates the Open/Closed Principle
  - Scattered logic across the codebase
- How to fix:
  - Use polymorphism — replace the switch with a method call on a type-specific subclass or strategy.

CODE SMELLS YOU MIGHT SEE

God Classes

- What it is:
  - A class that knows too much or does too much — often becoming a dumping ground for unrelated logic.
- Why it’s a problem:
  - High complexity
  - Low cohesion
  - Tightly coupled with everything
- How to fix:
  - Break it up into smaller, well-focused classes with single responsibilities.

FIXING SMELLS WITH SOLID

- SRP can help us remember to split logic
- OCP fixes help us to eliminate giant switches
- DIP to reduce tight coupling (Interfaces, DI, etc)

Think about LSP and ISP

- how can we use these to minimize the smells we leave behind?

PAUSE & PRACTICE

Goal: Practice identifying and understanding common code smells by writing short Java snippets that intentionally contain one smell each.

What To Do:

- Write One Java Class per Smell
- Create a short example (e.g., a class or method) that demonstrates each of the following code smells:
  - Long Method
  - Data Clumps
  - Shotgun Surgery
  - Feature Envy
  - Switch on Type
  - God Class

Keep each example around 10–30 lines — this doesn't need to compile or be a full app, just enough to clearly show the smell

- Name each file smell1.java, smell2.java, etc
- Push each file to your folder in my fall3350-video-work repo
  - fall3350-video-work/vid55/UMNETID/*.java
  - (follow the readme for folder setup if needed)