Code Smells and Refactoring Techniques

- A *code smell* is an aspect of a program that makes you uneasy. A code smell might indicate a design issue.
  - Code smells are not bugs, but they may lead to problems later on if they are not refactored away.
- A *refactoring technique* is a recipe for rearranging code.
  - These techniques are used to remove code smells.
- Source Making does a nice job explaining code smells and refactoring techniques (and in fact the website gave shape to this lecture). We base our treatment here heavily on its content.
  - There are dozens of similar websites and viewpoints — just a quick web search away.

Bloaters

- A *bloater* is a section of code that is excessively long.
- **Long Method**: any method longer than 10–15 lines of actual code should make you suspicious.
- **Large Class**: any class that feels large.
- **Primitive Obsession**: using several primitives that could be encapsulated.
- **Long Parameter List**: more than 4 parameters is smelly.
- **Data Clumps**: groups of variables that repeatedly appear together.

Example: Long Method

- Source Making's treatment of Long Method

Object-Orientation Abusers

- An *object-oriented abuser* is a violation of object-oriented programming principles.
- **Switch Statements**: often, a switch statement or a sequence of if statements indicate that inheritance and polymorphism might lead to cleaner code.
- **Temporary Field**: an instance variable that is only needed in specific situations, and is otherwise ignored.
- **Refused Bequest**: if a subclass only uses some of the inherited methods, and the other methods are either unused or rewritten to throw exceptions.
- **Alternative Classes with Different Interfaces**: two classes that provide quite similar functionality but with different method names.

Change Preventers

- A *change preventer* is a situation where changing one part of the code causes a ripple effect.
- **Divergent Change**: when making changes to a class, many unrelated methods need to be edited.
- **Shotgun Surgery**: Adding a feature or fixing a bug requires many small changes across several classes.
- **Parallel Inheritance Hierarchies**: When writing a subclass causes to create another subclass elsewhere in the program.

Dispensables

- A *dispensable* is code that is unnecessary and distracting.
- **Comments**: When a method has many explanatory comments.
- **Duplicate Code**: When two code fragments are nearly identical.
- **Lazy Class**: When a class exists only to handle a small task.
- **Data Class**: A class that is a container for instance variables.
- **Dead Code**: A piece of code that is no longer used.
- **Speculative Generality**: A piece of code that was created to handle a future situation that hasn't yet arisen.

Couplers

- A *coupler* is a smell that indicates excessive coupling.
- **Feature Envy**: When a method uses data from another object more than its own.
- **Inappropriate Intimacy**: When one class relies on the internal representation of another class.
- **Message Chains**: A series of method calls where the return value of each one is used to call the next method.
- **Middleman:** When a class does most of its work by asking another class to do that work.

Study for the test

- We reserve the right to test you on the Code Smell categories on these slides. Before the tests, **read about at least two Code Smells from each category** (Bloaters, Object-Orientation Abusers, etc.). We might ask you to write about them.
- Make sure you can explain the various refactoring techniques that solve the Code Smell.

Industry conversation

- Professional programmers talk about code, of course.
- Refactoring terms are a part of the language they use. You can use this in interviews.
- Good news! You can practice this while you study for the test:
  - Find a partner or a small group of classmates.
  - Have each of you pick a Code Smell and study it.
  - Find an example bit of code (it's a web search away).
  - Teach each other about the smell you chose, including the refactoring techniques that solve it.