Activity 5-1: Code Smells

A **code smell** is a characteristic of the code that suggests something is wrong, typically a weakness or funamental problem with the *design*. Code smells are not usually bugs or any obvious problem that prevents the code from running.

Typically the remedy for code smells is refactoring.

Comment smells

- **Obsolete comments** or comments holding information that should be maintained elsewhere, like in the version control system.
  Example: information about the version number or when changes where made
  Remedy: Delete them! Give Git a chance!
- **Redundant comments** that do little more than restate what the code says. Not useful, and might not change when the code changes, resulting in misleading comments.
  Remedy: Delete them!
- **Commented-out code**. By now probably totally out of sync with the system.
  Remedy: Terminate with extreme prejudice!

Naming smells

- **Non-descriptive** or ambiguous names.
- Names at the **wrong level of abstraction**.
- **Type encoding** in names.
- Names that **hide side-effects**.
- Names that **disinform**.
  Example: function name that does not correctly say what the function does
  Remedy for all: Rename them until you find a good name.

Function smells

- **Too many arguments.**
  Remedy: Extract arguments that often go together into a parameter object.
- **Flag arguments (booleans).** These suggest a function is doing too many things.
  Remedy: Create separate functions or convert the booleans into to some kind of enum if that is more appropriate.
- **Wrong level of abstraction**. The code in each method should be at the same level of abstraction.
  Remedy: Extract methods, giving suitable names to the new methods.
- **Feature envy.** A function that uses more methods from another class than its own possibly belongs to that other class.
  Remedy: Move the function to the other class.
- **Inappropriate static functions.** When creating static functions, make sure they would not more naturally be housed in the class of one of their parameters.
  Remedy: Move the function to the other class.

- **Long function**. Probably means the function is doing way too much.
  Remedy: Extract till you drop!

Behavioral smells

- **Obvious behavior not implemented:** Functions should implement reasonably expected behavior.
  Remedy: Add tests for this behavior and change your function to match.
- **Incorrect behavior at boundaries.**
  Remedy: Add tests for this behavior and change your function to match.
- **Duplicate code.** Try not to repeat yourself; you never want to have to change the same code in multiple places.
  Remedy: Extract Method.
- **Too much knowledge.** Parts of the code know more than they need to.
  Remedy: Find ways to reduce the coupling between classes!
- **Dead code.** Code that will never be reached. IntelliJ will actually complain about that.
  Remedy: Figure out what's wrong!
- **Artificial coupling.** Similar to *too much knowlege*. Different parts of your application should not know about each other if they do not have to.
  Remedy: Remove unnecessary dependencies. Might require moving some functions around.
- **Misplaced responsibility.** Code should go where you expect to find it, not where it is most convenient for the programmer.
  Remedy: Move functions around, or extract methods to create new wrappers. And keep thinking of your user.
- **Hidden temporal couplings.** A sequence of statements needs to happen in a particular order, but nothing in the code forces this to happen.
  Remedy: Make each statement depend on some output from the previous statement.
- **Transitive navigation / Train wrecks.** A sequence of getting calls, like: `o.getA().getB().getC().doSomething()`. The object `o` needs to know too much about the system that way.
  Remedy: Hide the chain behind a method that expresses more directly what you are doing, and/or make some of the classes in the middle do the same.
- **Divergent change.** When a module changes for many different reasons. It is a sign that this module is doing too many different things.
  Remedy: Find a subset of the methods with common functionality; extract them as a delegate or a superclass.
- **Shotgun surgery.** When many modules must be changed to effect a single behavioral change. This means that functionality that changes for the same reasons has been needlessly spread across many classes.
  Remedy: Move the relevant functionality around to where it should be.
- **Data clumps.** Data that tends to stick together should be in a class.
  Remedy: Extract a new class out of those common elements.

- **Primitive obsession.** The overuse of primitive types.
  Remedy: Create a new class to host the functionality related to those primitive types.
- **Speculative generality.** Unneeded abstractions that make the code harder to read.
  Remedy: Eliminate with Extreme Prejudice! YAGNI!

Form

- **Vertical Separation.** Variables and functions should be close to their use.
  Remedy: Move them closer to each other.
- **Inconsistency.** Similar tasks are performed differently, or similar variables are named differently.
  Remedy: Use rename, extract method and other refactorings to do things in a similar ways when possible.
- **Obscured intent.** Various aspects of the code make it hard to discern the code's intent, e.g., lack of explanatory variables.
  Remedy: Extract methods/variables to give names to things; rename as needed to give them better names.
- **Magic constants.** Literal values used "as is" in the text, without clear meaning.
  Remedy: Extract constant. Choose a good name for it.
- **Complex conditional tests.**
  Remedy: Extract methods for them to make them more readable. And find ways to simplify the code.
- **Temporary field.** A field that is not always set/used.
  Remedy: It may belong to a different class.
- **Null checks** all over the code.
  Remedy: Separate your public methods from your internal methods, and only allow null inputs from outside calls.
  Remedy: Consider creating a "Null Object" class to represent meaningful functionality for a "null object"; pass this around instead of the value null.

For more on code smells see:

- *Refactoring: Improving the Design of Existing Code*, chapter 3[1] by Martin Fowler.
- *Clean Code*, chapter 17[2] by Robert Martin.

[1] https://learning.oreilly.com/library/view/refactoring-improving-the/9780134757681/ch03.xhtml
[2] https://learning.oreilly.com/library/view/clean-code/9780136083238/chapter17.html