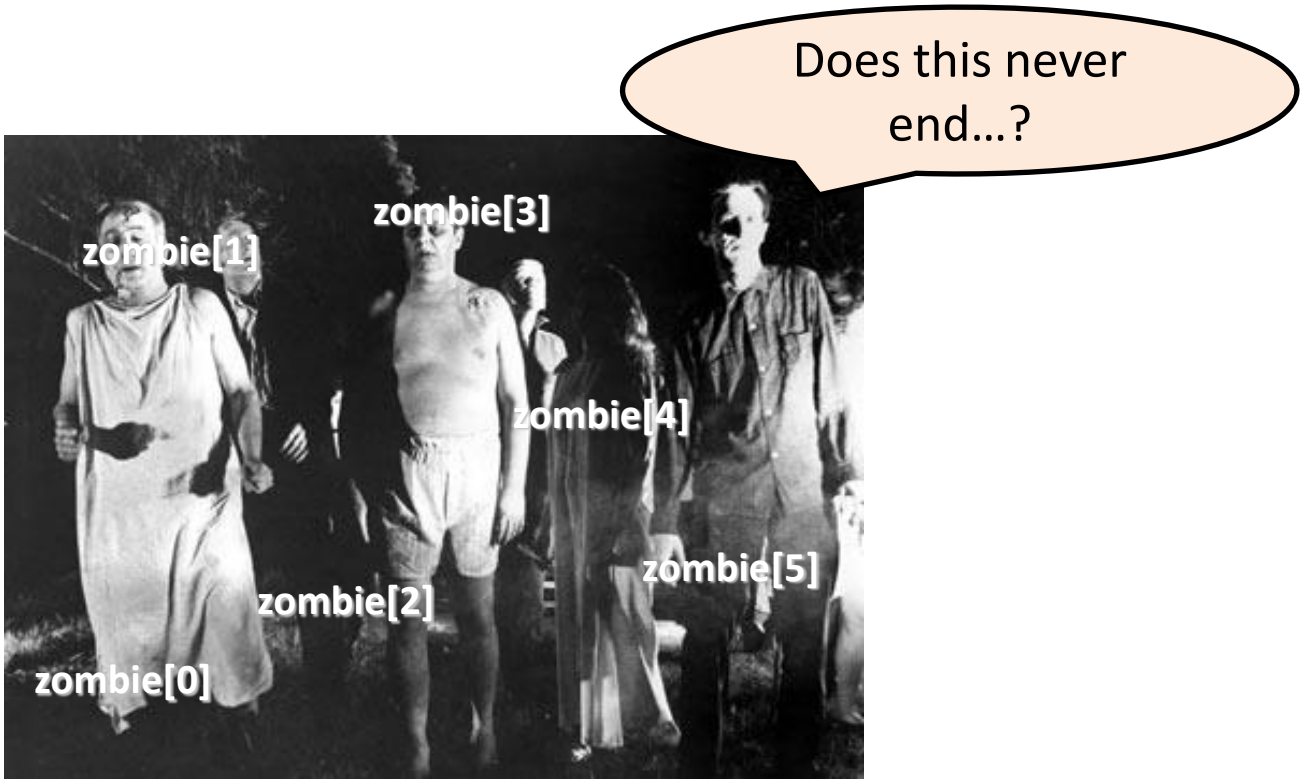
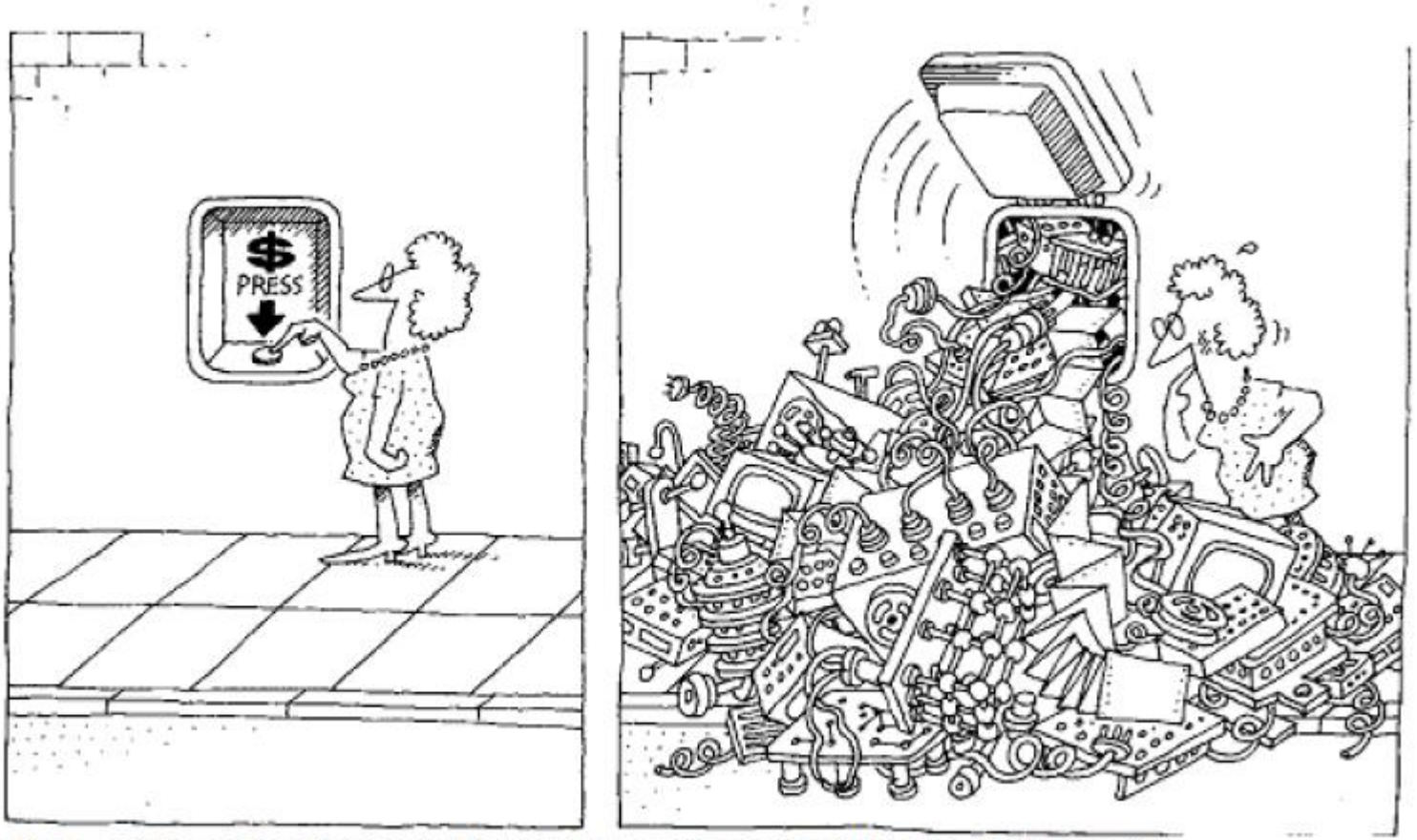
Problem Decomposition Revisited (Again and Again): Object Oriented Design Part III



Overview

- Object Oriented Design
  - Simplicity
  - Abstraction
  - Encapsulation
  - Modularity
  - Abstraction Hierarchy
  - Strong Data Typing
  - Concurrency
  - Object State, Behavior and Identity
  - Classes vs. Objects
  - Inheritance

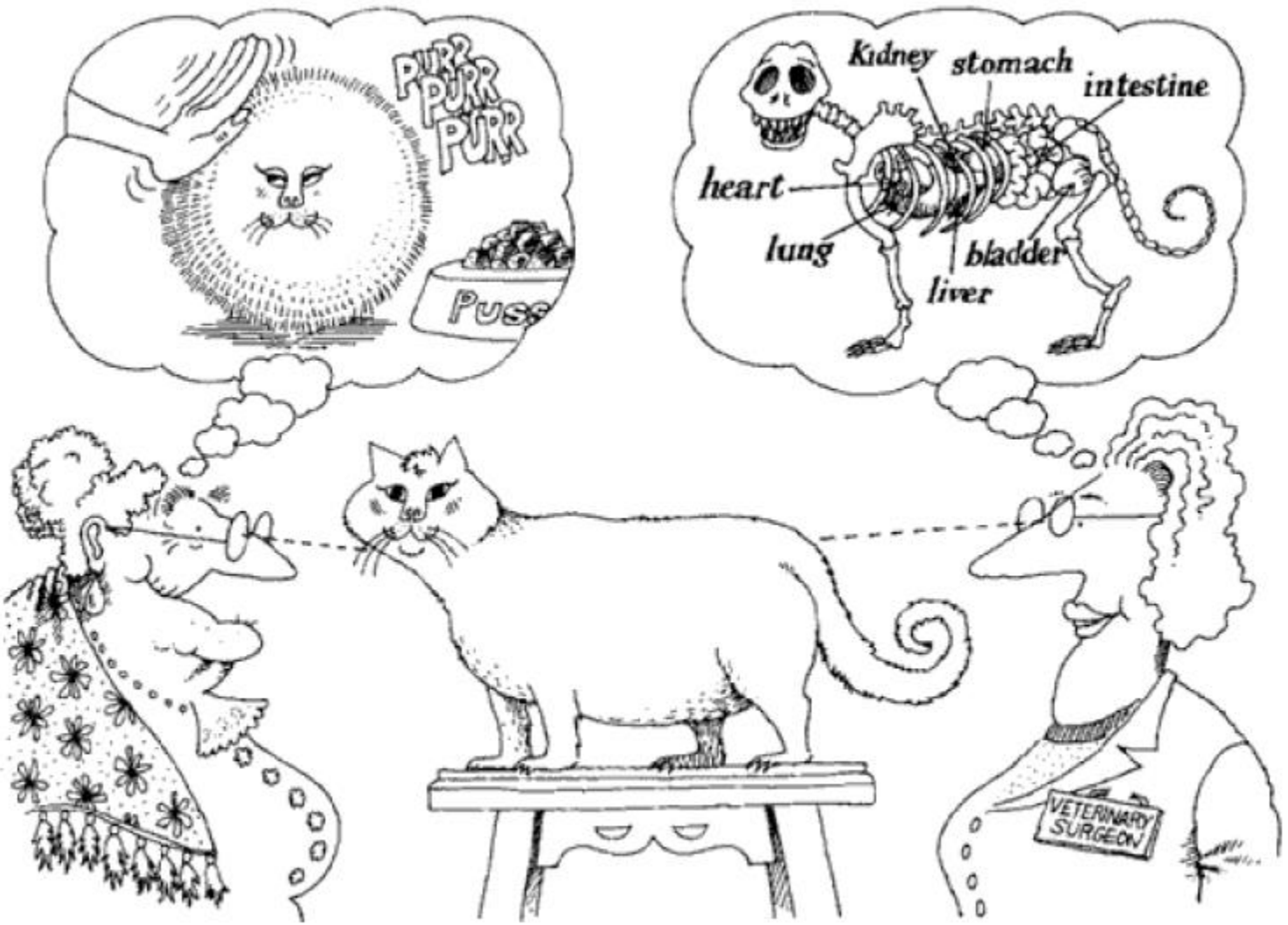
Good Design: Simplicity



The task of the software development team is to engineer the illusion of simplicity.

From “Object Oriented Design with Applications” by Grady Booch

Good Design: Abstraction



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

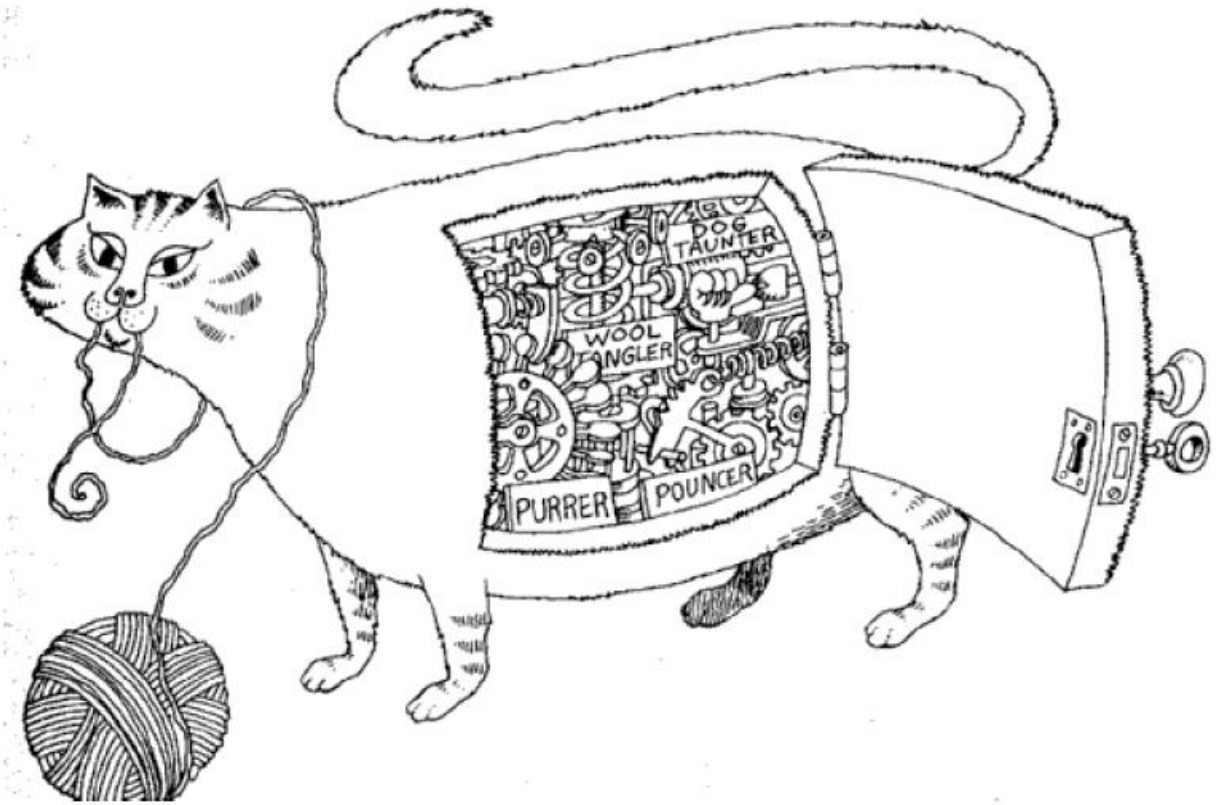
From "Object Oriented Design with Applications" by Grady Booch

Good Design: Abstraction

"An abstraction denotes the ***essential characteristics*** of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, ***relative to the perspective of the user***."

From "Object Oriented Design with Applications" by Grady Booch

Good Design: Encapsulation



Encapsulation hides the details of the implementation of an object.

"Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential characteristics."

From "Object Oriented Design with Applications" by Grady Booch

Good Design: Modularity

Modularity packages abstractions into discrete units.

"Modularity … creates a number of well-defined documented boundaries within the program. These boundaries, or interfaces, are invaluable in the comprehension of the program."

"Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules."

From "Object Oriented Design with Applications" by Grady Booch

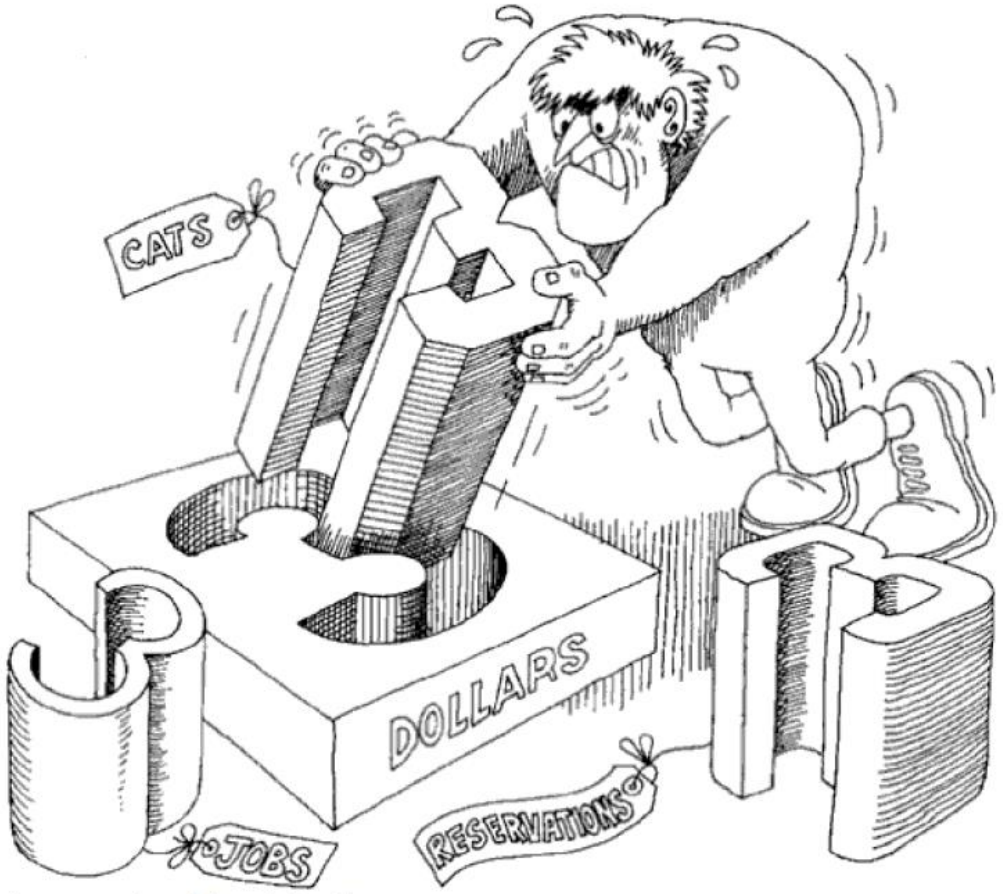
Good Design: Abstraction Hierarchy

Abstractions form a hierarchy.

"Hierarchy is a ranking or ordering of abstractions."

From "Object Oriented Design with Applications" by Grady Booch

Good Design: Strong Typing

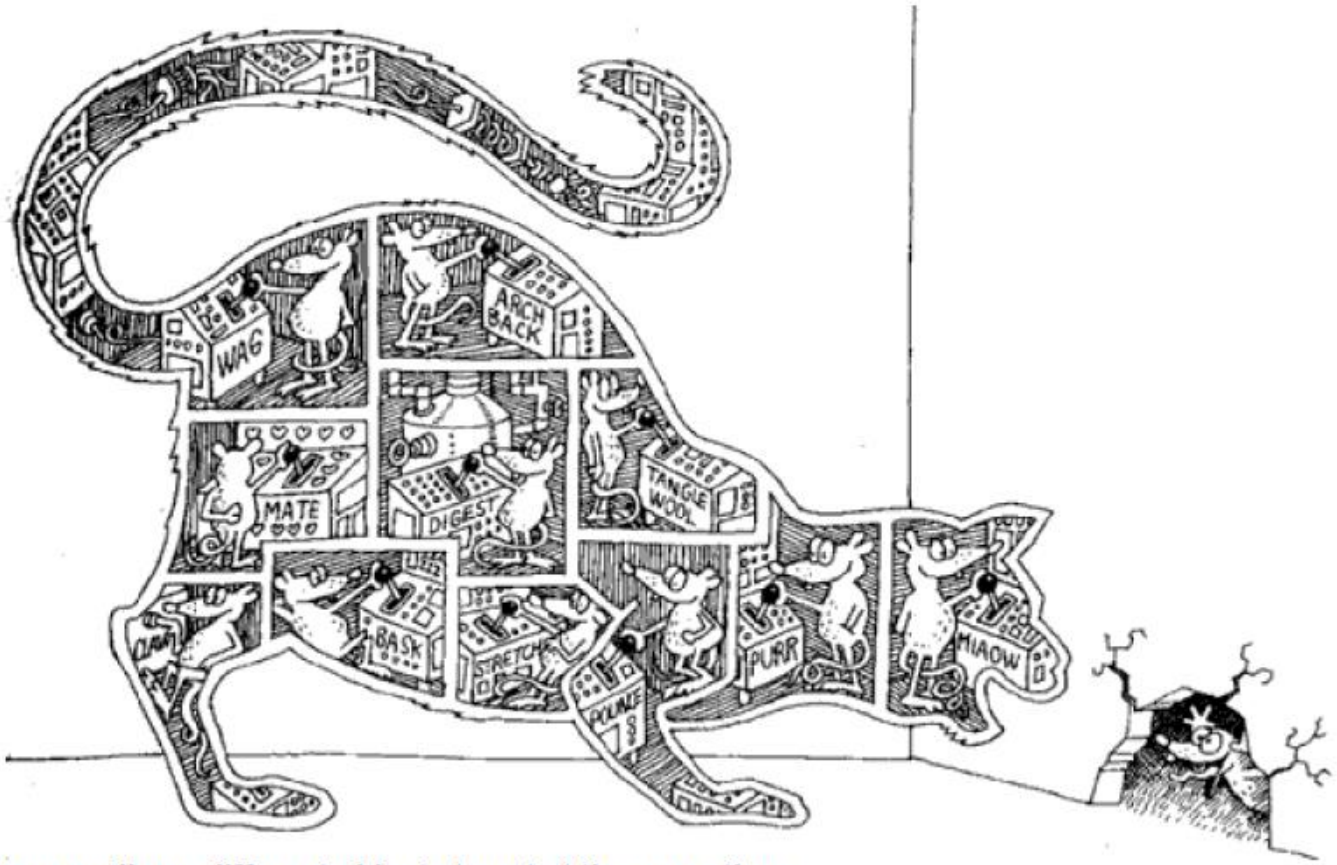


Strong typing prevents mixing abstractions.

"Typing is the enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways."

From "Object Oriented Design with Applications" by Grady Booch

Good Design: Concurrency

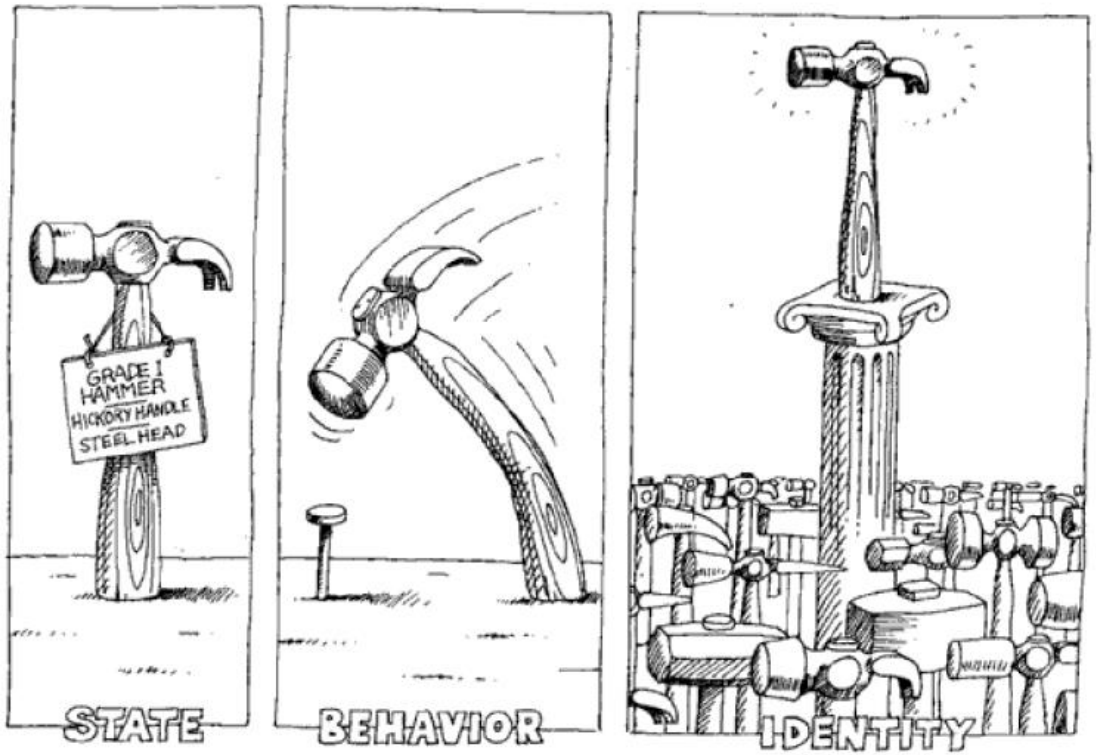


Concurrency allows different objects to act at the same time.

"Concurrency is the property that distinguishes an active object from one that is not active."

From "Object Oriented Design with Applications" by Grady Booch

Good Design: State, Behavior, Identity



An object has state, exhibits some weii-defined behavior, and has a unique identity.

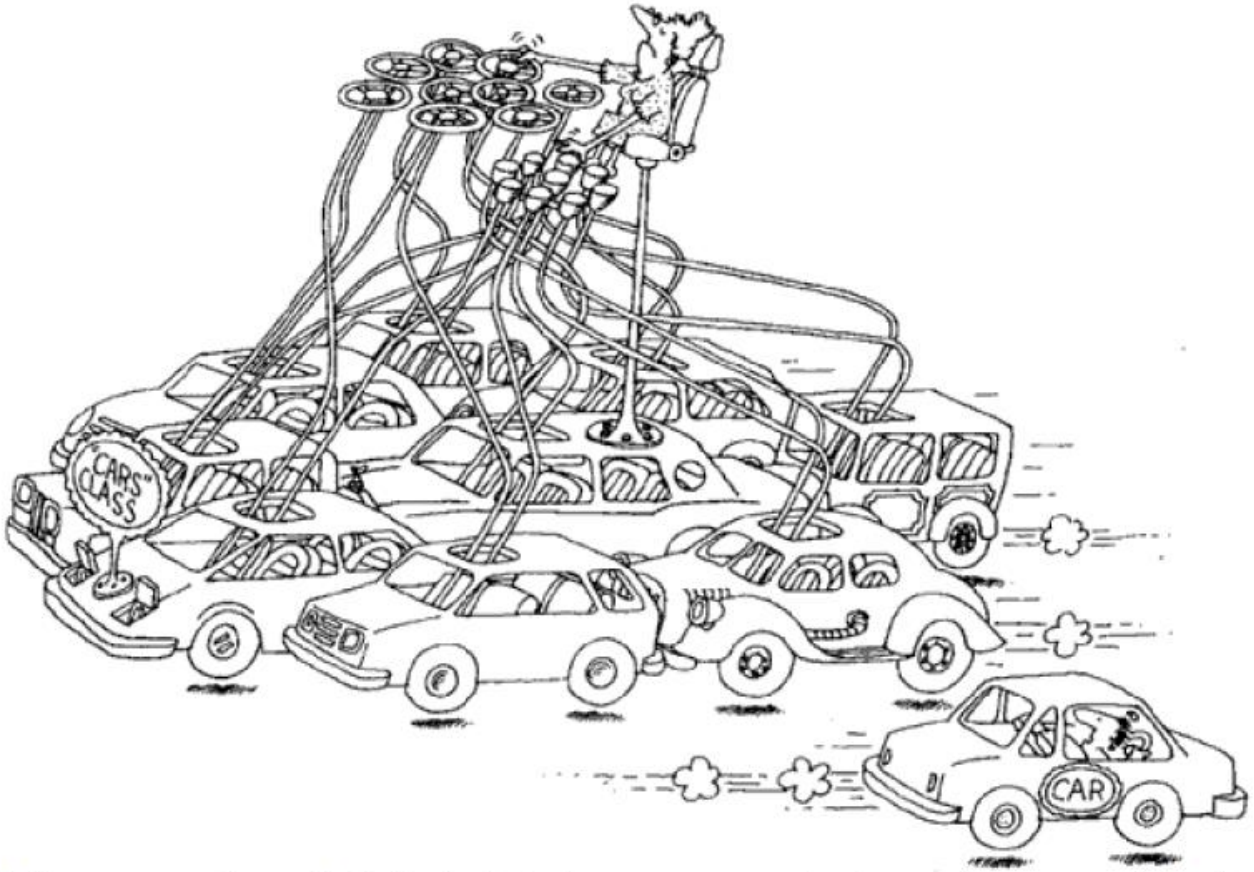
"The state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of those properties."

"Behavior is how an object acts and reacts, in terms of its state changes and message passing."

"Identity is that property of an object which distinguishes it from all other objects."

From "Object Oriented Design with Applications" by Grady Booch

Good Design: Classes vs. Objects



A class represents a set of objects that share a common structure and a common behavior.

"A class is a set of objects that share a common structure and a common behavior."

"A single object is simply an instance of a class."

From "Object Oriented Design with Applications" by Grady Booch

Good Design: Inheritance

A subdass may inherit the structure and behavior of its superdass.

From “Object Oriented Design with Applications” by Grady Booch

We haven’t talked about how this works in Java yet… but don’t worry, we will. ☺

Software Development Life Cycle

1. Requirements Analysis (Understanding the Problem - thoroughly)
2. Design (Working out the Logic)
3. Implementation (Converting it to Code)
4. Test/Debug
5. Maintenance

Measures of Good Design

- Coupling
  - How tightly modules or classes are tied together
  - We want to minimize this
- Cohesion
  - How tightly a module or class is tied to itself
  - We want to maximize this
- Sufficiency
  - Whether a class meets all the needs of its clients
- Completeness
  - Whether a class contains all that it needs
- Primitiveness
  - How simple is a module or class

Summary

- Object Oriented Design
  - Simplicity
  - Abstraction
  - Encapsulation
  - Modularity
  - Abstraction Hierarchy
  - Strong Data Typing
  - Concurrency
  - Object State, Behavior and Identity
  - Classes vs. Objects
  - Inheritance