Key Concepts

- Overall goal: reduce complexity
  - Dependencies
  - Obscurity
- Good design is an investment
  - Tactical vs. strategic programming
- Complexity is incremental: zero tolerance
- Abstraction: find simple ways to think about complicated things
- Information hiding
  - Interface vs. implementation
- Classes should be deep
- General-purpose classes are deeper
- Different layers should have different abstractions
- Pull complexity downward, push specialization upward
- Comments should describe things that aren’t obvious from the code
- Comments are at a different level of precision than code
- Names matter!
- Define errors out of existence
- Code should be obvious
Red Flags

- Shallow classes
- Information leakage
  - Dependencies
  - Conjoined methods/classes
- Temporal decomposition
- Pass-through methods
- Code duplication
- Special cases
- Inconsistencies
- Comment duplicates code
- Implementation contaminates interface documentation
- Documentation has to be long to be complete
- Vague names
- Code is not obvious
Wrapup

- This class is just a beginning
- To keep learning, need
  - Experience
  - Feedback
- What to do when you join a company?
  - Find a company that cares about design
  - Do as much as you can in your own code
  - Look for opportunities to influence
  - As you get more senior, change the organization
Workloads

- If Project 1 was 1.0 units of work, how many units were
  
  - Project 2: 0.8, 0.75, 1.5, 0.5, 0.8, 1.2, 0.5, 0.8, 1.2, 0.7, 0.67, 0.7, 1.5, 1.0, 0.8, 0.6, 0.7, 0.8
  
  - Project 3: 1.2, 0.55, 1.0, 1.0, 1.3, 1.0, 1.0, 1.35, 0.8, 0.5, 0.67, 1.2, 1.0, 0.75, 1.2, 0.8, 1.2, 1.0