Design In Construction
Good Design **Manages Complexity**

- “Seven plus or minus two” (Miller’s Law)
- The goal of all software-design techniques
  - Break complicated problems into simple problems
- Separation of concerns
  - Focus on one at a time
Design is hard

- Design is an art, not a science
- Large/infinite design space, not enumerable
- Requirements, constraints, trade-offs, and priorities
Virtues of a good design

- Minimal complexity
- Ease of maintenance
- Loose coupling
- Reusability
- Standard techniques

Plus....
Virtues of a good design, cont.

(which does not belong)

A) Extensibility
B) High Fan-in
C) Leanness
D) Symmetry
E) Low-to-medium Fan-out

utility piece

each class only uses a few other classes
Keep Coupling Loose

- small interfaces (few methods, few arguments/method)
- obvious (interactions through parameter passing)
- flexible
Which kind of coupling should cause concern?

A) ClassA calls a method of ClassB passing primitive data types
B) ClassA instantiates an object of ClassB
C) ClassA calls a method of ClassB passing an object of ClassC
Decomposition

- identify the objects and their attributes (methods and data)
- determine what can be done to each object
- determine what each object is allowed to do to other objects
- determine which parts of objects are visible to other objects
- define each object's public interface
Design Practices (tools for your toolbox)

- iterate
- divide-and-conquer
- top-down and bottom-up (decomposition vs. composition)
- experimental prototyping
- collaborative design/brainstorming
Best half paragraph of the book? (p. 119)

- Treat design as a wicked, sloppy, heuristic process.
- Don't settle for the first design that occurs to you.
- Collaborate.
- Strive for simplicity.
- Prototype when you need to.
- Iterate, iterate, and iterate again.
- You'll be happy with your designs.
Example: Adventure

- Contain the game
  - Take user input
  - Interpret user command
  - Handle invalid input (direction)
  - Map user input to room
  - Terminate the program

- Current room

- Layout
  - Verify floorplan
  - Parse JSON
  - Room name to room object

- Room
  - Print directions available
  - Print room description

- Direction
AStar a = new AStar
for (Graph g) { 
  g.doThing(); 
}
To Dos for Thursday

- Read Ch. 6 (Working Classes)

For next code review:
- Extend your A* implementation to handle arbitrary graphs
  - E.g., adjacency list based graphs
  - Use inheritance from an abstract base class or an interface
Graph of Illinois cities

```json
{
    "edges": [
        {"node1": "Bloomington", "node2": "Champaign", "weight": 54},
        {"node1": "Decatur", "node2": "Champaign", "weight": 54},
        {"node1": "Danville", "node2": "Champaign", "weight": 35},
        {"node1": "Lincoln", "node2": "Springfield", "weight": 37},
        {"node1": "Decatur", "node2": "Springfield", "weight": 44},
        {"node1": "Decatur", "node2": "Mattoon", "weight": 59},
        {"node1": "Mattoon", "node2": "Champaign", "weight": 53},
        {"node1": "Lincoln", "node2": "Peoria", "weight": 46},
        {"node1": "Lincoln", "node2": "Bloomington", "weight": 36},
        {"node1": "Kankakee", "node2": "Champaign", "weight": 75},
        {"node1": "Kankakee", "node2": "Chicago", "weight": 62},
        {"node1": "Joliet", "node2": "Chicago", "weight": 51},
        {"node1": "Joliet", "node2": "Kankakee", "weight": 50},
        {"node1": "Joliet", "node2": "Bloomington", "weight": 98}
    ]
}
```