Section 2
Basics of Object-Oriented Design

1. OO design overview
2. Object design categories
3. UML class diagrams

Section 2.1
Object-Oriented Design Overview

1. Software engineering
2. Data abstraction
3. Encapsulation
4. Principle of least privilege

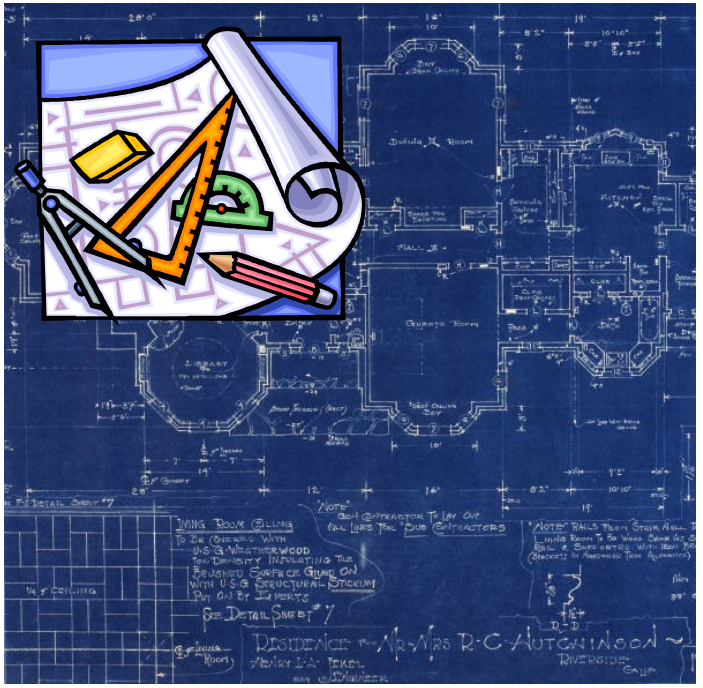
2.1.1 Software Engineering

- Industrial quality software must be:
  - reliable
  - easily modifiable
  - reusable
- Why?

Building a House

Really?

We need a plan!



The Plan

- Software development life cycle activities:
  - requirements analysis
    - making sure that we build what the client wants
  - design
    - making sure that what we build can be easily modified, extended
  - implementation
    - making sure we follow the design
  - testing
    - making sure that what we build works the way it should

Object-Oriented (OO) Design

- Think **before** you code, not after...
  - a good house doesn't "just happen"
  - neither does good software
  - this is a creative process
    - it's not algorithmic
    - it takes a lot of practice

- Just getting code to work isn't good enough
  - industrial quality software is too big to "wing it"
  - it has to follow principles of good software engineering
  - this mostly happens in the design phase

OO Design (cont.)

- Think:
  - what objects do you need?
    - data
    - behaviour
  - can you reuse classes from another source?
  - what do your classes have in common with each other?
  - what information should be hidden inside each class?

OO Design (cont.)

- How do we design good OO software?
  - single responsibility classes
    - create objects that have **one** purpose
  - data abstraction
    - separate **what** an object does from **how** it does it
  - encapsulation
    - protect runtime objects from bad code
  - principle of least privilege
    - access to runtime objects must be on an as-needed basis

2.1.2 Data Abstraction

- Goal of data abstraction
  - separate **what** a class does from **how** it does it
  - separate the class interface from the implementation details
    - abstract properties of objects
      - shared with the class users
    - concrete details of implementation
      - not shared with the class users
      - details remain confidential to the class developers

Data Abstraction (cont.)

- Goal of data abstraction (cont.)
  - always hide your class implementation
    - this is *not* about separating code into **files**
    - this is about separating code into **classes**
      - it's hiding the implementation details inside your class
      - includes underlying data structures, algorithmic details, etc.

Data Abstraction (cont.)

- Why data abstraction?
  - biggest enemy of timely software development is **change**
    - clients change their minds about what they want
    - designers misunderstand the requirements
    - clients want more features
  - consequences
    - change is always disruptive, consequences must be mitigated
  - what developers can do:
    - design classes that can change without impacting other classes
    - make sure class users don't rely on specific class implementation

Data Abstraction (cont.)

- Approach:
  - design objects that model the real world
  - a good class interface should be:
    - simple and intuitive
    - easy to use
    - not require knowledge of implementation details
  - useful breakdown
    - control objects
    - boundary (UI/view) objects
    - entity objects
    - collection objects

2.1.3 Encapsulation

- What is encapsulation?
  - grouping together common data and functionality
  - granting the least amount of access to other classes
- Goal
  - maximize reuse of existing code
  - design new classes that can be reused

Encapsulation (cont.)

- Approach
  - find the similarities between your classes
  - reuse the code that implements these similarities
  - give data members private or protected access only

Encapsulation (cont.)

- Key tools for encapsulation:
  - composition
  - inheritance
  - principle of least privilege

2.1.4 Principle of Least Privilege

- What is the principle of least privilege?
  - it's a design technique
  - it requires that you:
    - grant access permission to your runtime objects only as needed
    - never grant more permission than needed
- This applies to:
  - variables, parameters, objects
  - class members