Discovering Classes

- **Goals:**
  - Identify what classes / objects we can reuse
  - Identify what new classes / objects we need
  - Identify what methods the classes / objects need
  - Identify how the classes / objects interact

The Noun-Verb Approach

- **Idea:**
  - **Use nouns from the problem domain to identify classes**
  - **Use verbs associated with the nouns to identify methods for the classes**
- **Example: The Bank Simulation:**
  - **Nouns: Bank, Teller, Client, Person, Line-up (Queue)**
  - **Verbs:**
    - Bank: Open, Close
    - Person: Arrive, Leave
    - Teller Serve Client
    - Client: Line-up, Get Served

DALHOUSIE UNIVERSITY
Inspiring Minds

Design is…

- Iterative
  - You will make mistakes and will revisit the design
- About tradeoffs and priorities
  - Review your software quality objectives
- Nondeterministic
- Managing complexity and restrictions
- Heuristic

Design – Manage Complexity

- **Accidental Complexity**
  - Complexity that we inherit from the processes, environment, choices
  - Often disconnected from the base problem itself
  - Eg. iOS look-and-feel for an iPhone
    specifics of how SQL works in a mySQL database
    libraries (or lack thereof) in our programming language
    constraints on managing resources

Design – Manage Complexity

- **Essential Complexity**
  - Complexity that arise from the problem or the interlocking set of concepts in the solution
  - Arise no matter where or how we deploy the solution
  - Eg. dividing information among tables in a database connection between a user interface and a simulation model balancing a binary tree for an efficient search algorithm details of Dijkstra's algorithm for finding shortest paths

Design

- Minimize the quantity of essential complexity that you need to remember at one time
- Prevent accidental complexity from proliferating throughout the solution

Good Design?

- **Loose coupling and high cohesion**
- **SOLID properties**
- **Code Complete list**
  - Minimal complexity
  - Ease of maintenance
  - Loose coupling
  - Extensibility
  - Reusability
  - High fan-in
  - Low-to-medium fan-out
  - Portability
  - Leanness
  - Stratification
  - Standard techniques

Levels of Design

- From biggest to smallest
  - Software system
  - Subsystems or packages
  - Common subsystems
    - Business rules
    - User interface
    - Database access
    - System dependencies
  - Classes
  - Methods
  - Method algorithms

Architectures often help here (later in the course)

Class-level Design Steps

- Identify the objects and their attributes
- Determine what can be done to each object
- Determine what each object is allowed to do to other objects
- Determine parts of each object that will be visible to other objects
- Define each object's public interface

Class-level Design Steps

- **Identify the objects and their attributes**
- **Determine what can be done to each object**

Single Responsibility
Open / Closed Principle
Mind Map
Noun-Verb Approach

- **Determine what each object is allowed to do to other objects**

Liskov Substitution Principle
CRC Method

- **Determine parts of each object that will be visible to other objects**

Interface Segregation
Dependency Inversion

- **Define each object's public interface**

DALHOUSIE UNIVERSITY
Inspiring Minds

Assignment 1

- Write a Java class, called “RecipeBook” that holds and converts cooking recipes between different units of measures and scales recipes for people who don’t like to adapt recipes on their own. The converter will
- Provide rounding of quantities, based on what it is told of the rounding tolerance
- Select an appropriate unit of measure when more than one is available
- Represent quantities as integers and fractions, not as decimal numbers
- Scale and/or adapt ingredient quantities in the list of ingredients and in the instructions
- Interpolate between two measurement systems A and B through a third measurement system C if there is no direct mapping of A to B.

Assignment 1

- Write a Java class, called "RecipeBook" that holds and converts cooking recipes between different units of measures and scales recipes for people who don't like to adapt recipes on their own. The converter will
  - Provide rounding of quantities, based on what it is told of the rounding tolerance
  - Select an appropriate unit of measure when more than one is available
  - Represent quantities as integers and fractions, not as decimal numbers
  - Scale and/or adapt ingredient quantities in the list of ingredients and in the instructions
  - Interpolate between two measurement systems A and B through a third measurement system C if there is no direct mapping of A to B.

Assignment 1

(cooking) recipes
Units of measure
People
Converter
Rounding of quantities
Rounding tolerance
Quantities
Integers
Fractions
Decimal numbers
Ingredient quantities
List of ingredients
Instructions
Measurement systems

(cooking) recipes
- list of ingredients
  - ingredient quantities
    - integers
    - fractions
    - decimal numbers
- instructions

Measurement systems
- units of measure

Converter
- rounding of quantities
- rounding tolerance

~~People~~

DALHOUSIE UNIVERSITY
Inspiring Minds

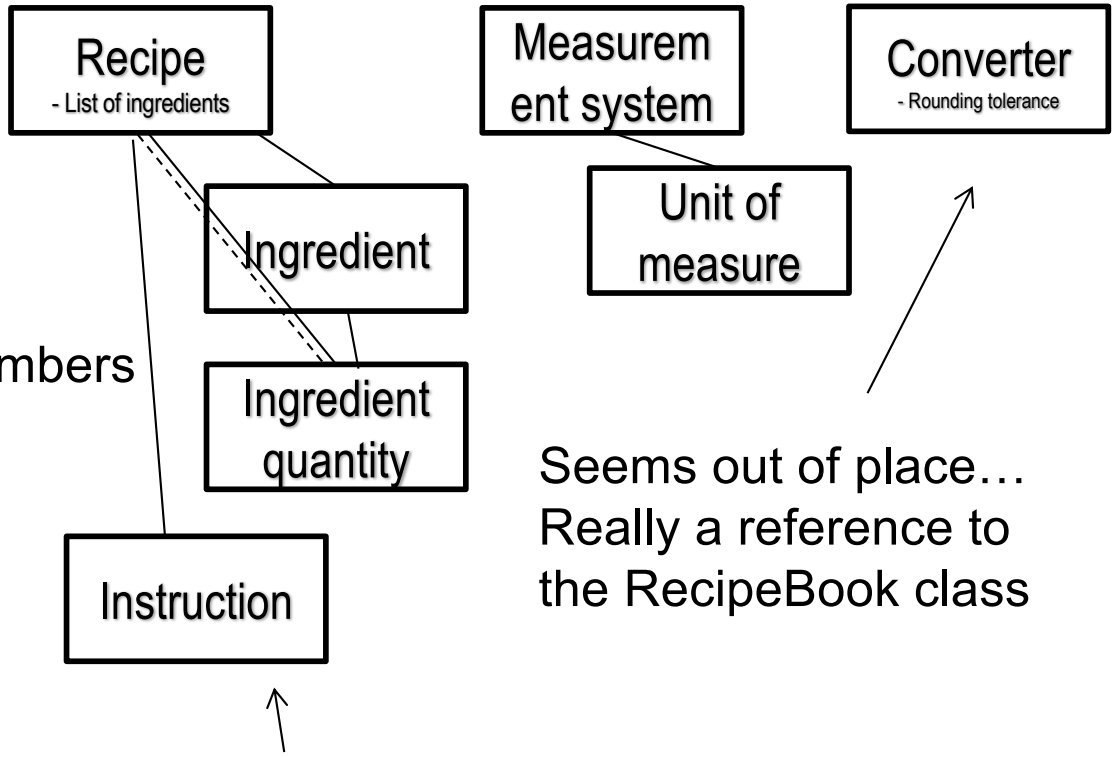
Assignment 1

(cooking) recipes
- list of ingredients
  - ingredient quantities
    - integers
    - fractions
    - decimal numbers
- instructions

Measurement systems
- units of measure

Converter
- rounding of quantities
- rounding tolerance

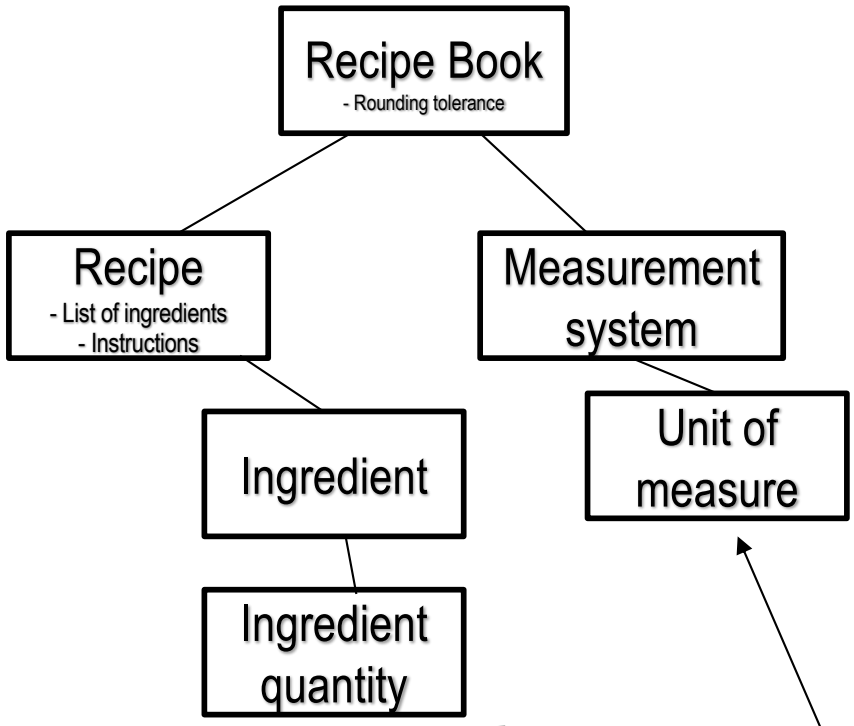
~~People~~



Seems out of place…
Really a reference to the RecipeBook class

Just a string, so a full class seem excessive

Assignment 1



Might still be too little distinct to be its own class

Assignment 2

- Write a class called “AmortizedTree” that accepts data values and then lets you search the list to see if a value is in the list. The key part of the class is in the data structure that it uses to store the data.

- At its core, the AmortizedTree is an unbalanced binary search tree. The tree actually has two underlying data structures: the unbalanced tree and an array of values that are waiting to be added to the unbalanced tree.

- New values are added to the array of values. Once that array reaches a sufficient size, we insert all the value of the array into the tree. Since we have all the values in the array, we can select an order of insertion that avoids the worst case unbalancing of the tree. For example, if the array values are 7, 12, 18, 23, and 45 then inserting the values in the order 17, 12, 18, 23, 45 unbalances the tree while inserting in the order 18, 7, 12, 23, 45 has a better chance of keeping the tree balanced. The idea is then to insert the middle element first and then the middle elements of what is left to either side of that element.

- The size of the array of values awaiting to be added into the unbalanced tree should be the ceiling of the logarithm (base 2) of the number of values in the unbalanced tree. Start the array with size 1. When the unbalanced tree has 3 values, the array can hold 2 values. When the unbalanced tree has 5 values, the array can hold 3 values. When the unbalanced tree has 9 nodes, the array can hold 4 values. For simplicity, if

  elements are deleted from the unbalanced tree, you do not need to shrink the array.

Assignment 2

- Write a class called “AmortizedTree” that accepts data values and then lets you search the list to see if a value is in the list. The key part of the class is in the data structure that it uses to store the data.

- At its core, the AmortizedTree is an unbalanced binary search tree. The tree actually has two underlying data structures: the unbalanced tree and an array of values that are waiting to be added to the unbalanced tree.

- New values are added to the array of values. Once that array reaches a sufficient size, we insert all the value of the array into the tree. Since we have all the values in the array, we can select an order of insertion that avoids the worst case unbalancing of the tree. ~~For example, if the array values are 7, 12, 18, 23, and 45 then inserting the values in the order 17, 12, 18, 23, 45 unbalances the tree while inserting in the order 18, 7, 12, 23, 45 has a better chance of keeping the tree balanced. The idea is then to insert the middle element first and then the middle elements of what is left to either side of that element.~~

- The size of the array of values awaiting to be added into the unbalanced tree should be the ceiling of the logarithm (base 2) of the number of values in the unbalanced tree. ~~Start the array with size 1. When the unbalanced tree has 3 values, the array can hold 2 values. When the unbalanced tree has 5 values, the array can hold 3 values. When the unbalanced tree has 9 nodes, the array can hold 4 values.~~ For simplicity, if elements are deleted from the unbalanced tree, you do not need to shrink the array.

DALHOUSIE UNIVERSITY
Inspiring Minds

Assignment 2

- Write a class called “AmortizedTree” that accepts data values and then lets you search the list to see if a value is in the list. The key part of the class is in the data structure that it uses to store the data.

- At its core, the AmortizedTree is an unbalanced binary search tree. The tree actually has two underlying data structures: the unbalanced tree and an array of values that are waiting to be added to the unbalanced tree.

- New values are added to the array of values. Once that array reaches a sufficient size, we insert all the value of the array into the tree. Since we have all the values in the array, we can select an order of insertion that avoids the worst case unbalancing of the tree.

- The size of the array of values awaiting to be added into the unbalanced tree should be the ceiling of the logarithm (base 2) of the number of values in the unbalanced tree. For simplicity, if elements are deleted from the unbalanced tree, you do not need to shrink the array.

Assignment 2

- Write a class called "AmortizedTree" that accepts data values and then lets you search the list to see if a value is in the list. The key part of the class is in the data structure that it uses to store the data.

- At its core, the AmortizedTree is an unbalanced binary search tree. The tree actually has two underlying data structures: the unbalanced tree and an array of values that are waiting to be added to the unbalanced tree.

- New values are added to the array of values. Once that array reaches a sufficient size, we insert all the value of the array into the tree. Since we have all the values in the array, we can select an order of insertion that avoids the worst case unbalancing of the tree.

- The size of the array of values awaiting to be added into the unbalanced tree should be the ceiling of the logarithm (base 2) of the number of values in the unbalanced tree. For simplicity, if elements are deleted from the unbalanced tree, you do not need to shrink the array.

Assignment 2

Data values
List
Value
Class
Data structure
Data
AmortizedTree
Unbalanced binary search tree
Tree
Underlying data structures
Unbalanced tree
Array of values
New values
Array

Sufficient size
Values of the array
Tree
Values in the array
Order of insertion
Size
Array of values
Unbalanced tree
Logarithm
Number of values
Unbalanced tree
Elements unbalanced tree
Array

Data values / value / data
AmortizedTree / Tree
- unbalanced binary search tree
  - number of values
  - elements
- array of values
  - new values
  - values of the array
  - size

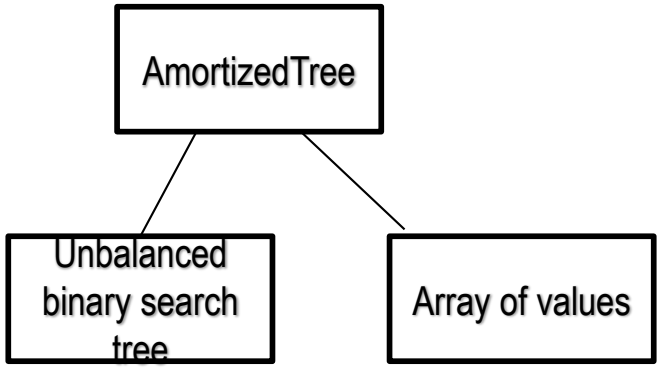
Logarithm
Order of insertion

DALHOUSIE UNIVERSITY
Inspiring Minds

Assignment 2

Data values / value / data
AmortizedTree / Tree
- unbalanced binary search tree
  - number of values — Size (integer)
  - elements — Key -> String
- array of values
  - new values — Key -> String
  - values of the array — Key -> String
  - size — Size (integer)

Logarithm — Math library already
~~Order of insertion~~



DSU Online Elections

- The DSU Executive will call an online election. They will appoint an election officer who will then validate and publish a slate of candidates for each DSU position as well as zero or more plebiscite yes/no questions. The election officer will obtain a list of valid students from the Registrar's Office as candidate voters. At the election, the election officer will open voting for a pre-determined period of time. Candidate voters will authenticate themselves to the voting system and then cast a vote for each DSU position and plebiscite. After the close of the election, the election officer will tabulate and publish the results.

DSU Online Elections – Nouns and Verbs?

- The DSU Executive will call an online election. They will appoint an election officer who will then validate and publish a slate of candidates for each DSU position as well as zero or more plebiscite yes/no questions. The election officer will obtain a list of valid students from the Registrar's Office as candidate voters. At the election, the election officer will open voting for a pre-determined period of time. Candidate voters will authenticate themselves to the voting system and then cast a vote for each DSU position and plebiscite. After the close of the election, the election officer will tabulate and publish the results.

Do's and Don't of the Noun-Verb Method

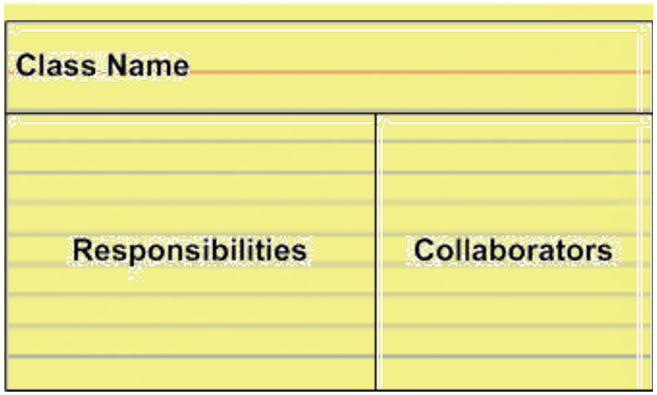
- Nouns should be concrete or conceptual from the problem domain:
  - E.g.,
    - Good classes: Student, Course, Grade, Grade Summary
    - Bad classes: Grade Sorter, Grade Summarizer
- Existing Classes may be named differently
  - E.g., Line-up vs Queue
- Avoid turning actions into classes
  - E.g., SortGrades
- Don't over-do. Decide when various data items in your program can be represented using basic types

What's Next?

- After identifying classes we need to figure out
  - What they do: Responsibilities
  - How they interact: Collaborate
- This is an iterative process
  - We consider each class and ask the above two questions
  - We then revisit the classes we have looked at earlier and refine our answers
- We use the CRC method

The CRC Method

- **CRC = Class, Responsibilities, Collaborators**
- **Idea:**
  - **Use an index card for each identified class**
  - **Divide card into three section:**
    - Class name
    - Responsibilities : The verbs / methods that the class is responsible for implementing
    - Collaborators : Other classes / objects that will be used to implement the responsibility
  - **Iterate through all the verbs / methods and add them to the responsibilities section of the respective class**
  - **Identify the collaborating classes**
  - **Ensure that collaborators provide the necessary methods in their responsibilities**



DSU Online Elections – CRC

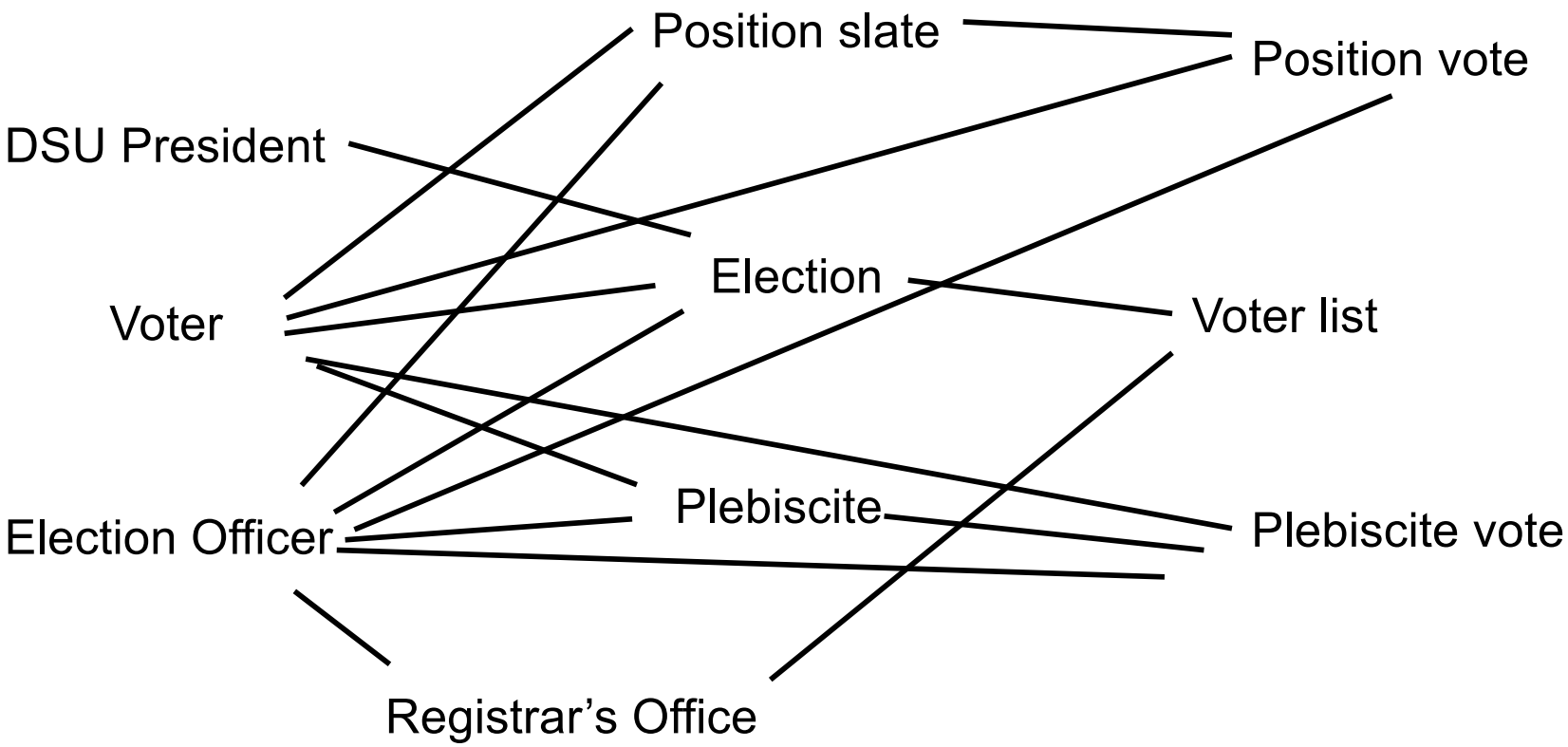
DSU President	
Appoint election officer	Election

Election officer	
Create slates Validate slates Publish slates Create plebiscite Load valid students Open voting Close voting Tabulate results Publish results	Position slate Plebiscite Election

Voter	
Authenticate Vote for position Vote on plebiscite	Election Position slate Position vote Plebiscite Plebiscite vote

Election	
Set position slate Set plebiscite Open Close Authenticate voter Record position vote Record plebiscite vote Tabulate results	Election officer Voter Position slate Plebiscite

DSU Online Election



Messy, but it's a start … not the end design.

CRC Outcome

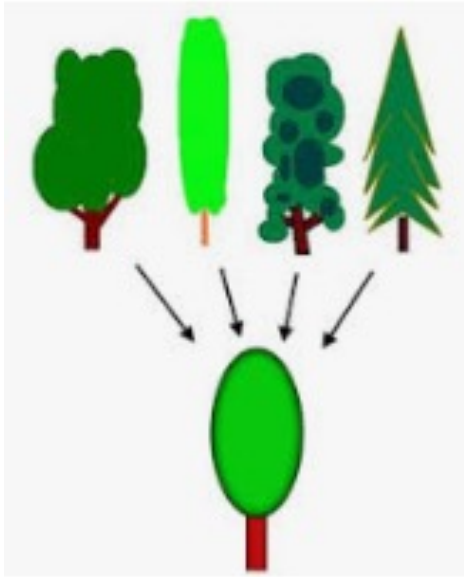
- Obtain a preliminary map of classes and relations between classes.
- Apply refinements to these classes
  - SOLID properties
  - Abstractions
  - Other heuristics

Heuristics for Design

- Form abstractions
- Encapsulate implementation details
- Use inheritance
- Hide secrets
- Identify areas likely to change
- Anticipate different degrees of change
- Keep coupling loose
- Look for common design patterns

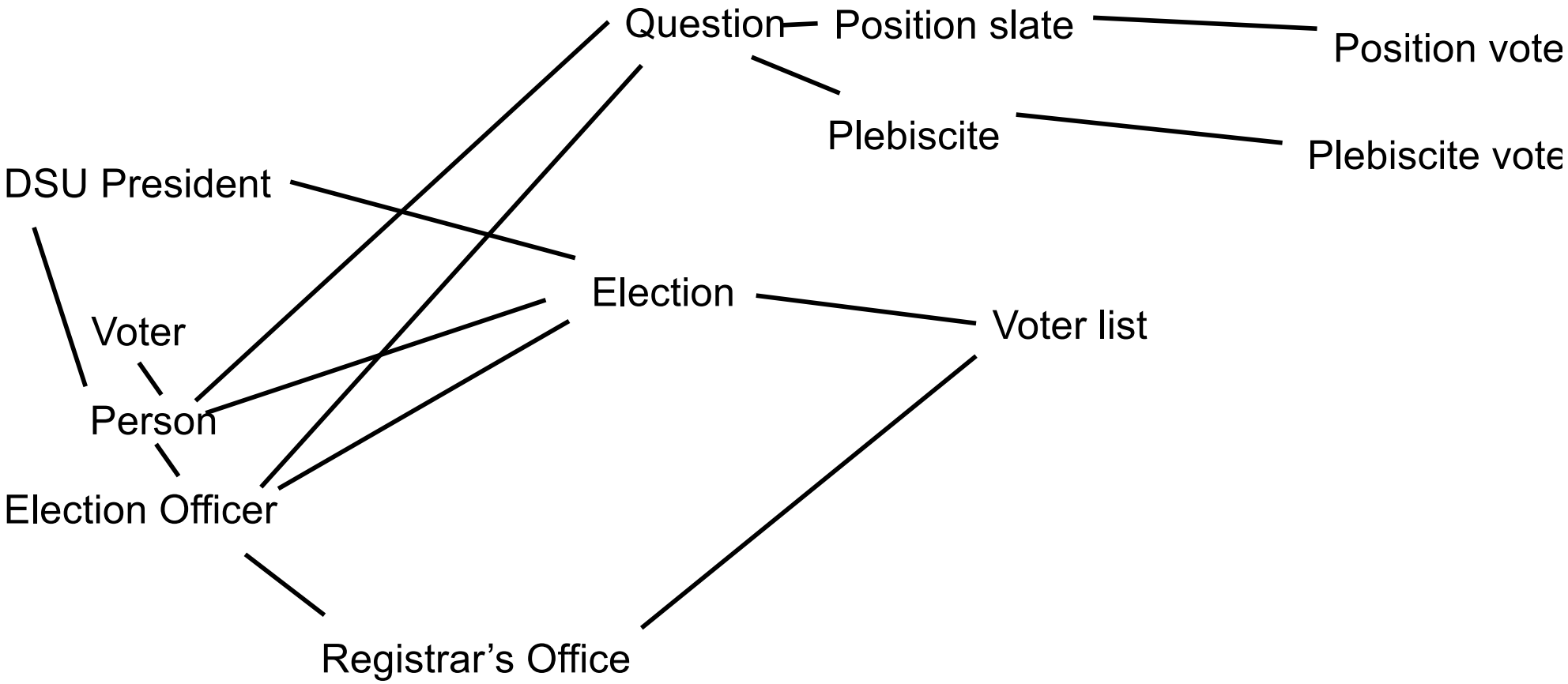
Form Abstractions

- Keep to abstracts to focus on the big picture.
- Include abstractions whenever possible to allow for
  - Portability
  - Delaying the point when you need to commit to implementation details
- See SOLID Dependency Inversion



DALHOUSIE UNIVERSITY
Inspiring Minds

DSU Online Election

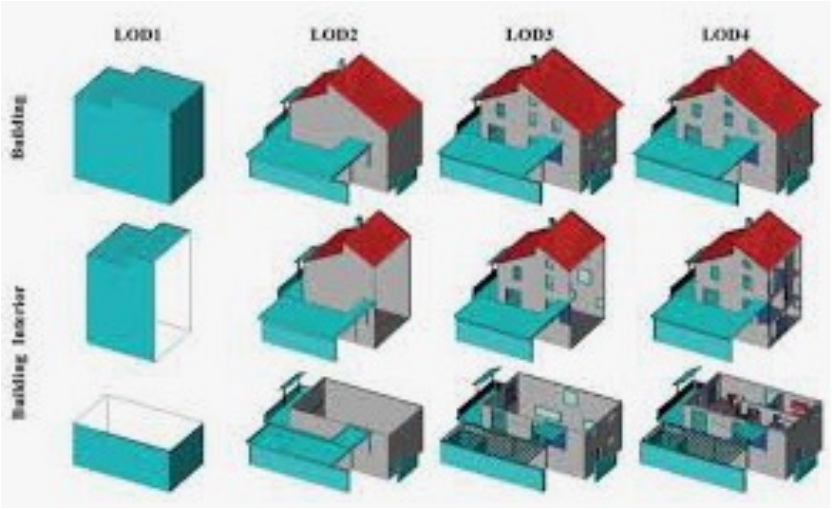


Messy, but it's a start … not the end design.

Encapsulate implementation details

- Resist exposing implementation details
- Provide a consistent level of access across all public methods of a class
- Provide a consistent level of abstraction across all classes

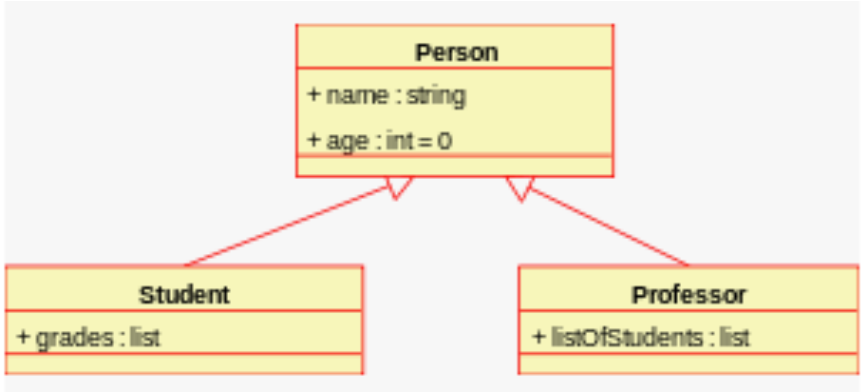
- Complements abstraction



DALHOUSIE UNIVERSITY
Inspiring Minds

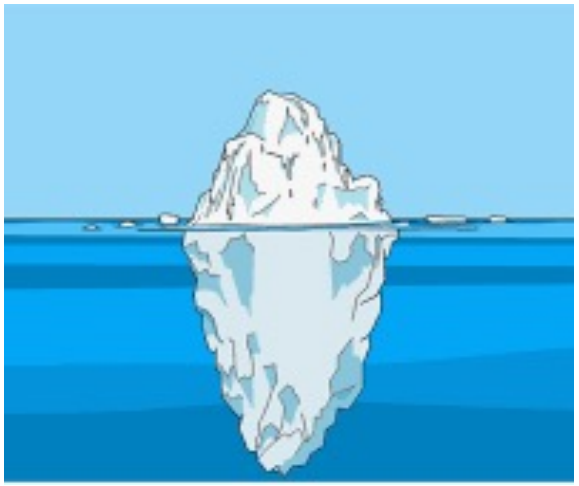
Use inheritance

- **Seek commonalities across classes**
  - Gather the commonalities into a base class
  - Encode the common code and attributes as the base class



Hide secrets – information hiding

- Do not let other packages or classes access the details of another class
  - Resist public (or even protected) attributes
  - Avoid having other classes rely on knowing which algorithm you are using
- Hide the complexity of the task or the solution
- Hide or isolate areas that are more likely to change



DALHOUSIE UNIVERSITY
Inspiring Minds

Barriers (perceived or not) to hiding secrets

- Some information is used everywhere / must be distributed
  - Opportunity to redesign to simplify and centralize the key distributed data
- Design includes circular dependencies of information
  - Re-encapsulate data, but beware breach of single responsibility
- Confusion between class data and global data
  - "Global data" probably belongs in a separate config class
- Performance penalties (perceived or real)
  - Question whether it's truly performance or convenience