Design Review Presentation

- PowerPoint mandatory
  - Use standard school format and logos available here (https://sites.google.com/a/scu.edu/soe-intranet/current-undergraduate-students/proposals-presentations)
- 10 minute presentations, 5 minute Q/A
- All team members must present
- All elements of your design must be presented
- Speak fluently, demonstrate good body language, transition speakers fluidly
- Dress professionally
- Stay for every group's presentation
- PRACTICE

Presentation Evaluation

Group Members:					
Project:					
Evaluator:					
Criteria:	Excellent	Good	Average	Fair	Poor
Quality of Content					
Knowledge of Subject					
Verbal Communication					
Nonverbal Communication					
Quality of PowerPoint slides					
Meets Dress Code					
Total Score					

Improving Presentations

- Slides
- Practice
- Dress Code

Kelsey Dedoshka

- https://developer.apple.com/videos/play/wwdc2017/239/

Improving Presentations

- Slides
  - Less is more
  - Bullets, not sentences
  - Don’t read
  - Face the audience
  - Set up what’s coming next
  - Be very picky about errors
  - Use SCU Engineering template
- Practice
  - Memorize your part
  - At least 3 times in room where you’ll present
  - Record practices
  - Bring dongles
- Dress Code

Design Concepts

- Formally define
  - **Modularity**
  - **Abstraction**
  - **Refinement**
  - **Information hiding**

Modularity

- Dividing a system into independent components that can be independently managed
  - Based on idea that solving two complex problems together is more effort than solving the two problems separately
  - Lets implementers focus and understand what they're doing
- How many modules?
  - Too few means too much complexity in each
  - Too many means too much complexity in connections between modules

Abstraction

- A design method that temporarily hides lower-level details of systems
- Common in people dealing with complex issues
  - Data abstraction: door instead of collection of knob, hinges, panels, …
  - Procedural abstraction: opening a door
  - Control abstraction: semaphores, locking, events, …
- All programming languages support procedural abstraction
- Most PL support data abstraction
- Few PL support control abstraction

Refinement

- Sometimes referred to as **stepwise refinement** or **elaboration**
- Design method that uses top-down approach to successively reveal levels of detail
- Abstraction and refinement are complementary
  - Refinement works top-down to gradually reveal details
  - Abstraction hides un-implemented details of lower levels

Information Hiding

- Modules hide design decisions from other modules
  - Details of data, procedural operations
- If details change, client modules are shielded from the change

OO Design Principles

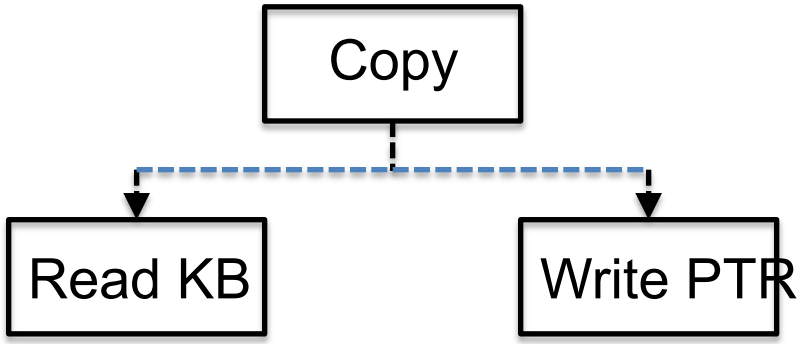
- Single-Responsibility
  - Classes should have one clearly understood responsibility
  - Relatively narrow rather than broad focus
  - Creates high level of cohesion
- Open-Closed
  - “An artifact should be open for extension but closed for modification in ways that affect its clients”
  - Especially applies to modules, classes and functions
  - Implies an artifact can be extended without changing existing code

OO Design Principles (cont.)

- Liskov Substitution
  - “Subclasses should be substitutable for their base class”
  - Implies any code referencing a base class will operate correctly if passed a subclass

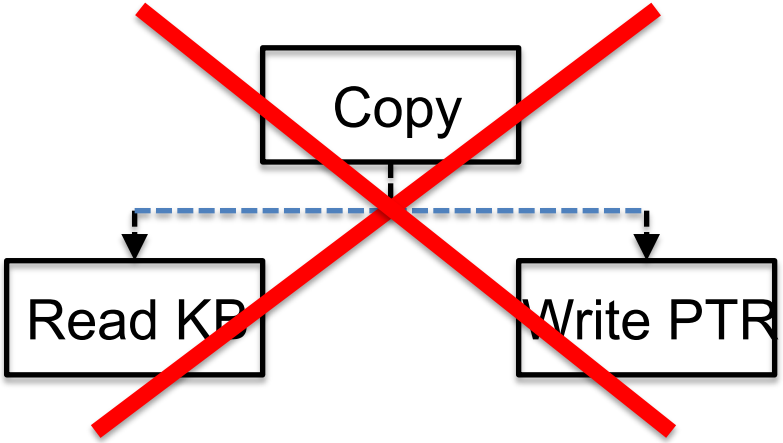
OO Design Principles (cont.)

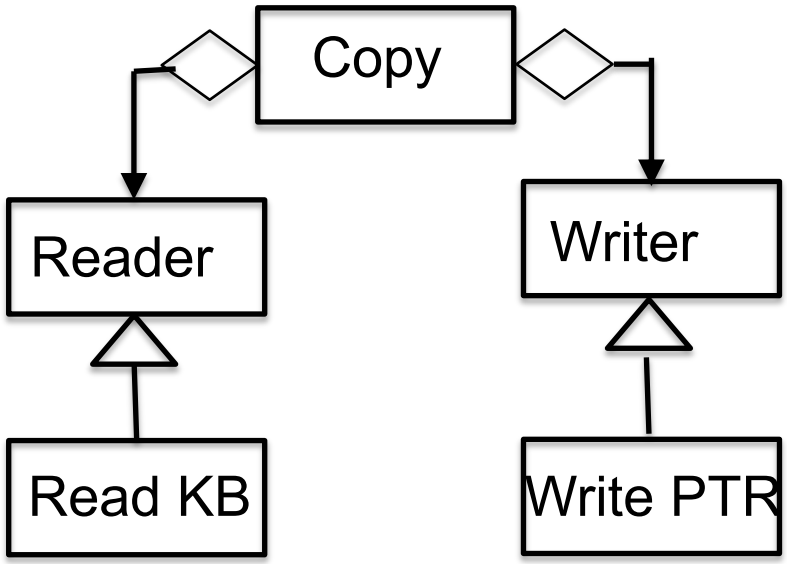
- Dependency Inversion
  - "Depend on abstractions, don't depend on concretions"
  - "Details should depend on abstractions, not the other way around"



OO Design Principles (cont.)

- Dependency Inversion
  - "Depend on abstractions, don't depend on concretions"
  - "Details should depend on abstractions, not the other way around"





OO Design Principles (cont.)

- Interface Segregation
  - "Many client-specific interfaces are better than one general-purpose interface"
  - Package methods into small, manageable sets of related methods
    - Don't package unrelated methods just because they're needed together

Architecture to Design Process

- Start with architectural model, domain, and use cases
  - Identify classes inherent in architecture and in problem domain
- Add classes that connect architecture classes with domain classes
  - Start with riskiest or highest priority elements first, try alternatives
- Refine design, make elements consistent, ensure design is complete

Architecture to Design Process (cont.)

- (for each class) Specify class invariants
- (for each method) Specify pre- and post-conditions, create activity diagrams, write pseudocode
- (for each unit/module) Sketch unit test plan
- Inspect test plans and design, refine as needed

Specifying Classes

- Identify all the attributes for this class listed in the SRS
- Add additional attributes required by detailed design
- Associate and name a method with each of the requirements satisfied by the class
- Name additional methods required by detailed design
- Create class model for the class
- State class invariants