Object Orientation

Good Practices

Conceptual Modeling

Conceptual Modeling

- What?
  - Capture ideas and concepts in a form that allows other activities, reasoning, analysis, prioritization, design, etc.
- Why?
  - To better understand a real-world system
  - To visualize a system as it is
  - To envision a system as we want it to become
  - To discover and document key decisions
  - To specify a solution (design)
  - To create a plan that can guide construction
  - To COMMUNICATE efficiently

Conceptual Modeling

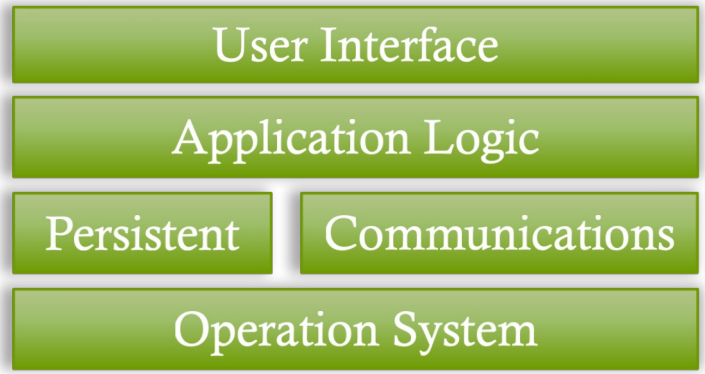
- When
  - Any time during the software development process, starting at very beginning, and from a variety of perspectives:
    - Analysis
    - Design
    - Specification
    - Implementation
    - Deployment
- See slides called “Introduction to Conceptual Modeling”

Conceptual Modeling with UML

- UML (Unified Modeling Language) is a collection of conceptual modeling languages, including
  - Use case diagrams – for describing those who need a software system and their goals
  - Class diagrams – for describing the classes of objects that make up a software system and their relationships with each other
  - Interaction diagrams – for describing how objects interact/communicate with each other
  - State charts – for describing the internal behavior of objects in a class or of a method
- See slides called "UML – Quick Reference Guide"

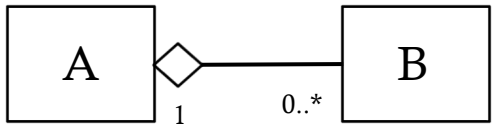
Organize Software into Loosely Coupled Layers

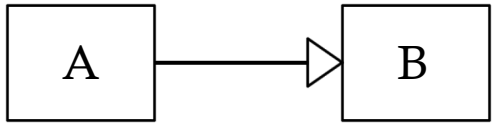
- Design your software system in layers
  - Each layer should provide a well-defined abstraction to higher-level layers
  - Each layer may use the components in its own layer
  - Each layer may use the abstractions provide by lower-level layers
- Example of layers:



Prefer Aggregation Over Inheritance

- In class-based languages, there are two fundamental ways to achieve reuse:
  - Aggregation, e.g., Object A contains object B (and therefore can "reuse" object B)
  - Inheritance, e.g., Class A inherits from Class B (so, A objects are B object and so they have all the capabilities B objects)
- Aggregation relationships can be changed at runtime and therefore can more flexible
- Therefore, when all other consideration are equivalence, choose aggregation over inheritance.
- Note: composition is a stronger form of aggregation





Program to an Interface or Abstraction

- One way to manage coupling that helps remove accidental complexity and eliminate unnecessary dependencies it to "program to an interface or abstraction"
- Interfaces, in the same of "class interfaces", are a type of abstraction
- A class's interface is all of its "public" methods or properties

Use Identifiers that Improve Readability and Maintainability

- Readability and maintainability can be greatly improved by choosing identifiers (variable name, method names, class names, etc.) that are meaningful within their scope.
- A class represents sets of objects and should be named after a prototypical instance in that set. For example, a set of widgets, should be called Widget. A class whose objects are themselves sets of widgets, would be called Widgets.
  - Class names should be nouns or noun phrases
  - Class names are typically in Pascal Casing, e.g., RighHandedWidget
- A variable holds an object, and therefore its name should represent the purpose or intended use of that object
  - Variables names typically use lowerCamelCasing, e.g., myWidget.
- Methods perform actions, and therefore should have verb or verb phases for names, e.g., Display or computeCost.

Testing with Executable Unit Test Cases

- Testing individual units of code with executable unit test cases
  - Helps find design errors
  - Helps developers verify correctness of the code, with repeatable test cases
  - Helps document the usage of a method or class
  - Enables regression testing
- Use testing frameworks to facilitate the construction and execution of executable test cases

WHAT IS NEXT…

UML