Concepts in Object Orientation

Object Identity

- Every object is a unique entity in the universe, distinguished by its properties.
  - Many of those properties may not be observable or interesting to a software system
  - In a software system, even if two objects have the same known properties, they are not the “same” object

Encapsulation

- See David Parnas' paper from 1972
- Encapsulation is the act of "hiding" or placing decisions in software component, like a method or a class
- In software systems, good encapsulation involves both the
    - Hiding of design decisions, and
    - Localization of design decisions

Abstraction

- Abstraction is the act of summarizing or generalizing something to focus on the ideas most relevant to a conversation or certain kind of communication.

- In object orientation, abstraction (the verb) is the creation of interfaces of a components, i.e., a class, that exposes certain details necessary for working with that component.

- An abstraction (the noun) is a description that leaves out unnecessary details.

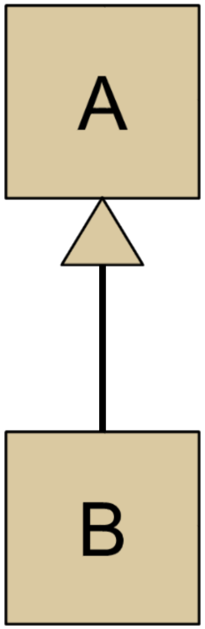
- “interfaces”, as found in C# and Java, are abstractions, but so are abstract classes, abstract methods, the publically visible classes in a namespace, and more.

Classification

- Classification (the verb) is the process of grouping objects together into sets based on common properties
- A classification (the noun) or "class" is a set of objects that share certain common properties
- In object orientation, we use classification during analysis to better understand the problem and during design to help structure a solution
- The "class" construct found in most OO languages comes from this idea of classification.
- A class in most strongly typed OO languages is an Abstract Data Type (ADT) that defines the common properties (e.g., data attributes and behaviors) of the objects in the class.
  - Typically, it does not direct keep track of the objects of the class.
  - In a sense, it is a template from which new instances of the class can be created

Sub-classification (Specialization)

- Sub-classification, also called specialization, involves identity subsets of another class.
- Conceptual, if class B inherits from class A, then all the objects in class B are also in class A.
  - The set of objects in B is a subset of those in A
  - All properties or capabilities common to all objects A are also common to all objects B
- In software, class inheritance is a reuse mechanism, where a class B's definition is based on (reuses) class A's definition



Coupling

- Coupling is the manner and degree of interdependence between software components
- When there is high coupling, then there is a change to a software components causes a ripple changes in other components
- Where there is low coupling, then a developer can make change or extension with minimal impact on other components

Types of Coupling

- Causes of coupling, in an approximate severe-to-moderate order
  - **Content coupling** (as called pathological coupling): one module depends on encapsulated concepts within another module instead of an abstract
  - **Common coupling**: modules share un-encapsulated global data
  - **Control coupling**: one module controls the flow of another
  - **Stamp coupling**: Modules share a composite data structure but only need part of it
  - **Data coupling**: Modules share individual, elementary chunks of data, typically via parameter passing
  - **Message coupling**: State is decentralized, and components communicate only via messages passing or method calls

Cohesion

- Cohesion refers to the degree to which the elements of a module belong together
- Cohesion is a measure of the strength of relationship between pieces of functionality with a given module.
- High cohesion leads to
  - Reduced module complexity
  - Increased maintainability
  - Increased reusability

Types of Cohesion

- Causes of bad cohesion, in an approximately severe-to-moderate order
  - **Coincidental cohesion**: components grouped together arbitrarily
  - **Logical cohesion**: components grouped together because they are logically categorized to do the same kind of thing
  - **Temporal cohesion**: components grouped together because they process close together in time
  - **Procedural cohesion**: components grouped together because they always follow a certain sequence of execution
  - **Communicational/informational cohesion**: components grouped together because they operate on the same data
  - **Sequential cohesion**: components grouped together because the output of one is the input of another
  - **Functional cohesion**: components grouped together because they all contribute to a single well-defined task

Modularization

- Modularization (verb) is the process by which developers break up a system into cohesive and loosely coupled components

- A modularization (noun) is the specific decomposition characterized by the components' abstractions

- The "goodness" of a modularization can be roughly assessed in terms of the
  - Coupling and cohesion of the components
  - Localization of design decisions
  - Strong encapsulations
  - Abstractions that reveal only what other components need
  - Whether a developer can fully understand a component's purpose and functionality by only looking at its implementation, and perhaps those components that use it directly or the components that it uses directly.

WHAT IS NEXT…

Objects and Classes

1. Galaxy
2. Video Camera
3. Mouse
4. Flash Drive
5. Pencil
6. Sound Mixer
7. Cellphone
8. Cable
9. Compact disc
10. Vinyl record
11. Microphone
12. Notebook

13. Cathedral
14. Stand
15. Table
16. Goalkeeper
17. Mask
18. Picture
19. SD card
20. Doll
21. Trash Bin/Garbage pan
22. Scissors
23. Soccer player
24. Soccer ball