ENSF 480 Notes

Brian Pho

October 11, 2017

Contents

Chapter 1

Review

1.1 Object Oriented Programming

- 4 Pillars of OO: Abstraction, Encapsulation, Hierarchy, Modularity
- Abstraction: ignore inessential details
- Encapsulation: information hiding
- Hierarchy: “has a” and “is a” relationship. Inheritance, aggregation, composition
- Modularity: dividing code up into loosely coupled modules
- 4 Properties: Identity, Properties, Functions, States
- Rule of Big 3 (C++)

1.2 Relationships

- Association: No hierarchy (Separate life, no ownership, no whole-part)

  Uses it in a method
- Aggregation: Whole/part (Separate life, ownership)

  Has a pointer to the object
- Composition: Lifetime (Connected lives, ownership, whole-part)

  Allocates memory for the object
- Inheritance: Generalization to specialization

  Extends a class
- Multiple Inheritance: A class can be derived from two or more superclasses (solve using virtual base)
- Polymorphism: Different objects react differently to the same message

- Realization: One element realizes (implements/executes) the behavior that the other model element specifies
- Delegation: Passing a duty off to something else (alternative to inheritance)

Chapter 2

Object Modeling

2.1 Introduction

- Model: simplification of reality
- Reasons to make Models: Low cost, verify understanding, test ideas, ease of communication
- UML: Unified Modeling Language

2.2 Classes and Objects

- Class name, attributes, functions
- + (Public), - (Private), # (Protected), / (Derived),   (Package)
- *italics* (Abstract), parameter (Generic/Template), underline (static)
- Packages are represented by grouping classes
- Navigability: Arrows on association pointing to class that it can change
- Cardinality/Multiplicity: Expresses quantity of relationship
- Stereotype: Defines a new model element

2.3 Interaction (Sequence) Diagram

- Shows the interaction between a set of objects and their relationships
- Is dynamic and aids in knowing which classes should implement which functions
- Sequence Diagram: an interaction diagram that emphasizes the time ordering of messages
- Shows successful interactions
- Focus of Control: A tall, thin rectangle that shows the period of time during which an object is performing an action

2.4 State Transition Diagram

- Shows the dynamic flow of control from state to state of a particular entity, as well as the behavior of classes in response to external stimuli
- States: represent conditions/situations during the life of an object
- Transition: arrow showing the path between different states. **Must be labeled**
- Initial State: Solid circle and only one may exist
- Final State: Bull's eye and multiple my exist
- Choice: Diamond representing a condition
- Can have reflexive transitions as well as terminating (marked with arrow towards an X)
- Guard Conditions: A Boolean expression that is evaluated when the transition is triggered
- Composite States: A state that has sub-states (nested states)

2.5 Activity Diagram

- Shows the flow from one activity to another activity (flow chart)
- Deals with dynamic aspects of the system and deals with all types of flow (sequential, branched, concurrent)
- Initial and final are same as state diagram
- Use bar to show a forking of control

Chapter 3

Design Patterns

3.1 Introduction

- Challenges: changes due to requirements, scaling, new technology.
- Design Pattern: represent the best practices used by experienced OO developers
- Benefits: saves times, common vocabulary, design reuse, documentation
- Three Types of Patterns: Creational (abstracting the object-instantiation process), Structural (how objects can be combined to form larger structures), Behavioral (communication between objects)

3.2 Strategy Pattern

- Separate changeable behaviors
- Program to interface not implementation
- Create concrete classes responsible for changeable behaviors
- Strategy: algorithms are separated from a class and encapsulated as a separate class
- Each strategy implements one behavior
- Allows changing the object's behavior dynamically without extending/changing the object itself
-

3.3 Observer Pattern

-