340 Review Fall 2016

Midterm 1 Review

Concepts

A. UML Class Diagrams

1. Components: Class, Association (including association name), Multiplicity Constraints, General Constraints, Generalization/Specialization, aggregation/composition, attributes
2. Conceptual Model
3. Design Model
    a. at higher level Many to Many often preserves or replicates associations or aggregations
    b. 1 to many, many to 1, and 1 to 1, (also 0:*) usually become attributes
4. Draw a UML class diagram to model something
    a. Include specific components
    b. Syntactically correct
    c. Complete

B. Design Principles

1. Single Responsibility Principle/Cohesion
2. Information Hiding -- hide implementation details
    a. Specification/Abstract View
        1. Domain
            a. atomic
                1. restricted atomic
            b. composite/aggregation
            c. structured (set, ordered set, multi-set, sequence, tree, map, graph)
            d. Invariants
                1. Instance
                2. Class Invariants
                    a. Treating the class as an object
        2. Behavior/Method Specification
            a. Pre-condition
            b. Post-condition
            c. What about static methods?
    b. Implementation in languages
        1. Use of visibility modifiers such as public, private, protected
    c. In languages: C++ JavaProblems
        1. Spec vs. Implementation
            a. C++ .h vs .cpp files
            b. Java: syntactically it really doesn't
                1. Javadoc
                2. Use of Interfaces and Javadoc
            c. Pre-conditions/Post-conditions
            d. Domain definition
                1. Invariants
                2. Class Invariants
    d. Good practices
        1. Make as many fields and methods private as possible

a. Make the “interface” as thin as possible – fewest methods possible
2. Keep parameters as few and simple as possible
3. Hide inherited fields

3. Coupling/Cohesion
   a. Class perspective
   b. Method perspective
   c. Less Coupling == Higher Cohesion

C. Generalization/Specialization
1. Conceptual
2. Implementation
   a. Inheritance
      1. Why this is not real generalization/specialization
   b. Composition
3. Design by Contract
   a. Contract perspective
   b. Pre-condition
      1. Which person should satisfy the pre-condition?
      2. If the pre-condition is false will the method fail?
      3. When/where should be pre-condition be checked?
      3. Use of assertions/exceptions as defensive programming
         a. Frowned upon by some
         d. defensive programming – checking requirements
   c. Post-condition
      1. Which person should satisfy the post-condition?
      2. When is the developer bound to satisfy the post-condition?
      3. Whose fault is it if the pre-condition is true but the post-condition is false?
   d. Math equation
4. Good practice
   a. Access a class only through methods
   b. Every field is private
   c. Why are protected fields in Java a little bit of a problem
      a. Does making all fields private solve the problem?
   d. Don’t let names expose unnecessary detail

D. Patterns -- How used and why useful?
1. Proxy
   a. Remote proxy
2. State Pattern
3. Façade Pattern
4. Observer Pattern
5. Singleton
6. Descriptions
   1. What is the problem?
      a. Give a specific example
   2. Describe general solution using
      a. UML like diagram
      b. Using English
   3. Give an example of how it can be used

a. How it was used in Catan

b. Why it was useful

7. Given a problem or partial example, demonstrate how you would use the pattern and how you would implement it in java.

E. Layers

1. Benefits
   a. Layer reuse, modification, replacement
   b. Reduce dependency (how?)
   c. Easier to understand
2. Behavior
   a. Down to Bottom then Up (Scenario I)
   b. Down to Intermediate Level then up (Scenario II)
   c. From Bottom to Top (Scenario III)

H. Dependency Inversion – disconnect to minimal abstraction

1. Problem – A.x calls B.y but doesn't want to be dependent on B.y's signature and semantics (pre-, post- conditions) which can be changed by B at any time
2. Solution
   a. Separate B's implementation from its interface
   b. Put B's interface and semantics where A has sole control (in A or A's package).
   c. Alternative view: Dependency Inversion depends on two basic ideas. First, we should program to abstractions, not concretions. Second, the caller defines the interface through which the call is made, not the callee.

I MVC

1. What are the model, view, and controller and their responsibilities
   a. A controller is often many controllers each with own "View" perspective and "Model" Perspective
2. Two views
   a. V <-> P<->M – often called Model View Presenter
      1. What is a View – set of views
         a. Each view object has a corresponding presenter
      2. What is a Model
      3. Interactions

1 – Presenter/Controller queries Model for data

2 – Presenter/Controller tells View what data to display

3 – View draws data on screen

4 – View passes user input to Presenter/Controller

5 – Presenter/Controller

i. Queries state of View (if needed)

ii. Tells Model to change its state

iii. Tells View to change its state (if needed) (e.g., enable/disable, error messages, sort, select/unselect)

6 – Model notifies Presenter/Controller that the model state has changed

7 – Presenter/Controller queries Model for new state

8 – Presenter/Controller tells View what data to display

9 – View draws new data on screen

b. V -> C -> M -> V

c. How are connections made

1. Call backs (handlers)
2. Observer Pattern

d. How does a “Server” fit in?

J SQA

1. Verification vs Validation
2. Reviews – one of two primary types of SQA activity
    1. How to Conduct
    2. Most effective!
3. Testing – one of two primary types of SQA activity
    1. Theory
    2. Black box – description or use on example
        a. Equivalence Partitioning
        b. Boundary Value Analysis
        c. Additional Forms
            1. Error Guessing
            2. State Transition
            3. Comparison
            4. Testing race conditions
            5. Performance
            6. Limit
            7. Stress
            8. Random
            9. Security
            10. Usability
            11. Recovery
            12. Configuration
            13. Compatibility
            14. Documentation
        d. Given code, how would you do black box testing (Equivalence and BVA)
    3. White box
        a. Coverage
            1. Line
            2. Branch
            3. Complete Condition
        b. 4 Forms – explain
            1. Relational
            2. Loop
            3. Internal Boundary
            4. Dataflow
        c. Given code, what test cases would you consider to perform the 4 types of white box testing, why?

4. Regression Testing
    a. What, Why, How
    b. Automation
5. Formal Verification
6. Unit, Integration, System, Acceptance Testing