340 Final Review Fall2017

A. UML Class Diagrams

1. Components: Class, Association (including association name), Multiplicity Constraints, Notes, Generalization/Specialization, aggregation/composition, attributes
2. Conceptual Model
3. Design Model
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. Implementation in languages
        1. Use of visibility modifiers such as public, private, protected
    b. In languages:
        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. 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. Minimize Dependencies
    a. Demeter's Law

C. Generalization/Specialization

1. Conceptual: Sets, property preservation, substitutability
2. Dynamic vs. Static specialization
3. Member of one vs many classes
4. Implementation
    a. Static-Single: simple inheritance
    b. Static-Multiple: multiple inheritance
        1. How can you do it in Java
    c. Dynamic-Single: Prestige effect (can use dynamic-multiple but guarantee single)
    d. Dynamic-Multiple: Decorator
        1. How to implement roles (multi-specialization, dynamic membership)
    e. Composition alone is NOT specialization, it does not provide substitutability
        1. Which to use for "has-a", "uses-a", static "is-a"
        2. How to use composition for reuse
            a. Why is it bad to use inheritance for reuse?
        3. How to use composition for specialization
5. 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 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. Mathematical formulation (pre $\Rightarrow$ post)
   e. Specialization and design by contract
      1. Relationship between pre-conditions
         a. $\text{pre}_g \Rightarrow \text{pre}_s$ or, if changed, pre-conditions must only be weakened
      2. Relationship between post-conditions
         b. $\text{post}_s \Rightarrow \text{post}_g$ or, if changed, post-conditions must only be strengthened
      3. Informal semantics of behavior specialization
         c. Don't lie
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 –
   a. Why?
   b. How?
3. Façade Pattern
4. Observer Pattern
5. Singleton
6. Abstract Factory
7. Adapter, Decorator, and Proxy –
   a. How are they similar
   b. How are they different
   c. Be able to show how to use each one
8. Template method pattern, Strategy pattern – How are they similar, how are they different
   c. Be able to show how to use each one
9. Visitor pattern - implement
10. Plugin pattern
11. Command pattern
12. Template Method Pattern -- implement
13. 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 Ticket to Ride

b. Why it was useful

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

E. Layers

1. Benefits

a. Layer reuse, modification, replacement

b. Reduce dependency (how?)

c. Easier to understand

F. Minimizing Dependency

1. Dependency Inversion – Problem that solves, benefits, how to write code

2. Dependency Injection – Problem that it solves, benefits, how to write code for it

a. Dependencies injected into client, dependencies defined by Interface, different dependencies (e.g. Mock objects) can be used

b. Implementation

1. Classes should implement interfaces so that mock objects can be used

2. Classes should not internally create dependencies using new.

3. Classes should have all dependencies passed in using either the constructor or setters

3. Factory Pattern – How it works and how it is this different from Dependency Injection

G. MVC/MVP

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?

H. 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

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. 3 Forms

1. Relational

2. Loop

3. Internal Boundary

c. Given code, what test cases would you consider to perform the 3 types of white box testing, why?

4. Regression Testing

a. What, Why, How

b. Automation

5. Formal Verification

6. Unit, Integration, System, Acceptance Testing

I. Refactoring

A. Bad Smells

1. Duplicated code
2. Long method
3. Large class
4. Long parameter list
5. Divergent change
6. Shotgun surgery
7. Feature envy
8. Data clumps
9. Primitive obsession
10. Switch statements
11. Lazy class
12. Speculative generality
13. Temporary field
14. Message chains
15. Middle man
16. Inappropriate intimacy
17. Data class
18. Refused bequest
19. Comments

B. Violated principles

C. How to correct them – there may be more than one way