Design Review Items

(Red text implies it is required later in the semester)

1. Style

A. Good Names

1. Not too long (not a common problem)

2. Not too short (common problem)

a. Examples

1. No Simple characters (e.g. x, v)

2. Abbreviations should be obvious to the most casual coder

a. Bad Example – srt

3. Don’t just extract vowels

3. Methods

a. Action verbs

b. No Conjunctions (see Cohesion.A.2 below)

4. Variables

5. Classes

B. Consistent Indentation

C. Consistent use of {…}

1. Control Statements with single statements should still have {} if the statement is on a subsequent line.

D. Proper use of comments

1. Define “why” a possibly confusing design decision was made

2. Used to explain connection between implementation and specification

2. Maximize Cohesion

A. Class Cohesion

1. Represents a single concept (though it might be abstract)

a. Single Responsibility Principle

2. Everything (including assumptions) about the concept are in the class

3. Nothing pertaining to some other class is in this class

4. No primitive obsession – abstracting all the way

5. Avoid collection of method classes though a few may be good (e.g. the Math class)

B. Method Cohesion

1. Name should be active verb (with implicit application to containing class name)

2. Should not have conjunctions(and/or) in name

3. Belongs in this class (if not, the implicit application is to an instance of another class)

5. Don’t use switch statements where polymorphism is better

5. Should follow single Responsibility Principle

3. Minimize Coupling or Dependencies

A. Use dependency inversion where possible

B. No undocumented assumptions – try your best though this is hard to detect and grade

4. Proper Decomposition
   - A. Avoid Hypo- and Hyper-decomposition
   - B. Proper use of Packages and sub-packages
   - C. Avoid Speculative Generality
   - D. Classes and Methods of proper complexity
     - 1. Not too complex, this is often measured by size.
5. Proper Algorithm and Data Structure selection
   - A. Proper tradeoff in quality of choices such as speed, space, understandability, etc.
6. Hide as much information as possible in an implementation
   - A. Make as many things as possible, private
   - B. Make fields or attributes private
     - 1. Even inherited attributes – for inheritance make the getters and setters protected
   - C. Avoid message chains
7. Minimize code duplication
8. Proper Implementation of Basic Abstraction Constructs
   - A. Cognitive vs Implementation
     - 1. Separate Specification and Implementation
       - a. Domain
         - 1. Invariants
       - b. Methods/Constructors
         - 1. Well written pre-conditions
           - a. often implemented as “CanMethods”
         - 2. Well written post-conditions
       - c. Use Javadoc
       - d. Use Interface when needed
   - B. Aggregation – also see decomposition
   - C. Classification
   - D. Specialization
     - 1. Proper use of Inheritance for Specialization
       - a. Do NOT use inheritance for reuse (that is done with composition)
       - b. All Fields/Attributes should be private (see 6.B)
         - 1. Should be accessed via getters and setters
           - a. Even for inheritance of methods
     - 2. Proper use of Composition for Specialization
       - a. Object in multiple classes or roles
       - b. Object with differing behaviors (depending on state) – State Pattern
9. Constraints
   - A. Can Methods – often implements pre-conditions
   - B. IsValid Method --- often implements invariants
10. Use of common methods/constructors – not needed in every case but probably appear often
    - A. Appear often: toString method, equals method
    - B. Appear when needed: Default Constructor, copy constructor, iterator for collections