1. Items on midterm review sheet: Object-Oriented Concepts/Principles, UML Class Diagrams, CAD / CAM case study, Introduction to Design Patterns, Facade Design Pattern, Adapter Design Pattern, Agile Development Processes, Use Cases, Eclipse, CVS (w/Eclipse), Testing, and Chat Client.

2. Design Principles
   (a) design from context through “complexification”
      i. know that Alexander says “it is only possible to make a place which is a live by a process in which each part is modified by its position in the whole”
      ii. understand that “complexification” describes the process of starting with the “big picture” and using that as context to add more details one at a time
      iii. understand the “thinking in patterns” approach to software design
      iv. understand why this approach cannot be applied generally
   (b) open-closed principle
      i. know what the open-closed principle is and what it means
      ii. be able to explain why extension is better than modification
      iii. given a system, be able to identify changes that can be made through extension and those that require modification
      iv. be able to refactor a system so that it is open to extensions that will support a specific feature
      v. understand that the open-closed principle is a goal rather than a rigid requirement
   (c) design from context
      i. know what the dependency inversion principle is and what it means
      ii. know that coupling should be done at a conceptual level
      iii. be able to recognize code where the dependency inversion principle is violated
      iv. know what Liskov’s substitution principle is and what it means
   (d) encapsulate variation
      • understand why it is desirable to encapsulate variation
      • understand how encapsulating variation facilitates polymorphism and decoupling
   (e) one rule, one place
      • understand the importance of putting each rule in only one place
   (f) separating use from instantiation
      i. understand why it is recommended to defer deciding how to create objects until you’ve figured out what objects are needed
      ii. understand how factories decouple the code the logic of how to use an object from the decision of what objects to create
      iii. understand how using factories can increase cohesion
      iv. understand how using factories can decrease coupling
      v. understand how factories can help make code more open to extension

3. Commonality and Variability Analysis
   (a) understand that each class hierarchy should contain classes that have some commonality but which vary in some dimension
   (b) understand how commonality and variability analysis gets reflected in a class hierarchy at the conceptual, specification, and implementation levels
   (c) given a description of a design problem (set of requirements), be able to identify commonalities and variations
   (d) know what the analysis matrix is
   (e) understand how the analysis matrix can be used to systematically identify commonalities and variations
   (f) given a description of requirements, be able to write it in an analysis matrix
   (g) understand how an analysis matrix can help identify missing requirements
   (h) be able to read an analysis matrix
4. Design Patterns

(a) Strategy Design Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. understand how the Strategy Design Pattern can be used to encapsulate varying algorithms

(b) Bridge Design Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. understand how the Bridge Pattern can be used to decouple variations in an abstraction/interface from variations in implementation
iii. understand how it can be applied to the CAD/CAM problem

(c) Decorator Design Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. understand under what circumstances a Decorator can be an alternative to subclassing
iv. understand what advantages using Decorator has over subclassing (when it is appropriate)
v. understand how the Java I/O stream classes reflect the Decorator Design Pattern

(d) Observer Design Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. understand how the Observer increases decoupling and inverts the dependency between the caller and callee
iv. understand how listeners are used in AWT and implemented in the chat client

(e) Abstract Factory Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. understand that an Abstract Factory is appropriate when a family of classes that belong together needs to be instantiated in a consistent manner

(f) Template Method Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. understand how template methods facilitate reuse while being open to extension through subclassing
iv. understand that template methods only facilitate variation in a single dimension

(g) Singleton Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. be able to write code to implement a singleton
iv. know that the implementation using lazy instantiation of the Singleton is not thread-safe without additional synchronization
v. know that Double-Checked locking is broken

(h) Object Pool Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. understand how the Object Pool can provide a place for object management rules

(i) Factory Method Pattern
i. know the intent, problem, solution, participants and collaborators, consequences, and implementation
ii. be able to recognize situations in which it is appropriate
iii. know that factory methods are used to produce iterators in the Java, C#, and C++ collection libraries
iv. recognize the similarities between the Factory Method and Template Method design patterns

5. UML Sequence Diagrams

(a) be able to read and understand UML sequence diagrams
(b) given program code, be able to draw a sequence diagram for a particular operation