The midterm will test for your knowledge of the class prerequisites, as well as for the topics we’ll cover until the midterm in the class itself.

To help you find review materials for each of the prereqs/topics, [70.m.d] refers to the slide set of month $m$, day $d$, of the W 2010 edition of my CSE 70 class slides available at http://bit.ly/cwe9MR. You’ll easily find the corresponding segment within the slide sets. This CSE 70 website also has further pointers to the literature with detailed explanations for each topic. Specifically, you’ll find the following reference useful: http://bit.ly/9ceuqZ; it is available online from within the UCSD network. Please note that the example questions are intended to help you review the material and get a feeling for the type of question you might encounter; they can’t (and don’t attempt to) cover all the prerequisites equally, nor all possible question types.

1. Java Basics [70.1.7] [70.1.12]

   a. Be able to write Java classes with encapsulated data and access methods for a given problem.
      ⇒ **Example Question**: Write a Java class that models a message sequence for a chat system. Use appropriate data structures for the message sequences, assuming a given class `Message`. Provide access methods for adding and removing messages.

   b. Be able to create and use Java interfaces.
      ⇒ **EQ**: Create a Java interface for the class you have just written, and use it in a rewrite of that class.

   c. Be able to write Java code for processing basic data structures, such as Vectors, Lists, and Maps.
      ⇒ **EQ**: Write a Java class that models a simple buddy list (for our chat system) using a `HashMap` with appropriate access methods.

   d. Be able to declare, throw and catch Exceptions in Java.
      ⇒ **EQ**: Write a Java class that has a method `divide-by` that accepts two integer values $x$ and $y$ and returns $x$ divided by $y$ iff $y$ differs from 0. When $y$ is 0, `divide-by` throws a `WrongInput` exception (define it!) that indicates the wrong value. Write another method that properly uses `divide-by`. 
2. Development Processes and Development Pipeline [70.1.14] [70.2.5]

a. Be able to explain the activities, benefits and challenges of the waterfall process and their ordering over time.
   ⇒ EQ: Name the key problem with executing the traditional Waterfall process if requirements change frequently.

b. Be able to explain the key values and practices of XP.
   ⇒ EQ: XP places more value on running code than on extensive documentation of requirements. Name two XP practices that help capture requirements as well as ensuring quality of the code.

c. Be able to explain the key differences between agile and plan driven development processes.
   ⇒ EQ: Contrast agile and plan driven development using the following categories: team size it works for, degree of customer-involvement, main product, iteration length, and path to quality.

d. Be able to name and relate the key elements of a software development pipeline.
   ⇒ EQ: Name the set of tools that facilitate the development pipeline for a medium-size project and team.

e. Be able to tell what a requirement is.
   ⇒ EQ: Define the term “requirement”.

f. Know the difference between functional and non-functional requirements.
   ⇒ EQ: Functional requirements define what the system is supposed to do. Name the two main types of constraints non-functional requirements place, and give an example for each.

g. Be able to characterize requirements into functional and non-functional aspects.
   ⇒ EQ: Decide whether the following is a functional or non-functional requirement; explain your rationale: “In case of an impact, the airbag must deploy within 10 milliseconds.”

3. Unified Modeling Language [70.2.9] [70.2.16]

a. Be able to draw and interpret basic UML class, activity and sequence diagrams.
   ⇒ EQ: Draw a UML class diagram capturing the following relationships between the classes Message, Chat, Call, Client, Conversation, ConversationRepository, as well as the interface IClient. A ConversationRepository stores multiple Conversations. A Conversation consists of a sequence of Messages. Chat and Call are specific types of Conversations. Be sure to list the appropriate access methods. A Client implements the IClient interface, which has the methods getName and setName. A Client can have multiple Conversations.
b. Be able to translate basic UML diagrams into Java code and vice versa.
   ⇒ **EQ**: Translate the class diagram from the previous question into correct Java code.

4. Design Patterns [70.2.11] [70.2.18]

   a. Be able to explain the purpose, tradeoffs, structure and basic behavior of the Adapter, Strategy, Template Method, Observer, and MVC patterns using UML diagrams and Java code.
      ⇒ **EQ**: Convert the inheritance-based design from [70.2.11], slide 41, into a composition-based design using the Strategy pattern using a class diagram.
      ⇒ **EQ**: Explain the difference between the Strategy and Template Method patterns. Which one allows you to dynamically "plug-in" design options at runtime?
      ⇒ **EQ**: Explain the major interactions among Model, View and Controller of the MVC pattern using a UML sequence diagram.

   b. Be able to select a pattern (from the list above) for a particular design context.
      ⇒ **EQ**: Assume you have a class S that has a method `getValue` to provide a certain `Integer` value. Furthermore, you have a class C that expects to consume a `Float` version of the values S instances can provide. Which pattern would allow you to compose S and C, and why?