**MPP Midterm Review Points**

The midterm will consist of 30 points. Short Answer questions may require English explanations, discussion, UML diagrams and Java code.

1. Bring a pencil, eraser, and a well-rested, well-nourished, coherent, alert, brain.
2. We follow DE test rules:
   1. No translation devices
   2. No cell phones (calls, messages, time, etc.)
   3. Once you have seen the exam you cannot leave the room until you hand it in.
   4. The test must be turned in by 12:00 noon.
   5. Any cheating results in zero points for the whole test
3. You can ask for clarification of the question but do not ask me to evaluate your answer during the test.

Review Recommendations:

**Lessons 1 and 2**

1. Discovering classes from a problem statement
2. Difference between analysis and design
3. Differences between association, dependency, inheritance
4. Associations, dependencies in code
5. Difference between one-way, two-way associations
6. Properties as attributes, properties as associations
7. Association adornments: name, role, multiplicity,
8. Association matrix
9. Significance of different multiplicities in code
10. Aggregation vs composition vs association
11. Reflexive associations

*Skills:*

* Create a class diagram with attributes, operations, associations, based on a problem statement
* Translate a class diagram into Java code
* Reverse engineer a class diagram from Java code

**Lesson 3**

1. Good uses of inheritance vs bad uses
2. Inheritance rules
   1. Rules for inheriting/overriding static methods
   2. Order of execution
3. IS-A and LS principles (subclass must be able to be substituted for the superclass)
4. Benefits of inheritance
5. Rectangle-square problem (1 side vs. 2 sides for computing area)
6. Principle: Design for inheritance or prevent it – class final or private constructor
7. How to replace inheritance by composition, or supplement inheritance with composition (Duck App – person/personRole), in a design and in code

*Skills*

* Solve a design problem by introducing composition
* Solve a problem using inheritance and polymorphism.
* Transform, in code, an inheritance relationship into a composition relationship

**Lesson 4**

1. Syntax of sequence diagrams – use of activation bars; how to show looping; how to show message passing and self-calls; iteration marker and interaction frame; return arrows
2. Using fragments of a sequence diagram starting from a reference point; introducing UI and Controller classes to model full use cases; when an actor should be shown and when a reference point can be used instead
3. Sequence diagrams as a way to model a use case
4. Centralized control vs distributed control in sequence diagrams
5. Syntax of object diagrams; purpose of object diagrams
6. The meaning of delegation and propagation
7. The meaning of polymorphism and late binding
8. The reason why static, private, and final methods have early binding
9. The template method design pattern. Recall how it was used in the lab on calcCompensation and in the DataParser example in the slides.
10. Open-Closed Principle

*Skills*

* Create a sequence diagram based on a use case description.
* Create an object diagram, given information about a system of objects and their attributes.
* Solve a problem using polymorphism.
* Use the template method pattern to solve a design problem.
* Converting Java code to a sequence diagram

**Lesson 5**

1. Definitions concerning abstract classes
2. Differences between abstract class and interface (in Java 7)
3. UML notation for abstract classes and interfaces
4. The Object Creation Factory pattern (know the diagram and what it means)
5. The simple factory method pattern
6. Know the benefits of these patterns and some examples of when to use them
7. The “Diamond Problem” for languages with multiple inheritance
8. Benefits of using interfaces -- recall our Duck lab
9. Refactoring / extending a design using interfaces (example in the slides)
10. What does Program to the Interface mean? Why is it a good practice? What are some examples?
11. What is the Evolving API Problem?

*Skills*

* Solve a problem using polymorphism.
* Create a factory method in a class
* Use a factory to implement a 1:1 bidirectional relationship or 1:many bidirectional relationship
* Turn a class into an *immutable* class.

**Lesson 6 and Project**

1. Be able to explain the flow of control from JavaFX source.
2. Know how we handle events to link our UI to the business logic in an application
3. Using Interfaces to implement /simulate DataAccess

NOTES

1. For the exam, we will adopt the convention that all sequence diagrams make use of an actor to initiate action; the actor talks to a UI class; and the UI class talks to a Controller class. The Controller class then communicates with the domain classes that have been identified.
2. When we need to create a new class (like a book or an employee), we show the new() method from the class doing the creating to the class being created.
3. We use comments in our sequence diagrams to show iterations and clarify returns, explain assumptions for success, and branching.