**MPP Midterm Review Points**

The midterm will consist of Multiple Choice (20%), Short Answer (50%), and Design (30%) questions, as well as an SCI question (3 points). Short Answer questions may require English explanations and discussion, but may also require UML and Java code.

**Lessons 1 and 2**

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

*Skills:*

* Create a class diagram with attributes, operations, associations, based on a problem statement
* Translate a class diagram into 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
4. Bad Stack example
5. Benefits of inheritance
6. Rectangle-square problem
7. EnhancedHashSet problem – inheritance violates encapsulation
8. Principle: Design for inheritance or prevent it
9. How to replace inheritance by composition (Employment/Manager equals method, Stack class), or supplement inheritance with composition (Duck App), in a design and in code

*Skills*

* Solve a design problem by introducing composition
* 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 (success scenario)
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 exercise 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 several examples of these patterns and what they illustrate
7. The “Diamond Problem” for languages with multiple inheritance
8. Benefits of using interfaces
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.

**The SCI Question**: You can prepare for your SCI question in advance. This is the question you will be asked on the exam:

*Write one or two paragraphs relating a point from the course to a principle from SCI*.

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.