MPP Midterm Review Points

The midterm will consist of approximately 30% short answer questions (including possibly some true/false questions) and 70% skill questions. Skill questions require you to write code, draw diagrams, or supply other forms of documentation. The exam will be a paper exam; you will not have access to laptops, internet, phones, books, or notes. There will be some coding on the exam, so you should be prepared to write code without the help of Eclipse.

Lessons 1 and 2

1. What is the SDLC? What are the main activities?
2. What is the Static Model that is created during analysis? What kind of diagram is used in this model? Describe an approach to building a Static Model based on a problem statement (in other words, how did we do it in this class?).
3. What is the Dynamic Model? Name some diagrams that are used in this model.
4. Use case diagram
5. Use case descriptions (ATM, SRS)
6. Actors for a use case
7. Discovering classes from a problem statement
8. Difference between analysis and design
9. Differences between association, dependency, inheritance
10. 3 types of relationshps and their differences
11. Associations, dependencies in code
12. Difference between one-way, two-way associatons
13. Properties as attributes, properties as associations
14. Association adornments: name, role, multiplicity,
15. Association matrix
16. How association multiplicities are translated into code
17. Reflexive associations
18. Association class – how is an association class represented in a more detailed design?

Skills:

- Create a class diagram with attributes, operations, associations, based on a problem statement
- Translate a class diagram into Java code
- Be able to identify the main flow of a use case and write up a use case description, following the rules for creating them (for instance, branching logic is now shown)

Lesson 3

19. Good uses of inheritance vs bad uses
20. Inheritance rules
    a. Rules for inheriting/overriding static methods
    b. Order of execution
21. IS-A and LS principles and the bad Stack example
22. Benefits of inheritance
23. Rectangle-square problem
24. EnhancedHashSet problem – inheritance violates encapsulation and the Ripple Effect
25. Principle: Design for inheritance or prevent it
26. Ways to prevent inheritance
27. How to replace inheritance by composition (examples: Person/PersonWithJob equals method, Stack class), or supplement inheritance with composition (Duck App, using a PersonRole), in a design and in code

Skills

- Solve a design problem by introducing composition, or composition together with inheritance (as in the Duck App)
- Transform, in code, an inheritance relationship into a composition relationship (recall Circle and Cylinder problem)

Lesson 4

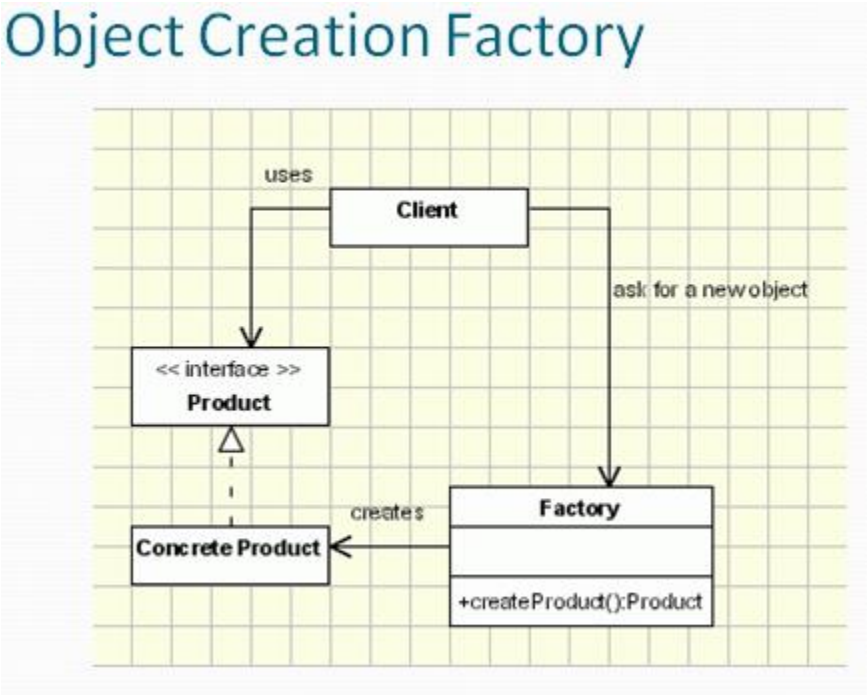
28. Syntax of sequence diagrams – use of activation bars; how to show looping; how to show message passing and self-calls; iteration marker and interaction frame; return arrows
29. Using fragments of a sequence diagram starting from a reference point (an *endpoint*); introducing UI and Controller classes to model full use cases; when an actor should be shown and when an endpoint can be used instead
30. Sequence diagrams as a way to model a use case (main flow)
31. Centralized control vs distributed control in sequence diagrams
32. Syntax of object diagrams; purpose of object diagrams
33. The meaning of delegation
34. The meaning of polymorphism and late binding
35. The reason why static, private, and final methods have early binding
36. Rules regarding inheritance and method overriding as they pertain to static methods
37. The template method design pattern. Recall how it was used in the exercise on calcCompensation and in the DataParser example in the slides.
38. 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 (reverse engineering)

Lesson 5

39. Definitions concerning abstract classes
40. Differences between abstract class and interface (in Java 7)
41. UML notation for abstract classes and interfaces
42. The Object Creation Factory pattern (know the diagram and what it means)



43. The simple factory methods and also *parametrized* factory methods (which accepts one or more arguments that are used to determine which object is created); recall the RulesFactory.
44. Know several advantages to using factory methods in place of public constructors with supporting examples
45. The "Diamond Problem" for languages with multiple inheritance
46. Benefits of using interfaces
47. Refactoring / extending a design using interfaces (example in the slides)
48. What is the Evolving API Problem?

Skills

- Solve a problem using polymorphism.
- Create a factory method in a class in place of a public constructor
- Use a factory to implement a 1:1 bidirectional relationship or 1:many bidirectional relationship

Lesson 6

49. You will not need to write JavaFX code, but you will need to be able to *read* it. There may be a question that asks you to decide what a screen will look like when a piece of JavaFX code is run.

The SCI Question: (3 points) You will be given an insight/principle from SCI and you will be asked to explain what it means and give an example of how it is exemplified or illustrated by a Computer Science concept. This is a short essay; richer content will be awarded more credit.