TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2009

COMP463
DESIGN PATTERNS

Time Allowed: 3 Hours

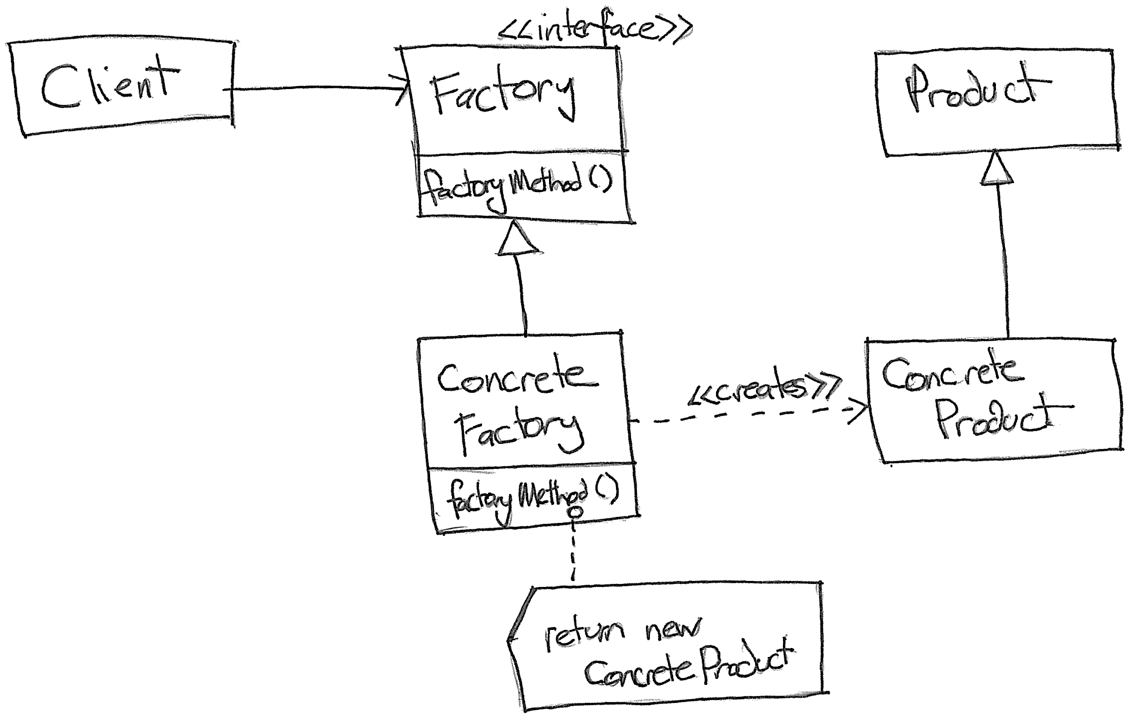
Instructions:

- This examination will be marked out of **180** marks.
- Read each question carefully before attempting it.
- Answer all six questions. Each question has the same value, and should take approximately 30 minutes to answer.
- You may answer the questions in any order. Make sure you clearly identify the question you are answering.
- Many of the questions require you to discuss an issue, or to express and justify an opinion. For such questions, the assessment will take into account the *evidence* you present and any *insight* you demonstrate.
- Some of the questions ask for examples from object-oriented languages or of design patterns. Your answers need only refer to object-oriented languages or patterns discussed in the course, but you may refer to other programming languages and patterns if you wish.
- This exam is open book. Non-electronic reference books and hand-written notes are permitted.

continued...

Question 1. Creational Patterns [30 marks]

The **Factory Object** pattern is an alternative creational pattern that is not discussed in the *Design Patterns* book. In this simple pattern, a Factory object implements a Factory interface that contains a Factory Method. The Factory Method returns an instance of a Product class.



(a) [20 marks] How is the Factory Object pattern related to the other four creational patterns? (Abstract Factory, Builder, Prototype, Singleton). How is Factory Object related to the Strategy pattern?

(b) [10 marks] Do you consider it worthwhile to treat the Factory Object pattern as a pattern in its own right? Explain why or why not.

Question 2. Structural Patterns [30 marks]

The Composite pattern is one of the most useful design patterns, but its use in practice can cause as many design problems as it solves. For this reason, many other patterns are often used alongside the Composite pattern. Three of these patterns are:

(a) [10 marks] Flyweight

(b) [10 marks] Bridge

(c) [10 marks] Visitor

For each of the above three patterns:

1. Describe a **problem** caused by Composite that this pattern can solve.
2. Describe **how** you would combine this pattern with the Composite pattern.
3. Explain **how** this pattern resolves the problem caused by Composite.

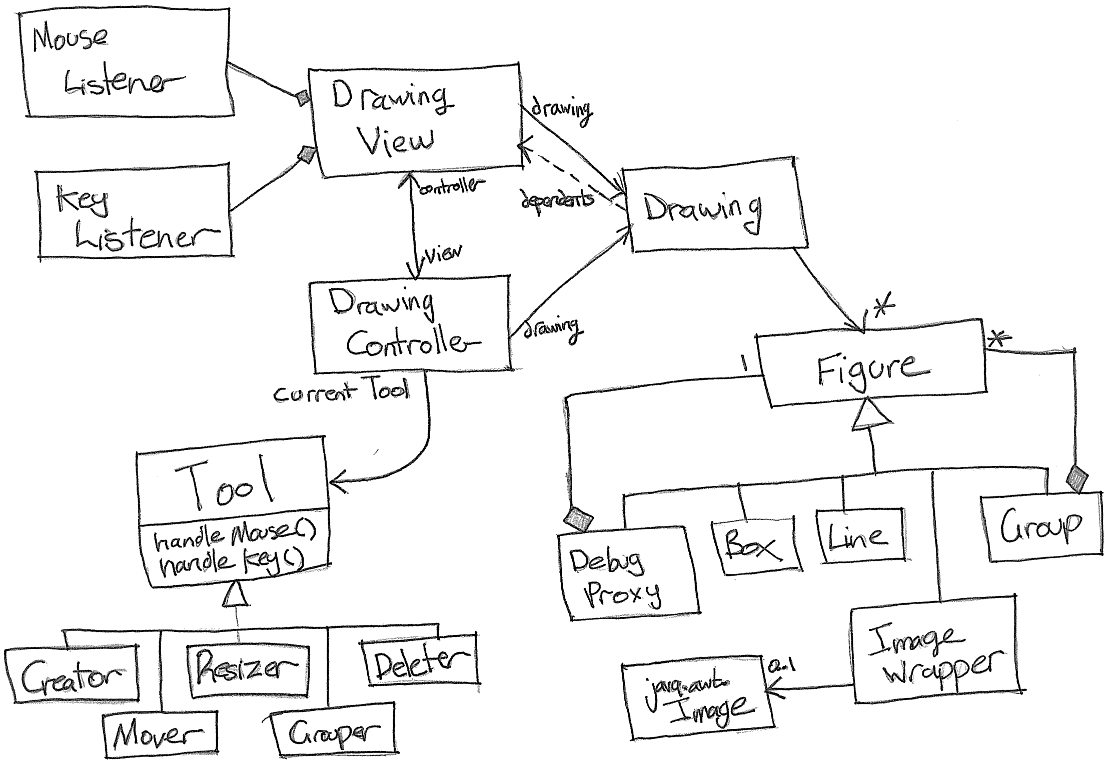
Question 3. Behavioural Patterns [30 marks]

(a) [15 marks] *Design Patterns* describes how some patterns (like Decorator) change the "skin" of an object, while others (like Strategy) change an object's "guts". Identify two other patterns that change an object's skin, and two other patterns that change an object's guts. Explain where you could use the "skin" patterns, and where the "guts" patterns would be more appropriate.

(b) [15 marks] The Interpreter and Observer patterns both allow dynamic behaviour to be associated with objects in programs. Do you think these two patterns could provide alternative solutions to similar problems? Could you combine Observers and Interpreters to solve a more complex problem? Explain your answers.

Question 4. Patterns in System Design [30 marks]

The following class diagram shows the design of an extensible drawing editor framework.



This design includes a number of design patterns.

Identify **three** patterns used in this design.

For each of these three patterns:

(a) [1 mark] Name the pattern.

(b) [2 marks] List each of the **Participants** of the pattern, and name the corresponding concrete class(es) in the design.

(c) [7 marks] Describe the design problem *in drawing editor framework* that the pattern solves, and explain why the pattern solves that problem.

Question 5. Inheritance vs Composition [30 marks]

(a) [10 marks] The Adaptor pattern has two versions, class adapter (using inheritance) and object adapter (using composition). How would you choose between these two variants of the pattern?

(b) [10 marks] How does the class adapter relate to the "second principle of object-oriented design":

> Favor object composition over class inheritance
>
> *Design Patterns*, Gamma, Helm, Johnson, Vlissides, p.20.

(c) [10 marks] The Template Method pattern makes great use of inheritance. Do you think similar problems can be solved via composition? More generally, do you think that *any* pattern that uses inheritance could be replaced with a pattern using composition? Do you think that patterns using composition can be replaced by patterns using inheritance? If so, explain how? If not, explain why?

Question 6. Patterns vs Languages [30 marks]

> When I see patterns in my programs, I consider it a sign of trouble.
>
> *Revenge of the Nerds*, Paul Graham

Paul Graham argues that patterns are evidence of missing features in programming languages, and so that rather than design our programs using patterns, we should adopt more powerful programming languages with language features that remove the need for patterns. For example, consider the Facade pattern and modules or packages; the Factory Method pattern and gBeta's class families; or the Iterator pattern and co-routines such as C♯'s `yeild return`.

Choose three patterns — from these three or any other patterns — and discuss how and why the patterns relate to specific language features (that is, how language features could remove the "need" for the patterns). You should give short illustrative code samples with and without the language feature.

Do you think *all* patterns can be replaced by programming language features? Or, can you give examples of patterns that cannot easily be captured by language features? In either case, explain why. In general, do you think patterns should be incorporated into programming languages, or left to programmers?

****