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

EXAMINATIONS — 2010

END OF YEAR

SWEN425

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.
- You may write code samples and/or draw diagrams to illustrate your answers to any of the questions.
- This exam is open book. Non-electronic reference books and handwritten notes are permitted.

Question 1. Creational Patterns [30 marks]

An Abstract Factory can create products using either the Factory Method pattern or the Prototype pattern.

(a) [5 marks] Sketch a brief code example in your favourite Object-Oriented language of an Abstract Factory using a Factory Method, and explain, briefly, how it works.

(b) [5 marks] Sketch another brief example of an Abstract Factory using Prototypes, and explain, briefly, how it works.

(c) [5 marks] Can the Client of an Abstract Factory tell how its products are created? — that is, can the Client determine whether the Abstract Factory uses Prototype or Factory Method? Why or why not?

(d) [5 marks] Would it make sense for a single Concrete Factory subclass to be implemented using both Prototype and Factory Method? Why or why not?

(e) [5 marks] In the Singleton pattern, access to the singleton instance is provided by a Factory Method typically called something like "`getInstance()`". Can Singleton be implemented using the Prototype pattern instead? If so, explain how, and sketch a brief code example. If not, explain why not.

(f) [5 marks] Would it make sense to use the Factory Method, Prototype, or Abstract Factory patterns to help to implement a Builder? If so, explain how and why. If not, explain why not.

Question 2. Structural Patterns [30 marks]

(a) You overhear three of your development team members arguing over the design of a `TextFile` class and its associated `WindowsFileWrapper` class:

```
class TextFile {
 TextFile(String filename);        // open filename
 void writeLine(String text);      // write text
 String readLine();                // read text
}
class WindowsFileWrapper implements WindowsFile {
 WindowsFileWrapper(TextFile tf);  // wrap a TextFile
 void writeLine(String text);      // write text
 String readLine();                // read text
 void writeString(String text);    // write text
 String readString();              // read text
 void flush();                     // ensure any cached data is
}                                  // written to the underlying TextFile
```

Diane argues that the `WindowsFileWrapper` is a **Decorator**, because it adds a responsibilty to the underlying `TextFile`: the `WindowFileWrapper` caches the aruments of `writeLine` and `WriteString` methods, and only writes them to the underlying `TextFile` when the `flush` method is called. **Alice** argues that the `WindowsFileWrapper` is an **Adaptor**, because it lets a `TextFile` be used wherever a `WindowsFile` is expected. **Peter** argues that the `WindowsFileWrapper` is a **Proxy**, because it provies a surrogate for a `TextFile`.

(i) [8 marks] Who do you think is correct (Diane, Alice, or Peter) or are they all wrong? Explain why.

(ii) [7 marks] Alice, Diane, and Peter then ask you if you think the `WindowsFileWrapper` class is a good design as it is, or if you think the design should be changed. If you think `WindowsFileWrapper` should stay as it is, explain why. If you think it should be changed, describe your changes and explain why the team should accept them.

(b) The Bridge pattern is rarely used in practice.

(i) [5 marks] Explain how the Bridge pattern and the Abstract Factory pattern can often be used to solve similar problems. What are the advantages of a Bridge, and why isn't the Bridge pattern used more often?

(ii) [5 marks] Explain the difference between a design using the Strategy pattern, and a simlar design using a Bridge. Can you distinguish between these two patterns using only the structure of the program implementing them, or do you need to know about the intent of the program's design?

(iii) [5 marks] Could you use a Facade to help build a Bridge? Explain why or why not.

Question 3. Behavioural Patterns [30 marks]

The Observer 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 Observers. Three of these patterns are:

(a) [10 marks] Mediator

(b) [10 marks] Template Method

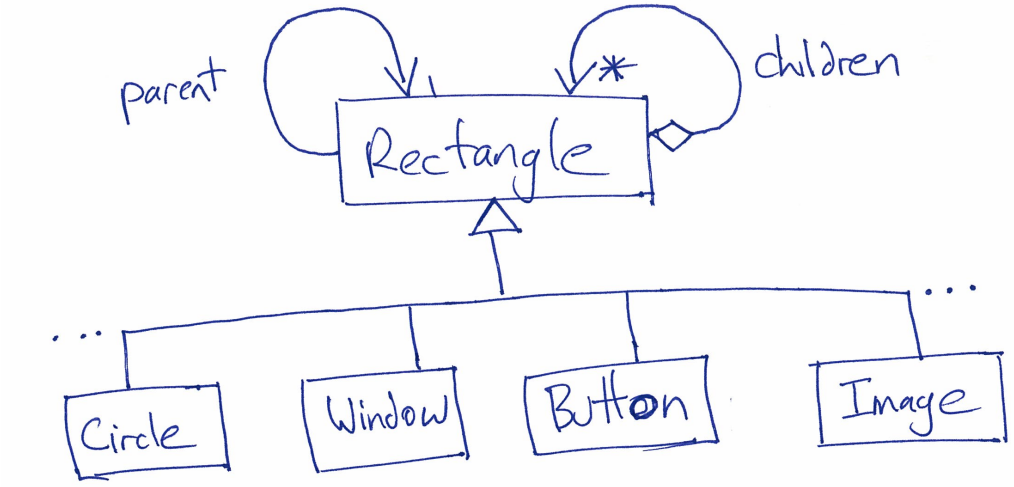
(c) [10 marks] Iterator

For each of the above three patterns:

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

Question 4. Patterns in System Design [30 marks]

This diagram shows the core of "RectDraw", a drawing editor framework. The code below it shows part of RectDraw's `Rectangle` class, implemented in a Java-like dynamic language:




```
class Rectangle {
 Rectangle parent;           // parent Rectangle
 List<Rectangle> children;   // child Rectangles
 Point topLeft;              // top left corner
 Point botRight;             // bottom right corner

 // draw this rectangle and all children on Graphic g
 void drawOn(Graphics g) {
     g.drawRectangle(topLeft,botRight);
     foreach (r : children) {r.drawOn(g);}

 // handle mouseEvent() and keyEvent() mesages
 // methodMissing is called if any method is not implemented by this object
 // here we try to send the method up to its parent Rectangle
 Object methodMissing(String name, List<Object> args) {
     return parent.invokeMethod(name,args); }
}
```

This design includes variants of two design patterns — 1) Composite, and 2) Chain of Responsibility. For these two patterns:

(a) [3 marks] List each of the **Participants** of the pattern, and name the corresponding concrete class(es) in the design.

(b) [6 marks] Describe the design problem *in drawing editor framework* that the pattern solves, and explain why the pattern solves that problem.

(c) [6 marks] Describe how the variant of the pattern embodied in this design differs from the primary variant presented in *Design Patterns*. Explain the advantages and disadvantages of this variant of the pattern. Do you think the drawing editor framework is better served by this variant of the pattern, or the *Design Patterns* variant?

Question 5. Interactive Interpretation [30 marks]

(a) The Command and Memento patterns can both be used to implement undo and redo.

(i) [5 marks] Describe how the Command pattern can support undo and redo.

(ii) [5 marks] Describe how the Memento pattern can support undo and redo.

(iii) [5 marks] Describe how you could choose between Command and Memento to implement undo and redo in an interactive music editor.

(b) [15 marks] Describe how you could combine the Composite, Interpreter, and Visitor patterns to design a simple domain specific language to implement the business rules logic of a share trading application. Your design needs to be able to execute these rules; format them for printing (to be audited by regulators); and to translate them into Java code source code for quicker execution. Here's an example business rule for the system:

```
RULE "Telecom"
 IF Price(TEL.NZ) < 100
   THEN Buy(TEL.NZ, 5000)
 IF Price (TEL.NZ) > 1000
   THEN Sell(TEL.NZ, ALL)
```

Question 6. Patterns vs Designs [30 marks]

> Once we have built the gate,
> we can pass through it
> to the practice of the timeless way.
>
> *The Timeless Way of Building*, Christopher Alexander

> "Do nothing because it is righteous or praiseworthy or noble to do so; do nothing because it seems good to do so; do only that which you must do and which you cannot do in any other way."
>
> *The Farthest Shore*, Ursula K. Le Guin.

> Only ever add a pattern when you cannot avoid it.
>
> Attributed to Thomas J. "Tad" Peckish by Brian Foote.

> Design patterns are targets for refactorings.
>
> *Design Patterns*, Gamma, Helm, Johnson, Vlissides (p.353)

Christopher Alexander argues that patterns (the "gate") are not things to be aimed for in designs, but that once we know the patterns we can use them to design (the "timeless way"). Thomas J. "Tad" Peckish (perhaps inspired by Ursula K. LeGuin) argues that we should only use patterns as a last resort. Gamma et al claim that patterns can be targets for refactorings — that is, not for initial designs, but for program *redesigns*.

(a) [15 marks] Choose three patterns and discuss how those patterns can (or cannot) be used in initial program designs, and can (or cannot) be used in refactoring. You should give short illustrative code samples to illustrate your arguments.

(b) [15 marks] Discuss whether patterns are always good things for programs. Can introducing patterns lead to unnecessary complexity or otherwise "bad" designs? Do you think *all* patterns should only be introduced by refactoring? In general, do you think patterns should be incorporated into designs from the beginning, or left to be introduced later in the development lifecycle?

****