University of Canterbury
Department of Computer Science and Software Engineering

End of Year Examinations 2006

Prescription Number(s): Cosc314

Paper Title: Software Engineering

Time Allowed: Three hours

Number of Pages: 11

- Answer *all* questions.
- Check carefully the number of marks allocated to each question. This suggests the degree of detail required in each answer, and the amount of time you should spend on the question.
- This is an *open book* assessment item. You are allowed any non-electronic reference material.
- You *are* permitted to use calculators.
- Use the separate *Answer Booklet* for answering *all* questions.
- No form of collaboration is permitted.
- This item of assessment is worth 100 marks.

1. **[15 marks for whole question]** Each of the following question parts (a–e) names a design principle and a GoF pattern. For each part, write a sentence or two to demonstrate that you know what the given principle means (don't just re-word it, explain it), and describe how the given pattern supports the principle.

   (a) **[3 marks]** *Separation of Concerns*; Observer.

   (b) **[3 marks]** *Program to the interface, not the implementation*; Factory Method.

   (c) **[3 marks]** *Favour composition over inheritance*; Strategy.

   (d) **[3 marks]** *Information Hiding*; Iterator.

   (e) **[3 marks]** *Open/closed principle*; Template Method.

2. **[3 marks for whole question]** The architect Christopher Alexander dislikes "master plans" that fully prescribe entire development projects. How has Alexander's thinking about master plans influenced software engineering? Be as specific as possible.

3. **[32 marks for whole question]** Parts a–d refer to the design shown in Figures 1 and 2, and the following notes.

   Notes:

   - This system models foreign exchange bank accounts. Each Account holds an amount of money in some currency, e.g. pesos.
   - The possible currency types are defined as `static int`s in the Currency class. (This set is not expected to change frequently, so can be hard-coded.)
   - An Account cannot change its currency, but money can be transferred from one Account to another.
   - Each Transfer object records the date and amount of funds transferred in one transaction from one account to another.
   - The Currency class contains a `static` array of current exchange rates, indexed by currency. (Exchange rates typically change daily.) An exchange rate is a multiplier that converts a unit of currency into its equivalent in ounces of gold. To convert from one currency to another, the original amount is multiplied by the source currency's exchange rate and divided by the target currency's exchange rate.
   - A transfer is made by calling `srcAccount.transfer(amount,dstAccount)`. A lot of the functionality of the system is in this method, so Figure 2 shows how it works.
   - Every Account is owned by a Customer. A Customer can have many Accounts. Customer extends ArrayList<Account> to store the Accounts.

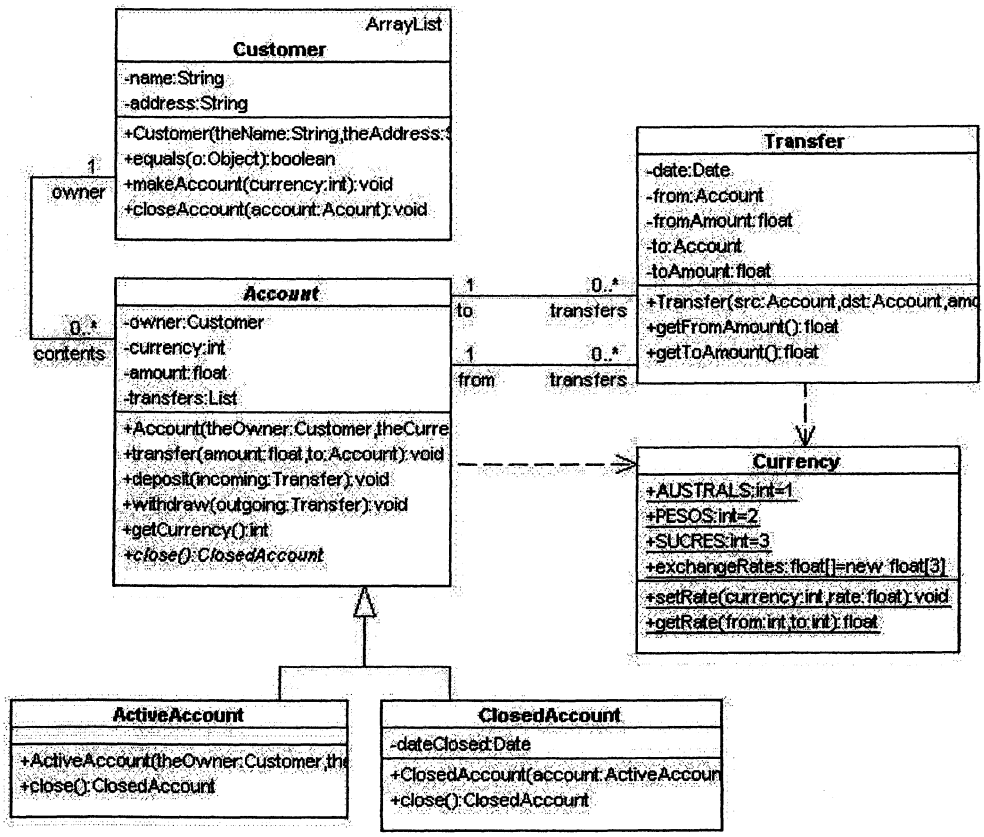

Figure 1: Foreign exchange accounts class diagram

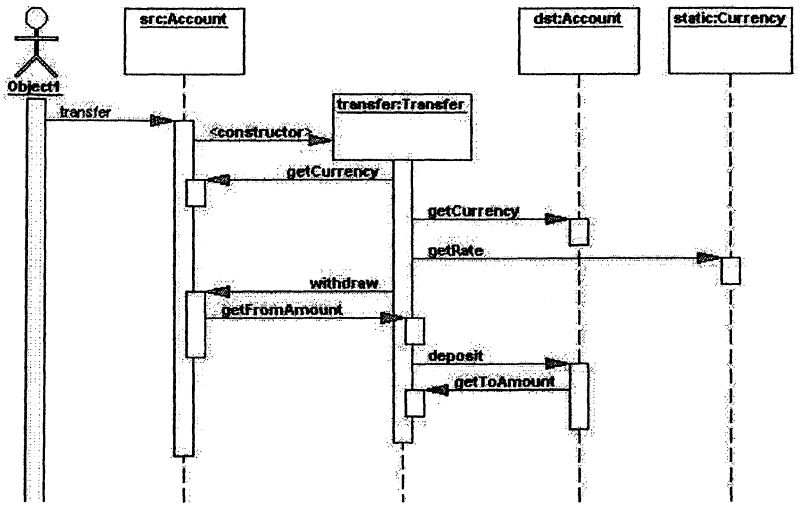

Figure 2: Transfer method sequence diagram

- When an Account is opened by a Customer, an ActiveAccount is constructed. If the ActiveAccount's `close()` method is called, it makes a ClosedAccount containing the same data. The `close()` method of a ClosedAccount just returns itself.

(a) **[16 marks]** Identify four significant flaws or weaknesses in the given design. For each one, name the problem, explain why it is a problem, and show how you would change the design to fix it. Use UML or Java to show clearly how you would fix each flaw.

(b) **[6 marks]** Account currently stores Transfers (incoming and outgoing) in a List, in the order they are added. The designer assumed they would be added chronologically, but a recent change in requirements means they might now be added out of sequence. Show how you would modify the program so that each Account stores Transfers in chronological order (i.e. ordered by Transfer date), regardless of what order they are added. Show all Java code necessary to accomplish the new behaviour.

(c) **[6 marks]** Using good OO design and appropriate GoF pattern(s), draw a UML class diagram to show how you would modify the design to support different kinds of Customers, such as an individual, company, or consortium, so that the different kinds can have different behaviour and data. A company, for example, would store date of incorporation and web URL, while an individual would store a birth date, etc. A consortium is a group of any kind of Customers, including other consortia, but it can be used like a single Customer. Make sure you include methods to add and remove Customers from consortia.

(d) **[4 marks]** Your answer for the previous question should include a method (or methods) to add a Customer to a consortium. Document the contract for this method (or methods). Explain how the contract conforms to the rules of *Design by Contract.*

Coupling type	Weight
Independent	0 (weakest)
Data	1
Stamp	3
Common	4
Control	5
Hybrid	7
Content	9 (strongest)

Cohesion type	Weight
Coincidental	0 (lowest)
Logical	1
Temporal	3
Procedural	5
Communicational	7
Sequential	9
Functional	10 (highest)

Table 1: Ranking coupling and cohesion (see question 4)

4. **[5 marks for whole question]** Numeric values were used in class to describe and rank levels of coupling and cohesion. Table 1 shows the types which were studied.

   What measurement scale would apply to the information summarised in Table 1? Briefly justify your answer and indicate any consequences for the practical uses of these rankings for coupling and cohesion on real software project data.

5. **[10 marks for whole question]**

   Figure 3 shows an XML file illustrating one way to represent cocktail recipes in XML. You may recognise it as part of a file used in the lab test but this is not important for this question. The corresponding DTD is given in Figure 4. Note that some data appears in the form of attributes.

   (a) **[8 marks]** When designing a DTD, a software engineer will consider whether data items should appear as attributes or as elements. Discuss the changes that would be required in order to obtain an alternative XML representation of cocktails, together with a corresponding DTD, without using attributes at all. You need not provide a complete data file and DTD corresponding to the counterparts of those appearing in Figures 3 and 4, but should provide sufficient detail that your approach is clear.

   (b) **[2 marks]** What are the advantages and disadvantages of using attributes? When is it most appropriate to use attributes?

```
<?xml  version="1.0"?>
<!DOCTYPE drinks SYSTEM "drinks-att.dtd">
<drinks>
  <drink glass="cocktail">
    <name>East India</name>
    <ingredients>
      <ingredient>brandy</ingredient>
      <ingredient quantity="0.75">pineapple juice</ingredient>
      <ingredient quantity="2" unit="tsp">orange curacao</ingredient>
      <ingredient unit="dash">angostura bitters</ingredient>
    </ingredients>
    <method>Shake the ingredients together, with plenty of ice,
            in the shaker and strain into the glass. </method>
    <decoration>Drop a maraschino cherry into the glass.</decoration>
  </drink>
</drinks>
```

Figure 3: Representing cocktails in XML (with attributes)

```
<!ELEMENT drinks (drink)*>
<!ELEMENT drink (name, description?, ingredients, method, decoration?)>
<!ATTLIST drink glass (rocks | cocktail | highball) "cocktail">
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT ingredients (ingredient+)>
<!ELEMENT ingredient (#PCDATA)>
<!ATTLIST ingredient quantity CDATA "1"
                     unit (oz | ml | tsp | dash) "oz">
<!ELEMENT method (#PCDATA)>
<!ELEMENT decoration (#PCDATA)>
```

Figure 4: The DTD corresponding to Figure 3

6. **[6 marks for whole question]** One of the goals of industrial metrics programmes is the use of metrics to predict the defect density (number of defects per KLOC) expected after the releases of software products.

   Typical results show a significant positive correlation between defect density and fan-out, but no correlation with fan-in. Does this result seem reasonable? Clearly and concisely explain your answer.

7. **[20 marks for whole question]** Figure 5 shows a listing of the `endElement()` method. It comes from the Waitress class used in the lab test—though this is not directly relevant to the question.

   (a) **[6 marks]** Sketch the flow graph corresponding to the structure of `endElement()`. Be particularly careful to indicate the correspondences between nodes in the graph and lines of code in the listing. Note that line numbers have been included in Figure 5 to help you.

   (b) **[5 marks]** What is the value of $\nu(endElementListing)$, the cyclomatic complexity for this method? Be sure to show all relevant working. How can you check that your answer is correct?

   (c) **[5 marks]** What is the value of $NPATH(endElementListing)$? Be sure to show all relevant working.

   (d) **[4 marks]**

      i. Sketch what you would expect a graph of cyclomatic complexity (vertical axis) against LOC (horizontal axis) for all the methods of the Java application in which `endElement()` appears. Indicate where on this graph you would expect the point corresponding to `endElement()` to lie. Briefly justify your answer.

      ii. What would be an appropriate way to illustrate the relationship between NPATH and LOC? Sketch the graph you would expect to see for all the methods of the application. Indicate where on this graph you would expect the point corresponding to `endElement()` to lie. Briefly justify your answer.

```
public void endElement(String namespaceURI, String localName,
                       String rawName)
    throws SAXException {

    if(rawName.equals("ingredient")) {
        // content contains the ingredient name
        if(ingredientMissing) {
            // Don't look further because we are already missing something
        } else if(!barman.haveIngredient(content.toString())) {
            System.out.println("Sorry honey, I can't make your " + drinkName
                               + " because we're fresh out of " +  content);
            ingredientMissing = true;
        }
    } else if(rawName.equals("name")) {
        // content contains the drink name
        System.out.println("One " + content + " coming up!");
        drinkName = content.toString();
    } else if(rawName.equals("drink")) {
        if(!ingredientMissing) {
            System.out.println("Here's your " + drinkName + ".  Enjoy it!");
        } else {
            System.out.println("Would you like something else?");
        }

        drinkName = "";
        ingredientMissing = false;
    }
    content.delete(0, content.length());
}
```

Figure 5: The endElement method

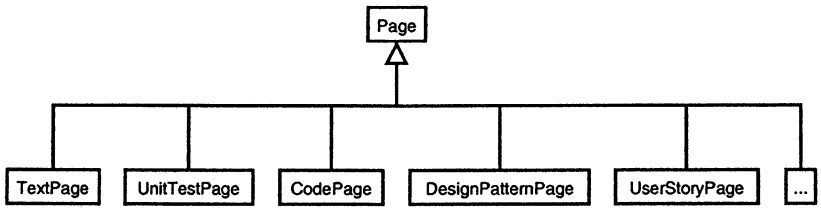

Figure 6: NGW page types

8. **[9 marks for whole question]** Read the following fragments of a dialogue between two Java developers and answer the questions which follow. Arthur and Martha are members of a team which is developing NGW, a Next Generation Wiki product. The data for page content and structure is stored as XML files. One of NGW's features is pages which have special types, each of which has particular formatting and behavioural aspects. In the NGW application, these are implemented by classes as shown in Figure 6. Martha has been working on the code for presentation of the formatted pages to the user.

   Arthur How are you getting on with that coding problem you were having trouble with last week?

   Martha Fine thanks. I have got it all working now.

   Arthur How did you decide to do it?

   Martha I wrote a really cool method. Since all the pages are read from their XML form on disc into a DOM, all the user has to do is provide the page as a Node of the DOM and say how it is to be formatted.

   Arthur Hmmm... How is it used?

   Martha Well, you would say something like `formatPage(NGWConstants.TEST_PAGE, thisPage)` — the method just looks at the kind of format you want and applies all the relevant transformations to add the custom presentation information.

   Arthur I think I see. It's header must look something like `formatPage(int how, Node what)` ...

   Martha ... That's right—and then I just have a big switch to do the appropriate things depending on what is required for each type.

   Arthur I see. And you have a class called NGWConstants that has distinct `static final int` values for each possible kind of page.

   Martha Exactly—neat, eh?

(a) **[3 marks]** How would you describe the level of cohesion that *formatPage()* is likely to have? Is this desirable?

(b) **[3 marks]** How would you describe the coupling between *formatPage()* and the methods which call it in order to satisfy their page formatting requirements?

(c) **[3 marks]** Which class(es) is(are) most likely to contain a *formatPage()* method? Suggest an alternative design and briefly explain why is is better.

End of Paper