THE UNIVERSITY OF MELBOURNE

School of Computing and Information Systems

SWEN30006 Software Modelling and Design

Examination: 2018 End of Semester 1

Reading Time: 15 minutes **This paper has 32 pages.**

Writing time: 120 minutes **Total marks for this paper: 120**

Student Number: _____

Authorised Materials

- *Calculators:* Not permitted.
- *Dictionaries:* Paper-based language translation dictionaries are permitted provided they are not annotated in any way.
- *Notes:* Two double-sided A4 sheets of notes are permitted.

Instructions to Invigilators

- Students should initially receive just this exam paper with 32 pages**.**
- Provide script books on request.
- Students may **NOT** keep the exam paper after the examination.

Instructions to Students

- Write your student number on this exam paper during writing time.
- Attempt all questions.
- The marks for each question are indicated at the beginning of each question.
- The marks are an indication of how much time should be spent on the question.
- Answer questions only in the boxes on the exam paper.
- The blank pages can be used for rough working and notes and will not be marked.
- State clearly and justify any assumptions you have made.
- Write legibly in blue or black pen.
- Draw diagrams legibly: pencil is permitted for drawing diagrams.
- You may request and use a script book if you run out of space in a box for your answer: ensure your answers in the script book are clearly labelled with the question number.
- Mobile phones, tablets, laptops, and other electronic devices, wallets and purses must be placed beneath your desk.
- All electronic devices (including mobile phones and phone alarms) must be switched off and remain under your desk until you leave the examination venue. No items may be taken to the toilet.

This page intentionally left blank.

Question 1. [15 marks]

For each of the following terms, briefly define it and explain its importance in relation to software design:

a. Representational gap
b. Coupling
c. Cohesion

Question 2. [10 marks]

In the context of domain modelling, for each of the following elements: define it, describe when it should be used, and provide an example.

a. Description class
b. Composition

This page intentionally left blank.

Question 3. [10 marks]

a. Describe the Information Expert pattern and provide an example to illustrate its application.
b. Describe the Creator pattern and explain why particular Creator options are given priority over others.

Question 4. [7 marks]

In the context of mapping a design to code:

a. [2] The guideline for translating design classes to code is to work from least coupled to most coupled. Briefly justify this guideline.
b. [5] Explain how design associations with multiplicities greater than one can be mapped to code, and what should guide the choice of representation. Provide an example.

Question 5. [10 marks]

Question 5 Part 1. [4 marks]

Briefly describe architectural analysis and why it is important.

Question 5 Part 2. [6 marks]

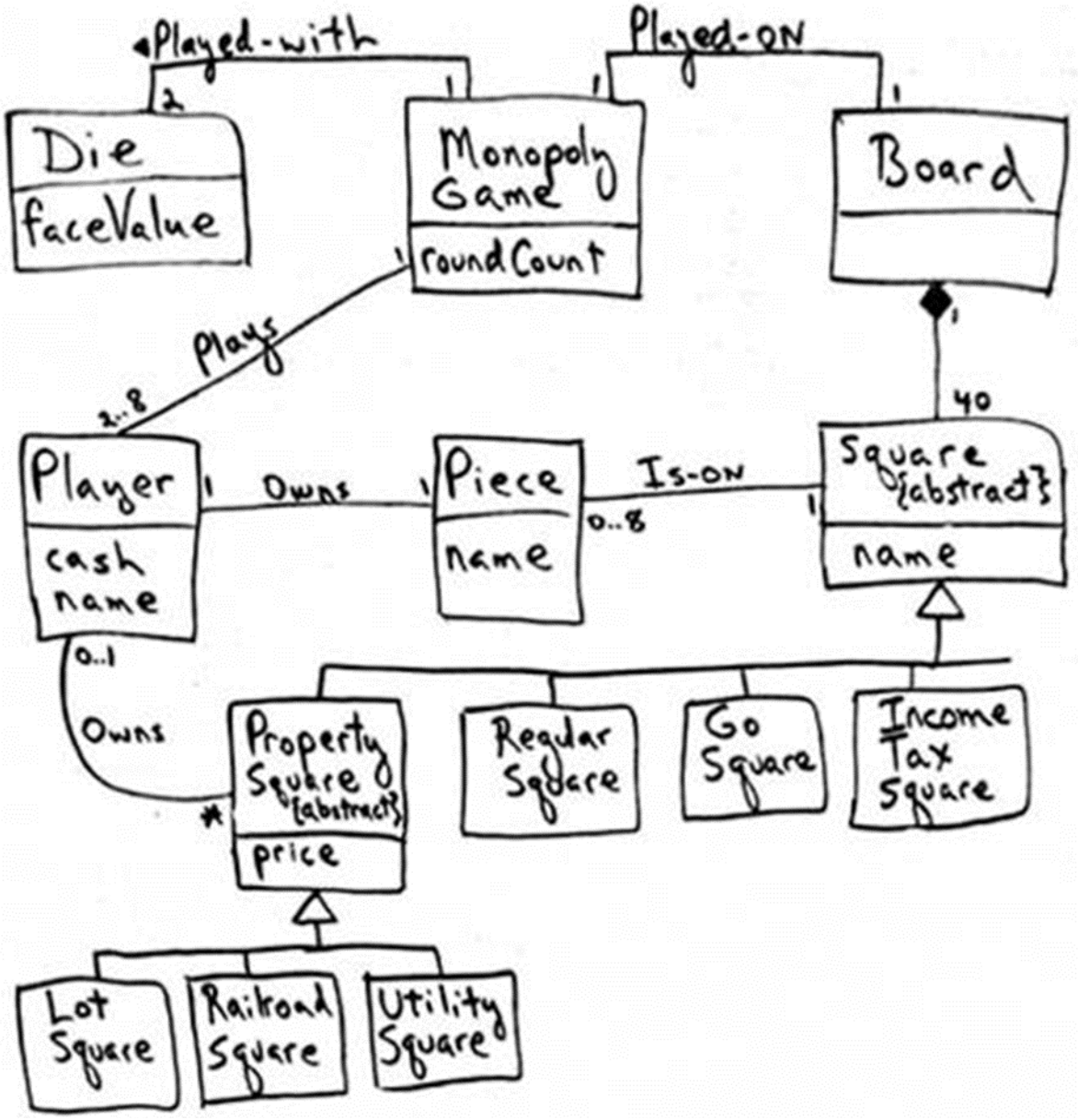
For each of the following, provide an example and briefly explain why it is architecturally significant:

a. An architecturally significant functional requirement.
b. An architecturally significant non-functional requirement.

This page intentionally left blank.

Question 6. [18 marks]

This question relates to the domain model below. This is the iteration-3 domain model for the Monopoly case study from Applying UML and Patterns, 3rd Ed. by Larman.



Below is a list of changes to the Monopoly game. For each change, describe precisely or draw the corresponding change (if any) that should be made to the Monopoly domain diagram. You should not redraw the whole diagram.

a. [2 marks] A new move rule is introduced. If a Player P is about to move their Piece to a Square, and there is already another Player's Piece on that Square, Player P will instead move their Piece to the next Square without a Piece already on it.
b. [2 marks] One of the Railroad Property Squares is replaced by another Income Tax Square.
c. [6 marks] A new kind of square called a Park is created; there are four different Parks. Every Player is required to choose their favourite Park at the start of the game (they can choose the same Park if they wish). However, Players can't own Parks.
d. [8 marks] *For this change you should assume that: (i) every turn taken by a player is numbered in sequence starting with 1; (ii) Properties can be bought and sold by Players; and (iii) once a Property has been bought, it will always be owned by some Player.* New rules are introduced to the game that depend on knowing the Properties owned by every Player in every turn. An example of one of these rules is: when a Player lands on a Property, if they previously owned that Property they get a discount on rental based on the total number of turns for which they owned that Property. Note that we still need to know who owns which Properties currently (as per the Owns association).

This page intentionally left blank.

Question 7. [27 marks]

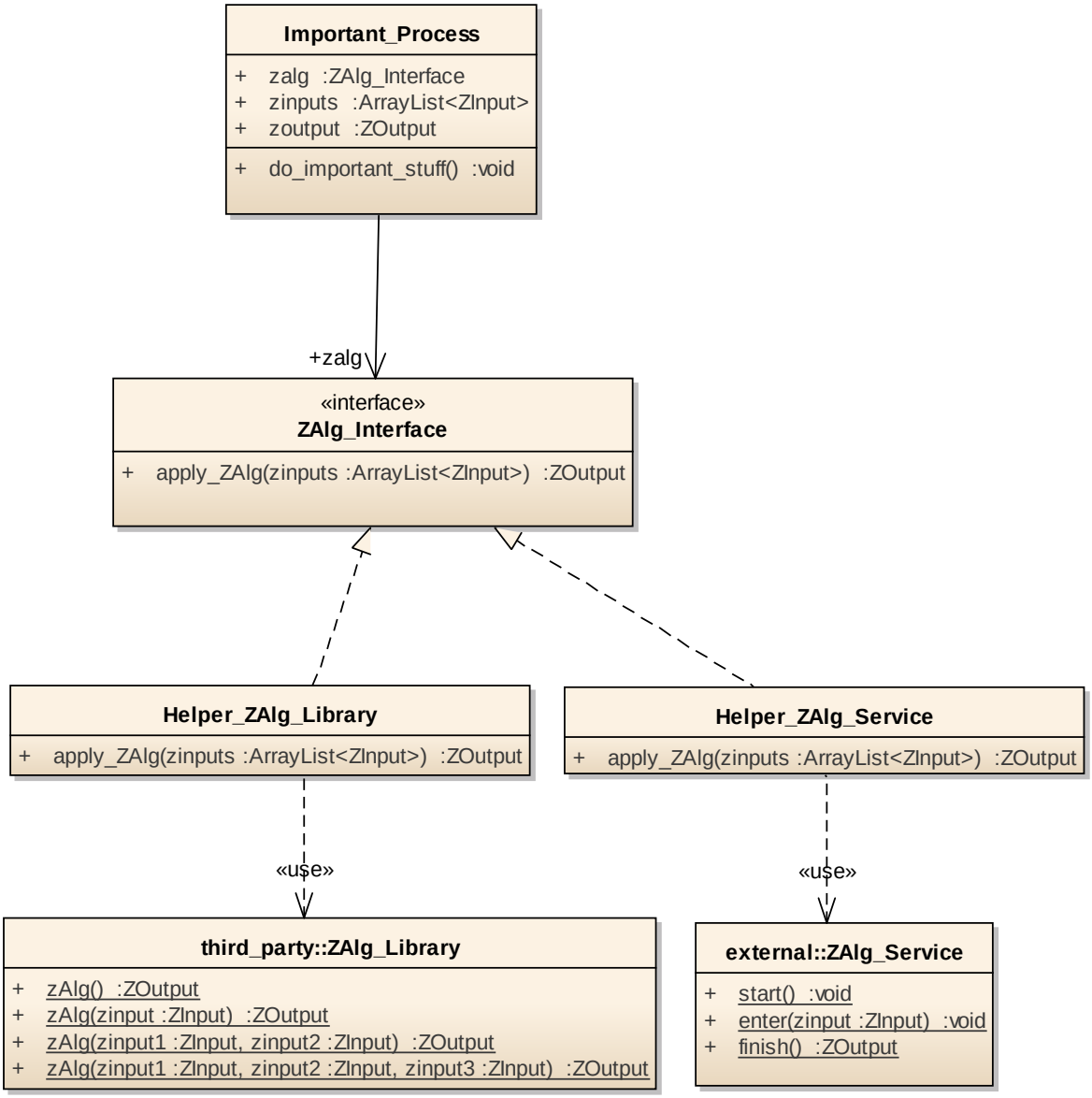
The passage below describes the ZProcess software system. It is important that this system maintains a high degree of availability; the box below describes how this is to be achieved.

> The *Important_Process* object needs to *do_important_stuff()* involving a special algorithm *ZAlg*, which operates on a sequence of *ZInput* to generate a *ZOutput*. We have access to two different versions of *ZAlg*, one implemented in a library *ZAlg_Library* and one available through an external service *ZAlg_Service*. The external service can deal with larger input sequences than the library, so the plan is to use the service when available, but resort to the library otherwise.

Question 7 Part 1. [5 marks]

This ZProcess system requires protected variation. What is protected variation, what are the two different kinds of variations we need to consider, and which of these kinds is required by ZProcess?

A partial design class diagram (DCD) showing public elements and some code snippets from the implementation for ZProcess are provided below.



```
public void do_important_stuff(){
    generateZInputs();
    zoutput = zalg.apply_ZAlg(zinputs);
    if (testZOutput(zoutput)) {
        useZOutput();
    } else {
        zoutput.displayFail();
    }
}
public class Helper_ZAlg_Service implements ZAlg_Interface {

    public ZOutput apply_ZAlg(ArrayList<ZInput> zinputs) {
        ZAlg_Service.start();
        zinputs.forEach(zinput -> ZAlg_Service.enter(zinput));
        return ZAlg_Service.finish();
    }

}
public class Helper_ZAlg_Library implements ZAlg_Interface {

    public ZOutput apply_ZAlg(ArrayList<ZInput> zinputs) {
        switch (zinputs.size()) {
            case 0: return ZAlg_Library.zAlg();
            case 1: return ZAlg_Library.zAlg(zinputs.get(0));
            case 2: return ZAlg_Library.zAlg(zinputs.get(0), zinputs.get(1));
            case 3: return ZAlg_Library.zAlg(zinputs.get(0), zinputs.get(1), zinputs.get(2));
            default: System.out.println("Too many inputs!"); return null;
        }
    }

}
```

Question 7 Part 2. [7 marks]

What GoF design pattern is most prominent in the ZProcess design? Describe this pattern, with clear reference to the ZProcess system as a concrete example.

Question 7 Part 3. [15 marks]

Consider a case where *do_important_stuff()* generates exactly two *ZInputs* and applies *ZAlg* to them. Illustrate this scenario with a design sequence diagram, using the methods in the design/implementation elements provided above. Be sure to cover the behaviour for both the service and library cases. You can assume that zalg is already initialised and all relevant objects already exist.

This page intentionally left blank.

Question 8. [23 marks]

The following questions refer to the passage below which describes the Pennand ATM (Automated Teller Machine). The sentence numbering is provided to make it easier for you to track the various elements of the description.

1. The ATM allows an account holder to check the balance or make a withdrawal on one of their accounts.
2. When the ATM is idle, they can insert their transaction card and enter their PIN; the ATM will authenticate (via the banking network) the id from the card and their PIN.
3. If authentication is successful, the network will return the list of accounts associated with the card. (Otherwise the ATM returns to the idle state.)
4. The customer can then select an account.
5. The ATM will retrieve (via the banking network) and then display the account balance.
6. The customer can then choose to make a withdrawal from the selected account or choose another account.
7. If the customer chooses to make a withdrawal, they then enter the amount; if their account has sufficient balance, the ATM dispenses the amount in cash.
8. The customer can choose to end their transaction at any point other than when the ATM is dispensing cash; on ending the transaction, the ATM will return to the idle state.
9. A successful withdrawal is recorded via the banking network.
10. The ATM will return the account holders transaction card before returning to the idle state (unless the card is identified as stolen – see below).
11. The Pennand ATM also has features designed to support ease of use, maintenance, and security.
12. When a customer first inserts their card, the ATM will photograph the customer and check the card details against a list of stolen cards (stored locally); a stolen card will be kept in the ATM and a message sent to the Police Card Agency, before the ATM returns to the idle state.
13. If the ATM at any stage detects a fault, it switches itself into out-of-service mode; a maintenance technician can then use this mode to conduct diagnosis/repairs, before shutting down and restarting the ATM.

Question 8 Part 1. [5 marks]

Draw a use case diagram, covering the Pennand ATM description above.

Question 8 Part 2. [18 marks]

Draw a state machine diagram for the Pennand ATM, based on the description above.

End of the Exam Questions

End of the Exam Paper