

EXAMINATIONS – 2016

TRIMESTER 2

SWEN222

Software Design

Time Allowed: TWO HOURS

CLOSED BOOK

Permitted materials: No calculators permitted.
Non-electronic Foreign language to English dictionaries are allowed.

Instructions: Answer all questions

Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.

Question	Topic	Marks
1.	Object-Oriented Design	30
2.	Functional Design & Contracts	30
3.	Design Patterns I	30
4.	Design Patterns II	30
Total		120

Question 1. Object-Oriented Design

[30 marks]

Consider the following (incomplete) description for a simple board game called *Dungeon*.

“The game board is made up of twenty locations. Each location is adjacent to one or more other locations. Each location is itself made up from one or more tiles which are arranged into a grid where each tile has a specific position. Each location is either a room, a corridor or an outside area. Rooms and corridors are either lit or unlit. An item can be stored in a location on one or more tiles. Every item has a name and description. Some items represent furniture and cannot be moved. All other items can be picked up by one of the players.”

(a) [9 marks] Provide a *class diagram* covering those aspects of the game outlined in the description above. Your diagram should contain at most nine classes.



(b) [9 marks] Provide suitable *Class-Responsibility-Collaborator (CRC)* cards describing the following classes from the game.

Board:

Player:

Location:

(c) [12 marks] Provide a straightforward *Java implementation* for each of the three classes in part (b). **You need only consider those aspects of the game described on Page 2.** For example, you do not need to implement player movement, the rules for game over, player input, or other aspects not described. Your classes should include *constructors* and *methods* as necessary.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 2. Functional Design & Contracts

[30 marks]

Consider the following class for representing a simple FIFO buffer:

```
1  class QueueBuffer {
2      private int[] items;
3      private int writePos;
4      private int readPos;
5
6      public QueueBuffer(int len) {
7          items = new int[len];
8      }
9
10     public boolean isFull() {
11         return writePos == items.length;
12     }
13
14     public boolean isEmpty() {
15         return readPos == writePos;
16     }
17
18     public void write(int item) {
19         items[writePos] = item;
20         writePos = writePos + 1;
21     }
22
23     public int read() {
24         int item = items[readPos];
25         readPos = readPos + 1;
26         return item;
27     }
28 }
```

(a) [4 marks] An important aspect of the functional programming paradigm is *immutability*. Briefly, discuss what immutability means and whether or not `QueueBuffer` is *immutable*.

(b) This question is concerned with method *side-effects*.

(i) [4 marks] Briefly, discuss what is meant by the term *side-effect free*.

For each of the following methods, briefly discuss whether or not it is *side-effect free*.

(ii) [2 marks] `QueueBuffer.isFull()`

(iii) [2 marks] `QueueBuffer.write(int)`

(iv) [2 marks] `QueueBuffer.read()`

(c) [8 marks] Rewrite the `QueueBuffer` class to use a functional design.

(d) The question concerns the original implementation of `QueueBuffer` on page 6. For each method below, provide appropriate *preconditions* and *postconditions*:

(i) [2 marks] `QueueBuffer(int len)`

REQUIRES:

ENSURES:

(ii) [2 marks] `write(int item)`

REQUIRES:

ENSURES:

(e) [4 marks] Give an appropriate *class invariant* that you would enforce for the `QueueBuffer` class, and discuss how you would enforce the invariant.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.

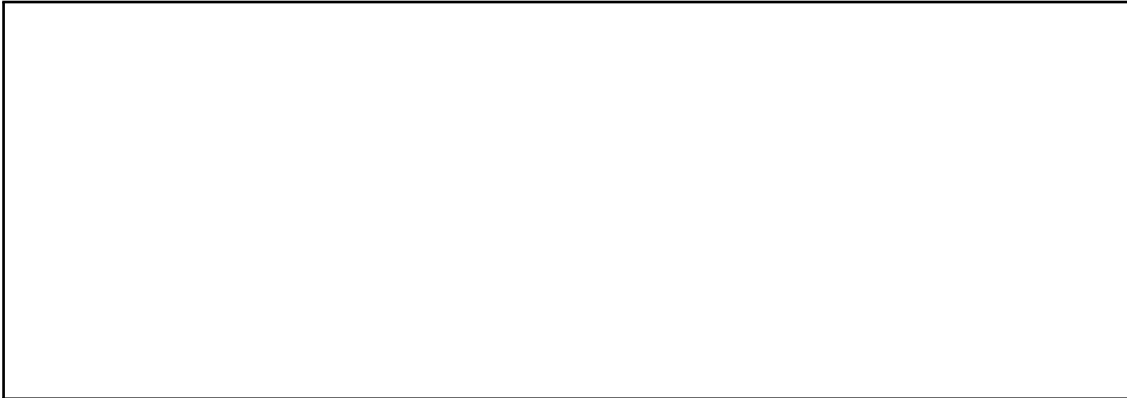
Specify the question number for work that you do want marked.

Question 3. Design Patterns I

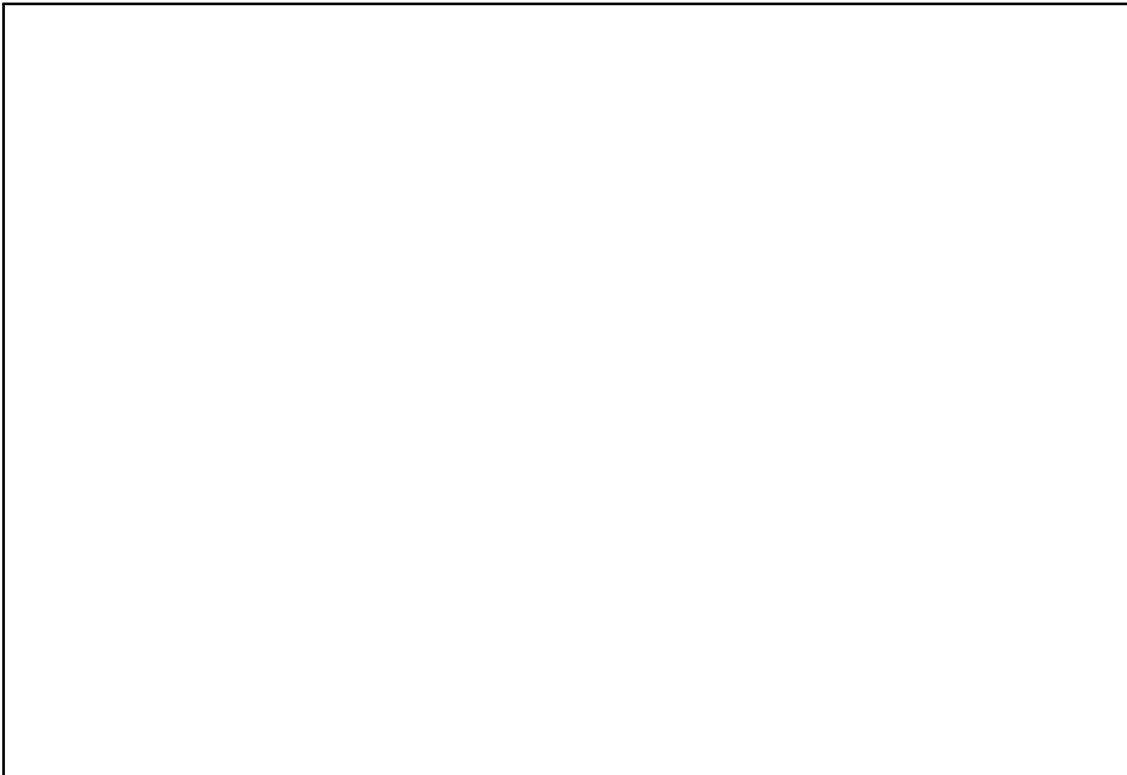
[30 marks]

(a) This question is concerned with the OBSERVER pattern.

(i) [4 marks] Briefly, describe the *problem* being solved by the OBSERVER pattern.



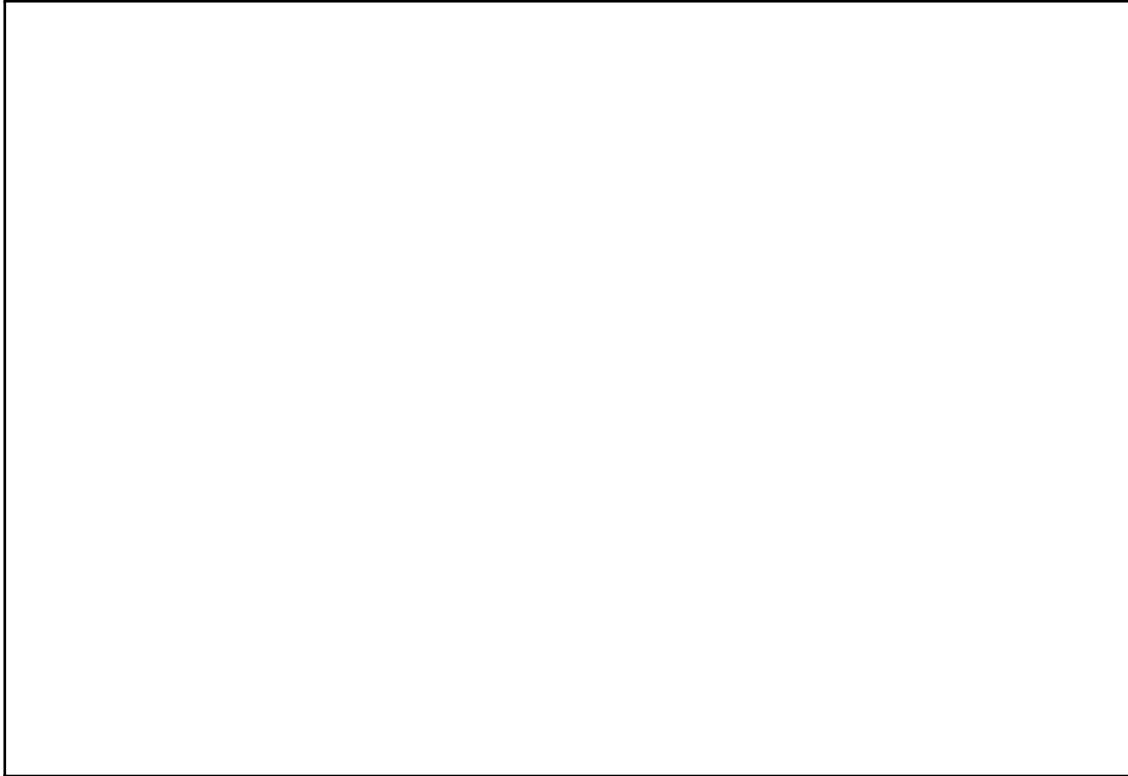
(ii) [5 marks] Provide an appropriate *class diagram* which describes the OBSERVER pattern in an abstract sense.



(Question 3 continued)

(iii) [6 marks] A *listener* can be added to certain *items* that produce *events*. For example, *alarms* may produce *activation* and *deactivation* events. Different kinds of listener can respond to these events in different ways.

Sketch an implementation of these classes which uses the OBSERVER pattern.



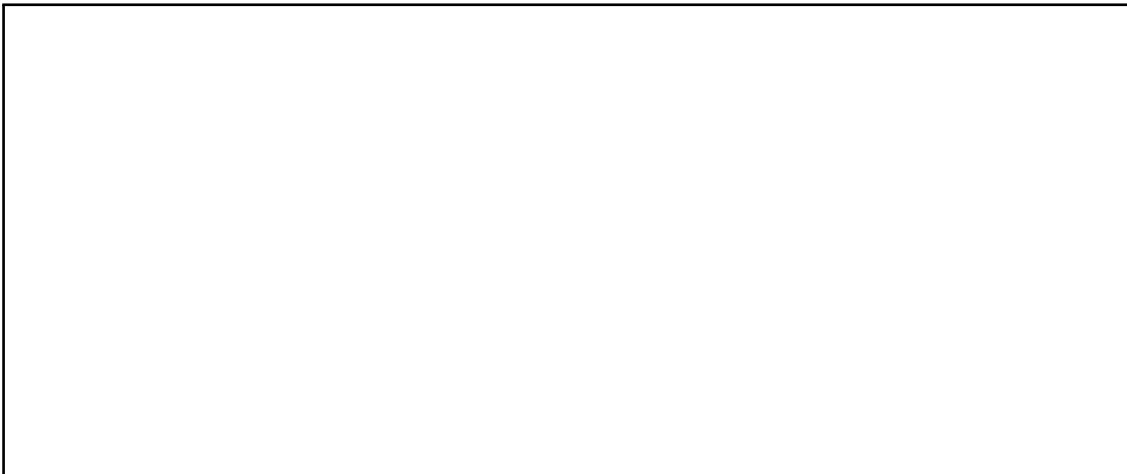
(Question 3 continued)

(b) This question is concerned with the COMPOSITE pattern.

(i) [4 marks] Briefly, describe the *problem* being solved by the COMPOSITE pattern.



(ii) [5 marks] Provide an appropriate *class diagram* which describes the COMPOSITE pattern in an abstract sense.



(Question 3 continued)

(iii) [6 marks] An *XML Object* has the form “<tag attrs>...</tag>”, where *tag* is the *object name*, and *attrs* a sequence of zero or more *attributes*. Each attribute has the form “attr=str”, where *attr* is the *attribute name* and *str* a string constant. An XML Object may contain zero or more XML Objects within it. The following illustrates a simple example:

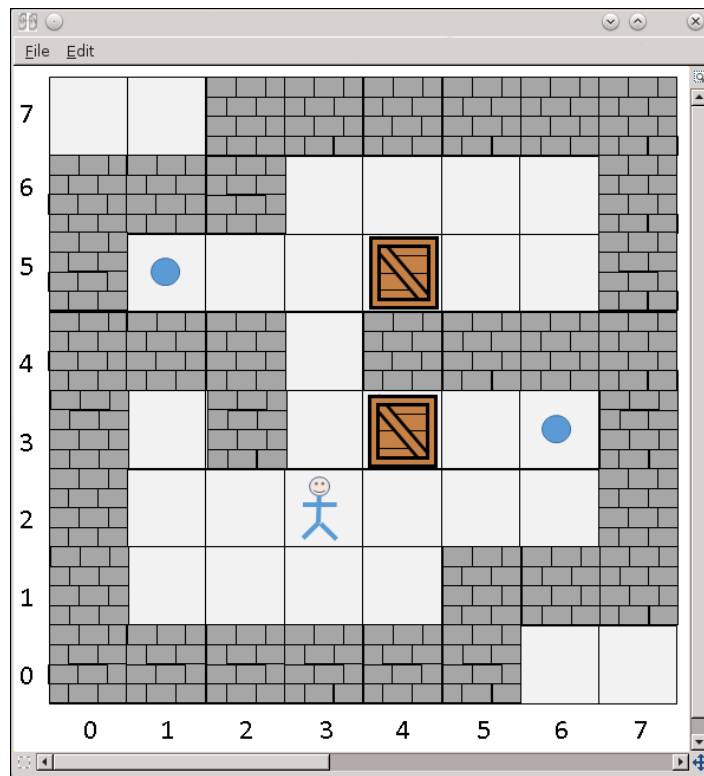
```
1 <project name="wyc">
2   <target name="compile">
3     </target>
4   <target name="test" depends="compile">
5     </target>
6 </project>
```

Sketch a Java implementation for representing XML Objects which uses the COMPOSITE pattern.

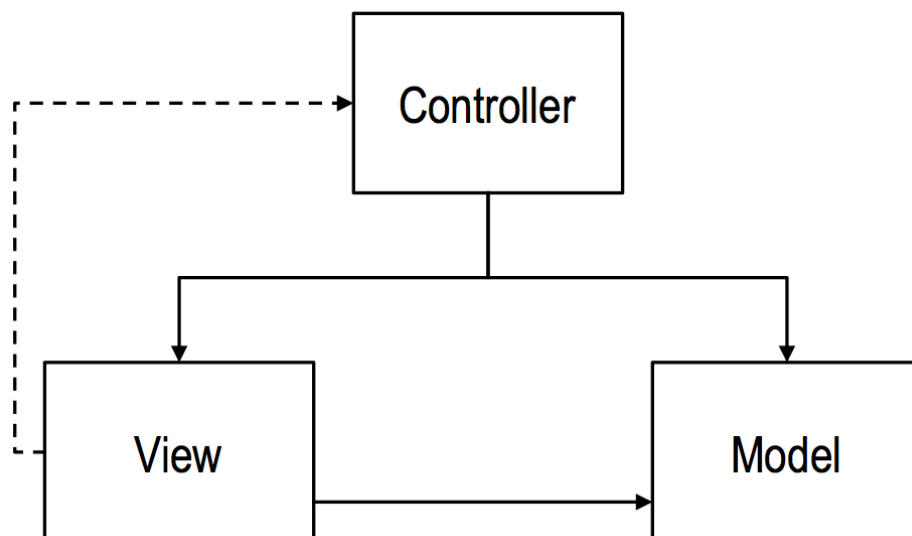
Question 4. Design Patterns II

[30 marks]

You have been asked to design an implementation of the popular game *Sokoban* which uses a *Graphical User Interface (GUI)*. In the game, the player uses the keyboard to move his/her *character* around the board. The goal is to push *crates* into *storage locations* and, when every *crate* is placed in a *storage location*, the game is over. The player's character cannot move through *crates* or *walls* and cannot push more than one *crate* at a time. The following illustrates how the game might look:



You intend to use the *Model/View/Controller* design pattern in your implementation:



(a) [5 marks] Briefly, discuss the features of the game that the *Model* component would be responsible for.

(b) [5 marks] Briefly, discuss the features of the game that the *View* component would be responsible for.

(c) [5 marks] Briefly, discuss the features of the game that the *Controller* component would be responsible for.

(d) [5 marks] The DECORATOR pattern is often used when implementing a graphical user interface. Briefly, outline how this pattern might be used in the Sokoban game.

(e) Someone suggested implementing a *command-line* version of the Sokoban game. This would draw the board using a simple text-based user interface, rather than a graphical user interface.

(i) [5 marks] Briefly, discuss how the Model/View/Controller pattern makes it easier to add a text-based user interface.

(ii) [5 marks] Briefly, discuss how implementing both command-line and GUI versions of Sokoban might uncover issues with the separation of *Model* and *View*.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.