

EXAMINATIONS – 2015
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
All questions are of equal value

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.	Software Design	30
2.	Design by Contract	30
3.	Software Design Qualities	30
4.	Design Patterns	30
Total		120

Question 1. Software Design

[30 marks]

Consider the following (incomplete) description for a simple board game called *Land and Water*.

“The game board is made up of 37 hexagonal *territories*. Each territory is *adjacent* to at most six other territories. Each territory either represents *land* or *water*. Land territories are further classified as *plains*, *forests* or *mountainous regions*. Forests contain a certain number of trees and, when all trees are chopped down, automatically become plains.

Up to six *players* can play the game. Each player owns one or more territories and can harvest resources of *gold* or *wood* by mining them or logging them. Players can spend their resources by building *houses* or *castles* on territories they own.”

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



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

(c) [12 marks] Provide a straightforward *Java implementation* for each of the three classes identified in part (b). You need only consider those aspects of the game described on Page 2.

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. Design by Contract

[30 marks]

Consider the following implementation of a Character Buffer, which compiles without error:

```

1  public class CharBuffer {
2      private char[] buffer;
3      private int length = 0;
4
5      public CharBuffer(int max) { buffer = new char[max]; }
6
7      public CharBuffer(char[] buffer) {
8          this.buffer = buffer;
9          this.length = buffer.length;
10     }
11
12     public void append(char c) {
13         if(length == buffer.length) {
14             // not enough space in buffer!
15             char[] nbuffer = new char[buffer.length * 2];
16             // copy elements from old buffer to new buffer
17             System.arraycopy(buffer,0,nbuffer,0,buffer.length);
18             // activate new buffer
19             buffer = nbuffer;
20         }
21         buffer[length] = c;
22         length = length + 1;
23     }
24
25     public char charAt(int index) {
26         return buffer[index];
27     }
28
29     // set the character at a given index
30     public void set(int index, char c) {
31         buffer[index] = c;
32     }
33
34     // Return size of buffer's active portion
35     public int length() { return length; }
36 }

```

(a) For each method listed below, provide appropriate *preconditions* and *postconditions*:

(i) [2 marks] `CharBuffer(int max)`

REQUIRES:

ENSURES:

(ii) [2 marks] `CharBuffer(char[] buffer)`

REQUIRES:

ENSURES:

(iii) [2 marks] `set(int index, char c)`

REQUIRES:

ENSURES:

(iv) [2 marks] `length()`

REQUIRES:

ENSURES:

(b) [4 marks] Method contracts can help establish who is to *blame* when an error occurs. Briefly, discuss what this means.

(c) [5 marks] An important aspect of the functional programming paradigm is that methods are *side-effect free*. Briefly, state what this means.

(d) [5 marks] The precondition/postcondition of a method often itself includes calls to other methods. Briefly, discuss why it is helpful for such methods to be side-effect free.

(e) [8 marks] Rewrite the `CharBuffer` class to use a *functional design*.

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. Software Design Qualities

[30 marks]

(a) [4 marks] Define what is meant by the term *coupling* in software design.

(b) [5 marks] Describe the *positive* implications of *strongly coupled* object oriented designs.

(c) [5 marks] Describe the *negative* implications of *strongly coupled* object oriented designs.

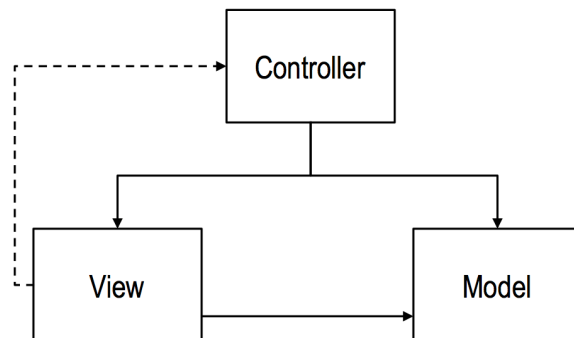
(d) [6 marks] Describe three benefits of *inheritance* in Java software designs.

(e) [6 marks] Provide a *UML class diagram* of the key features of the COMPOSITE pattern.

(f) [4 marks] Explain the role that inheritance plays in the COMPOSITE design pattern.

Question 4. Design Patterns

[30 marks]



(a) [10 marks] Explain what is meant by the phrase “We need **SMART Models**, **THIN Controllers**, and **DUMB Views**.”

The code sample provided below provides an implementation of a Controller for a piece of software.

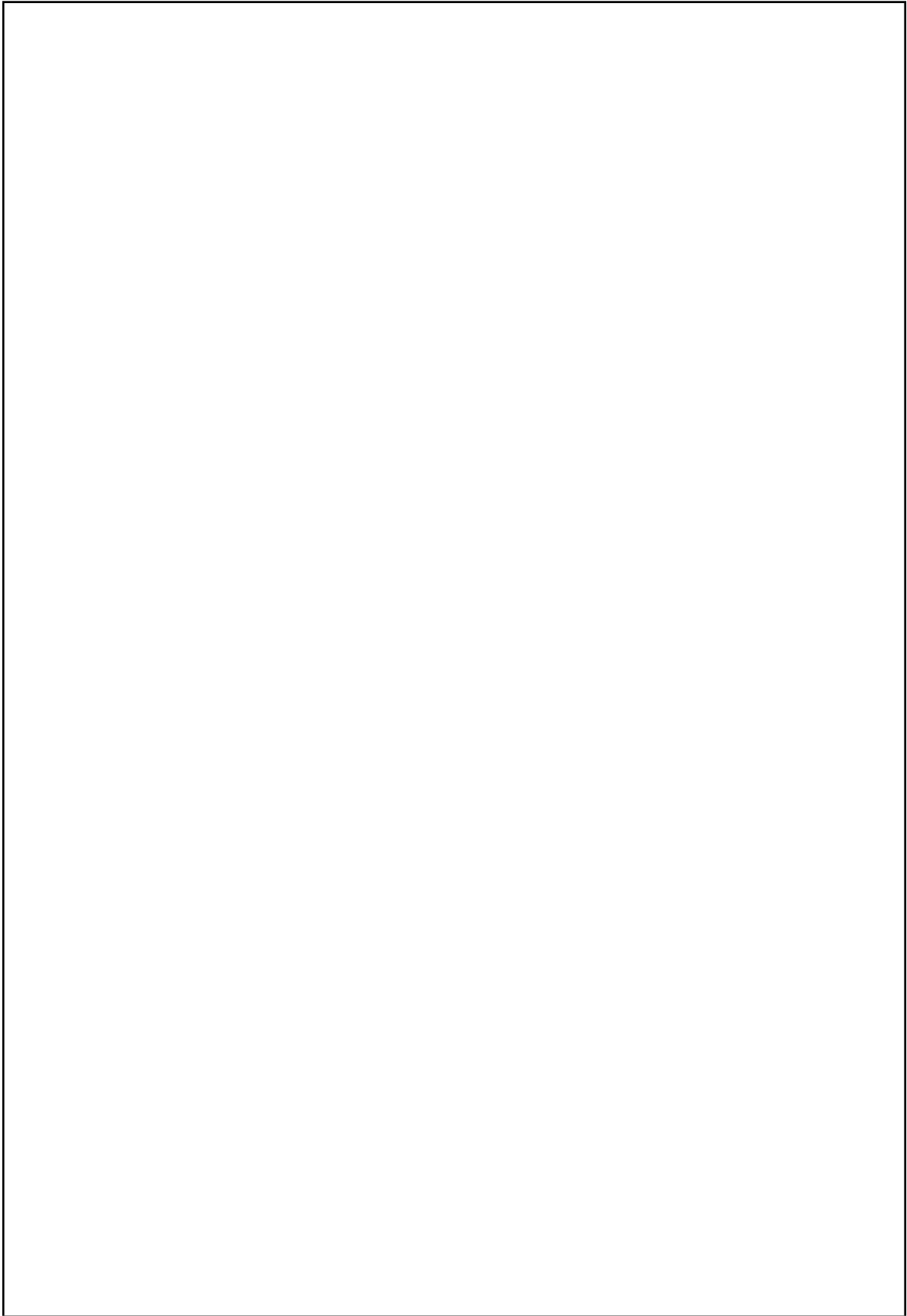
```

1  public class GameController implements KeyListener {
2      private GameBoard board;
3      private GameView view;
4      private Point currentLocation;
5      private GameObject objectInHand;
6
7      // *** Setup code omitted for clarity ***
8
9      public void keyPressed(KeyEvent e) {
10         switch (e.getKeyChar()) {
11             // Move up command
12             case 'w':
13                 Point newLocation = new Point(currentLocation);
14                 newLocation.y += 1;
15                 GameSquare newSquare = board.getSquare(newLocation);
16                 if (checkSquareCanBeEntered(newSquare) == true) {
17                     Point oldLocation = new Point(currentLocation);
18                     currentLocation = newLocation;
19                     view.drawPlayerMovingToSquare(oldLocation, newLocation);
20                 } else {
21                     view.playInvalidMoveSound();
22                     view.drawPlayerStandingInSquare(currentLocation);
23                 }
24                 break;
25             // Pick up object command
26             case 'q': {
27                 GameSquare square = board.getSquare(currentLocation);
28                 GameObject squareObject = square.getObject();
29                 if (squareObject == null) {
30                     view.playInvalidCommandSound();
31                 } else {
32                     square.removeObject();
33                     objectInHand = squareObject;
34                 }
35                 view.drawPlayerStandingInSquare(currentLocation);
36                 break;
37             }
38             // Drop current object command
39             case 'e': {
40                 GameSquare square = board.getSquare(currentLocation);
41                 square.addObject(objectInHand);
42                 objectInHand = null;
43                 view.drawPlayerStandingInSquare(currentLocation);
44                 break;
45             }
46             default:
47                 view.playInvalidCommandSound();
48                 view.drawPlayerLookingBored(currentLocation);
49         } }

```

```
50
51  public boolean checkSquareCanBeEntered(GameSquare theSquare) {
52      if (theSquare == null) {
53          // Trying to move off the board
54          return false;
55      } else if (theSquare.type == GameBoard.WALL_SQAURE) {
56          // Trying to into a wall
57          return false;
58      } else if (theSquare.content != null) {
59          // Square is already occupied
60          return false;
61      }
62      return true;
63  }
64 }
```

(b) [20 marks] Refactor this code to improve the GameController implementation in line with the statement made in part (a). Please clearly indicate any consequential changes to the GameBoard and GameView classes by providing the relevant code.

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a student to draw or write their answer.



* * * * *

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.

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.