

EXAMINATIONS — 2014**TRIMESTER 2****SWEN222****Software Design****Time Allowed:** THREE HOURS**Instructions:** Closed Book.

There are 180 possible marks on the exam.

Answer all questions in the boxes provided.

Every box requires an answer.

If additional space is required you may use a separate answer booklet.

No calculators permitted.

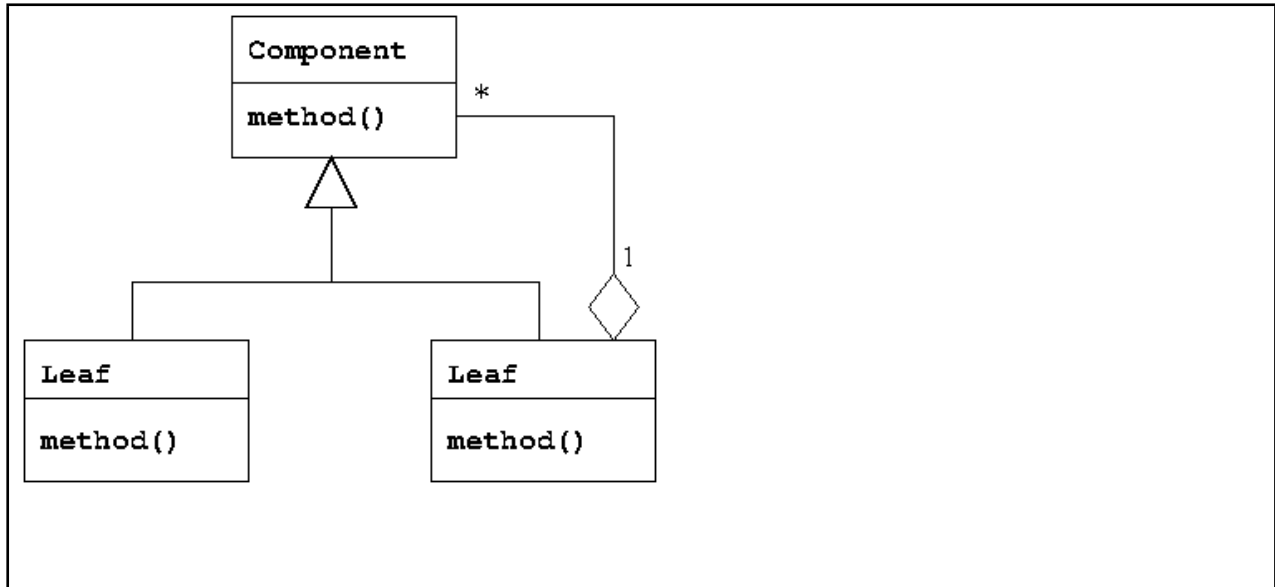
Non-electronic Foreign language to English dictionaries are allowed.

No reference material is allowed.

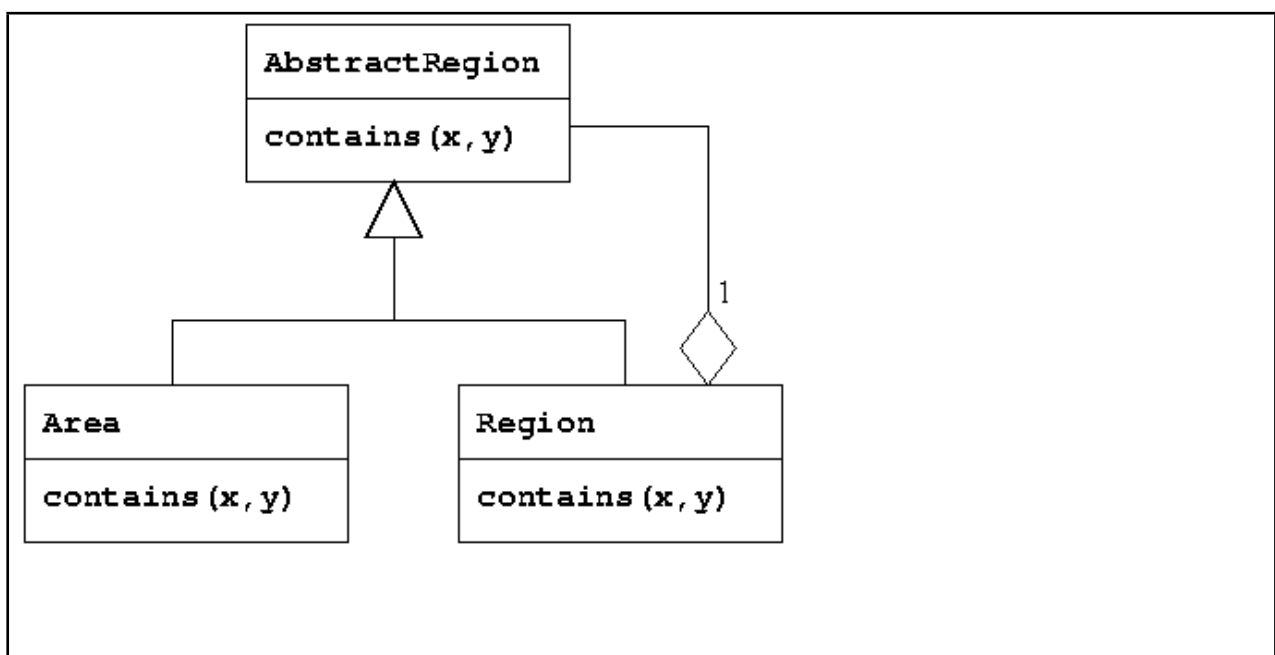
Question	Topic	Marks
1.	Design patterns 1	30
2.	Functional design	30
3.	Design by Contract	30
4.	Software Design Qualities	30
5.	Design Patterns 2	30
6.	Refactoring	30
Total		180

Question 1. Design Patterns 1

[30 marks]

(a) [8 marks] Provide a *class diagram* which describes the COMPOSITE pattern.Consider the following description for describing *areas* and *regions* in a geographical application:

“An *area* is a square section of land with dimensions measured in Kilometres (km^2). A *region* contains one or more areas or sub-regions. For example, a country can be considered as a region containing counties or states, which themselves are regions. An important concern is whether or not a given point (x, y) is contained within a region.”

(b) [4 marks] Provide a *class diagram* for describing regions and areas.

(c) [10 marks] Provide a *Java implementation* for describing regions and areas.

```
1 interface AbstractRegion {
2     boolean contains(int x, int y);
3 }
4
5 public class Area implements AbstractRegion {
6     private int x;
7     private int y;
8     private int width;
9     private int height;
10
11     public Area(int x, int y, int width, int height) {
12         this.x = x;
13         this.y = y;
14         this.width = width;
15         this.height = height;
16     }
17
18     public boolean contains(int px, int py) {
19         return x <= px && px < (x+width) &&
20             y <= py && py < (y+height);
21     }
22 }
23
24 public class Region implements AbstractRegion {
25     private ArrayList<AbstractRegion> regions;
26
27     public Region(List<AbstractRegion> regions) {
28         this.regions = new ArrayList<AbstractRegion>(regions);
29     }
30
31     public boolean contains(int x, int y) {
32         for(AbstractRegion r : regions) {
33             if(r.contains(x,y)) { return true; }
34         }
35         return false;
36     }
37 }
```

(d) Consider the following additional requirements regarding regions. For each, briefly discuss whether or not this is true of your implementation.

(i) [4 marks] The structure of regions represents a tree.

The structure of regions in my implementation can be viewed as a tree, where instances of `Area` represent the leaf nodes and those of `Region` the non-leaf nodes. However, nothing in my implementation prevents one region from being contained in another more than once. Note that it is essentially impossible for a region to contain cycles, since there is no easy way to add to a region after it is created. Therefore, my implementation more closely describes a DAG rather than a tree.

(ii) [4 marks] A region cannot be contained in a region more than once.

Yes, a region or area can be contained directly in another region more than once, or indirectly in one or more its subregions. We can prevent a region being repeatedly contained in a given node by using a set to remove duplicates. This would require appropriate `equals()` and `hashCode()` methods be defined. However, it is harder to recursively ensure that a region does not appear in all those regions contained in another.

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

[30 marks]

Consider the following class for representing *binary data*.

```
1
2 public class BitSet {
3     private boolean[] data;
4
5     public BitSet(boolean[] data) {
6         this.data = data;
7     }
8
9     public boolean get(int index) {
10        return data[index];
11    }
12
13    public void set(int index, boolean bit) {
14        if(index >= data.length) {
15            // Make sure there is enough space
16            boolean[] nData = new boolean[data.length * 2];
17            System.arraycopy(data, 0, nData, 0, data.length);
18            data = nData;
19        }
20        data[index] = bit;
21    }
22 }
```

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

A function is side-effect free if it:

1. Does not modify any state that existed before the function was called.
2. Does not call any methods which are not themselves side-effect free.
3. Does not perform any I/O (e.g. printing to the console, etc).

(b) For each of the following `BitSet` methods, briefly discuss whether or not it is side-effect free.

(i) [2 marks] `BitSet.get(int)`

This function is side-effect free as it does not modify any state or perform I/O, etc.

(ii) [2 marks] `BitSet.set(int, boolean)`

This function is not side-effect free as it modifies the `data` field which existed before the function was called.

(c) Another important aspect of the functional programming paradigm is *immutability*.

(i) [4 marks] Briefly, discuss whether or not the `BitSet` class is *immutable*.

An immutable class is one which cannot its state modified after creation. The `BitSet` class is not immutable because the `set` method allows its state to be modified.

(ii) [4 marks] Briefly, discuss the following statement:

“Immutable classes can only have methods which are side-effect free.”

This is not true. Certainly, classes which only have side-effect free methods are immutable. However, an immutable class may have methods with side-effects, provided those methods do not modify its own state. They could, for example, perform I/O and/or modify other objects passed in as parameters.

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

```

1 public class BitSet {
2     private boolean[] data;
3
4     public BitSet(boolean[] data) {
5         this.data = new boolean[data.length];
6         for(int i=0;i!=data.length;++i) {
7             this.data[i] = data;
8         }
9         // could e.g. also use Arrays.copyOf(), etc
10    }
11
12    public boolean get(int index) { return data[index]; }
13
14    public BitSet set(int index, boolean bit) {
15        if(index >= data.length) {
16            // Make sure there is enough space
17            boolean[] nData = new boolean[data.length * 2];
18            System.arraycopy(data, 0, nData, 0, data.length);
19            return new BitSet(nData);
20        } else {
21            BitSet copy = new BitSet(data);
22            copy.data[index] = bit;
23            return copy;
24        }
25    }
26 }

```

(e) [5 marks] Using the `BitSet` class as an example, briefly discuss why programs using a functional design typically have fewer software bugs.

Aliasing is a critical problem in a typical object oriented program. This is where two or more references are held to the same object and, furthermore, this object is mutable. In such case, changes to the object through one reference are visible through the other references and this can make it difficult to track down where a problematic state change is actually occurring. Programs in a functional design tend to promote the use of immutable objects, which eliminates this kind of problem. Furthermore, the behaviour of a function will be consistent provided the same input, making it easier to test and debug.

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 by Contract

[30 marks]

Consider the following class which is used as part of an application for *matrix multiplication*.

```

1 public class Matrix {
2     private int[] items;
3     private int width;
4     private int height;
5
6     public Matrix(int width, int height) {
7         this.items = new int[width * height];
8         this.width = width;
9         this.height = height;
10    }
11    public int width() { return width; }
12    public int height() { return height; }
13
14    public int get(int x, int y) {
15        return items[x + (y * width)];
16    }
17    public void set(int x, int y, int item) {
18        items[x + (y * width)] = item;
19    }

```

(a) [2 marks] Briefly, state what a *pre-condition* is.

A precondition is a condition which must hold true before a function is executed. The function's implementation can then exploit the assumption that this is true (e.g. that an argument is non-null, etc).

(b) [2 marks] Briefly, state what a *post-condition* is.

A postcondition is a condition which must hold true after a function is executed. The function's implementation is responsible for ensuring that the postcondition holds, assuming that its precondition did.

(c) [2 marks] Briefly, state what a *class invariant* is.

A class invariant is a condition over the fields of a class which must hold true for the entire lifetime of that class. Specifically, after the constructor has completed.

(d) For each of the following methods, given appropriate *pre-* and *post-conditions*.

(i) [4 marks] `Matrix.Matrix(int width, int height)`

```
REQUIRES: width >= 0 && height >= 0
ENSURES: this.width == width && this.height == height
ENSURES: this.items.length == width * height
```

(ii) [4 marks] `Matrix.get(int x, int y)`

```
REQUIRES: x >= 0 && x < width && y >= 0 && y < height
ENSURES: \result == this.items[x + (y * width)]
ENSURES: (in English) this.items is unchanged by the function
```

(iii) [4 marks] `Matrix.set(int x, int y, int item)`

```
REQUIRES: x >= 0 && x < width && y >= 0 && y < height
ENSURES: this.items[x + (y * width)] == item
ENSURES: (in English) every other item in this.items is unchanged by the function
```

(e) [4 marks] Given an appropriate *class invariant* for the `Matrix` class.

```
INVARIANT: width >= 0 && height >= 0
INVARIANT: this.items.length == width * height
```

(f) An important problem is to ensure the pre-condition of a method is respected by its callers and, similarly, that a method guarantees its post-condition holds. A simple solution is to use *runtime assertions*.

(i) [4 marks] Briefly, discuss how you would modify the `Matrix` class to use runtime assertions.

There are three steps:

1. For each method, add appropriate `assert` statements at the beginning to check the precondition.
2. For each method, add appropriate `assert` statements at the end to check the postcondition. To do this, the method may need to be updated. For example, if the postcondition refers to parameters which held on entry to the function, then these need to be saved if they are modified by the method. Likewise, if the function is permitted to exit via an exception then the postcondition needs to be asserted in **finally** block.
3. For each method, add appropriate `assert` statements at the beginning and end of the method to check the class invariant is maintained. This is not a complete solution, and long running methods could be handled with greater resolution.

(ii) [4 marks] Unfortunately, runtime assertions cannot guarantee a program meets its specification. Briefly, discuss why not.

Runtime assertions check that a method's specification holds for a given execution of the method. That is, for a particular input for the method. However, they cannot check that the method's specification will hold for all possible inputs to the method. Furthermore, it is in general infeasible to exhaustively try every possible input. In contrast, a compile-time verification system will attempt to prove that a method meets its specification for all possible inputs.

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 4. Software Design Qualities

[30 marks]

(a) Coupling is an important concept relevant to software design.

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

Coupling is an indication of the strength of the interconnections between the components in a design. Highly coupled systems have strong interconnections, with program units dependent on each other

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

- Performance
- Efficient integration of strongly related activities
- Simpler to implement initially
- Greater coordination between classes

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

- Maintenance, unpredictable changes may propagate through design
- Potential loss of reuse opportunity for individual classes
- Complex designs

(b) Inheritance is an important concept relevant to software design.

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

- Code reuse
- Models domains efficiently
- Simplifies modification of current classes
- Compilers can enforce static type checking

(ii) [10 marks] Describe two limitations of *inheritance* in Java software designs and suggest alternative design approaches that address those limitations.

1) Multiple inheritance. Java has no support for direct multiple inheritance and, instead, relies on interfaces for this. However this can still cause problems when one wants to extend multiple classes.

2) Fragile Base Class. Inheritance in Java introduces the so-called *fragile base class* problem. The issue arises when child classes rely too tightly on the behaviour of their parent classes. Should the parent classes change in unexpected ways, this can result in code which no longer compiles or — worse still — hidden bugs in child classes.

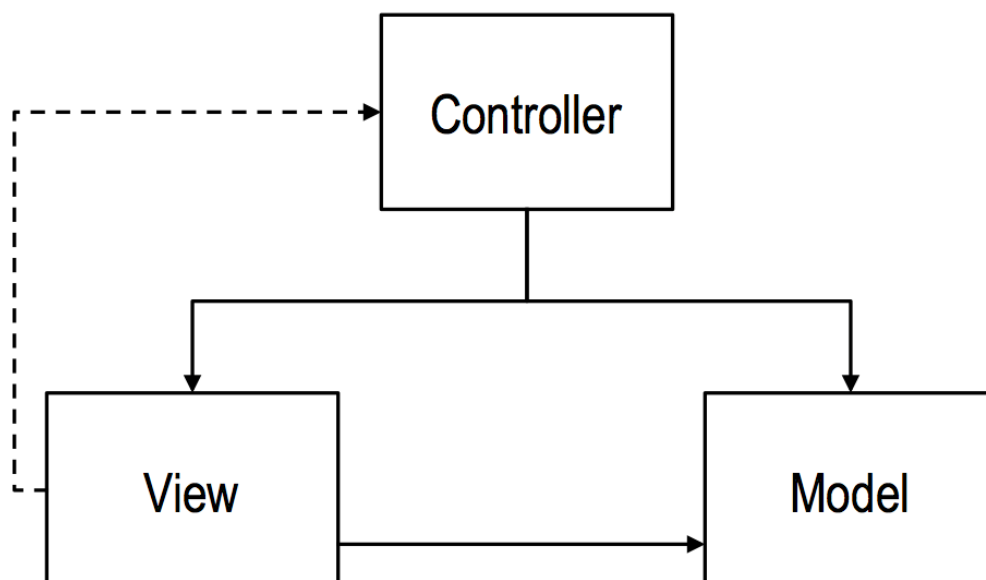
Question 5. Design Patterns 2

[30 marks]

You have been asked to design a Java GUI implementation of the popular board game *Scrabble*. Key features include:

1. The game is played on a board with 15 squares on each side.
2. Players have hands of seven tiles, drawn from a pool of 100 tiles.
3. The first player places a word in the middle of the board, and subsequently every player in turn places letters on the board such that they create words linked to the existing words.
4. All words, including those created by intersecting tiles, must be legal words listed in the Scrabble dictionary.
5. Scores are calculated based on the letters used and the squares occupied by the letters.
6. Some of the squares modify the score calculated for letters or words placed on them, others do not.
7. When all of the tiles in the pool have been drawn, the game is over.
8. For this simple implementation, all players use the same computer, taking turns to use the mouse and keyboard as needed.

You are required to use the Model/View/Controller design pattern in your design:



(a) [6 marks] What features of this game would be implemented in the Model, View and Controller sub-systems?

Model:

Player information, score, current tiles, score history including words placed
Current player, play order
Squares and their states and their point values Dictionary of valid words
Pool of available tiles
Turn management
Check that moves are valid, special case the first move of the game

View:

Board display
Scores
Remaining tiles in pool
Whos turn it is
Current players available tiles
Controls for creating and placing words
Display of whether word placement is legal

Controller:

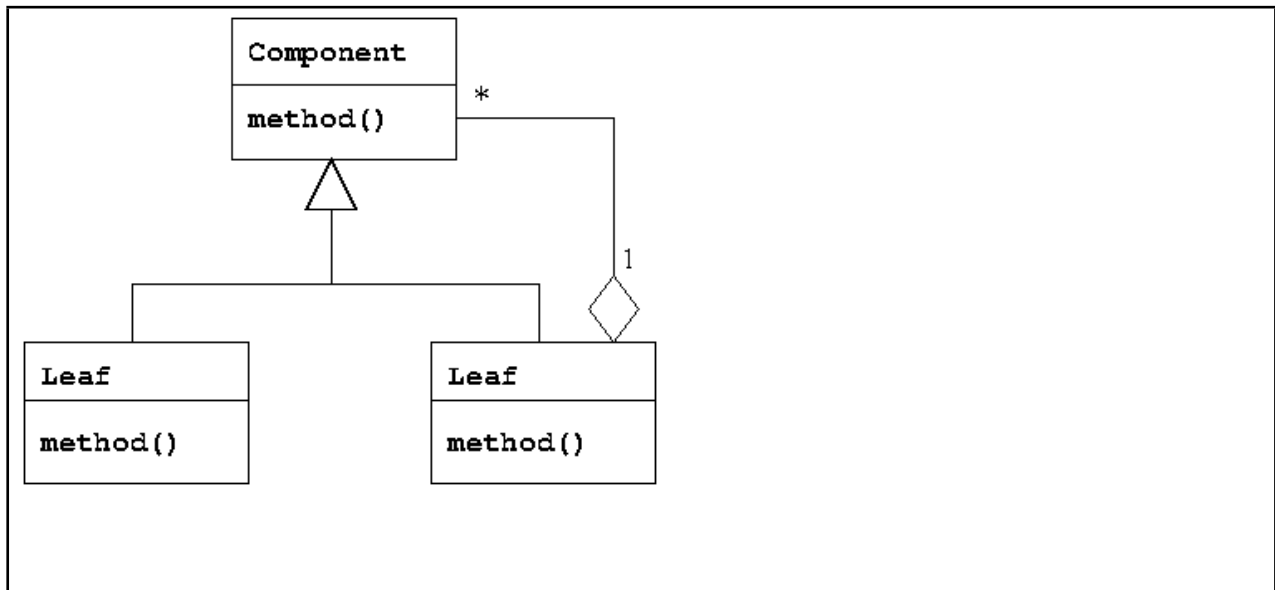
Response to control selection in view
Verification of actions with Model including real time response to dragging tiles onto squares
Keyboard, mouse and menu command responses

(b) View Design

(i) [2 marks] Identify a Design Pattern that could be effectively used to implement key elements of the View's graphical user interface design.

Composite

(ii) [4 marks] Provide a UML class diagram summarising the key features of the design pattern identified in **(b)(i)**.



(iii) [6 marks] Describe how using that pattern helps create a more effective design for the View.

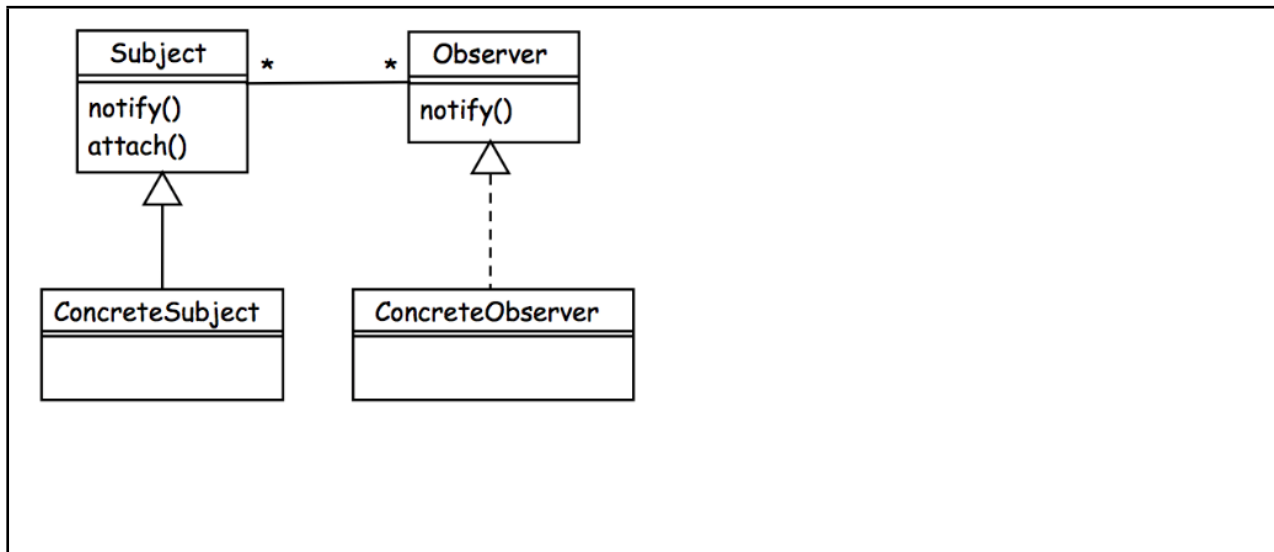
Models the composition arrangement of the visual layout of the various components, simplifies event handling, redrawing, management of various visual classes, layout can be handled more simply

(c) Model Design

(i) [2 marks] Identify a Design Pattern that could be effectively used to implement key elements of the Model's design.

Observer

(ii) [4 marks] Provide a UML class diagram summarising the key features of the design pattern identified in (c)(i).



(iii) [6 marks] Describe how using that pattern helps create a more effective design for the Model.

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 6. Refactoring

[30 marks]

Consider the following classes defined as part of the design of a simple online adventure game:

```

1  public class GameBoard implements KeyListener {
2      public enum SquareType {EMPTY, WALL, CAVE, WATER};
3      public GameSquare squares[][];
4      private static int max_x = 50;
5
6      /** load the squares from our configuration */
7      void setup (BufferedReader gameInfo) throws IOException {
8          String line = null;
9          int x = 0;
10         int y = 0;
11         while((line = gameInfo.readLine()) != null) {
12             squares[x][y] = new GameSquare();
13             squares[x][y].location
14                 = new Point(x,y);
15             squares[x][y].occupied = false;
16             switch (line) {
17                 case "Wall":
18                     squares[x][y].sq = SquareType.WALL;
19                     break;
20                 case "Cave":
21                     squares[x][y].sq = SquareType.CAVE;
22                     break;
23                 case "Water":
24                     squares[x][y].sq = SquareType.WATER;
25                     break;
26                 default:
27                     squares[x][y].sq = SquareType.EMPTY;
28             }
29             x++;
30             if (x > max_x){x = 0;y++;}
31         }
32     }
33
34     /** draw the board */
35     void drawSquares() { ... }
36
37     /** handle user commands to move squares */
38     public void keyTyped(KeyEvent e){ ... }
39     public void keyPressed(KeyEvent e){ ... }
40     public void keyReleased(KeyEvent e){ ... }
41 }

```

```
1 public class GameSquare {
2     public Boolean occupied;
3     public Point location;
4     public GameBoard.SquareType sq;
5     public Monster mIS1;
6     public Monster mIS2;
7     public Monster mIS3;
8     public Treasure tIS1;
9     public Treasure tIS2;
10    public Treasure tIS3;
11 }
12
13 public class Monster {
14     public String desc;
15     public Point pos;
16     public int hp;
17 }
18
19 public class Treasure {
20     public String desc;
21     public Point pos;
22     public int val;
23 }
```

(a) [18 marks] *Refactor* this design to improve it in *three significant and distinct ways*. For method bodies, you need only provide details needed to explain your refactorings. You can define new classes but only provide outlines of the methods for those classes.

Create a positionable interface and apply it to the Monster and Treasure classes, then change the GameSquare implementation so that it stores an array of positionable.

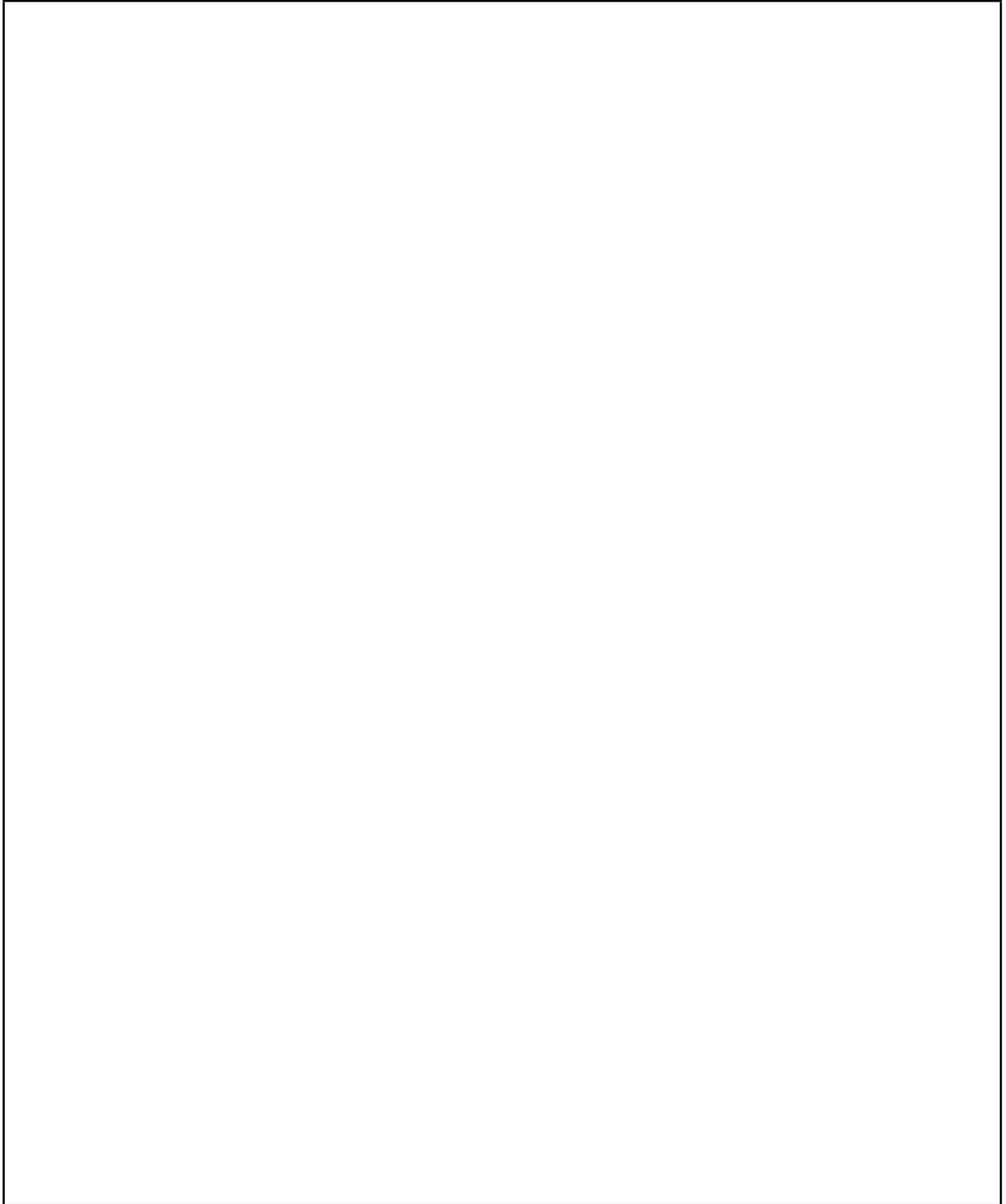
Change the class members to private and add get/set methods

Add a constructor to GameSquare that handles the assignment of the values in the setup method, bonus marks if they avoid exposing the location member by making a private copy

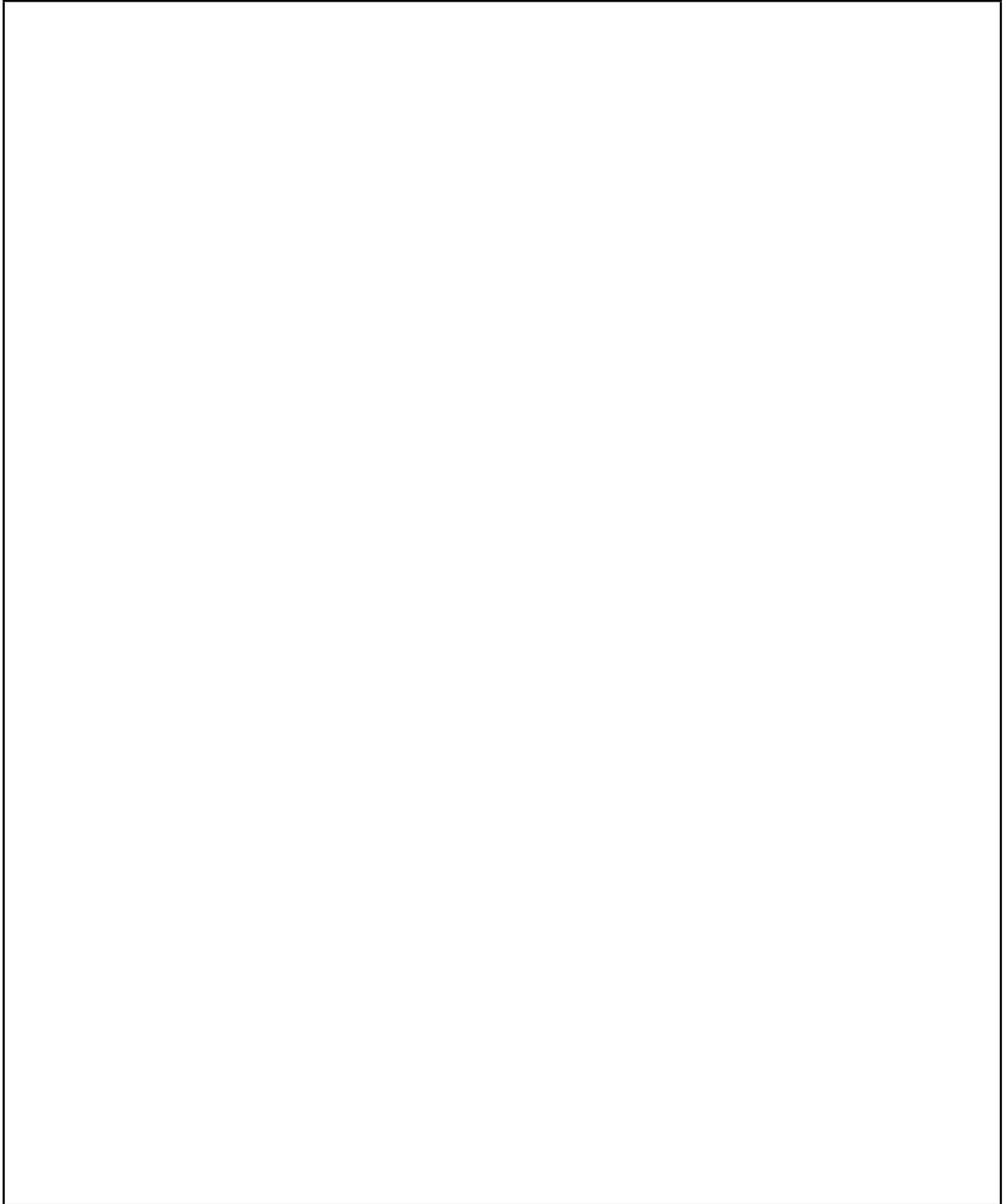
Change the square location handling so that the square location is managed by the GameSquare class and not by the GameBoard or vice-versa, not maintained in both classes

Changing method and variable names to be more clear half marks as not really a significant change

Student ID:

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:

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:

(b) [12 marks] Identify the three significant changes you have made to this design, and for each explain how it has improved the design.

1) 1 mark for a clear identification, 3 marks for a design explanation

2) 1 mark for a clear identification, 3 marks for a design explanation

3) 1 mark for a clear identification, 3 marks for a design explanation

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.