

EXAMINATIONS – 2018

TRIMESTER 2

<p>SWEN225</p> <p>SOFTWARE DESIGN</p>

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.

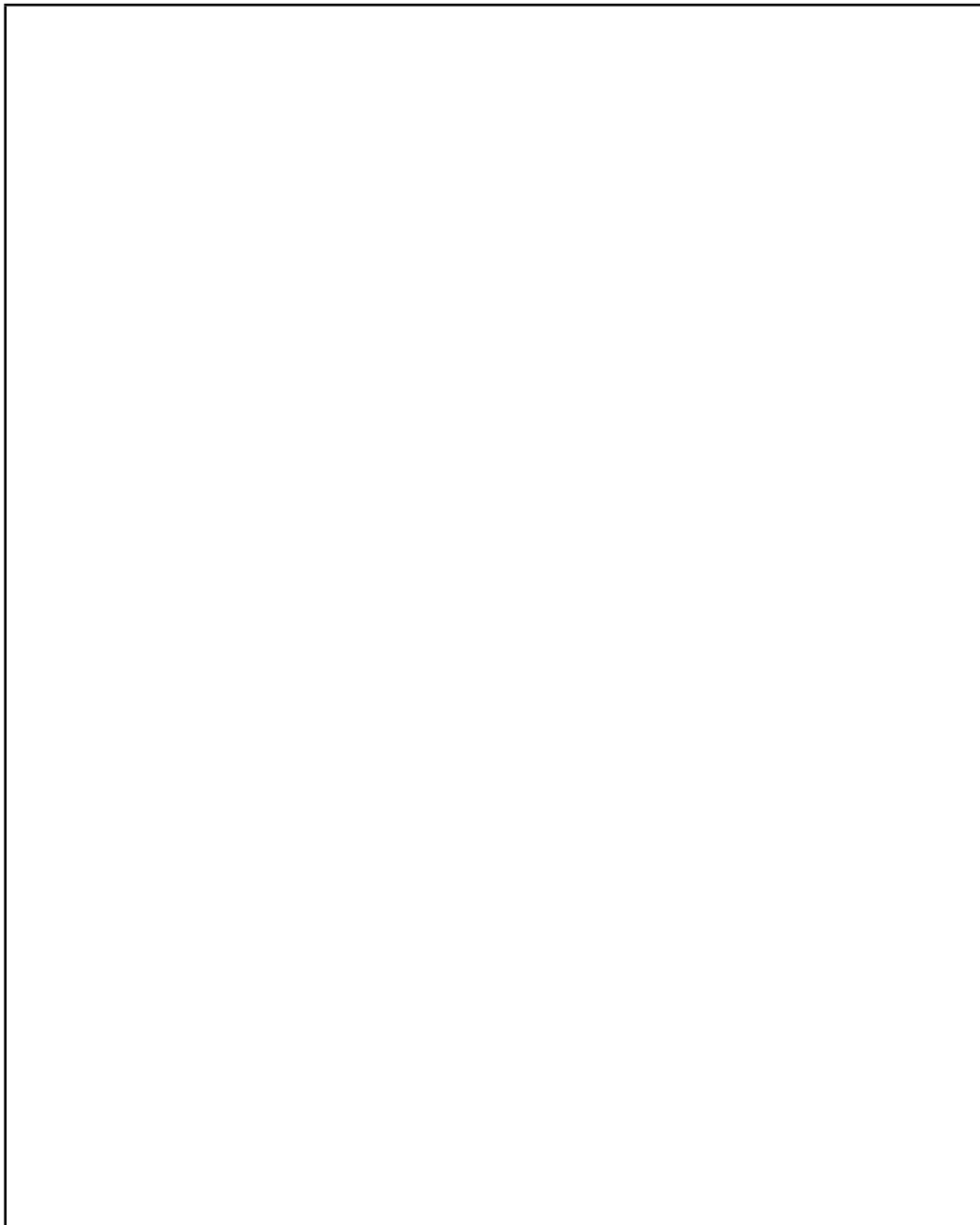
Question	Topic	Marks	
1.	Modelling	30	<input type="text"/>
2.	Design Patterns	30	<input type="text"/>
3.	Functional Design	30	<input type="text"/>
4.	Design by Contract	30	<input type="text"/>
Total		120	

1. Modelling

(30 marks)

- (a) **(15 marks)** A library offers both books and magazines to its users. Both books and magazines have titles and magazines additionally have an issue number. A single user may have up to 10 items on loan. Books can be loaned for up to five weeks and magazines up to three weeks. Users can make reservations for items and will be served according to their position in the reservation queue. Ensure that your design can be easily extended to deal with more library items such as DVDs.

Draw a UML class diagram using classes, associations, multiplicities, and inheritance that models the above library system. Make sure to use appropriate names for classes, association labels, and attribute names. Do not include any operations. For labelling associations use either labels for the whole association or role names at the association ends.



Student ID:

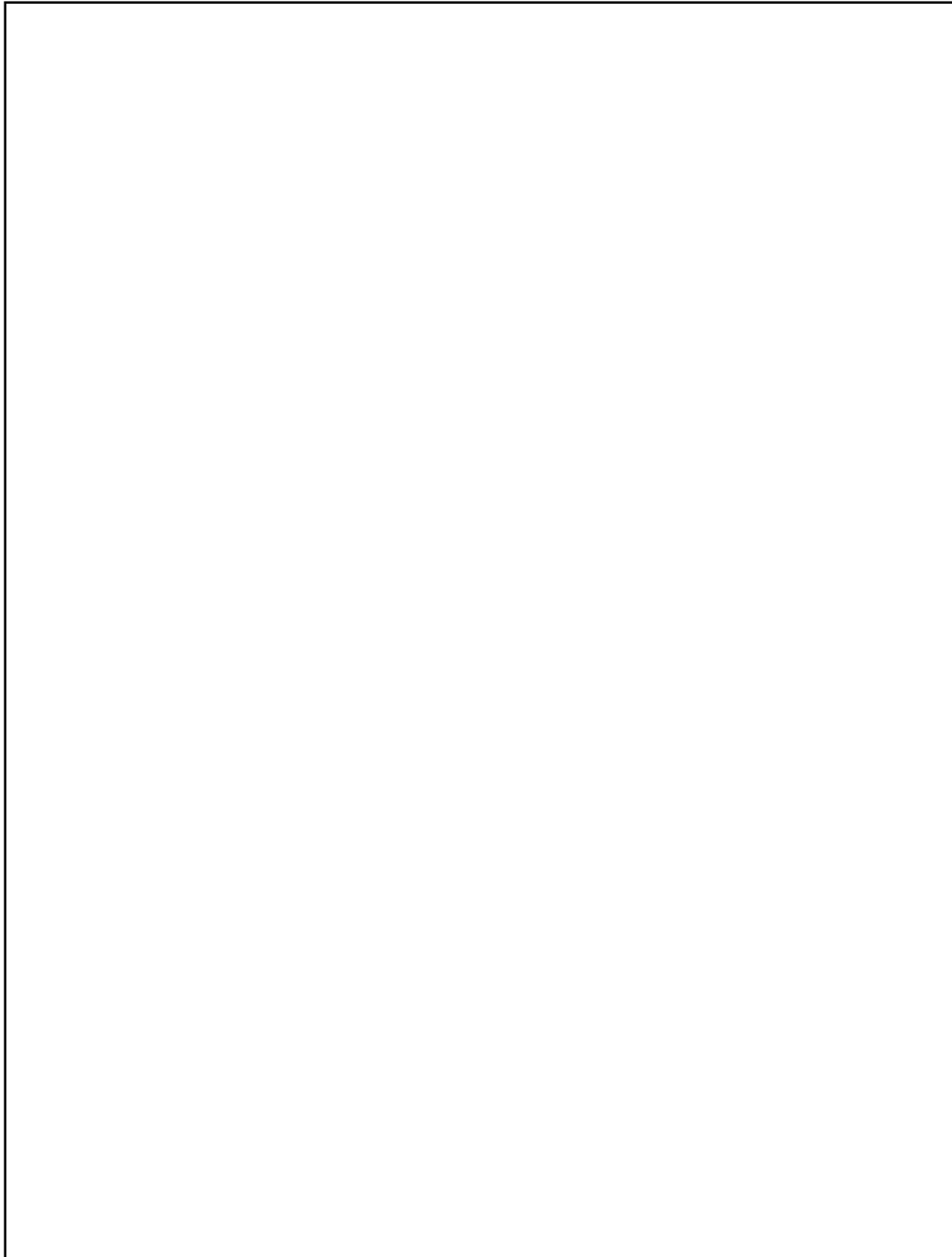
SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.

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

- (b) **(15 marks)** A cleaning robot is idle per default. After 48 hours of inactivity or when the user presses the “clean” button on a remote control, the robot starts cleaning the floor. The robot is capable of running in a “normal clean” and a “deep clean” mode. It will change to either mode depending on whether the user presses the “normal” or the “deep” button on the remote control. The remote also has buttons “fast” and “slow” which the user may press for noisier or quieter operation correspondingly. At any point in time during cleaning the robot’s battery level may become low in which case it will move to the charging station. After it has charged itself, it will continue with the cleaning operation (in the “normal clean” mode).

Draw a UML state diagram using states, transitions, concurrent states and substates as you see fit. Make sure to use appropriate names for states and transition labels.



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.

2. Design Patterns

(30 marks)

- (a) **(6 marks)** Briefly describe three advantages of using the Model-View-Controller design.

- (b) **(6 marks)** Name one design pattern used in the Model-View-Controller design. Briefly explain what the participants are and what their roles are.

- (c) **(6 marks)** The Composite pattern poses a challenge when used in a statically typed language like Java. Briefly explain what this challenge is by explaining what the problem is and what the forces at play are.

- (d) **(6 marks)** Name a design pattern that you haven't discussed so far that helps to achieve *low coupling*, briefly explain how it decreases coupling, and briefly describe the resulting advantages.

- (e) **(6 marks)** Tick all true statements. Ticking incorrect statements incurs a penalty.

- Language design may be informed by design patterns.
- Design patterns allow to reuse substantial amounts of pre-written pattern code.
- Design patterns embody good designs.
- Design patterns only make sense for object-oriented languages.
- Design patterns are a good way to pass on best-practice.
- It is easy to recognise design patterns in code due to the standardised names.

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.

3. Functional Design

(30 marks)

Consider the following interface for representing *shapes*.

```

1  interface MutableShape {
2      // Check whether point contained in shape
3      public boolean contains(int x, int y);
4      // Move shape by amount in x and y directions.
5      public void move(int dx, int dy);
6  }
```

- (a) An important aspect of the *functional programming paradigm* is that methods are *side-effect free*. Briefly discuss whether or not one would expect implementations of the following methods to be side-effect free.

- i. (3 marks) `Shape.contains(int, int)`

- ii. (3 marks) `Shape.move(int, int)`

- (b) (4 marks) Consider the following alternative design for representing shapes:

```

39 interface FunctionalShape {
40     // Check whether point contained in shape
41     public boolean contains(int x, int y);
42     // Move shape by amount in x and y directions.
43     public FunctionalShape move(int dx, int dy);
44 }
```

Briefly discuss how instances of `FunctionalShape` should behave compared with those of `MutableShape`:

(c) Consider the following implementation of the `MutableShape` interface:

```
8  class MutableRect implements MutableShape {
9      private int x, y, w, h;
10
11     public MutableRect(int x, int y, int w, int h) {
12         this.x=x; this.y=y; this.w=w; this.h=h;
13     }
14
15     public boolean contains(int px, int py) {
16         return px >= x && px < (x+w)
17             && py >= y && py < (y+h);
18     }
19
20     public void move(int dx, int dy) { x+=dx; y+=dy; }
21 }
```

i. (5 marks) Briefly discuss why `MutableRect` is not considered *immutable*.

ii. (5 marks) Briefly discuss how you would update this class to use a functional design and implement `FunctionalShape`.

(d) Consider the following implementation of the `MutableShape` interface:

```
23 class MutableUnion implements MutableShape {
24     private final MutableShape left, right;
25
26     public MutableUnion(MutableShape l, MutableShape r){
27         left=l; right=r;
28     }
29
30     public boolean contains(int x, int y) {
31         return left.contains(x,y) || right.contains(x,y);
32     }
33
34     public void move(int dx, int dy) {
35         left.move(dx, dy); right.move(dx, dy);
36     }
37 }
```

i. (5 marks) Briefly discuss whether `MutableUnion.contains(int, int)` can be considered side-effect free or not.

ii. (5 marks) Briefly discuss how you would update this class to use a functional design and implement `FunctionalShape`.

4. Design by Contract

(30 marks)

Consider the following interface for a *byte buffer*:

```

1  public interface ByteBuffer {
2      // Get the number of bytes currently stored in the buffer.
3      // Has no effect on the state of the buffer.
4      public int size();
5
6      // Get the maximum number of bytes which can be stored in
7      // the buffer. Cannot be less than the number of bytes
8      // stored in the buffer. Has no effect on the state of
9      // the buffer.
10     public int capacity();
11
12     // Write zero or more bytes into this buffer from a given
13     // array (which cannot be null). There must be enough
14     // capacity to hold the bytes being written. Afterwards,
15     // the size of the buffer is increased accordingly.
16     public void write(byte[] bytes);
17
18     // Read n bytes from this buffer into a given array (which
19     // cannot be null). The number of bytes being read cannot
20     // be negative and cannot exceed the number of bytes
21     // currently stored in the buffer. Afterwards, the size
22     // of the buffer is decreased accordingly.
23     public void read(byte[] bytes, int n);
24 }

```

- (a) For each method listed below, provide appropriate *preconditions* and *postconditions* using only the **public** methods of ByteBuffer:

- i. (2 marks) **int** size()

REQUIRES:

ENSURES:

- ii. (2 marks) **int** capacity()

REQUIRES:

ENSURES:

iii. (2 marks) `void write(byte[] bytes)`

REQUIRES:

ENSURES:

iv. (2 marks) `void read(byte[] bytes, int n)`

REQUIRES:

ENSURES:

(b) (4 marks) The precondition/postcondition of a method often itself includes calls to other methods. Briefly discuss why such methods should be side-effect free giving examples from the `ByteBuffer` interface.

- (c) Consider the following implementation of `ByteBuffer` which compiles without error:

```
1 public class MyByteBuffer implements ByteBuffer {
2     private byte[] bytes;
3     private int size;
4
5     public MyByteBuffer(int n) {bytes = new byte[n];}
6
7     public int size() { return size; }
8
9     public int capacity() { return bytes.length; }
10
11    public void write(byte[] src) {
12        if (src == null) { return; }
13        System.arraycopy(src, 0, bytes, size, src.length);
14        size = size + src.length;
15    }
16
17    public void read(byte[] target, int n) {
18        if(n>=16) {throw new IllegalArgumentException();}
19        size = size - n;
20        System.arraycopy(bytes, size, target, 0, n);
21    }
22 }
```

- i. (2 marks) Give an appropriate *class invariant* for the `MyByteBuffer` class.

- ii. (5 marks) Can your class invariant ever be violated? Justify your answer.

(d) Briefly discuss whether the following methods violate *Liskov's Substitution Principle* with respect to `ByteBuffer`:

i. (3 marks) `MyByteBuffer.write(byte[])`

ii. (3 marks) `MyByteBuffer.read(byte[], int n)`

iii. (5 marks) Assume "`class C implements I`". Briefly discuss how preconditions and postconditions of methods in `I` may differ from the corresponding methods in `C` *without violating Liskov's Substitution Principle*.

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.