TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2019

TRIMESTER 2

SWEN225

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.	Modelling	30
2.	Design Patterns I	30
3.	Design Patterns 2	30
4.	Functional Design and Contracts	30
	Total	120

1. Modelling **(30 marks)**

   (a) **(15 marks)** Draw a UML class diagram that captures the concepts relevant to a restaurant, including their relationships with each other. The restaurant has a dining room and a kitchen. The dining room contains tables which each can accommodate up to four restaurant clients. Each client may order a dish from a waiter. The dish is prepared by a cook who uses ingredients with a certain cooking style that depends on the dish the ingredient is used for. Waiters have to manage tables that are empty, have waiting clients, and have served clients respectively. For each person in the restaurant it must be known which room they occupy.

   Use classes, associations, multiplicities, and inheritance that models the above restaurant. 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. Make use of advanced notation when appropriate.

(b) **(15 marks)** Draw a UML state diagram that describes the states and events of a wireless phone with the following behaviour: Initially, the phone is idle. When an incoming call arrives, it keeps ringing until the user picks up or the caller aborts the call. In the former case the phone is connected to the calling party, in the latter case it becomes idle again. In the case of an outgoing call, when the user picks up the handle, the phone keeps accepting individual digits until a valid number has been dialled. In the latter case, it becomes connected to the called party. At any point during the dialling or while being connected, the user may hang up, causing the phone to become idle again. The phone may be charging or discharging its battery.

Use states and transitions to describe the above behaviour and use concurrent states and substates when appropriate. Make sure to use appropriate names for states and transition labels.

2. Design Patterns **(30 marks)**

   (a) **(6 marks)** Briefly describe the three major steps that occur in a Model-View-Controller design when the user changes a value of the model by interacting with the user interface.

   (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 Template pattern achieves a separation between two different types of code. Briefly explain what these two different types of code are and explain how the Template pattern combines those two types of code.

(d) **(6 marks)** How do many design patterns achieve *low coupling*? Briefly describe the general principle and name one design pattern, naming which two entities it decouples from each other.

(e) **(6 marks)** Tick any box preceding a concept that is relevant to design pattern descriptions. A correct tick yields two marks, an incorrect tick deducts two marks. The overall score cannot be lower than zero.

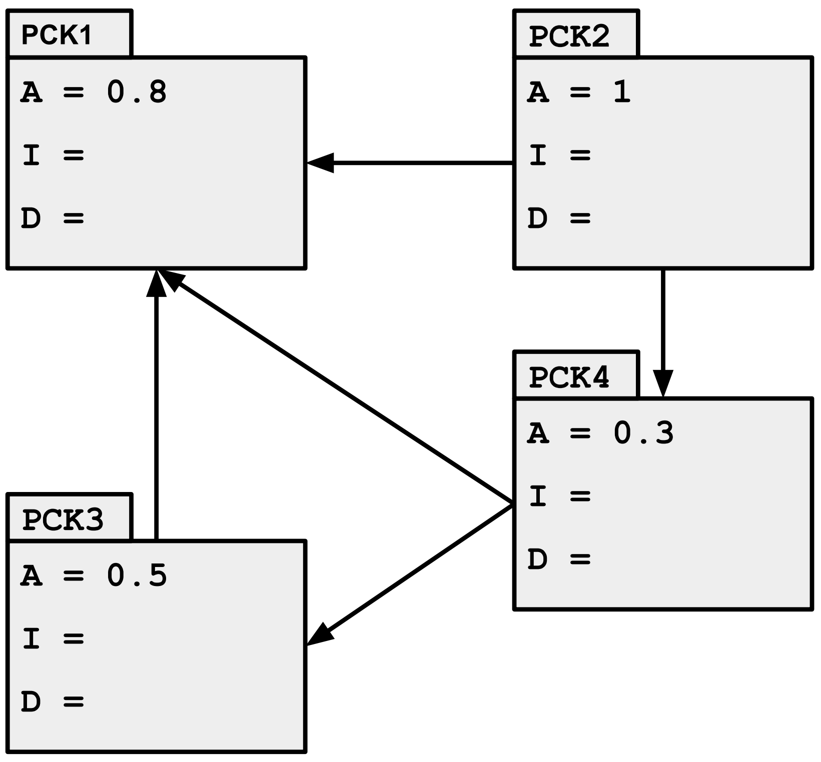
- ☐ Context in which the pattern may appear.
- ☐ Number of lines of code used in the pattern.
- ☐ Participants collaborating in the pattern.
- ☐ Class names required in the pattern.
- ☐ Language and library versions required for the pattern.
- ☐ Consequences of using the pattern.

3. Design Patterns 2 **(30 marks)**

(a) **(6 marks)** Which design patterns are predominantly used in the following XML parser and query technologies in Java? Answer this question by placing a tick into the appropriate table cells. There is exactly one correct tick per row. 1 mark for each correct tick, -1 for each incorrect tick. The overall score cannot be lower than zero.

	DOM Parser	SAX Parser	StAX Parser	XQuery	Not applicable
Observer	☐	☐	☐	☐	☐
MVC	☐	☐	☐	☐	☐
Domain Specific Language / Interpreter	☐	☐	☐	☐	☐
Iterator	☐	☐	☐	☐	☐
Builder	☐	☐	☐	☐	☐
Composite	☐	☐	☐	☐	☐

(b) **(14 marks)** Consider the following package dependency diagram.



i. **(8 marks)** Calculate the values for instability (`I`) and distance to main sequence (`D`), and put the respective values into the diagram.

ii. **(6 marks)** One of the packages will have a value for `D` that indicates a design flaw. State which package this is, and briefly discuss why the D value may indicate poor design.

(c) **(10 marks)** SQL is a query language used to query relational databases. A simple SQL SELECT statement to query for student details looks as follows:

```
SELECT NAME, FIRST_NAME FROM STUDENTS WHERE DEGREE='BE'
AND MAJOR='Software Engineering'
```

Assume your task is to develop an internal domain specific language (DSL) to help developers writing valid SQL statements. This should in particular help them with the following:

- to use the correct keywords (keywords used in the example are: `SELECT`, `FROM`, `WHERE`, `AND`)
- to ensure that each query has at least one column to select (column names are the comma-separated tokens following the SELECT keyword)

i. **(6 marks)** Sketch what the internal DSL you are tasked to develop could look like, by writing down the example query in the question defined using the DSL.

```
String sql =
```

ii. **(2 marks)** Name the design pattern that is widely used to implement internal DSLs.

iii. **(2 marks)** When implementing the DSL, how would you enforce the constraint that at least one column name must be provided in order to construct a valid `SELECT` statement?

4. Functional Design and Contracts **(30 marks)**

(a) **(16 marks)** A common approach to control aliasing is to use unmodifiable collection wrappers, such as the collections created by `java.util.Collections::unmodifiableCollection()`, instantiating `java.util.Collections$UnmodifiableCollection`. These wrappers are themselves collections which disable all methods in `Collection` that directly manipulate the state of the collection (methods like `add*` , `remove*, retain*`, `clear`).

i. **(6 marks)** Is this sufficient to ensure that the wrapped collection cannot be modified? Briefly discuss the capabilities and limitations of `Collections::unmodifiableCollection` in the context of the following scenarios:

- handling of subtypes of `java.util.Collection` such as lists and sets
- mutation of elements of the original (wrapped) collection by the application

ii. **(2 marks)** How are the methods modifying the state of the collection being disabled?

iii. **(2 marks)** Name the design pattern that is used by `Collections::unmodifiableCollection`. There are several variants and usages of this particular pattern. Also name the variant used here.

iv. **(6 marks)**

How are the roles in the design pattern mapped to types (classes and interfaces) in `java.util.Collections::unmodifiableCollection()`? Answer this question by adding role and type names into the empty spaces provided.

role in design pattern		name of class or interface
	⇒	
	⇒	
	⇒	

(b) **(5 marks)** Which of the following Java approaches are suitable to realise the respective contract element and/or signal contract violations? Answer this question by placing a tick into the appropriate table cells. There can be one or more correct ticks per row. 1 mark for each correct tick, -1 for each incorrect tick. The overall score cannot be lower than zero.

	Preconditions	Postconditions	Invariants	Not applicable
returning null	☐	☐	☐	☐
runtime exceptions	☐	☐	☐	☐
contract annotations	☐	☐	☐	☐
assertions	☐	☐	☐	☐
print warnings to console	☐	☐	☐	☐

(c) **(9 marks)** Assume there is a class `A` with a method `m` implemented in `A` (written as: `A::m`) and a subclass `B` of `A` overriding `m` (written as `B::m`). Consider the table below, and assess the impact changes made to the overriding method `B::m` have. Answer this question by placing a tick into the appropriate table cells. There can be one or more correct ticks per row. 1 mark for each correct tick, -1 for each incorrect tick. The overall score cannot be lower than zero. LSP means "Liskov Substitution Principle".

	violates LSP	results in a compiler error	no compiler error, but may result in runtime exceptions	not applicable
`B::m` declares a checked exception, but `A::m` does not	☐	☐	☐	☐
`A::m` declares a checked exception, but `B::m` does not	☐	☐	☐	☐
`A::m` is public, `B::m` is protected	☐	☐	☐	☐
`A::m` is protected, `B::m` is public	☐	☐	☐	☐
`B::m` declares a return type which is a subtype of the return type of `A::m`	☐	☐	☐	☐
`B::m` declares a return type which is a supertype of the return type of `A::m`	☐	☐	☐	☐
`B::m` throws an `UnsupportedOperationException`, but `A::m` does not (*)	☐	☐	☐	☐
`A::m` throws an `UnsupportedOperationException`, but `B::m` does not (*)	☐	☐	☐	☐
`A::m` never returns `null` (*), but `B::m` may return `null`	☐	☐	☐	☐

(*) Not only do those methods not throw `UnsupportedOperationException` or not return `null`, they also do not declare the intention to do so in their documentation.

* * * * * * * * * * * * * * *