Exam exam 7 July 2017, questions and answers

Object Oriented Design (Monash University)

Office Use Only

MONASH University

Sample
Examination Period

Faculty of Information Technology

EXAM CODES: **FIT2099**

TITLE OF PAPER: **OBJECT ORIENTED DESIGN AND IMPLEMENTATION – SAMPLE PAPER**

EXAM DURATION: 2 hours writing time

READING TIME: 10 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Berwick	☑ Clayton	☑ Malaysia	☐ Off Campus Learning	☐ Open Learning
☐ Caulfield	☐ Gippsland	☐ Peninsula	☐ Monash Extension	☐ Sth Africa
☐ Parkville	☐ Other (specify)			

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam. Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	**☑ YES**	**☐ NO**
CALCULATORS	**☐ YES**	**☑ NO**
SPECIFICALLY PERMITTED ITEMS **if yes, items permitted are:**	**☐ YES**	**☑ NO**

Instructions to students

This is a SAMPLE EXAM intended as a study aid.

The questions are of a similar style to and cover some of the same topics as the real exam.

The exam is formatted similarly to the real exam – each question is on a new page.

There are seven questions, of which you may attempt the first five and one of the last two, with a total of 79 marks available. On the real exam, there will only be one last question. For the sample exam, you have the opportunity to try two different ones. The estimate of two hours is based on only answering ONE of the last two questions.

The number of marks available for a question should be a rough guide as to how long you spend on it.

If you need clarification on some part of a question, you can ask on the Moodle Discussion Forum or come to consultation.

Question 1 – (6 + 6 = 12 marks)

You have joined a team that is working on a fully-featured version of the Star Wars game, complete with a graphical user interface, for commercial release. It's planned to re-use the engine code (which has gained a lot of new features, including GUI support) for similar games in the future.

The game runs reasonably well, but when you look at the engine code you discover that it contains a lot of "cut-and-paste reuse" – that is, repeated similar sequences of code, sometimes with a few very small changes.

a) Why do software engineers consider repeated code to be a bad thing? Write a paragraph or two about the risks of cut-and-paste reuse. (6 marks)

Repeated code makes software harder to maintain – if it turns out to contain a bug, or is in need of refactoring, the required changes have to be repeated everywhere the code has been copied.
Repeated code makes software harder to test – each repeat could contain a typo or error, and so it all needs to be tested separately.
Repeated code is a source of technical debt.
Repeated code makes software harder to read, because there's more of it.
If the code has been cut-and-pasted because the developer doesn't understand it, then it might not do what is needed (might even have security issues).
The developer might have intended to modify the code after cutting-and-pasting, but forgot to do so. The maintainer must work out whether it really is supposed to be identical to other code in the system.
(Other answers are possible; use your judgement as to whether they are valid.)

1 mark for mentioning a valid point; a further 2 marks for a *clear* explanation of the problem, to a maximum of 6 marks. If the explanation is vague but correct, it is worth 1 mark. Ignore anything about the **advantages** of cut-and-paste reuse; the question is asking specifically for **risks**.

b) Describe how you would go about fixing the code to reduce these risks. (6 marks)

Strategies:

- Refactor
  - make a series of small changes to the code structure
  - test at each step to ensure that the code still works properly
- General
  - identify areas of repeated code
  - if in same class, put repeated code into new method
    - call method wherever the repeated code is
  - if in different classes, decide which class should hold new functionality then proceed as above

Specific refactors to reduce repetition in code:

- methods: extract a method and call it wherever the repeated code appears
- inheritance: if repeated code exists in subclasses, it may be possible to pull it up into the superclass
- generics: if code is repeated to handle different input types, may be possible to make the type a parameter
- others – use judgment

Any of these basic approaches is acceptable.

6 marks: complete and clear explanation, mentioning the need to test/verify that the functionality hasn't been broken
4 marks: okay but incomplete, unclear, or lacking in detail
2 marks: mentions something relevant but doesn't say anything sensible about it
0 marks: no attempt (or word salad) – e.g. a description of what repeated code is.

If you can't decide which of two categories an answer falls in, split the difference (e.g. if torn between a 4 and a 6, give 5.)

Question 2 - (4 + 6 = 10 marks)

In mathematics, an integer is a number that is the member of the set {…,-2, -1, 0, 1, 2,…}

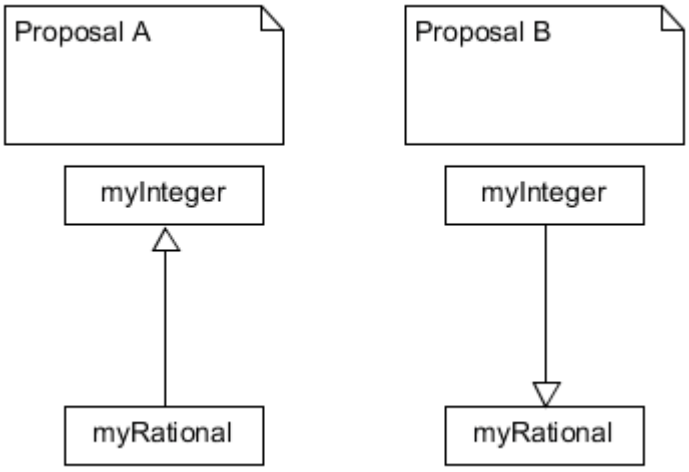
A rational number is any number that can be expressed as a fraction *p/q*, where *p* and *q* are integers and *q* is not 0.

Trivially, any integer *x* can be unambiguously expressed as a rational number x/1, so any integer is also a rational number.

Imagine you wished to define two classes, myInteger and myRational, that (initially) support the following operations:

- Constructing new objects with an initial value
- Incrementing the value by the value of another myInteger or myRational object.

Consider two class diagrams below that show proposed inheritance relationships between myInteger and myRational



a) In a couple of paragraphs, explain the advantages and disadvantages of both proposal A and proposal B, considering the design principles relating to inheritance discussed in FIT2099. (6 marks)
b) Propose an alternative design that avoids the disadvantages of both proposal A and proposal B. Draw a class diagram to illustrate it, and explain clearly how your proposal would implement the required operation. (6 marks)

a) This example has the same structure as the famous "*circle-ellipse*" problem. (Students are not required to know this, but if you want more information on this problem type, you can google it.)

Proposal A: if a myInteger stores one integer, then this would represent the numerator and myRational would only need to store one more integer to represent the denominator of the fraction. The disadvantage is that by the Liskov Substitution Principle, which says that you should always be able to substitute instances of the base class with instances of the subclass without breaking anything, we're essentially saying here that a myRational is a kind of myInteger. But rational numbers *aren't* integers. If anything, it's the other way around.

Proposal B doesn't have this problem, but it models a myInteger as a myRational where, presumably, the denominator is fixed at 1. That wastes a bit of space, but also introduces a potential problem: if myInteger inherits two integers from myRational, representing the numerator and the denominator, what guarantee will there be that the denominator will always be 1?
Key points to address for part a):

- Inheritance implies an "is-a" relationship, and it is not true that a rational number "is-a(n)" integer.

- Although an integer *is* a rational number, there are practical problems if you try to represent them this way.
- These points may be addressed with reference to data representation, to operations, or both (e.g. integer.add(rational) doesn't work – you can't change the class of a variable).

For full marks, must identify the first two points and justify them as indicated in the third.

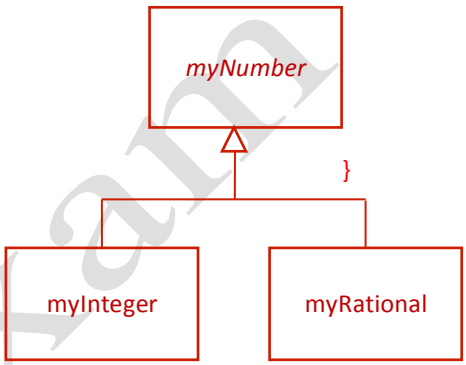
b) Perhaps it is best to model myInteger and myRational as subtypes of the same abstract superclass, myNumber. (It would also be fine to have myNumber be an interface that both myInteger and myRational implement.)

If inheritance was used, these classes could work like this:

In myNumber:

```
private int num;

protected myNumber(int num) {
   this.num = num;
```



In myInteger:

```
public myInteger(int num) {
   super(num);
}
```

In myRational:

```
private int denom;

public myRational(int num, int denom) {
   super(num);
   this.denom = denom;
}
```

3 marks for clearly describing a proposed solution (Java and UML are optional).
3 further marks if solution addresses the identified problem.

Question 3 - (4 + 5 + 6 = 15 marks)

Consider the following Java source code implementing a Unit class for a system similar to JavaUniversity from your labs:

```
/**
 * Represents a Unit at the university.
 *
 * @author Robert Merkel
 */
public class Unit {

	// the unit name
	private String name;

	// a list of assessments for this unit
	// invariant: the weight of all the assessments in this list must ALWAYS sum to
100.
	private ArrayList<Assessment> assessments;

	/**
	 *
	 * @param name the unit name
	 * @param code the unit code
	 * @param assessments the assessments for this unit. Their weights must sum to
exactly 100.
	 * @throws Exception if the assessment weights do not sum to 100.
	 */
	public Unit(String name, String code, Collection<Assessment> assessments) throws
Exception {
		this.name = name;
		int totalweight = 0;
		for(Assessment a: assessments) {
			totalweight += a.getWeight();
		}
		if (totalweight != 100) {
			throw(new Exception("Assessment weights do not sum to 100"));
		}
		this.assessments = new ArrayList<Assessment>(assessments);
	}

	/**
	 * Simple getter for unit name
	 * @return the unit name
	 */
	public String getName() {
		return name;
	}

	/**
	 * Simple getter for the list of assessments
	 * @return the assessment lists
	 */
	public ArrayList<Assessment>getAssessments() {
		return assessments;
	}
}
```

a) The class and the class methods both have Javadoc comments. However, the attributes do not – despite the fact that you can annotate attributes with Javadoc. In your own words, briefly explain the reason for using Javadoc comments in your code. Does it matter much if the comments for the attributes *in this class* do not use Javadoc? Why or why not? (4 marks)

It doesn't matter much, because these attributes are private. They can only be accessed from within the current class, so any programmer who wishes to use them must already be looking at the source code and so will be able to see the non-Javadoc comments.

2 marks if they just say "the attributes are private". Full marks if they explain why this means that Javadoc isn't needed.

b) There is a potential design problem with getAssessments(). Explain what is wrong, and write an alternative version of getAssessments() that does not have this problem. (7 marks)

This class has an invariant that says that the weights of the assessments must always add up to 100%. If getAssessments() returns a reference to the assessments list, then the client code that called it can modify the list at will. That means that the class can not maintain its documented invariant. (4 marks for the whole explanation; 2 marks for just saying that it returns a reference/returns the list/breaks the invariant without further explanation.)

To fix this, use defensive copying (as seen in labs and lectures):

```
public ArrayList<Assessment> getAssessments() {
   return new ArrayList<Assessment>(assessments);
}
```

Can also create a new empty ArrayList and iteratively copy the assessments across – the main thing here is that a copy is being made. 2 marks if the idea is obviously there; 3 marks if the code is reasonably correct (we're not worried about the generic syntax but it must be obvious what the method is called and what is being returned.)

c) The Java language specification states that if a method throws a *checked exception* (that is, any exception that is not an instance of RunTimeException or a subclass), that method must either:

- catch and handle them within its body; or
- declare in its signature that it can throw them back to the caller (e.g. as the constructor for Unit does)

Not all languages that support exception handling have this requirement. In Python, for instance, any method can raise any exception it chooses, and there is no requirement for the method signature to indicate this.

In about a paragraph, explain *one advantage* and *one disadvantage* of each approach. (6 marks)

- Advantage of declared exceptions: developers of client code can easily see which methods might throw exceptions, and therefore will know where they need to put try/catch blocks. The IDE and compiler can also check that exception handling code exists, which makes it less likely that a thrown exception will end up crashing the program.
3 marks for either of these explanations. Just saying "fewer crashes" or "easier for programmers" without clear reasoning is only worth 1.
- Advantage of undeclared exceptions: less tedious to write. Declarations are shorter. Compiler and IDE won't force you to write exception handling for "can't-happen" cases.

"Less tedious to write", on its own, is only worth 1.

Other valid/reasonable answers are fine – look for evidence that the student understands the principles of development that lead to high-quality software and isn't just thinking about the immediate experience of programming.

Question 4: 2 + 2 + 4 + 4 = 12 marks

Design By Contract is an approach to software design that is supported directly in some languages, and which can be applied in others, such as Java, through the use of extensions or other language features. In Design By Contract, the designer of a class tells the clients of the class what the class does and what it needs via the *specification* of the class.

(a) **Name two features of Java that allow a developer to write executable preconditions, and explain how these features can be used for this purpose. (2 marks)**

Assertions: allow the programmer to specify a logical expression which must be true at a certain point in the program. Assertions can be placed at the start of a method to ensure that its preconditions have been met. A possible disadvantage is that assertions need to be enabled via a parameter at the start-up of the Java runtime.

Exceptions: another way to write executable preconditions is to write some logical condition checking for the parameters (e.g. using an if…then structure) at the start of the method, and to throw an exception if a precondition is violated.

1 mark for each.

(b) **Why is it a good idea for a method not to try to cope with a violation of its preconditions, but simply to report the problem to the client that called it? (2 marks)**

Violation of preconditions is indicative of a bug, but not in the called method. Called method has no sensible way to know what the correct response to a bug higher up in the call stack is.

1 mark if they get it indicates a bug, 2 marks for a fuller explanation.

(c) **Give an example of a method for which it would be much easier to write a postcondition than to write the body of the method. Explain when and how writing such a postcondition would aid in software development. (4 marks)**

There are lots of possible examples. (e.g. a sort routine having a postcondition that array must be sorted, a routine that calculates the the square root of its input).

1 mark for giving a reasonable one

Potential points: before implementation as capturing requirements precisely. It becomes a specification that the implementer can use and test against. In testing and debugging, because it catches problems earlier and gives a clear indication of what the problem is.

3 marks for explanation of when and how.
Must mention at least two points for full marks. Explanation of each much be clear.
Max of 2 marks for a clear explanation of one instance where it is helpful.

(d) **Here is a simple Java method, in which the programmer would like to use a precondition. Choose a useful precondition for this method, explain what it is (in English), and write Java code to enforce the precondition (using standard Java, without using CoFoJa extensions). (4 marks)**

```
/**
 * Calculate the arithmetic mean of a List
 * The arithmetic mean is defined as the sum of the values
 * divided by the number of values in the list.
 * @param nums - the list
 * @return the mean value
 */
```

```
public static double average(List<Double> nums) {
	// FIXME: precondition code should go here
	double sum=0.0;
	for(Double num: nums) {
		sum += num.doubleValue();
	}
	return sum / (double) (nums.size());
}
```

Reasonable preconditions include:

- `nums` is non-null.
- `nums` is non-empty

2 marks for a clearly expressed precondition, 2 marks for Java code enforcing it.

Question 5: 10 marks

Below is some code from the act() method of a Droid in a version of the Star Wars game. This version implements some rather different behaviour for droids than the version in your Assignments.

Review this code for maintainability and extensibility.

For each design problem you identify:

a) **Describe the problem clearly.**
b) **In a sentence or two, explain potential negative consequences of the problem.**
c) **Outline a potential fix for each design problem. You do not need to write actual code, but explain your fix in enough detail to make clear how your fix would address the problem. Write up to a few sentences.**

Hint: consider "code smells"…

```
public void act() {

	// Code for other actions (omitted for brevity)

	// get a non-actor entity at this location, if any
	SWEntity potentiallyTakeable= getLocalEntity(this);
	if (potentiallyTakeable != null) {
	// if it's R2-D2
		if (this.model == 1) {
			SWAffordance takeaffordance =
			  getTakeAffordance(potentiallyTakeable);
			if(takeaffordance != null) {
				scheduler.schedule(takeable, this, 1);
				say("Bleepy bleep bloop");

			}
	 // if it's  C-3PO
		} else if (this.model == 2) {
			SWAffordance takeaffordance =
			 getTakeAffordance(potentiallyTakeable);
			if(takeaffordance != null) {
				scheduler.schedule(takeable, this, 1);
				say("I hope this doesn't get me into trouble.");

			}
		} else {
			say("Boss, there's something here!");
		}
	} else {
		// Do other things (omitted for brevity)
	}
}
	// Get the Take affordance for this entity, if any.
private SWAffordance getTakeAffordance(SWEntity entity) {
	for(SWAfforance aff : entity.getAffordances()) {
		if (aff instanceof Take) {
			return aff;
		}
	}
	return null;

}
```

Key problems with this code:

1. Duplicated code for R2-D2 and C-3PO.
2. Switching on type information (using embedded “magic number” literals!)
3. Use of instanceof to identify Affordances (less critical).

Accept other reasonable answers. Maximum 8 marks if they don’t get *both* 1 and 2 (or equally significant problems)

For each problem:

2 marks for a clear description of problem.
1 mark for description of consequences.
2 marks for quality of solution.

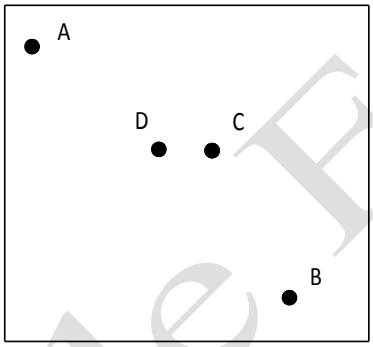
Question 6: 5 + 3 + 5 + 7 = 20 marks

Remember: answer EITHER Question 6 or Question 7, not both!

You're working on a system to monitor pairs of primary school students working on a geography assignment on the grounds of their school, which is too large for a teacher to keep all the students in sight at all times.

The students are supposed to work on the assignment in assigned pairs. Each student carries a smartphone with an app that periodically reports the position of that student to a monitoring system. The positions are reported as two numbers, x and y, which represent the offset in an easterly and northerly direction, respectively, in metres from an origin point in the center of the school. For instance, a student at the point (20.5, -5.3), would be located at a point 20.5 metres east and 5.3 metres south of the origin point.

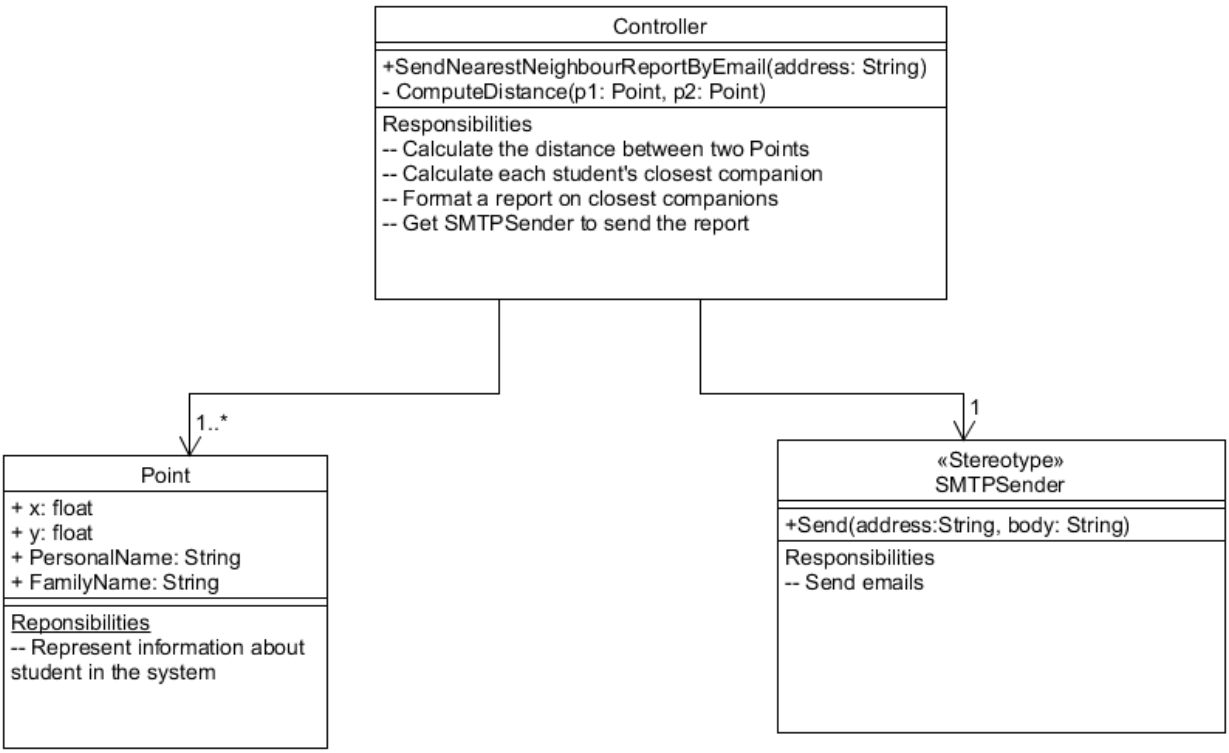
To help teachers verify that they are working in their pairs, the system should regularly email the teacher a report listing each student and the student closest to them at that moment. For instance, if there are four students, A, B, C, and D, located as follows:



The emailed report should read:

A is closest to D
B is closest to C
C is closest to D
D is closest to C

You and a colleague are working on a design for the part of the monitoring system that emails the report. They send you the following class diagram, annotated with responsibilities for classes.



Correction: Please ignore the <<Stereotype>> stereotype in SMTPSender.

a) Based on the description and the class diagram above, draw a sequence diagram for the "email nearest neighbour report". (5 marks)

UML syntax
3 marks: correct
2 marks: minor errors (e.g. incorrect "object: Class" labels for lifelines) that do not significantly impeded comprehension.
1 mark: significant errors that impede comprehension, or lots of minor errors such that the diagram is significantly harder to understand than it should be.
0 marks: bad enough that it is impossible to understand the functioning of the system.

Semantics
2 marks: plausibly correct semantics (that is, computeDistance repeatedly requests x and y from points to find nearest neighbour, then requests details from closest neighbour, and then a send request is passed to SMTP sender.
1 - in the ballpark.
0 - major errors.

b) How easy would it be to extend this system to support sending the report by other means? In a few sentences, explain the features of the design that make it easy or hard, and why they would do so. (3 marks)

Some plausible issues:

Sending is in a separate class – makes it somewhat easier

Controller directly invokes SMTPSender - dependency inversion principle not applied.

Interface of SMTPSender specifies address as a string rather than an abstract address class, makes it hard to modify if a non-string address is necessary.

Accept any other reasonable issues identified.

3 marks: expresses at least a couple of issues using appropriate terminology and correctly identifies their effect on modifiability.
2 marks: only identifies one issue but talks about it well, or identifies more than one but problems with language or analysis.
1 mark: makes some point relevant to modifiability in this instance.

c) Analyse the rest of the design in terms of the design principles we discussed in FIT2099. Write about half a page. (5 marks)

Potential issues:

Name of Point class (really bad)
public attributes in Point (really bad)
Feature envy in computeDistance

Accept all reasonable issues raised - list above is not exhaustive. Accept alternative phrasings/perspective of above points (e.g. Point contains info that should go in separate student class)

5 marks: identify several issues including both really bad ones and other reasonable ones, and discuss them clearly in terms of concepts such as connascence/dependency, cohesion, etc. Cannot get 5 if lots of irrelevant "hand-waving" is included.
4 marks: As above, but with minor issues with terminology, fewer issues, or some "hand-waving".
3: identify a couple of issues and talk about them in a coherent way, if not necessarily using professional terminology.
2: identify at least one issue and talk about it clearly, or talk about a couple of issues badly.
1: saying something vaguely plausible about the design.

d) Suggest ways to improve the design, and explain why your changes are an improvement. Write a maximum of one page. (7 marks)

Suggested improvements
4 marks: at least two suggestions that a) address actual problems, b) are well communicated, and c) reasonably good ways of addressing the problem. NB: renaming point can't be one of the two for full marks.
2 marks: one suggestion that addresses a, b, and c above, or two that fall down on one or more of these aspects.

Explanation
3 marks: clearly explain nature of two or more improvements using appropriate terminology.
2 marks: as above for one improvement, or terminology has problems, but still conveys some sense of why it's better.
1 mark: gave some vaguely plausible reason.

Question 7: 20 marks

Remember: answer EITHER Question 6 or Question 7, not both!

You are working on surgery management software for an animal hospital that treats cats and dogs.

A key part of the system is managing the wards, where animals stay before and after their treatment.

The management system will keep a record of animals, including their name, classification, and owner. In the case of dogs, small and large dogs are treated as separate classifications because of their different boarding requirements, and cats are classified separately because of their different responses to drugs.

In the ward, there are many "rooms" (cages) that animals can be placed in. However, because animals are very diverse, there are several different types of room.

There are small, medium, and large standard rooms. A cat can be kept in any size room. Small dogs can be kept in a medium or large room. Large dogs can only be kept in a large room.

Rooms are reserved for a period of a single day. If a longer stay is required, the users of the system will do another booking.

You can assume the existence of a "UserInterface" class that passes on user requests, and which is responsible for reporting results back to the user. Dates in the system are stored using the LocalDate class in the java.time system library, so you don't have to worry about the details of date representation.

There are two scenarios the initial version of the ward management system must support:

1. Find and reserve a room for an animal already registered in the system for a specified day.
2. Locate an owner's animal(s) currently staying in the wards.

Devise a design for this part of the system, and draw a class diagram and two sequence diagrams – one for each scenario – to illustrate your design. You must explicitly list high-level responsibilities (in words) for each class in your design. If you wish to add other information, to help explain your design, you may do so.

You must also explain the reasoning behind your design choices – take about half a page for this.

Diagram syntax
8 marks: CD and sequence diagram syntax is correct
6 marks: minor errors, e.g. incorrect "object : Class" labels for SD lifelines or wrong (but still comprehensible) arrow types in CD
4 marks: one diagram correct, one missing; or major problems that impede comprehension such as missing message labels in SD
2 marks: student has drawn some boxes and maybe some lines

Diagram semantics
6 marks: clearly supports required scenario; sequence of messages in SD makes sense; messages are being sent to classes that might reasonably be able to act on them; message passing in SD is reflected in associations in CD;

attention paid to design principles.
4 marks: reasonable attempt but CD and SD inconsistent (or SD missing, etc.). Or: some component of required functionality isn't supported in the model but could easily be added.
2 marks: seems to be modelling something relevant to the question

Explanatory paragraph
6 marks: Paragraph is consistent with the design expressed in the diagrams, makes sense, and addresses design principles seen in class (e.g. DRY, coupling/connascence, code/design smells, abstraction, etc.) A likely example here is use of inheritance to avoid repeated code and switch logic.
4 marks: An explanation that makes sense but does not use design-related terminology correctly. OR: a clear description of functionality that does not adequately address design concepts. OR: something that is mostly good but misses a major point/overlooks an obvious flaw in the design.
2 marks: some useful comment on the design.

END OF SAMPLE EXAM