SPARE PAGE FOR EXTRA ANSWERS
Student ID:

2016

Q1 Software Engineering

1.Not every code development counts as "software engineering". List three aspects that are characteristic of true "software engineering" projects.(-)

Software Engineering is programming under at least one of these two conditions:

1. more than one person builds and uses the program
2. more than one version of the program is created
3. Software Engineering is the practical application of scientific rationale on the design and the construction of program systems

2.There are four factors of software development of which the client may prioritize three, By improving what property of the software development process can one gain an advantage regarding all four factors? Briefly point out why this is the case for two of these factors.[6 marks] .(-)

1. Improving productivity of software development can one gain an advantage regarding all four factors (time, quality, functionality, cost)
2. productivity is a proportion that how well the project quality and functionality does, under constant cost and time. In addition, the four factors limit each other in a software project.
3. Therefore, we need to find the balance of the four factors, which is mostly helpful to improve productivity

3.Is there a difference between "building the right (desired) system" and "building the system right (properly)"? Briefly explain your answer using corresponding software engineering terminology. [6 marks](2) (-)

Aspects of Reliability
Correctness is defined as the conformance of the system to its specification
"Are we building the right system?"
Robustness is defined as the ability of a system to (continue to) perform despite being forced to operate outside specified parameters
"Are we building the system right?"

4.Briefly explain what the term "maintenance" means in software engineering and what typical activities are performed in the maintenance phase.[6 marks] (-)

Maintainability
How easy or hard is it to detect and correct errors in a system? How easy or hard is it to change the system?
» fix shortcomings » extend functionality » address changing
» error correction » change of specification » change of construction

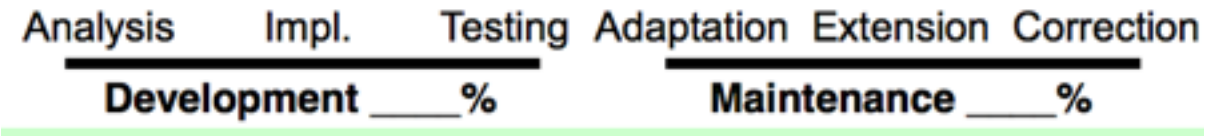
(a) 3Briefly discuss the meaning and significance of "maintenance" in software engineering.

[4 marks] (-)

How easy or hard is it to detect and correct errors in a system? How easy or hard is it to change the system?
Issues back then programming in the large, system structure, error propagation and change avalanches
» error correction („right“, ca. 20%)
» change of construction („better“, ca. 20%)
» chIn computing, an interface is a shared boundary across which two separate components of a computer system exchange information.ange of specification („different“, ca. 60%) 36-64

1. Maintenance is the continuous updating of a system to fix shortcomings, extend functionality and address ongoing changing requirements of the client.

2. Maintenance includes any change to the software after the product has been released and
3. it is estimated 60%-80% of a software's lifetime is spent in maintenance.
4. Maintenance is important in software engineering because software projects are rarely released without bugs which need to be fixed later.



(b) What are symptoms of a software system that is hard to maintain? [5 marks(-)

1. software systems are unreliable.
2. user requirements are not fulfilled
3. Software system are most costly and required more time to build from planned.

4 Moreover, reliability and correctness are rarely perfect.

5 software system belong to most complex and difficult to handle

(c) Fewer interfaces are considered better than many interfaces in software engineering. Briefly explain why this does not imply that zero interfaces are optimal. [5 marks] (-)

1. In computing, an interface is a shared boundary across which two separate components of a computer system exchange information.
2. Software interfaces provide access to computer resources (such as memory, CPU, storage, etc.) of the underlying computer system.
3. So, interface is necessary in software engineering, i.e. interface cannot be zero. If it is, we do not have access to computer resources.

(d) [4 marks] What are the four factors in software development of which the client may prioritize a maximum of three? (-)

(time, quality, functionality, cost)

[2 marks] Why do you think 40 years of software engineering practice and research have not been able to lead software development out of the above mentioned crisis? (-)

Development of software engineering is rather young and continually developing

It is also hard to do empirical studies, i.e. repeatability is a problem.

(d) [10 marks] Describe the steps a software engineer can take to minimise the possibility of software project failing using at least two of the examples discussed by at least two different group project presentations in the class.

1. Review and testing are an essential practice when writing code; especially high-risk code.
2. Version control is a software practice used by teams of software developers to ensure working pieces of code aren't lost or overwritten while someone else is editing the code
3. Automatic deployment is an effective method when updating elaborate systems with new functions.
4. Error prevention, traceability and alerts are useful tools to mitigate the damage of a fault as quickly as possible. Knights Trading Loss #Toyota ETCS

(b)The maintainability of a component correlates with the size of its interface. Briefly describe this correlation and mention two technical properties that components with the desirable interface size will typically exhibit. (2)

Interfaces » few, small, and explicit
Every module should communicate with as few others as possible.

(c) If a software system is hard to change because any change may break the system in some way, what is the system suffering from and what system property could address the problem? [4 marks] (2)

A software system has a high fragility if the software is easily broken, because just make small change of the program.
Modular Continuity is to avoid “change avalanche”

(d) Briefly discuss the potential benefits and dangers involved in reusing software components. (2)

Advantages:-
1. It can reduce the overall cost of software development as compared to other model.
2. It can reduce the risk.

3. It can save the time of software development. b/c testing of component is minimize.
4. Faster delivery of software.

Disadvantages:-

1. Organization using the reusable component, are not able to control the new version of component, this may lead to lost control over the system evolution.

2. once the reused piece of code contains bugs, the software system will contain unexpected errors because of the reused piece of code

Q2 Design Principles

1.Why are classes with low coupling desirable? Tick only one box. [2 marks]

□They increase cohesion.□They decrease cohesion. □ They increase rigidity. □ They decrease rigidity.

Rigidity It is hard to change because every change affects too many other parts of the system

2.Assume module A has five components with six connections and module **B** has five components with 8 connections. Which module is more likely to contain components better suited for reuse? Briefly explain your answer? [4 marks]

1. module A is more likely to contain components better suited for reuse. A has less connections than B, and it has fewer the interface count.
2. if a system is composed of n modules, the number of connections should remain much closer to the minimum, n–1 (C), than to the maximum, n (n – 1) /2 (B).
3. components in module A communication less than components in module B. Therefore, the components in module A has lightweight responsibility for each other module A is better suited for reused.

3.Which - "layers" or "partitions" - are useful to address modular continuity? Explain your answer.[6 marks] .(2)

1. Layering »strict layers only interact with adjacent layers, stops change avalanches, separates concerns
2. Partitioning means a module can be accessed by multiple other modules.
3. In this case, a small change in one module can affect the other modules it communicates and interacts with.
4. The purpose of continuity is to avoid "change avalanche"

5. Layering »strict layers only interact with adjacent layers, stops change avalanches, separates concerns This reduces the number of modules that could be affected by the changes, and thereby satisfies/aids in modular continuity.

4.Assume an implementation has been verified to be correct with respect to the specification. What other properties of the implementation may the customer be interested in?

1. The support and usability available for the implementation, including the forms that might take (documentation, call centers, etc.);

2. the ongoing costs for using and supporting the implementation;
3. information on any external resources used by the implementation;

5.Briefly list the pros and cons of pre-conditions and post-conditions for the purpose of achieving modular protection. [6 marks]

Protection

Pros:

1. An abnormal condition occurring at run time in a module will remain limited to that module, or at worst will only propagate to a few neighboring modules.
2. Preconditions is used for checking the validity of input data before it is used to ensure that the module will only run when the program is in a suitable state to do so.
3. Postconditions assert that the output of a module is as expected. This prevents errors propagating throughout several modules.

Cons:

1. Excessive checking of pre and post conditions can lead to a reduction in performance speed.
2. Undisciplined exceptions - unless caught at appropriate places can create and propagate errors without bounds.

(e) [4 marks] Briefly discuss which of ”using pre-conditions” or “using post-conditions” are a better means to achieve “modular protection”.

pre-conditions is a better approach to achieving modular protection because it checking the validity of the input data before it use.

1. An abnormal condition occurring at run time in a module will remain confined to that module, or at worst will only propagate to a few neighboring modules.
2. Preconditions is used for checking the validity of input data before it is used to ensure that the module will only run when the program is in a suitable state to do so.
3. Postconditions assert that the output of a module is as expected. This prevents errors propagating throughout several modules.

6. Briefly explain the relationship between coupling and information hiding.(2)

1. information hiding, make attributes and operations private
2. limit the association traversal
3. decrease coupling -> program to an interface, not an implementation

(c)Why do internal criteria for a software system matter, if the client is only concerned with external criteria?

The internal criteria of a software system determine how easy the software is to understand and maintain.

External Criteria is about what clients expect.

But not only external factors really matter. But the key to achieving them are the internal ones

For example, if the client would like to change or add a new part to the software system, the developers must be able to go back to find the solution which is under the internal criteria.

(d) What causes a software system to exhibit "rigidity"?

A software system has a high rigidity if it is hard to change because every change affects too many other parts of the system.
High rigidity is caused by lots of connections between components, which means the components have high dependencies on each other.

(e) What causes a software system to exhibit "fragility"?

A software system has a high fragility if the software is easily broken, because just make small change of the program.
In other words, a piece of software has lots of dependencies across components (high rigidity), this means changing a small part of one component may have drastic effects on other components that depend on it. Strong coupling

(a) Why are classes with high cohesion desirable?

1. High cohesion: a class has lightweight responsibilities in one area and collaborates with other classes to fulfil tasks
2. Classes with high cohesion are much easier to understand
3. Changes to a class with high cohesion only affect one (the intended) aspect

(b) Which of the five modularity requirements that were discussed in lectures can help to improve continuity? Briefly explain your answer.??????????

Continuity » avoiding the "change avalanche"

(c) If in the software architectures that it yields, a small change in a problem specification will trigger a change of just one module, or a small number of module

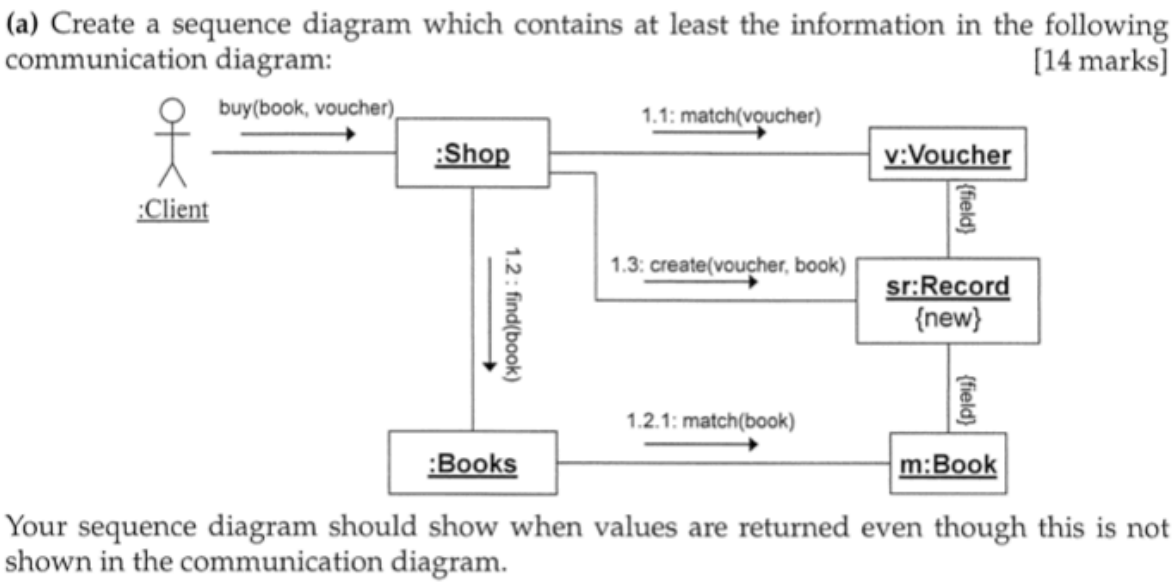
1. Decomposability
2. composability
3. Understadability
4. Continuity
5. Protection

Briefly explain why even a correct implementation does not guarantee full customer satisfaction and why this circumstance is not used to change the traditional development process.

- Correct implementation means achieve internal criteria of software system,
- customer satisfaction means achieve external criteria of software system.
- Therefore, internal criteria could only guarantee the correct of the software system. but it is unlike to fully satisfy all customer expectation of the software system.

- Traditional development process is characterized by a sequential series of steps like requirement definition, planning, building, testing and deployment.
- This means if internal criteria are met, every steps of customer expectation is satisfied.
- In this case, correct implementation can guarantee full customer satisfaction

Q3 Interaction Diagrams

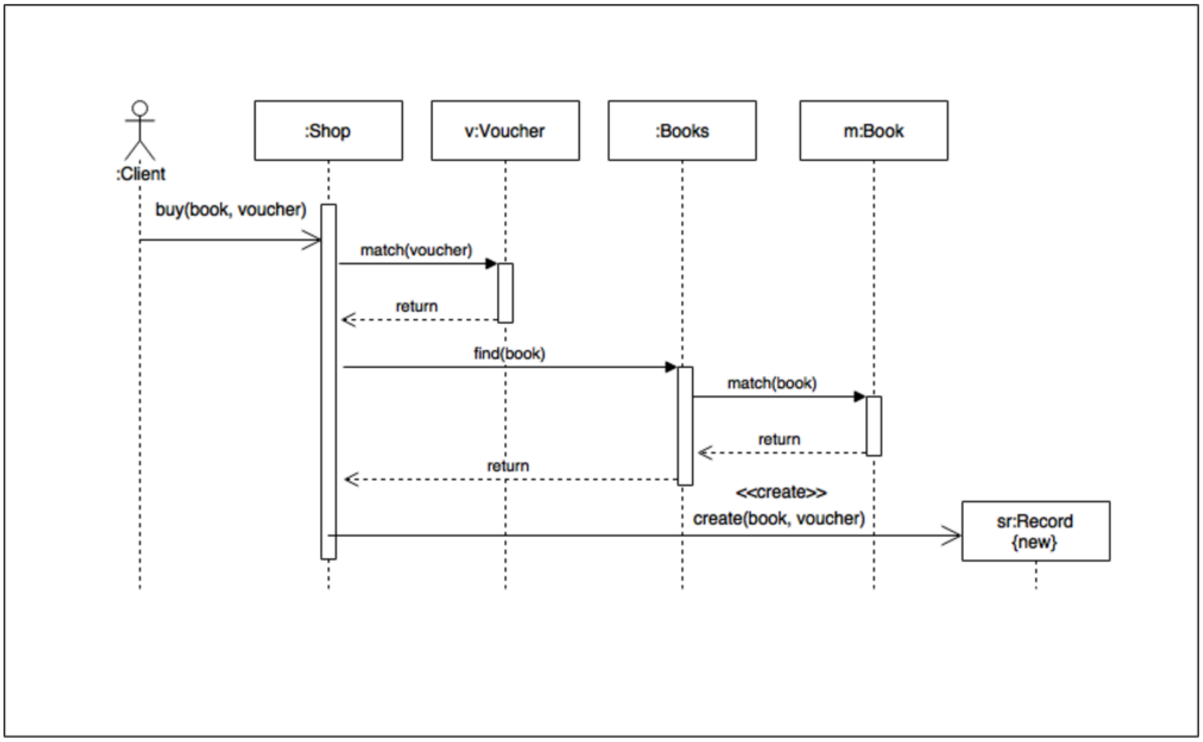


(a) Create a sequence diagram which contains at least the information in the following communication diagram: [14 marks]

Your sequence diagram should show when values are returned even though this is not shown in the communication diagram.

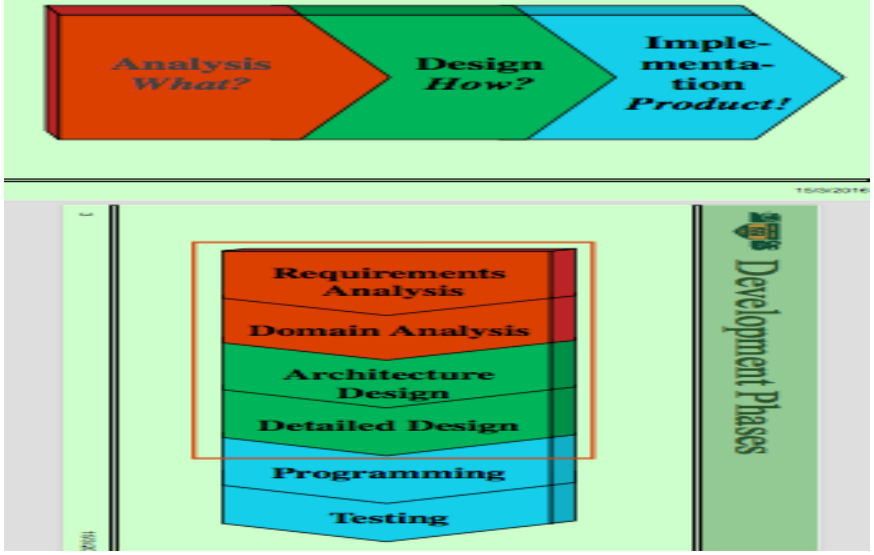
(2) <create>

-- (2)



---

2. In which development phases can one use interaction diagrams? Briefly describe the purpose of interaction diagrams for each phase you name. [6 marks]

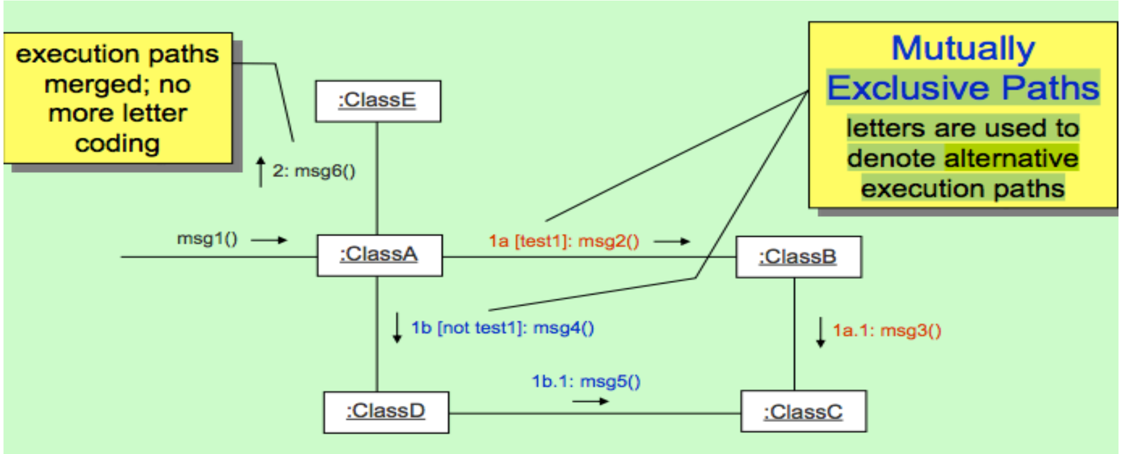


Design phases

during analysis, to improve individual or group understanding of inter-object behaviour
during design, to precisely (but typically partially) describe inter-object/process communication
during testing, the traces can be compared with those described in the earlier phases

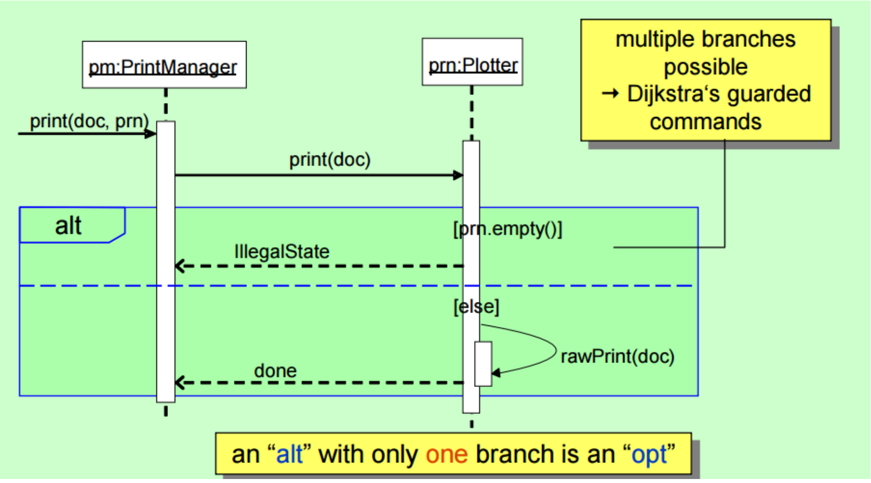
(a) In what way can you capture alternative execution paths in a communication diagram?

[2 marks] (3) Conditional Paths -> Mutually Exclusive Paths letters are used to denote alternative execution paths, and include message guards e.g. [valid], [invalid] to specify which branch should be followed and when.

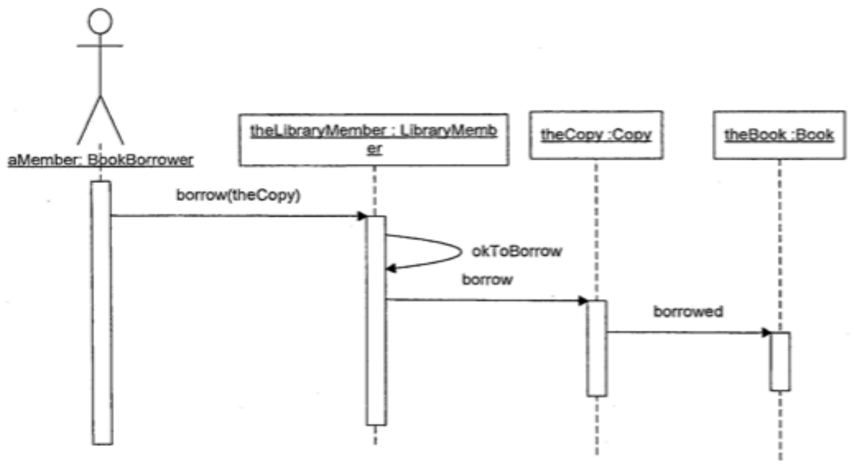


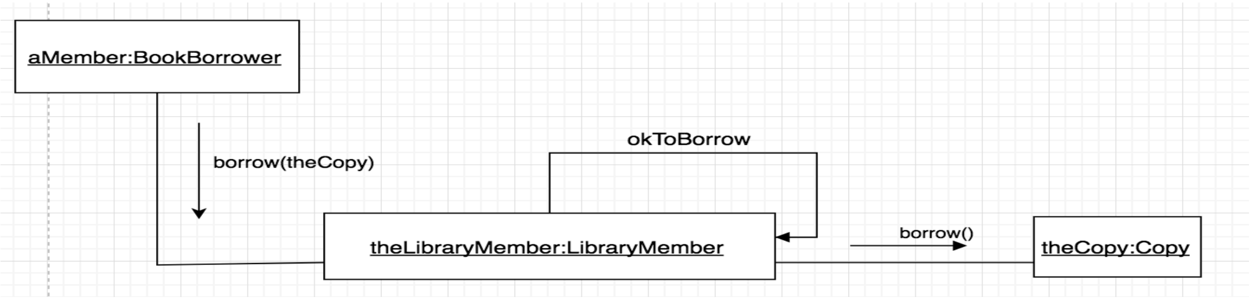
(a) In what way can you capture alternative execution paths in a sequence diagram? (2)

By using alt boxes which specify which split actions depending on certain guards that are specified. An alternate to alt boxes (pun intended) is when an action is optional, in which case, an opt box would be used.



(a) [10 marks] Create a **communication diagram** which contains at least the information in the following sequence diagram:





b) [5 marks] Compare and contrast sequence diagrams and communication diagrams. Discuss the advantages and disadvantages of both kinds of diagrams.

1. Sequence diagrams -> Strength: clearly show ordering of messages
2. Weakness: don't show links become very wide
3. Communication diagrams: -> Strength: show links & use space economically
4. Weakness: difficult to see message sequence

(b) Briefly explain how you could use interaction diagrams in both implementation and testing phases respectively (2)

(c) during analysis, to improve individual or group understanding of inter-object behaviour

(d) during design, to precisely (but typically partially) describe inter-object/process communication

(e) during testing, the traces can be compared with those described in the earlier phases

(c) Briefly explain for which purpose you would choose a . Sequence Diagram over a communication diagram and vice versa. 5 marks(2)

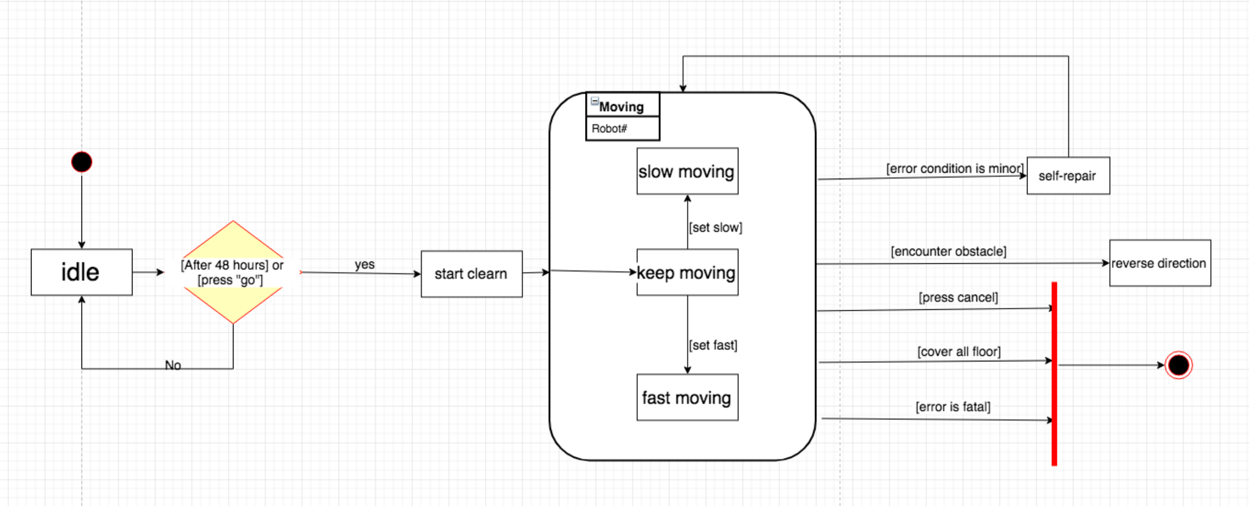
1. Sequence diagrams are for showing the order of system actions over time. I would choose a sequence diagram for showing the order of events and return sequences.
2. Communication diagrams are for showing the links between components in a system and how they communicate with each other. I would choose a communication diagram if I wanted to show a spatial perspective of the whole system.

Q4 State Diagrams

(a) Create a UML state diagram that describes the behaviour of a cleaning robot. Initially, the robot is idle. After 48 hours of inactivity or when the user presses the "go" button on a remote control, the robot starts cleaning the floor and normally keeps moving forward until it has covered all of the floor or the user presses the "cancel" button. The user can also set the speed to "fast" (noisier operation) or "slow" (quieter operation) at any time. If the robot encounters an obstacle, it turns by 15 degrees and then reverses direction - i.e., moves backwards if it has been moving forwards and moves forwards if it has been moving backwards. At any point in time an "error" condition may occur. If the "nature" of the error is "minor" the robot will self-repair and then continue. If the "nature" of the error is "fatal", the robot will terminate all activity immediately and indefinitely. (2)

Marks are awarded for using advanced notation that goes beyond using only states and transitions.

SPARE PAGE FOR EXTRA ANSWERS



Create a UML state diagram that describes the states and events of a phone with the following behaviour: (3)

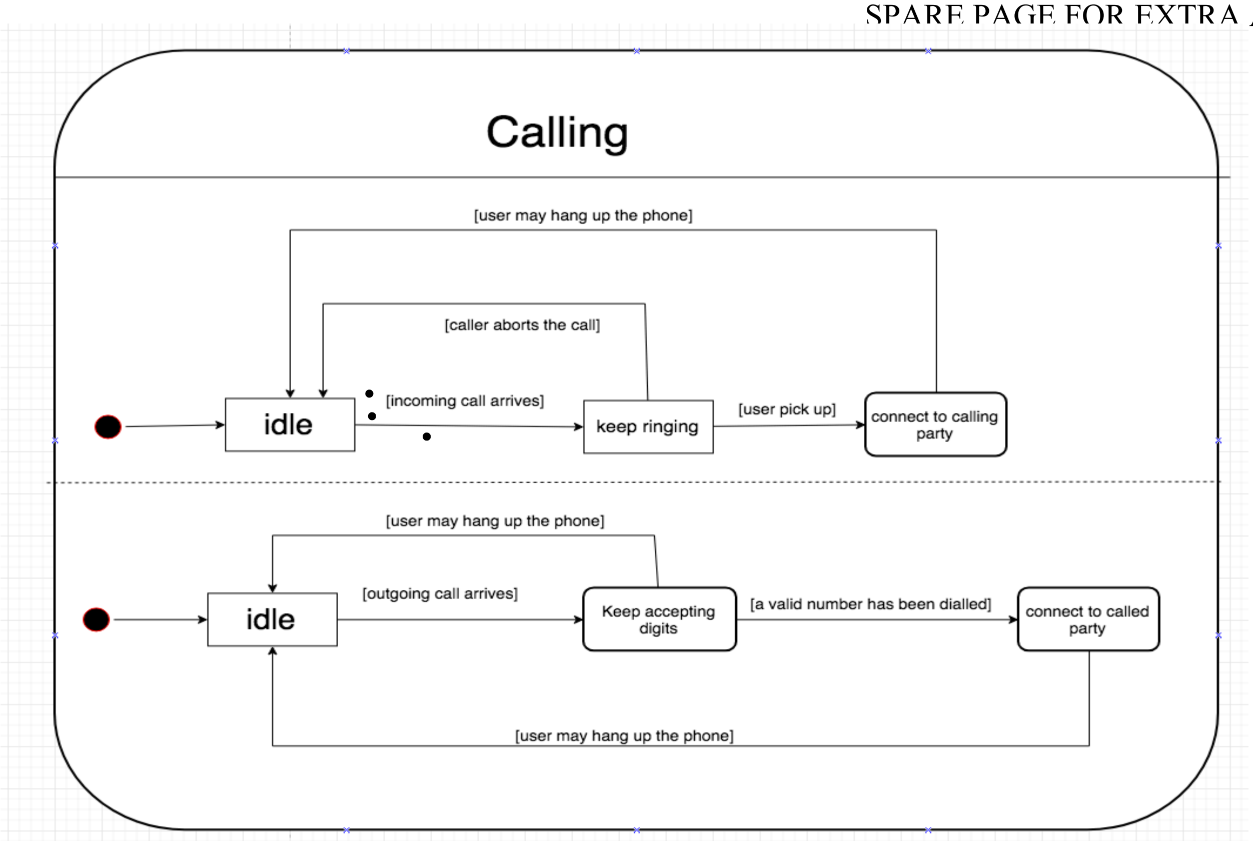
Initially, the phone is idle. When an incoming call arrives, it keeps ringing until the user picks up or the caller aborts the call. In the former case the phone is connected to the calling party, while in the latter case it becomes idle again.

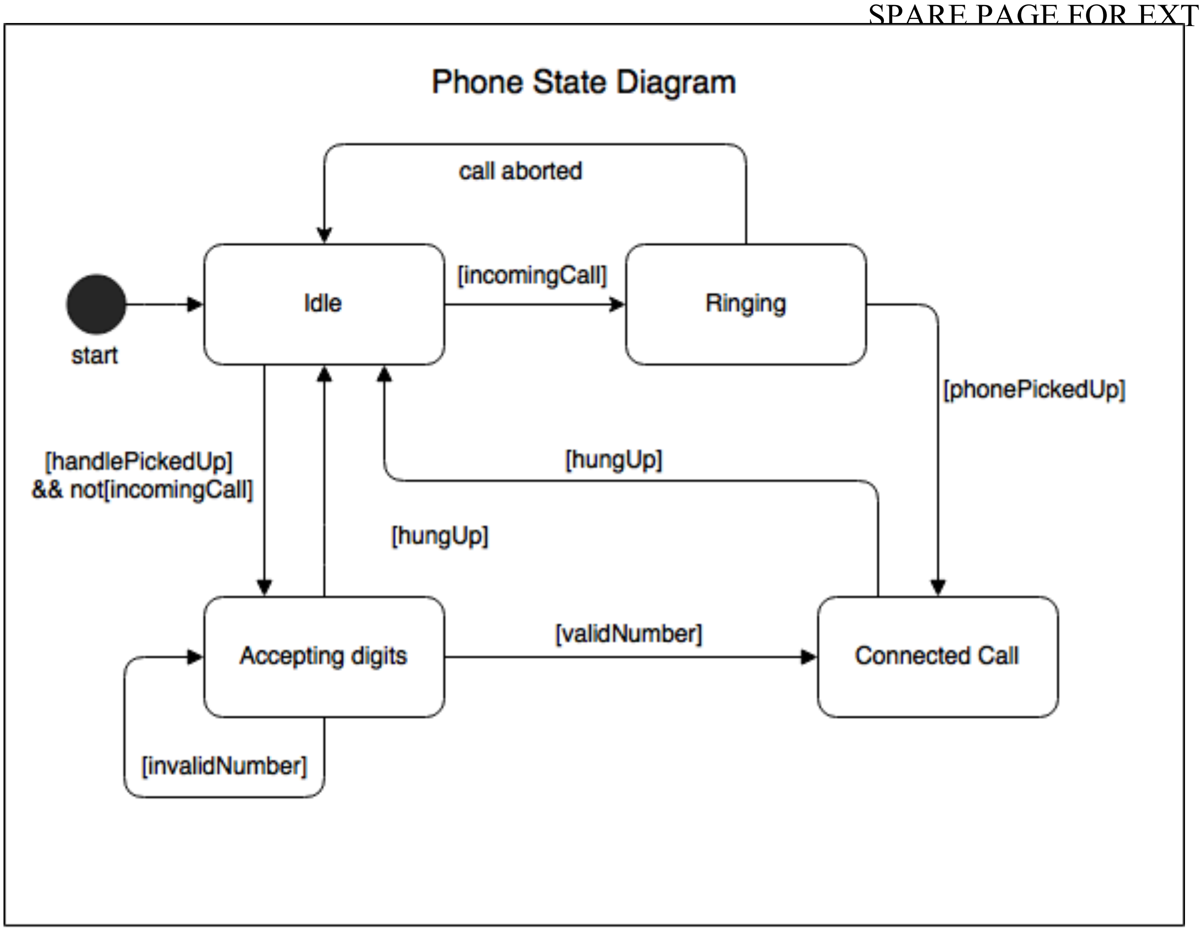
For an outgoing call, when the user picks up the handle, the phone keeps accepting digits until a valid number has been dialled. When this happens, the phone becomes connected to the called party.

At any point during the dialling or while being connected the user may hang up the phone, causing it to become idle again.

Note: Marks are also awarded for the appropriate use of advanced notation.

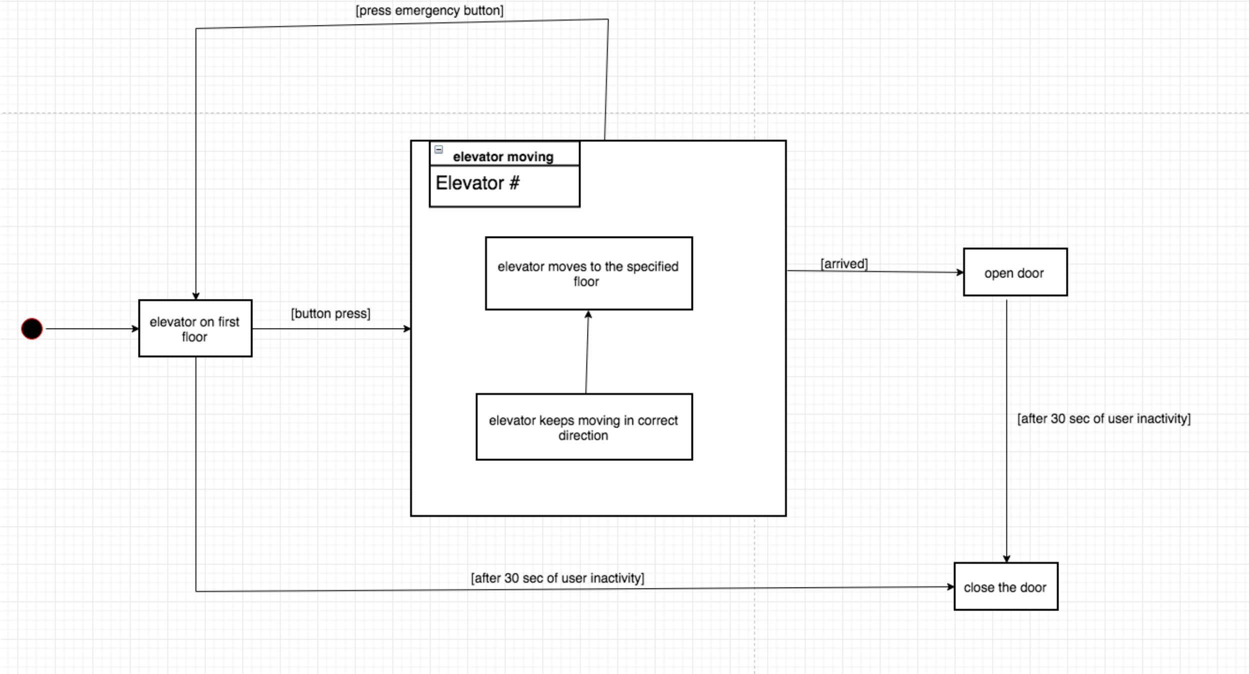
SPARE PAGE FOR EXTRA ANSWERS





(a) [15 marks] Create a UML state diagram that describes the behavior of an elevator. Initially, the elevator waits on the first floor. When a “button press” event occurs, the elevator moves to the floor number specified by the event. It is important for the elevator to move in the correct direction (up or down). When the elevator reaches the target floor, an “arrived” event is generated. The elevator should then open its doors. The elevator should close the door before it moves and return to the first floor after 30 seconds of user inactivity (i.e., no button presses). At any point during the elevator’s operation, it is possible to press the “emergency button” which will cause the elevator to return to the first floor.

Marks are awarded for the appropriate use of advanced notation.


[press emergency button]
elevator moving
Elevator #
elevator moves to the specified floor
[arrived]
open door
elevator on first floor
[button press]
elevator keeps moving in correct direction
[after 30 sec of user inactivity]
[after 30 sec of user inactivity]
close the door

(b) [5 marks] Briefly explain why substates, i.e., the ability to use concurrent lanes each specifying reactive behaviour that contributes to an overall combined behaviour, can be used to reduce the complexity of state diagrams. (2)

1. Concurrent Substates » parallel execution » reduces complexity by factorising multiplied states and corresponding transitions
2. When an object needs to accomplish 2 or more functions/actions at the same time(eg. Vacuum cleaner move and clean) And these need to be finished at the same time.

(b) Briefly explain what superstates are typically used for and why they are considered to be an important feature of state diagrams. （3）[6 marks]

1. Superstates » combination of several substates into one superstate » reduces complexity by hiding aggregated states and multiplied transitions
2. A super-state is used when many transition lead to a certain state.
3. Instead of showing all of the transitions from each state to the redundant state, a super-state can be used to show that all of the states inside of the super-state can transition to the redundant state.
4. This helps make the state diagram easier to read.

Name and very briefly describe three UML language/notation features designed to deal with complexity.fan

Complexity Reduction Through...

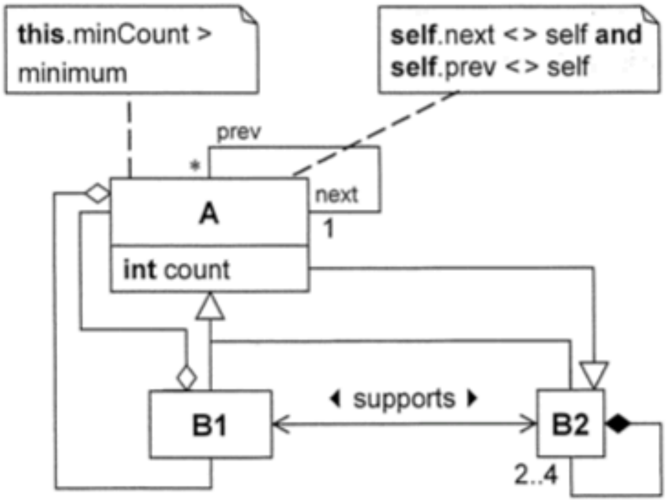
1. Superstates » combination of several substates into one superstate » reduces complexity by hiding --- aggregated states and multiplied transitions

2. Concurrent Machines » parallel execution » reduces complexity by factorizing multiplied states and corresponding transitions

3. Aggregation: Implies that an object is a part of another, but one can exist without the other. Eg. a car has four wheels, but a wheel can exist without a car and a car without a wheel.

Q5 Modelling

The following class diagram contains a number of errors/problems.



(a) List four errors/problems. For each, i) identify it with a numbered circle in the diagram, ii) briefly explain it, and iii) describe the least invasive way to correct it. [12 marks]

1)

2)

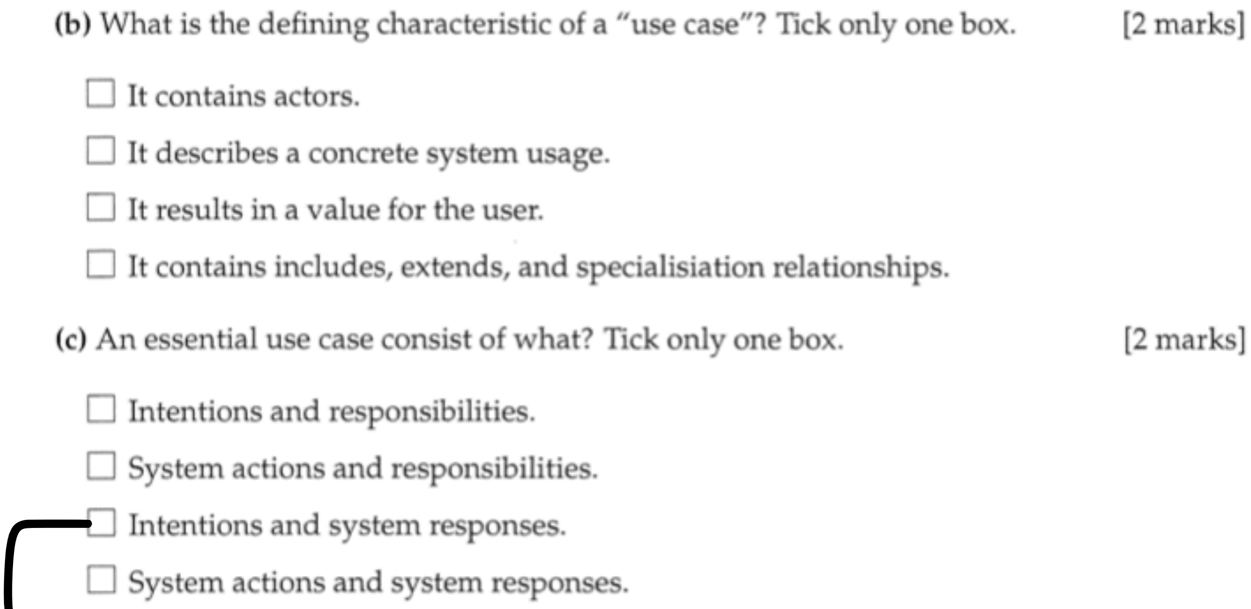
SPARE PAGE FOR EXTRA ANSWERS
Student ID:

1)The attribute of class A should be changed into count: int because the correct format should be visibility - name of variable – type of variable

2)delete one of aggregation between class A and B1 because aggregation should be anti-symmetric and transitive

3)change multiplicities of the composition of B2 into one.
because composition is used that parts cannot exist without the whole. B2 has a composition to itself, implying that it requires an instance of itself to exist, which is not possible without creating a paradox.

4)delete the generalisation from A to B2. The reason is that if A is parent class of B2, therefore B2 cannot be parent class of A



(b) What is the defining characteristic of a "use case"? Tick only one box. [2 marks]

□ It contains actors.

□ It describes a concrete system usage.

□ It results in a value for the user.

□ It contains includes, extends, and specialisiation relationships.

(c) An essential use case consist of what? Tick only one box. [2 marks]

□ Intentions and responsibilities.

□ System actions and responsibilities.

□ Intentions and system responses.

□ System actions and system responses.

b-> it contains includes, extends and specialisation relationships.
c-> ~~system action~~ and system responses
<include> → a common subpart is used within a containing use case
<extend> → a variant or exceptional situation extends the normal case

(b) A colleague asks you what the direction of the inheritance relationship between the concepts "Rectangle" and "Square" should be. Advise your colleague of three alternative options and briefly explain the rationale for each option. [8 marks] (3)

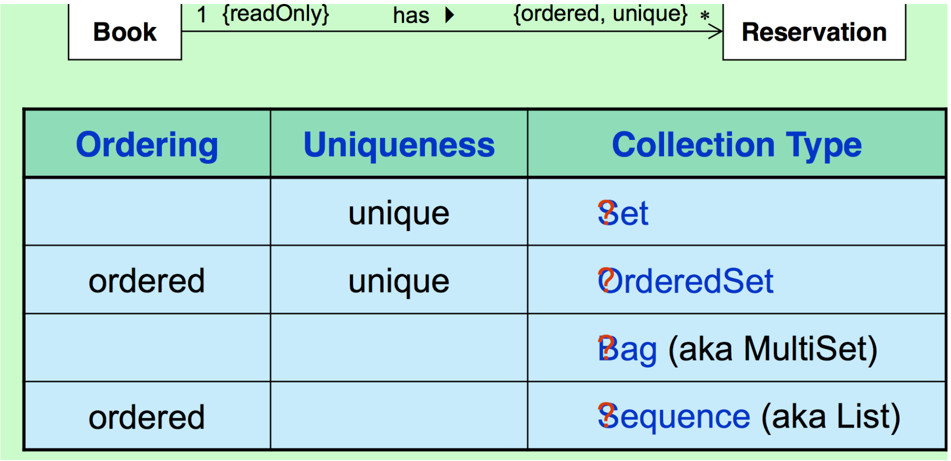
Option 1: A circle inherits the properties of an ellipse, however the circle must have a constant radius and the formula Area = π * r^2 must remain true

Option 2: An ellipse inherits the properties of a circle, however the ellipse will have two radius measurements and the formula Area = π * a * b must remain true

Option 3: Both of the shapes inherit from a shape interface. This is sensible because both circle and ellipse are shapes and all shapes will require the same methods, however the methods will be written differently. (e.g. getArea(), move(x, y), setColor(Color))

[4 marks] List and discuss the properties that make a good class diagram.

1. **Visibility of attribute and method in a class The permitted values**
2. **Multiplicity -> May have multiple associations between two classes if there are different relationships.**
3. Collection Types
4. Generalisation is used for classification (System understanding), code reuse(subclassing)
5. Aggregation--> part object can exist without the whole (anti-symmetric, transitive)
6. Composition –> part object cannot exist without the whole (transitive)

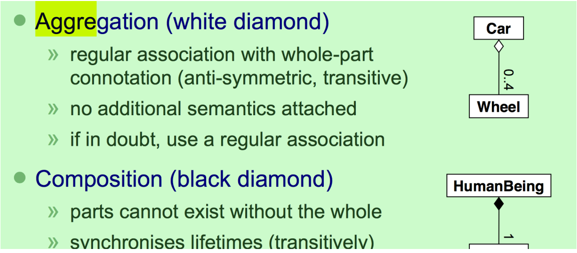

Book
1 {readOnly}
has ▸
{ordered, unique} *
Reservation

Ordering	Uniqueness	Collection Type
	unique	Set
ordered	unique	OrderedSet
		Bag (aka MultiSet)
ordered		Sequence (aka List)

[2 marks] Explain why class diagrams are already created in the **analysis stage** of software engineering.

1. All the possible use cases aka requirements of the system
2. class diagram should the core idea about the structure of the project, such as what the relation between each class.

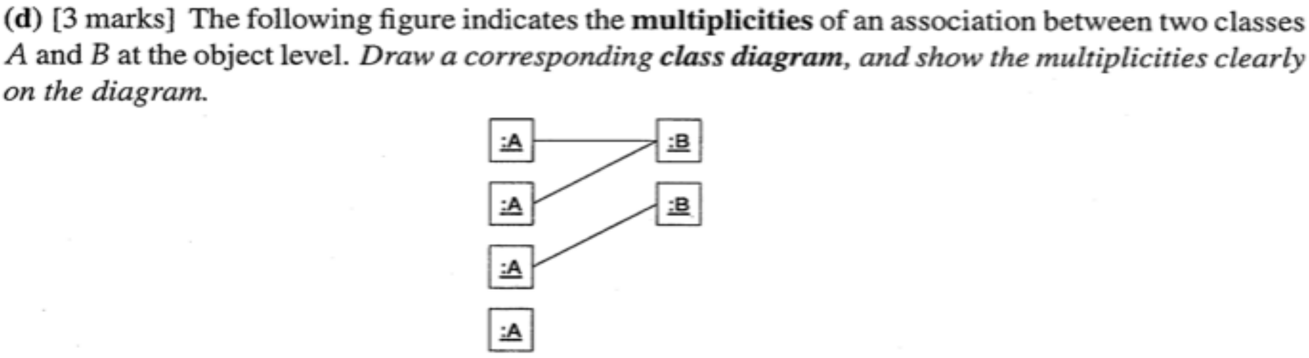
[6 marks] **Aggregation** and **composition** are two kinds of **association** in class diagrams. *Give an example for each of them.* Discuss the differences between aggregation and composition. *Use your examples to illustrate the differences.*



- Aggregation (white diamond)
  - » regular association with whole-part connotation (anti-symmetric, transitive)
  - » no additional semantics attached
  - » if in doubt, use a regular association
- Composition (black diamond)
  - » parts cannot exist without the whole
  - » synchronises lifetimes (transitively)

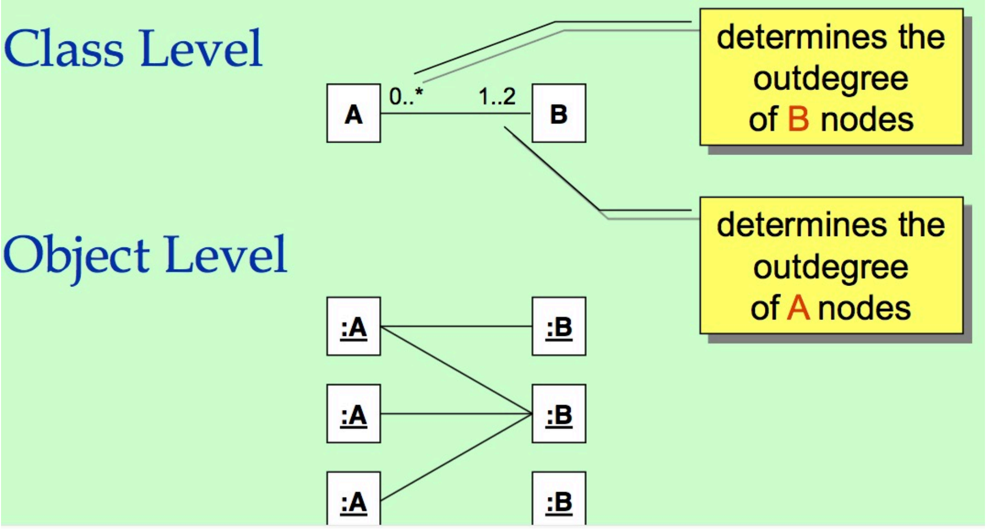
Car
0..4
Wheel
HumanBeing
1

SPARE PAGE FOR EXTRA ANSWERS
Student ID:



(d) [3 marks] The following figure indicates the **multiplicities** of an association between two classes *A* and *B* at the object level. *Draw a corresponding **class diagram**, and show the multiplicities clearly on the diagram.*

1..2 0..1



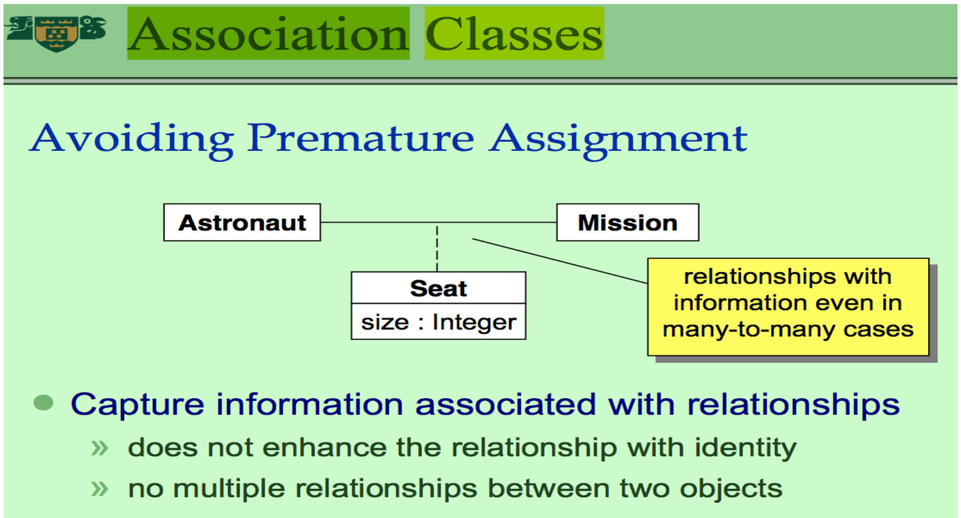
Class Level

Object Level

determines the outdegree of B nodes

determines the outdegree of A nodes

(a) [6 marks] Discuss why the use of association classes and higher-arity associations is a good idea. Give an example of a higher-arity association to illustrate the benefits.



Association Classes

Avoiding Premature Assignment

relationships with information even in many-to-many cases

- Capture information associated with relationships
  - » does not enhance the relationship with identity
  - » no multiple relationships between two objects

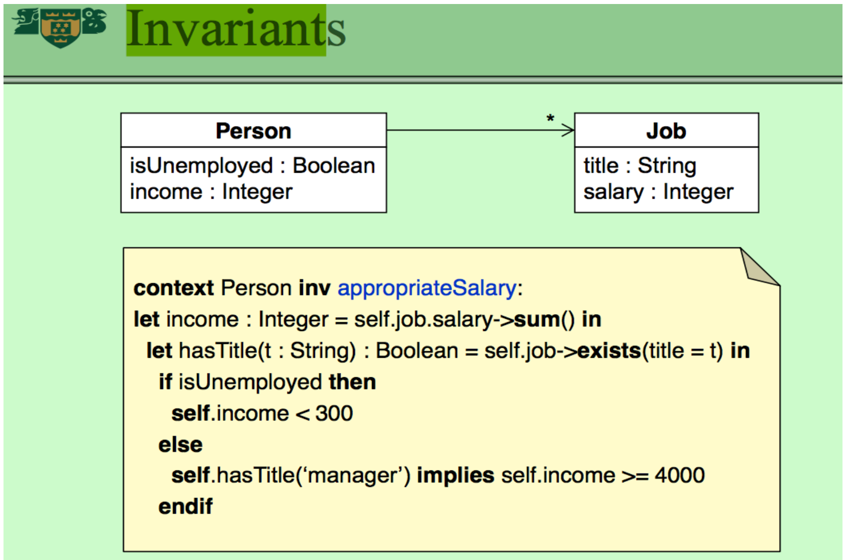
Avoiding Premature Assignment
Capture information associated with relationships » does not enhance the relationship with identity » no multiple relationships between two objects

(b) Explain the purpose of the Object Constraints Language (OCL).

As it is difficult to express too complex of diagrams through visual notation only; so, object constraint language can be used to apply constraints to relationships between classes.

(c) [6 marks] What are invariants in the context of OCL? Give an example of an invariant using OCL and explain its meaning.

A constraint that states a condition that must always be met by all instances of the component, type, or interface. Invariants must be true all the time.



```
context Person inv appropriateSalary:
let income : Integer = self.job.salary->sum() in
  let hasTitle(t : String) : Boolean = self.job->exists(title = t) in
    if isUnemployed then
      self.income < 300
    else
      self.hasTitle('manager') implies self.income >= 4000
    endif
```

UML

(a)Briefly explain what a "use case" is. Include the ultimate criterion that determines whether something really should be regarded as a use case.

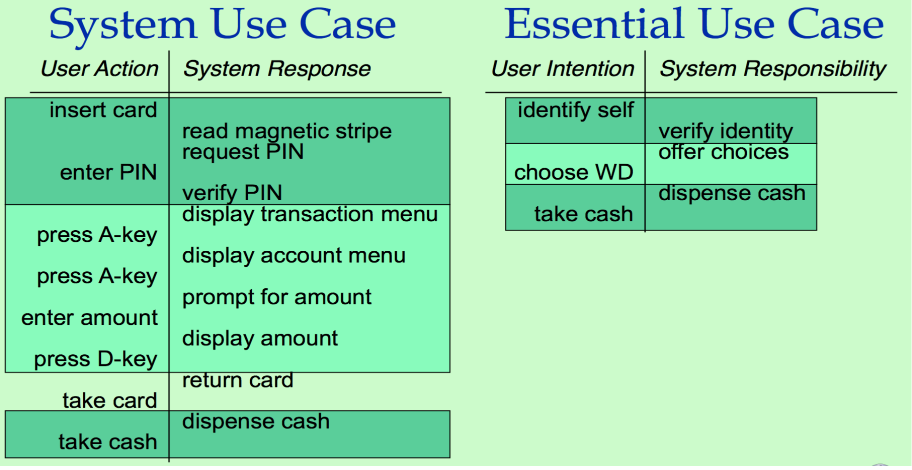
A use case is a detailed description of how a system is expected to interact with a user (or possibly another system).
Having such a description allows us to analyse it and develop specific:
functional requirements, actors, and interactions that the system must respond to in an appropriate manner.

(b)[2 marks] Briefly explain the idea of an "essential use case" (as opposed to a "system use case").

System Use Case » describes user actions and system responses at a technical level
Essential Use Case » describes user intentions and system responsibilities
a simplified and generalized form of use case ... intended to capture the essence of problems

through technology-free, idealized, and abstract descriptions

(b)Briefly explain the difference between a "system use case" and an "essential use case".



System Use Case

User Action	*System Response*
insert card	
	read magnetic stripe
	request PIN
enter PIN	
	verify PIN
	display transaction menu
press A-key	
	display account menu
press A-key	
	prompt for amount
enter amount	
	display amount
press D-key	
	return card
take card	
	dispense cash
take cash	

Essential Use Case

User Intention	*System Responsibility*
identify self	
	verify identity
	offer choices
choose WD	
	dispense cash
take cash	

System Use Case » describes user actions and system responses at a technical level
Essential Use Case » describes user intentions and system responsibilities

(a) Sally wrote the following OCL constraint in order express that if an employed person is a manager, the person's income must be at least $4000:

```
  context Person inv appropriateSalary:
                 if self.isEmployed then
                     self.isManager and self.income >= 4000
                endif
```

What would you change and why?

Easier to read. The first could imply that any employee is automatically a manager whereas the second one implies that self.isManager is a conditional.

```
Context Person inv appropriateSalary:
If isUnemployed then
   Self.income < 300
Else
  Self.hasTitle("manager") implies self.income >= 4000
Endif
```

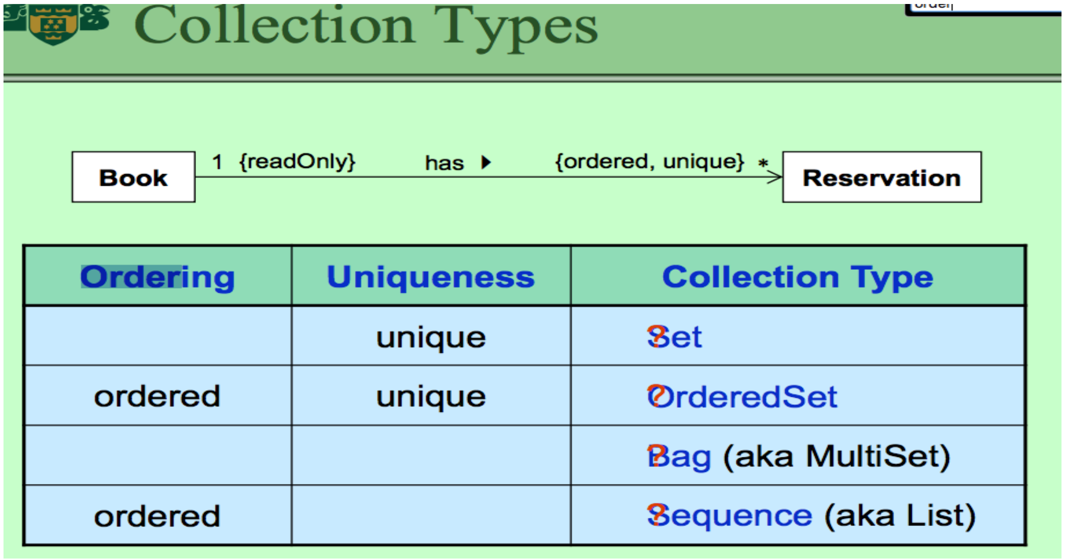
Briefly describe the UML's approach to characterising container types and write down how you

SPARE PAGE FOR EXTRA ANSWERS

Student ID:

would specify that a concept is used as i) an "Ordered Set" and ii) a "Sequence".

Unique → set. Ordered unique → Ordered Set Bag Ordered –sequence



Collection Types

Ordering	Uniqueness	Collection Type
	unique	Set
ordered	unique	OrderedSet
		Bag (aka MultiSet)
ordered		Sequence (aka List)