

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

EXAMINATION FOR
SEMESTER 1 AY2006/2007

CS2103 – SOFTWARE ENGINEERING

November 2006

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **FOUR (4)** questions and comprises **SIXTEEN (16)** printed pages, including this page.
2. Answer **ALL** questions within the space in this booklet
3. This is an **OPEN BOOK** examination.
4. Please write your Matriculation Number below.

MATRICULATION NO: _____

This portion is for examiner's use only

Question	Marks	Remarks
Q1 (max = 13)		
Q2 (max = 13)		
Q3 (max = 11)		
Q4 (max = 13)		
Total (max = 50)		

Question 1 (13 marks)

A local delivery service company, *SingEx*, seeks your help to develop a simple software system for them. *SingEx* intended to use this system to keep track of customer orders, which includes the general information about the package and its content. Information about the order and package are summarized below:

A single *ORDER* has the following information:

- Customer name
- Delivery address
- A *BOX* for containing all the delivery items

A single *BOX* has the following information:

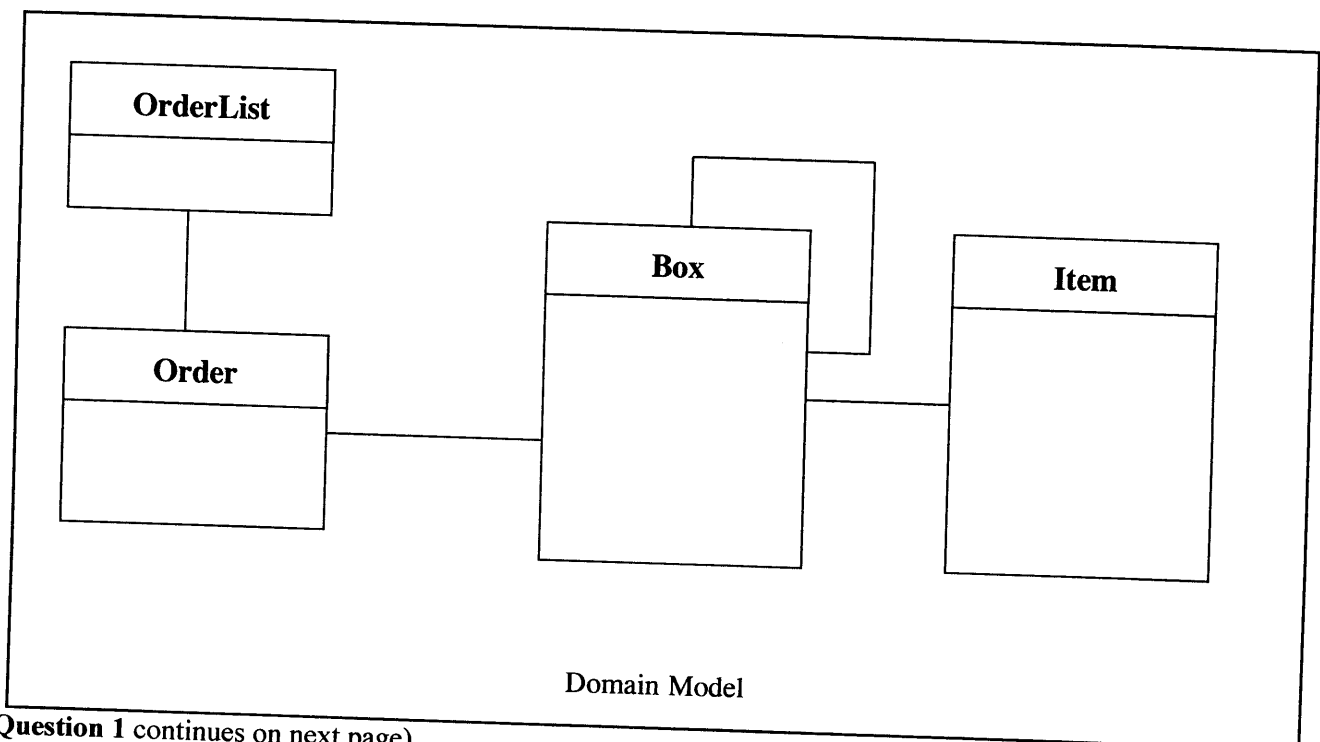
- Capacity (volume instead of dimension is used for simplicity)
- Remaining Capacity (after items are placed)
- Serial Number
- A number of *ITEMs*
- A number of smaller *BOXes* (known as sub-box)

A single *ITEM* has the following information:

- Name
 - Volume
 - Serial Number
-

Note that the “sub-box” is the same as a normal box (i.e. has the same information, can contain other item(s) and/or box(es)).

Part I) A draft *Domain Model* is given below. Fill in the attributes for all Entities and multiplicity for all Associations. (2 marks)



(Question 1 continues on next page)

Question 1

CS2103

Part II) Using the domain model in Part I, draw an **object diagram** for the following scenario. (2 marks)

Customer “FeddyEx” placed an order to be delivered to “NUS”. The order includes the following:

- A box (capacity 5000cm^3 , Serial Number 1234), which contains:
 - One “Printer” Item (volume 1250cm^3 , Serial Number 5555)
 - One smaller box (volume 3000 cm^3 , Serial Number 5556)
 - The smaller box contains:
 - Two “Cartridge” Items (volume 500 cm^3 , Serial Number 100 and 101 respectively)
-

Values for relevant attributes should be included. There is no need to draw the *OrderList* object.

(Question 1 continues on next page)

Question 1

CS2103

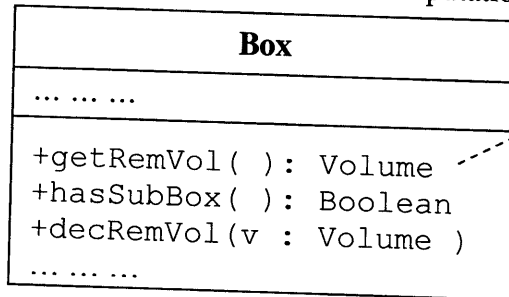
Part III) Study the interactions involved in *adding an item to a box* using **Sequence Diagram**. (3 marks)

The basic steps are:

1. If the item can be fitted in the box, then it is added directly.
 - a. Else, check whether there is any "Sub-Box"
2. If there is Sub-Box
 - a. Try to add the item to the Sub-Box

Hint: If there are multiple levels of message passing, just show up to the first "Sub-Box" level.

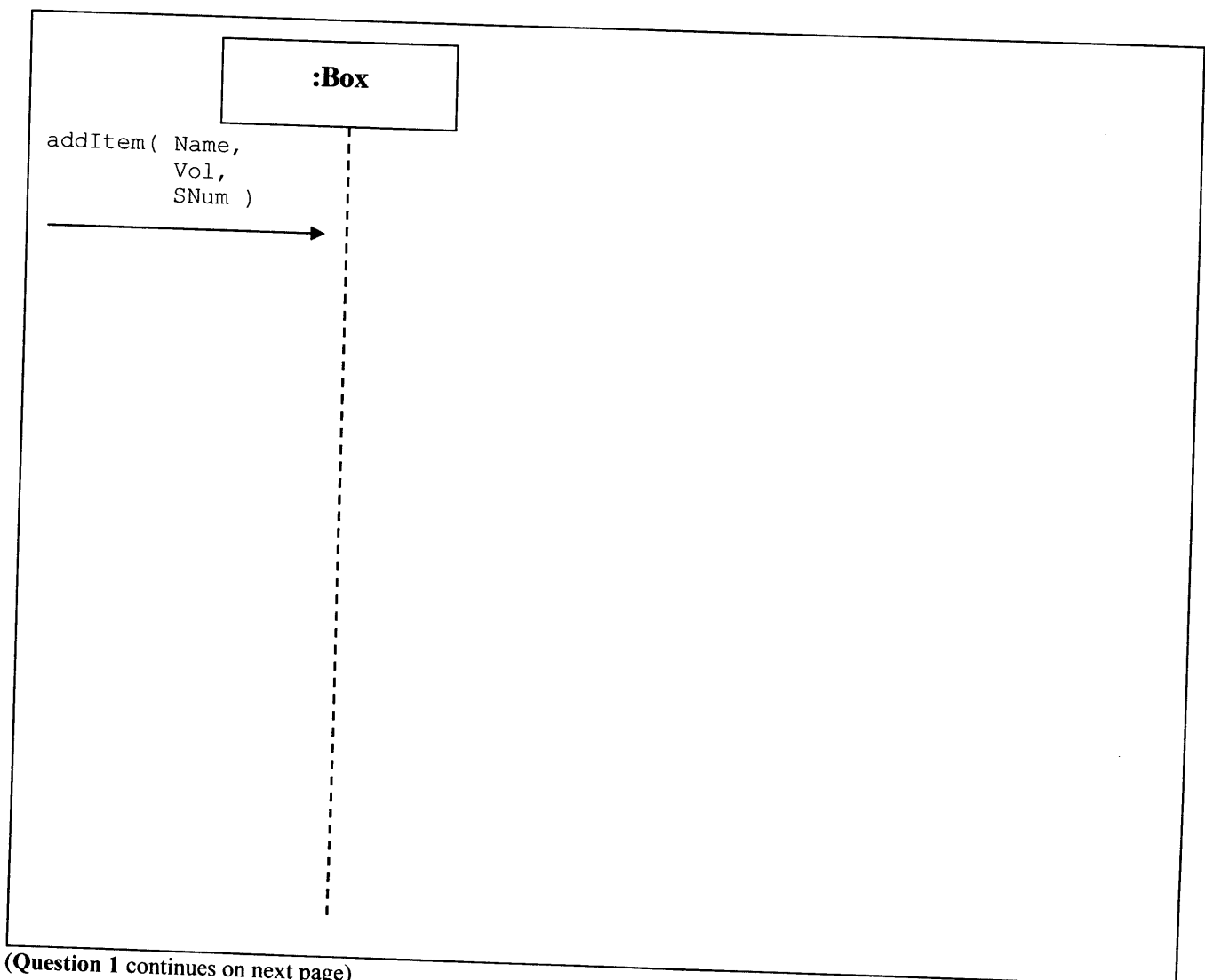
Some basic methods to show computation steps are added to the *Box* class for your reference:



Abbreviation:

- RemVol = Remaining Volume
- decRemVol = Decrease Remaining Volume

Use the partial sequence diagram below to fill in the interactions. You can add classifier roles and/or constraints as you see fit.



(Question 1 continues on next page)

Question 1

CS2103

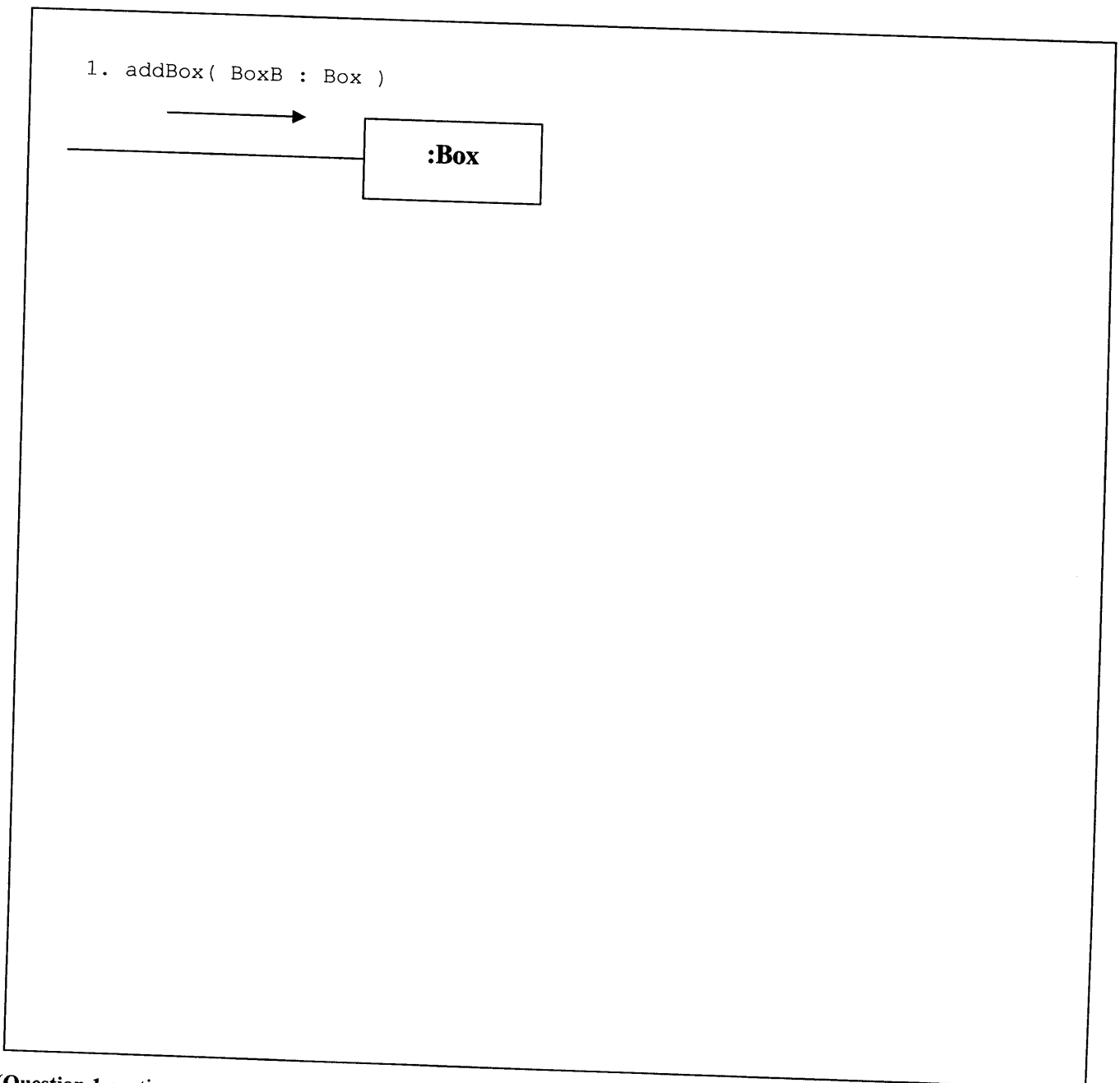
Part IV) Study the interactions involved in *adding an existing box (Box B) to another box (Box A) as "Sub-Box"* using **Collaboration Diagram**.

The basic steps are:

1. Get the **volume** of the "Sub-Box" (Box B)
2. Get the **remaining volume** for Box A
3. If Box B can be fitted in Box A
 - b. Add Box B into Box A

Basic methods introduced in Part III can be used.

Use the partial collaboration diagram below to fill in the interactions. You can add classifier roles and/or association roles as you see fit. **Label each association role with association stereotypes.** (3 marks)



(Question 1 continues on next page)

Question 1

CS2103

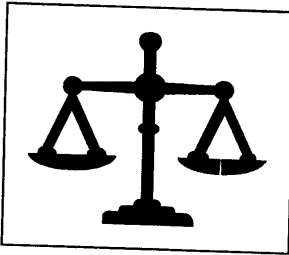
Part V) The class design in Part I can be improved by Design Pattern. There are two possible patterns for this problem: **General-Hierarchy Pattern** or the **Composite Pattern**. State which pattern is the best fit for this system and **briefly** justify your choice. Note that your reasoning should be based only on the information given in this question (including all parts). **(3 marks)**

Ans: The correct pattern is _____

Justification:

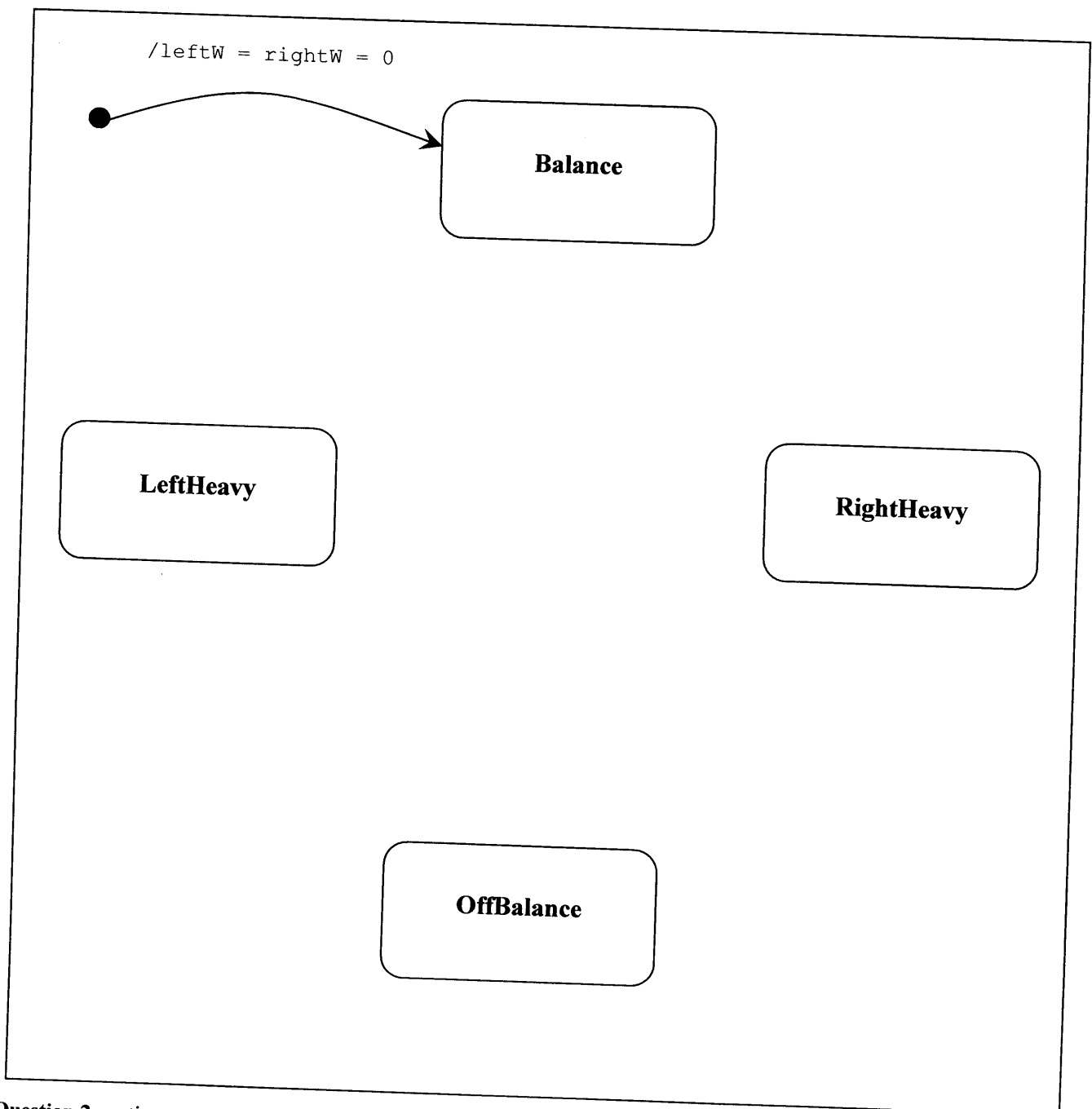
Question 2 (13 marks)

CS2103



A weighing scale (shown on the left) starts off in the *Balance* state. Weights can then be added to either left or right side of the scale. Whenever the weight on one side is more than the other, the scale will be tipped. For example, if left side weights more than right, the scale is in *LeftHeavy* state. When the weight difference is more than 20kg, the scale will be overturned, i.e. in *OffBalance* state and **cease to respond**.

Part I) Model the weighing scale using the partial statechart below. Assumes that the only triggers are **addL(w)** and **addR(w)**, where **w** is the weight in kg added to left or right side of the scale. Internally, the weights on both sides are kept as **leftW** and **rightW** respectively. To simplify the modelling, only model the transitions caused by addL(w). Add in all necessary transitions, guard conditions and actions. (4 marks)



(Question 2 continues on next page)

Question 2

Part II) Assumes that we are implementing the statechart using the switch-case approach, fill in the following skeleton code. **Only attributes declaration and implementation of the addL() method are needed. (4 marks)**

```
class WeighingScale
{
    //Declaration of relevant attributes

    //Implementation of the "addL" transition
    public void addL(
    {
```

```
}
```

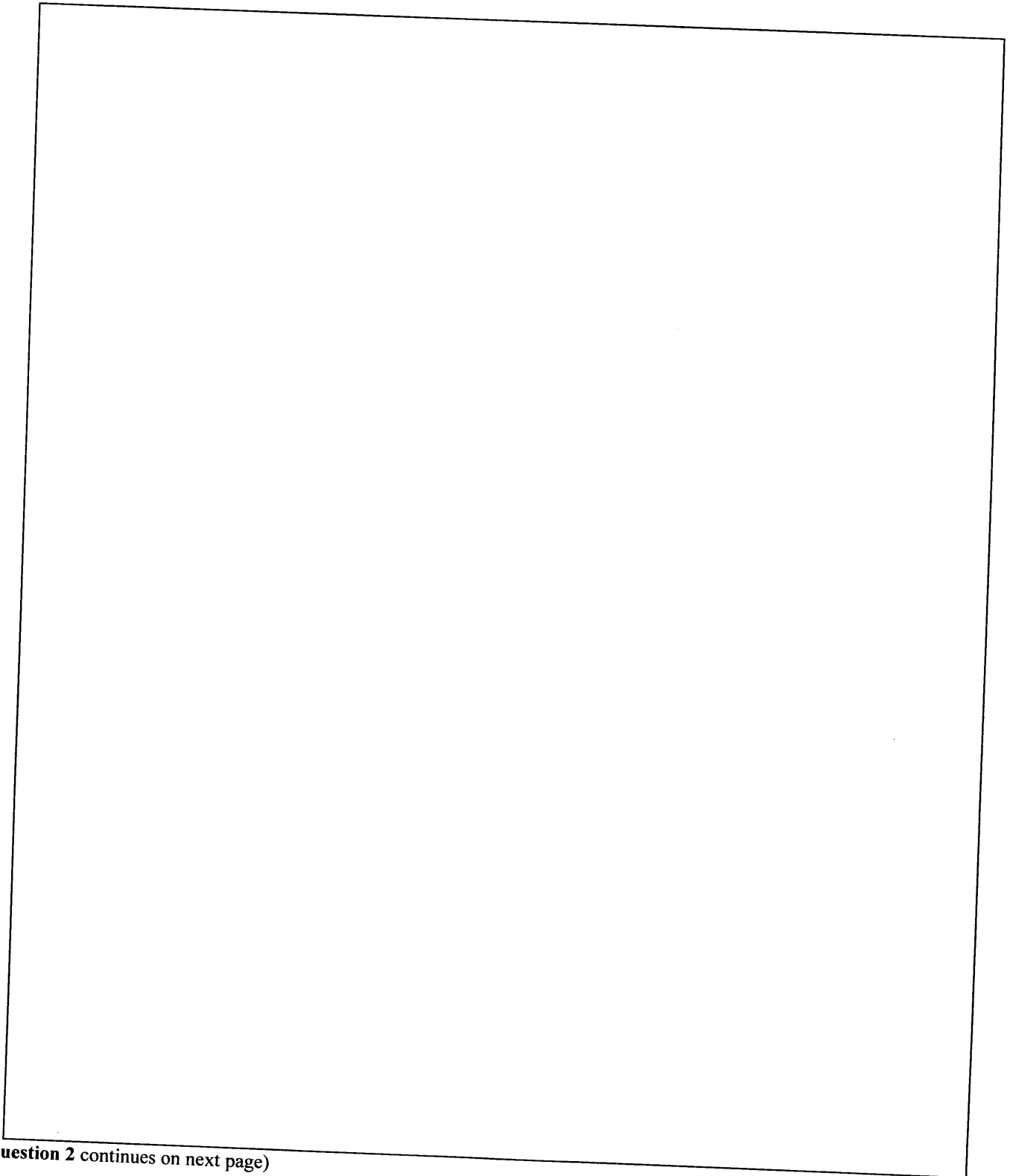
Question 2 continues on next page)

Question 2

CS2103

Part III) Assume that *State Pattern* is used to implement the statechart. A new class hierarchy with **ScaleState** as the superclass is added. The subclasses are **Balance**, **LeftHeavy**, **RightHeavy** and **OffBalance** respectively. Additionally, **addL()** and **addW()** returns a new state class instead of **void**.

Draw the class diagram for the **ScaleState** class hierarchy. All subclass icons should be included, however, attributes and methods information are only needed for the classes **ScaleState** and **LeftHeavy**. (2 marks)



Question 2 continues on next page)

Question 2

CS2103

Part IV) Give skeleton code for the **attributes, constructor** and **addL()** method in **LeftHeavy** subclass: (3 marks)

```
class LeftHeavy
{
    //Declaration of relevant attributes

    //Implementation of constructor

    LeftHeavy(
    {

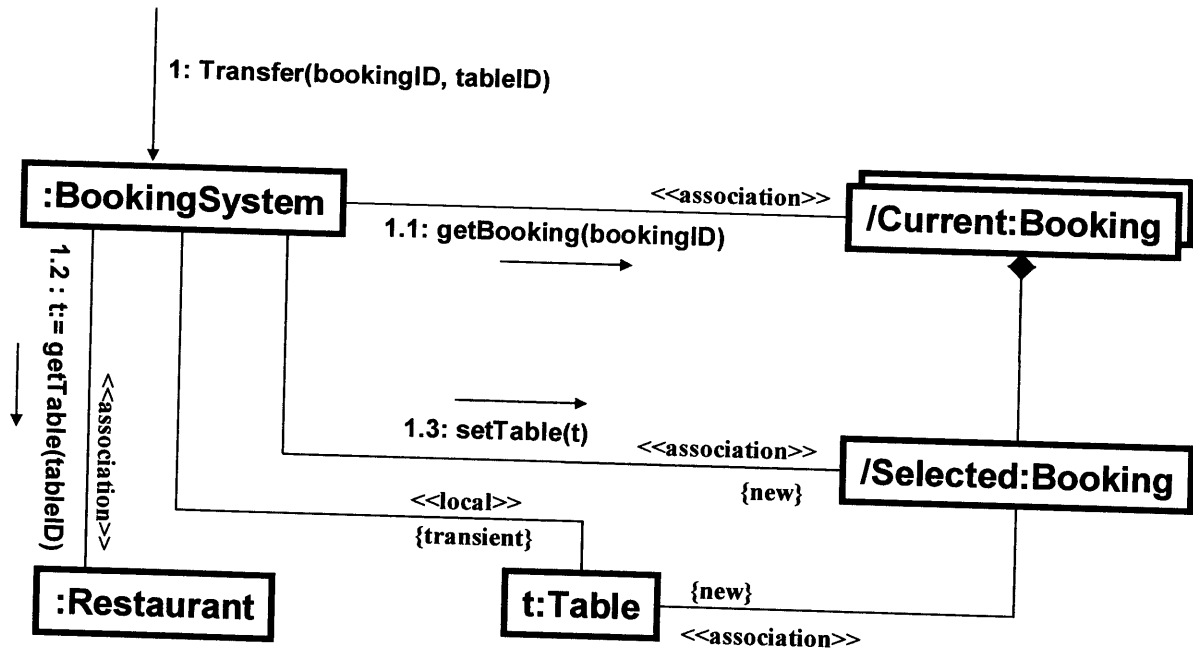
    } // End of LeftHeavy Constructor

    //Implementation of the "addL" transition
    public      addL(
    {

    } // End of addL( )
} // End of class LeftHeavy
```

Question 3 (11 marks)

Part I) Given below is a sequence diagram extracted from the design of a restaurant booking system. It models the operation of changing the table assigned for a booking. This operation is triggered when the system receives the message `transfer(bookingID, tableID)`. Parameters `bookingID` and `tableID` denote the identifiers for the Booking object in concern, and the Table object which should be assigned to the booking, respectively.



a) Complete the below operation contract for `transfer(bookingID, tableID)` operation. (4 marks)

Preconditions:

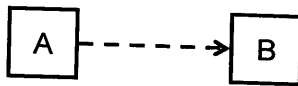
Postconditions:

(Question 3 continues on next page)

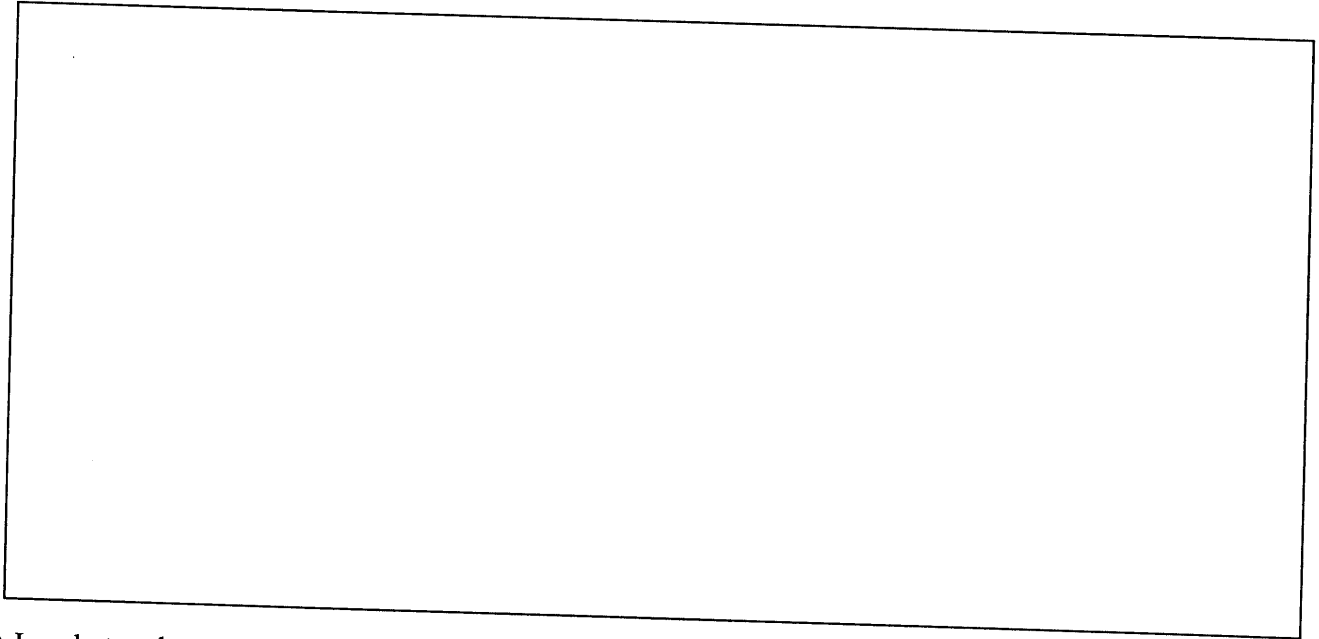
Question 3

CS2103

- b) What are the dependencies between the classes mentioned in this sequence diagram? Use the following notation to show dependencies. **(2 marks)**



A depends on B



- c) In what order would you implement the classes mentioned in answer *b*, if you are following a strict top-down approach to implementation? **(1 marks)**

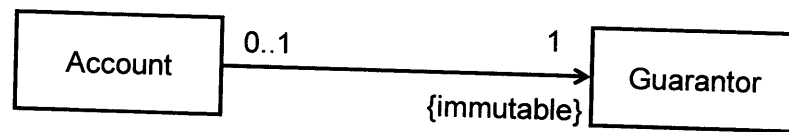
- d) Write a stub for the `getTable(tableID)` method of the `Restaurant` class. A stub need not be functionally complete, yet it should compile without errors, and should 'appear' to be fully functional to the users of the stub. **(2 marks)**

(Question 3 continues on next page)

Question 3

CS2103

Part II) Using a defensive approach to coding, give a suitable implementation to the following class diagram. (2 marks)

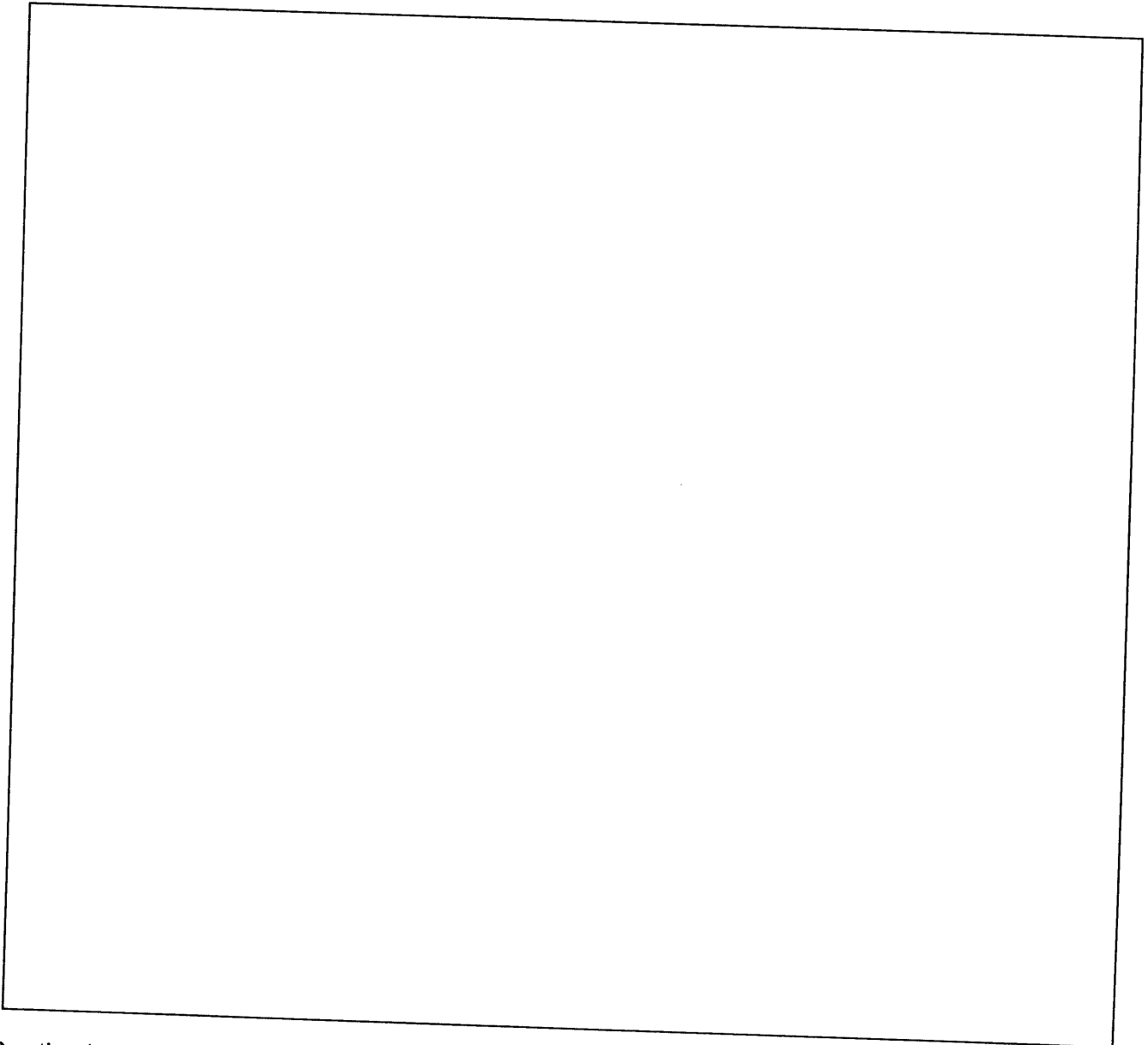


Question 4 (13 marks)

Function `printNumbers` takes two integer numbers as parameters, and prints all integers from the smaller number up to, but not including, the larger number. For example, `printNumbers(7, 4)` prints 4,5,6.

```
function printNumbers(int a, int b){  
    if (a>b){  
        int i = b;  
        while(i<a){ print i; i++;}  
    }else{  
        int i = a;  
        while(i<b){ print i; i++;}  
    }  
}
```

Part I) Draw a control flow graph for the `printNumbers` function. **(5 marks)**



(Question 4 continues on next page)

Question 4

CS2103

Part II) What is the cyclomatic complexity number for `printNumbers` function? (1 mark)

Part III) List independent basis paths. For each realizable basis path, also give test data to execute the path, and expected output (the numbers that would be printed). Note that the number of rows in the below table does not indicate the number of paths. (4 marks)

Path	Test data		Expected output
	a	b	

(Question 4 continues on next page)

Question 4

Part IV) Rewrite the `printNumbers` function to give the same functionality, yet to become easier to test using the same testing approach. That is, we should be able to test the new function to the same extent as before, but using fewer paths. Use only the language constructs already used in the function. (3 marks)

~ End of Paper ~