NATIONAL UNIVERSITY OF SINGAPORE

CS2103/T – SOFTWARE ENGINEERING

(Semester 2: AY2016/2017)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number only. Do not write your name.
2. This assessment paper contains **FIVE (5)** questions and comprises **ELEVEN(11)** printed pages.
3. Answer **ALL** questions within the space in this booklet.
4. This is an OPEN BOOK assessment.

Student Number:	A								

This portion is for examiner's use only

Questions	Marks	Remarks
Q1	/ 8	
Q2	/ 10	
Q3	/ 10	
Q4	/ 6	
Q5	/ 6	
Total	/40	

The **problem description** used for all questions in this assessment paper is given in the appendices (last page of this assessment paper).

Q1. [8 Marks]

(a) [2 marks] List down three different user roles for the software app.

(b) [4 marks] Give a *nice-to-have* user story for each of the user roles listed in **(a)** above.

(c) [2 marks] List down two non-functional requirements for the software app. Choose requirements that are more specific to **DragonEat**, as opposed to a requirement that is applicable to most other software.

Q2 [10 marks] Suppose that the implementation for managing the customer details is to be re-used from an existing in-house software product that has been developed previously. The constructor for the ***Customer*** class is given as follows:

public Customer(Name name, Address address, Phone phoneNumber)

(a) [8 marks] By making use of the ***Customer*** class above, model and implement a customer order, i.e. item (1) of Appendix A using the **OOP** paradigm. You are to write the relevant Java classes by including the following:

- All relevant data properties of each class
- Constructors for each class
- All relevant functionalities so as to facilitate the calculation of the price of the order. Note the different pricing plans.

Pay particular attention to the OOP concepts of abstraction, encapsulation, inheritance and polymorphism in your implementation. You may exclude program comments.

(b) [2 marks] What SOLID principles are reflected in your above implementation? Justify your answers.

Q3. [10 marks] The original design of the **DragonEat** app supports **local mode** and **network mode**. Local mode retrieves data from a file stored on hard disk, while the network mode retrieves the data from remote server via internet. Consider the following two simplified code snippet in the **Data** class:

```
public Data() {
    .....
    store = new Storage();
    .....
}

public void initializeData( ) throws loadDataException {
   try {
     if ( isLocalMode) {
        allData = store.loadDataFromFile( filepath );
     } else {
        allData = store.loadDataFromNetwork( url );
     }
   } catch (fileOperationException, networkOperationException ) {
       throw new loadDataException();
   }
}
```

(a) [3 marks] Use class diagram to model the **Data** class and **Storage** class. Your modeling should reflect **only** the code snippets above and be as accurate as possible.

(b) [2 marks] Explain the rationale of the **try-catch-throw** in the above code.

(c) [3 marks] Redesign the **Storage** class to satisfy the following criteria:

i. Improve Cohesion
ii. Enable Dependency Injection

Use a class diagram to model your new design.

(d) [2 marks] Using your design in **(c)**, rewrite the ***initializeData***() code. Highlight how dependency injection can be performed. If necessary, you can involve other methods in the Data class, e.g. constructor.

Q4. [6 marks]

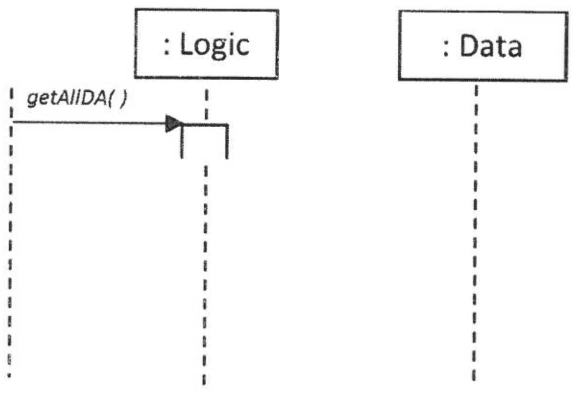
Suppose we have a ***DeliveryAssignment*** class to capture the delivery assignment, i.e. which dinner package is given to a deliverer to deliver. The current (incomplete) design of the class is given below:

DeliveryAssignment
- timeAssigned : Time - customerName: String - customerPhone: String - customerAddress: String
......irrelevant methods omitted......

Your development team previously decided on the following use case:

Use Case: Indicating Delivery Timing **Actor: Deliverer**
1. Deliverer requests a list of all his/her delivery assignments. 2. System shows a list of delivery assignments. 3. Deliverer requests to record the delivery time of a specific assignment. 4. System records the current time as the delivery time of the indicated delivery assignment. 5. [Use case ends]

(a) [3 marks] Use sequence diagram to model the interaction in the above use case. For simplicity, you can focus on the internal logic and ignore the UI classes. To reduce cluttering, you can use the shorthand "**DA**" to represent ***DeliveryAssignment*** in the diagram.



(b) [2 marks] Update the class diagram for ***Deliverer***, ***DeliveryAssignment*** based on your sequence diagram in **(c)**. Remember to model as close as possible to the sequence diagram.

(c) [1 marks] Consider ***DeliveryAssignment*** together with the existing classes in Appendix C, identify one drawback of the current design. Suggest a better alternative design for the ***DeliveryAssignment*** class and briefly justify.

Q5. [6 marks]
Singapore postal code is a six-digit number with the following restrictions:

- The first two digits from the left represent the sector code, only valid from "01" to "85".
- The third digit represents the street code, "0" to "9" are all valid.
- The remaining three digits represent the house / block number. Validity is dependent on the sector code:

Sector Code Range	Valid House/Block Number
"01" to "34"	"001" to "099"
"35" to "85"	"001" to "999"

Your team designed the following method for postal code validation:

public boolean isValidPostalCode(String postalCode)

(a) [4 marks] Suppose we know that the UI enforced that only string with exactly six digits can be entered as **postalCode**, design 4 valid and 4 invalid test cases for the method above. Demonstrate your understanding of test case design heuristic by giving simple justification for each test case.

postalCode	Expected Result	Justification for having this test case
	false (invalid)	
	false (invalid)	
	false (invalid)	
	false (invalid)	
	true (valid)	
	true (valid)	
	true (valid)	
	true (valid)	

(b) [2 marks] Classify the test case design in the above as Glassbox / Greybox / Whitebox. Briefly explain whether this test case design approach is appropriate for this example.

It is a Glassbox / Greybox / Whitebox approach (Circle the right answer).

Justification:

**~~~ End of Paper ~~~**

Given below is the **problem description** used for all questions in this assessment paper. You may detach this page from the assessment paper. There is no need to submit this page.

Appendix A. DragonEat product description

DragonEat Food Catering is an up and coming food catering company, whose niche business is in the home delivery of dinner on weekdays (Monday to Friday). To improve service efficiency, the company has decided to engage your software team to build a software app. The app, **DragonEat**, is to be used by both the customers and the company's employees. The typical usage scenarios are given below.

1. A **customer** places an **order** for one of the available monthly dinner packages and makes payment online.
    - The customer supplies his/her name, home address and phone number.
    - The price of the order will be calculated based on the type of dinner package, and the number of persons (pax) to be catered for. Details on the pricing is given below.
    - The dinner delivery will start on the following Monday for four weeks.
2. Each working day, the **food manager** uses the system to compile the number and type of dinner packages according to the postal code. The food packages are then prepared and assigned to the delivery rider, or simply termed a **deliverer**.
3. The **deliverer** will indicate the dinner as delivered after the each of the food package is sent to the customer's home.
4. At the end of the day, the manager uses the system to compute various statistics:
    - Number and type of dinner packages ordered and delivered.
    - The average of delivery time (from the time the package is assigned to the deliverer till the package is delivered to customer) for every deliverer.

Currently, the dinner packages come in two varieties:
- **Economy** dinner package: the price per pax (or per person) is fixed at 100 dollars.
- **Deluxe** dinner package: the regular price per pax is 125 dollars. However, **DragonEat** is keen to offer a discount on new orders that depends on the existing number of orders. Finer details have yet to be worked out.

Moreover, **DragonEat** has the intention to include more packages in future, with more variety in terms of menu selection, if the food catering business is deemed viable.

Appendix B. Suggested components for the system architecture

- **UI**: The UI is a GUI and uses JavaFX.
- **Logic**: The main logic of the application.
- **Data**: An in-memory collection of data objects needed for responding to users.
- **Storage**: Provide functionality to retrieve data from harddisk / network for the **Data** component. Used once at the start of the app.

Appendix C. Given classes and attributes for the Data component

- **Customer**: {name} {address} {phone number}
- **Deliverer**: {phone number} {car plate number}

**~~~End of problem description~~~**