NATIONAL UNIVERSITY OF SINGAPORE

CS3219 – SOFTWARE ENGINEERING PRINCIPLES and PATTERNS

(Semester 2 AY2016/2017)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Write your Student Number only. Do not write your name.
2. This assessment paper contains **EIGHT questions** and comprises **SIXTEEN** printed pages.
3. Answer **ALL** questions within the space in this booklet.
4. Use a pen to write descriptions. You can use a pencil to draw and label diagrams.
5. This is a CLOSED book assessment. An A4 size help-sheet is allowed.

STUDENT NO: ____________________________________

This portion is for examiner's use only

Question	Marks	Remarks
Section A	/18	
Section B	/26	
Section C	/26	
Total	/70	

(Blank Page)

Section A **18 marks**

Question 1: Circle True or False as appropriate for the given statements. **(3 marks)**

Statement	Circle your selection
The facade pattern is used to lower coupling between different subsystems.	True / False
In the facade pattern clients only interact with a subsystems' facade. They never interact with the subsystem directly.	True / False
The facade object only forwards a client request to the subsystem, it does not answer the request itself.	True / False
The mediator pattern requires an abstract Mediator class.	True / False
The Memento pattern allows the Originator to store a history of mementos.	True / False
Memento pattern is useful when you need to provide an undo mechanism in your applications.	True / False

Question 2: Identify the category (e.g. data, implementation etc) for each of the non-functional requirements. **(3 marks)**

Non-functional requirements	Category
The user clicks at the top of the name list to change the sort sequence.	
Only open source software available under the GNU General Public License can be used to implement the product.	
The Application must use Microsoft .NET framework 4.5	
ATM contains only $20 bills.	
Online payment may be made only through PayPal.	
All textual data used by the application shall be stored in the form of XML files.	

Question 3 (each 4 marks)

a) "Quality or non-functional Requirements are elicited after the Functional Requirements". How far do you agree with the above statement? Give reasons with your answer.

Answer :

b) “Evolving Software considerably is more difficult than developing software from scratch”. How far do you agree with the above statement? Give reasons with your answer.

Answer :

c) What kind of information can be useful to extract from code artifacts to reconstruct a software architecture?

Answer :

SECTION B **26 marks**

Question 4 (each 4 marks)

a) Assume that you are building a source code transformation toolkit. The toolkit will contain components that can be configured to perform specific sets of code transformations on source code. There are 3 categories of components: parsers that can read in source code and produce internal program representation, transformers (code formatters, code optimizers, etc.) that work on the internal program representation, and code-generators that can transform the internal program implementation back into source code. Assume that the toolkit should be easily extensible with new components and that the components can work incrementally on streams of source code or internal representation.

What architectural style would be most appropriate for this toolkit? Why?

Answer :

b) Assume that you are building an e-commerce system handling high volumes of transactions per day from on-line customers.

Which architectural style is most appropriate for the given requirement? Why?

Answer :

c) ***Caretaker* in memento pattern is similar to *invoker* in command pattern.**
How far do you agree with the above statement? In your answer describe the role of caretaker and invoker in the respective patterns.

Answer :

d) Many design patterns use two or more design principles. Select one design pattern, from the patterns covered in the module, that uses two design principles (covered in the module). Your answer should clearly write which design principles are used by the pattern.

Answer :

Question 5 (4+6 marks)

a) The method below returns the String `"rectangle"` if the given Shape instance is a Rectangle and `"circle"` if it is a Circle.

```
public static String getName(Shape shape) {
  if (shape instanceof Rectangle)
    return "rectangle";
  if (shape instanceof Circle)
    return "Circle";

}
```

Which design principle(s) is being violated in above and why?

Answer:

b) Following is an implementation of a controller that handles adding new comments from visitors to a blog post. To stop spam bots from posting comments it currently checks commenting visitors IP address' against an IPBlackList.

```
public class EntryController
{
    public void AddComment()
    {
        if(ValidateNotSpam())
        {
             //Save the comment to the database
        }
    }
    private bool ValidateNotSpam(string comment)
    {
        //Check if the IP-address that made the request
        //is known as a spammer by IPBlackList
    }
}
```

Which design principle(s) is being violated in above and why?

Answer:

SECTION C 26 marks

Question 6 (4 marks)

Use pipe & filter style to design components for the following requirement. Illustrate your design with clearly labeling the components of pipe & filter style.

Assume a new order arrives in our enterprise in the form of a message. The message could be encrypted to prevent eavesdroppers from spying on a customer's order. The message contains authentication information in the form of a digital certificate to ensure that orders are placed only by trusted customers. Duplicate messages could be sent from customers. To avoid duplicate shipments and unhappy customers, duplicate messages should be eliminated before subsequent order processing steps are initiated.

Answer:

Question 7 (4+12 marks)

Consider a large network of distributed components, such as computers, hubs, routers, software programs, and so forth. We wish to monitor and manage these components from a central location. Problems are typically infrequent and unpredictable, but when they do occur we wish to be notified without having to constantly poll the components. The notification may appear on a central console, pager, or e-mail reader. In the simplistic implementation, shown in Figure 1, a component (**Hub** and **Router** in the example) must send notification of problems to both the **Console** and the **e-mail reader**.

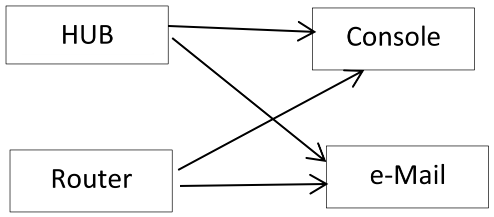

Figure 1

a) What kind of problem(s) do you see if we later wish to change the interface of the console or e-mail reader or add another notification receiver e.g. a paging system ?

Answer:

b) One of your team member suggests using **mediator** or **observer** or **publish-subscribe patterns.**
Illustrate use each of the above suggested patterns, through suitable diagrams, to replace the mechanism given in *Figure 1*.

Answer (next page) :

Mediator
Observer
Publish-subscriber

Question 8 (6 marks)

Given below is an implementation of a very useful pattern in application control. Identify the pattern and illustrate its corresponding design through a diagram.

```
public interface Order {
    public abstract void execute ( );
}

class StockTrade {
    public void buy() {
        System.out.println("You want to buy stocks");
    }
    public void sell() {
        System.out.println("You want to sell stocks ");
    }
}


class Agent {
    private m_ordersQueue = new ArrayList();

    public Agent() {
    }

    void placeOrder(Order order) {
        ordersQueue.addLast(order);
        order.execute(ordersQueue.getFirstAndRemove());
    }
}

class BuyStockOrder implements Order {
    private StockTrade stock;
    public BuyStockOrder ( StockTrade st) {
        stock = st;
    }
    public void execute( ) {
        stock . buy( );
    }
}

class SellStockOrder implements Order {
    private StockTrade stock;
    public SellStockOrder ( StockTrade st) {
        stock = st;
    }
    public void execute( ) {
        stock . sell( );
    }
}

public class Client {
    public static void main(String[] args) {
        StockTrade stock = new StockTrade();
        BuyStockOrder bsc = new BuyStockOrder (stock);
        SellStockOrder ssc = new SellStockOrder (stock);
        Agent agent = new Agent();

        agent.placeOrder(bsc); // Buy Shares
        agent.placeOrder(ssc); // Sell Shares
    }
}
```

Answer :

(Blank page)

(Blank page)

--- End of paper---