OOS Class Test 2 06-11-2023

Time: 50 mins. **Marks 35**

Name:

Roll Number:

Over all Evaluation Guideline: Every mistake = -1 mark

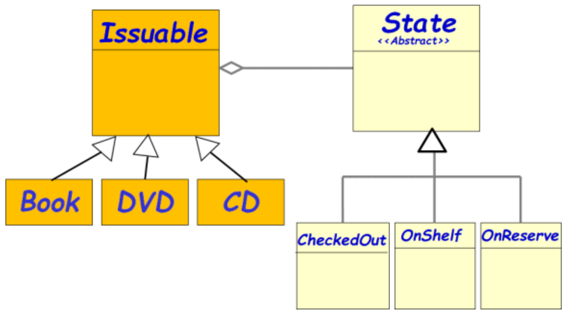
1. In the in-built Java **Observer** interface, why does the **update** method in the interface include a reference to the **Observable** (the object being observed)? Does the **Observer** not know what **Observable** it is observing? Restrict your answer to only one short sentence. **[2]**

Your Answer:

An Observer object may be observing multiple Observable objects.

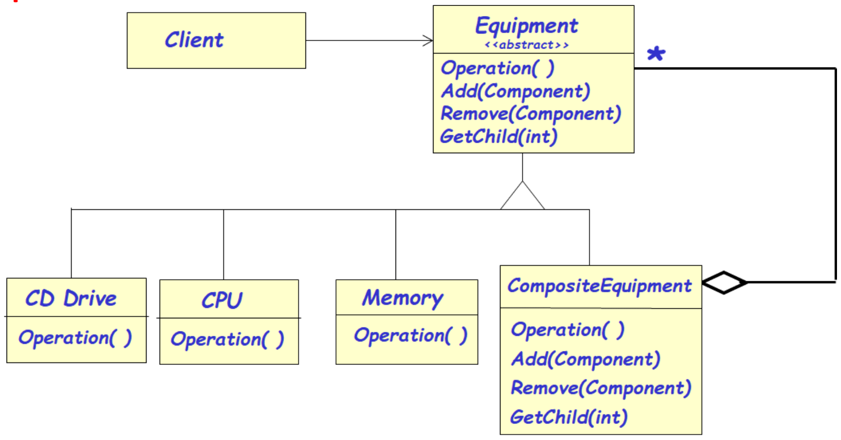
2. Consider a Library Automation application that needs to be developed to automate the book keeping activities of a large public Library. The **Library** has a large number of **Issuables**. Each **Issuable** is either a **book**, a **DVD**, or a **Music CD**. The state of each **Issuable** can either be **on shelf**, **issued out**, or **on reserve**. The response to an **issueOut()** request to an **Issuable** depends on its state. Design the class model of the **Issuable** class using state pattern. Restrict your class model to only what is described in the question. **[5]**

Ans:



3. Suppose you are a programmer employed by a company that manufactures various types of computerized systems. You have been entrusted with developing an application that would manage the sales of various computerized systems that your company is manufacturing. Your program would be used, among other things, to compute the cost of any computerized system from the costs of the atomic components used in constructing the system. Every computerized system is constructed from a few atomic components such as CPUs, buses, CD drives, memory modules, and also a few subsystems. Several variations of the basic components are being used in manufacture of different computerized systems, such as different versions of CPUs, different sizes and types of memory modules, etc. Each subsystem may be made up of CPUs, buses, CD drives, memory modules, and few subsystems. Draw a class diagram to model any arbitrary computerized system that your company is already manufacturing or may manufacture in the future. **[5]**

Ans: Composite pattern



4. Consider that you are developing a Java application program in which you want all messages interchanged among classes to be written to a log file by calling the **LogMessage()** method of the **LogManger** class. You don't want accidental creation of multiple **LogManger** objects, nor do you want any trouble that may arise when two different messages are attempted for logging in at the same time. Write skeletal Java code for the **LogManager** class, by implementing the Singleton pattern. **[5]**

Ans:

```
public class MessageLogManager {
    private static MessageLogManager instance = new
  MessageLogManager();
   private MessageLogManager(){      }
   public static MessageLogManager getInstance(){
     return instance;
     }
   public void synchronized logMessage()   {
     ….
  }
}
```

5. Consider the following Java code.

i) Encircle the statement(s) that are not compliant with any of the GRASP design patterns.

ii) Name the GRASP design pattern(s) with which it is/are not compliant.

iii) For your identified instructions, write down the code that would be obtained by using the GRASP pattern. Also Mention the line number(s) that would be replaced or the line number at which your code would be inserted. **[1+1+3]**

```
1.  class Order {
2.      private Customer customer;
3.      private String description;
4.      public String getDescription() {
5.          return description;
6.      }
7.     public Customer getCustomer() {
8.          return customer;
9.      }
10. }
```

```
11. class Customer {
12.       private String name;
13.       public String getName() {
14.           return name;
15.      }
16.  }
```

```
17. class OrderPrinter {
18.     private order Order;
19.     public void printOrder() {
20.        System.out.println("Order: " + order.getDescription()  + " by user " + order.getCustomer().getName());
21.     }
22.  }
```

Ans

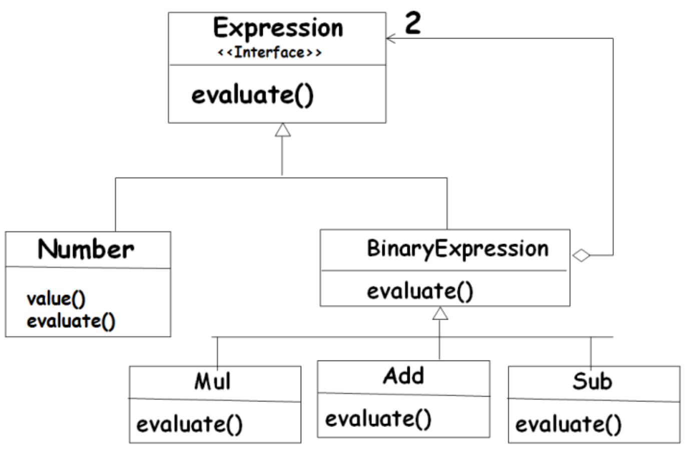
ii) Law of Demeter

iii)

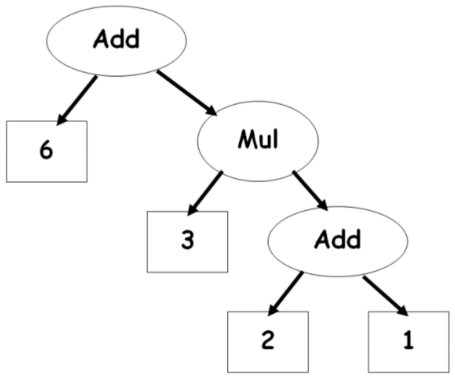
```
9.  public String getCustomerName() {
        return customer.getName();
    }
20.     System.out.println("Order details: " + order.getDescription() + " by user " +
    order.getCustomerName());
```

6. Suppose you are developing a Java application that requires you to handle binary expressions. Draw a class diagram to represent the class structure that can store and evaluate any arbitrary arithmetic expression **E**. The expression **E** can be an integral number (Terminal object) or an expression of the form **E op E**. Assume that **op** can be any of the arithmetic operators: **Add, Sub, Mul, and Div**. A few example expressions are: **1**, **1 Add 2**, **3 Add 4 Mul 5** etc. Give object diagram for the expression **6 + 3 * 1 + 2** **[5+1]**

Ans:



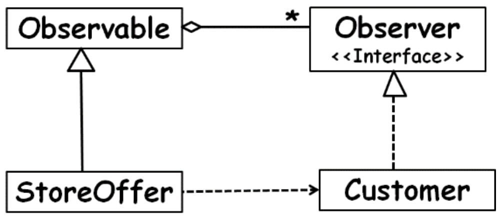
Object Diagram:



7. Assume that in a certain super market, customers can register their interest in the App of the supermarket to receive SMS notifications regarding any new discount schemes introduced by the super market. A member may at anytime opt out of the notification option for which he/she might have registered. Assume that whenever a new discount scheme is started by the store manager, the registered members would be instantly notified.

a. Give a solution to this problem (only class diagram) using the inbuilt Java Observer and Observable classes. **[4]**

Ans:



b. Write skeletal Java code for your class design solution in Part a). **[3]**

```
class StoreOffer extends Observable{
        private OfferScheme s;
        public void start(OfferScheme b) {
                this.s=b;
```

```
                    notifyObservers();
          }
          Public OfferScheme getDetails(){
                    return b;}
}
Class Member Implements Observer{
          void update(Observable sub, Object arg) {
                    sub.getDetails();}
```

---------- The End ---------