Sample EXAM

PROBLEM #1

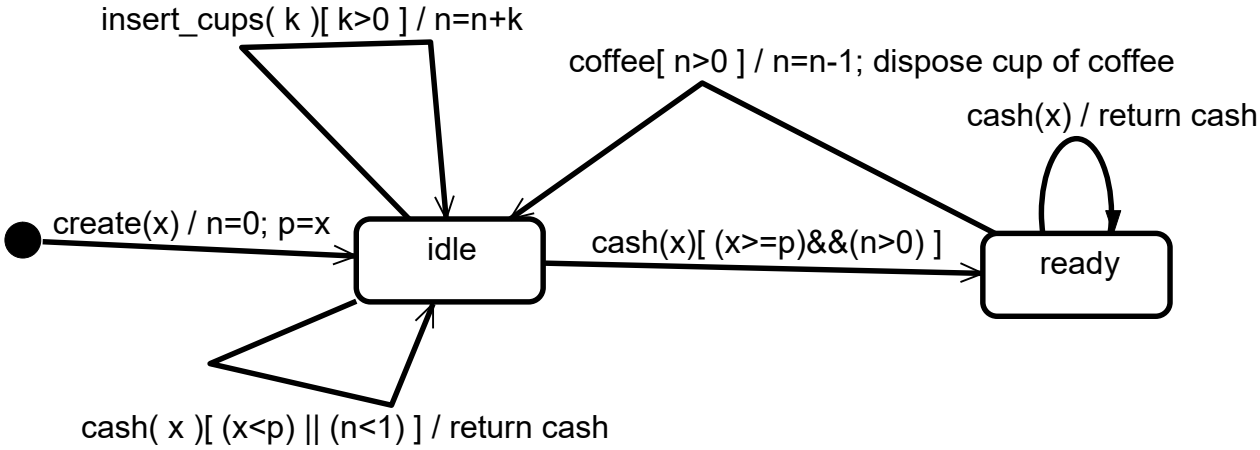
An EFSM (Extended Finite State Machine) of a component is shown below. The component supports the following operations: *create(float x), cash(float x), insert_cups(int k), coffee()*

Design the system using the **State design pattern**. You should use the **de-centralized** version of this pattern.

In your solution:

a. Provide a class diagram for the component. For each class list all operations with parameters and specify them using **pseudo-code**. In addition, for each class provide its attributes and data structures. Make the necessary assumptions for your design. Notice that the components in your design should be de-coupled as much as possible. In addition, components should have high cohesion.

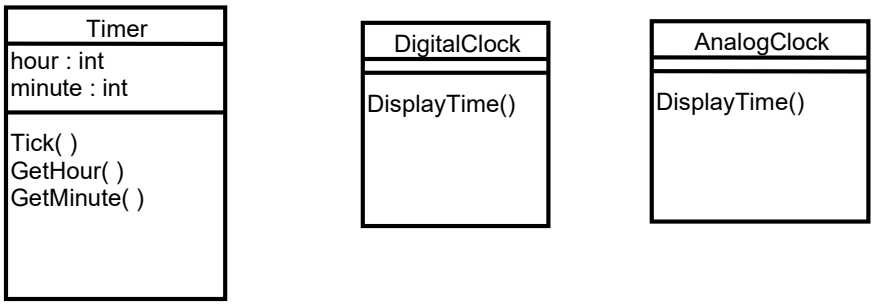
b. Provide a **sequence diagram** for the following operation sequence:
create(2.5), *insert_cups(10)*, *cash(3)*, *coffee()*



PROBLEM #2

In the system there exists a class *Timer* whose object stores and maintains the time of a day. This class supports the following operations: *Tick()*, *GetHour()*, and *GetMinute()*. The *Tick()* operation is called by an internal timer every 1 second. *Tick()* operation updates the *Timer*'s internal state (time data structure). Operations *GetHour()* and *GetMinute()* provide the interface for retrieving individual time units such as an hour and a minute.

In addition, there exist clock components in the system (e.g., *DigitalClock*, *AnalogClock*, etc.) that are responsible for displaying the time of the *Timer* component with a precision to a minute. Design a software subsystem using the **Observer** design pattern in which interested clock components can be updated about the current time of the *Timer* component.

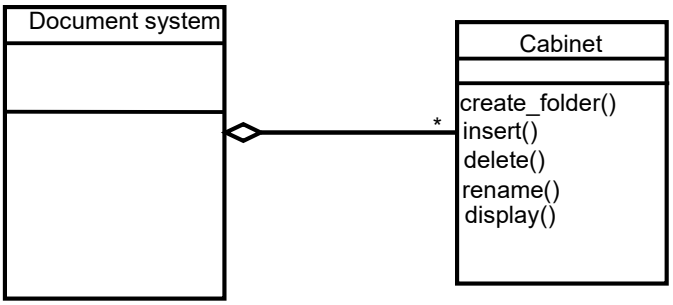


1. Provide a class diagram for the system. The class diagram should include classes *Timer*, *DigitalClock* and *AnalogClock* (if necessary introduce new classes and operations). In your design it should be easy to introduce new types of clock components (e.g., *AlarmClock*) that are interested in observing the time of the *Timer* component. For each class list operations and briefly describe their (operations) functionality. Notice that the components in your design should be de-coupled as much as possible. In addition, components should have high cohesion.

2. Provide a sequence diagram showing how the system notifies a registered digital clock and an analog clock about time change.

Note: Assume that the *Timer* and *Clocks* are in the same time zone and use the *24-hour time format* (e.g., 16:24).

PROBLEM #3
A document system consists of a set of cabinets as shown below:



Each cabinet contains folders and documents. A folder is a group of documents. In addition, a folder may contain another folders. Currently, the document system supports two types of documents: specification documents and design documents. In addition the system supports the following operations:

create_folder() – a folder is created
insert() – a document or a folder is inserted
delete() - a document or folder is deleted
rename() - a document or folder is renamed
display() - a document is displayed

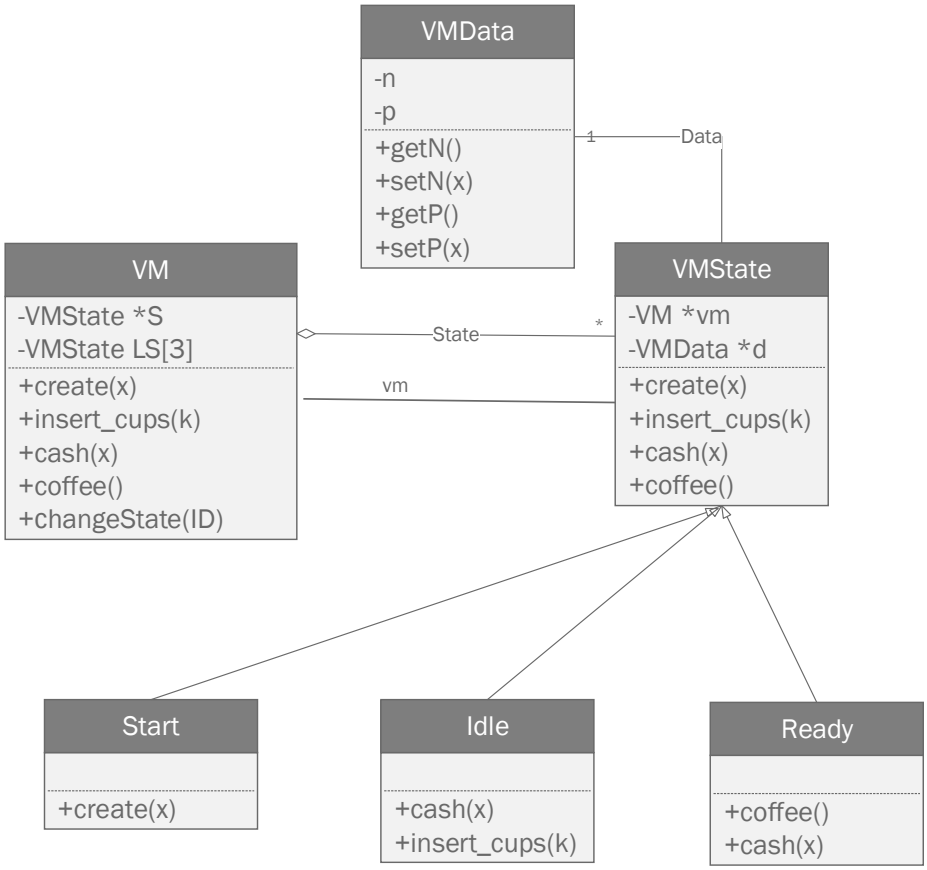
Develop a class diagram for the document system using the **Whole-Part design pattern**.

Provide a class diagram for the document system. Identify operations and major attributes for each class. Identify necessary changes to the design when a new type of document is incorporated into the document system. Notice that required changes should be **minimal**.

Note: You do not have to specify the operations in classes.

Problem #1

De-centralized State Pattern



Class "VM"

S //points to current state object
LS[0] //points to "Start" Object
LS[1] //points to "Idle" Object
LS[2] //points to "Ready" Object

S = LS[0] // initialize state object to "Start"

Operations

changeState(ID){
S = LS[ID]
}

create(x){
S->create(x)
}

```
insert_cups(k){
    S->insert_cups(k)
}

cash(x){
    S->cash(x)
}

coffee(){
    S-> coffee()
}
```

Class "VMState"

Operations

create(), insert_cups(), cash() and coffee() are abstract operations

Class "Start"

Operations

```
create(x){
    d->setN(0)
    d->setP(x)
    vm->changeState(1)          //change VM state from "Start" to "Idle"
}
```

Class "Idle"

Operations

```
cash(x){
    IF ( x >= d->getP() ) && (d->getN() > 0 ) THEN
        vm ->changeState(2)     //change VM state from "Idle" to "Ready"
    ELSE IF ( x < d->getP() ) || (d->getN() < 1 ) THEN
        return cash
    ENDIF
}

insert_cups(k){
    IF  k > 0 THEN
        numberOfCups = d->getN()
        numberOfCups = numberOfCups + k
```

```
        d->setN(numberOfCups)
    ENDIF
}
```

Class "Ready"

Operations

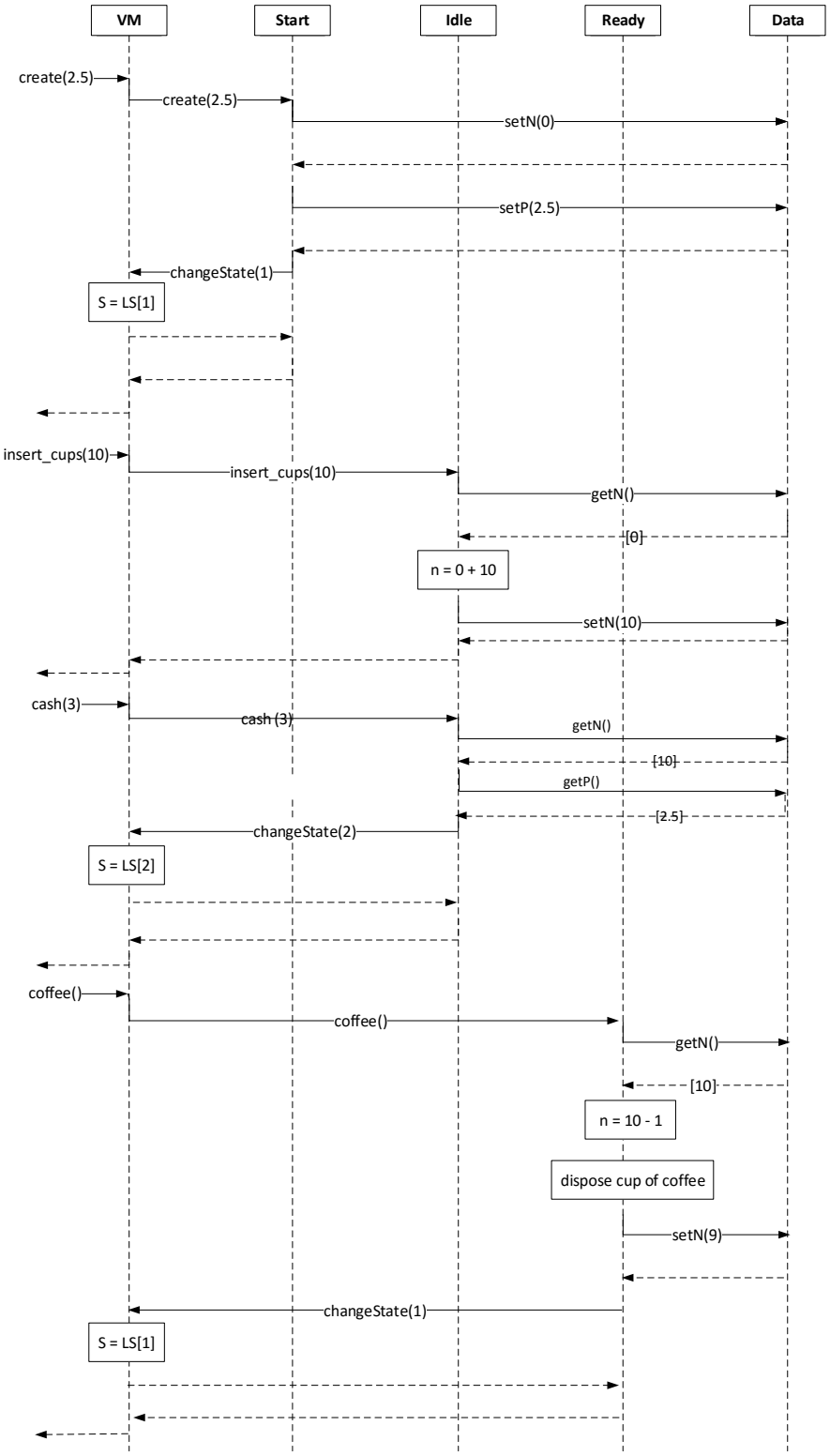
```
coffee(x){
    IF  d->getN() > 0 THEN
        numberOfCups = d->getN()
        numberOfCups = numberOfCups - 1
        d->setN(numberOfCups)
        dispose cup of coffee
        vm->changeState(1)      //change VM state from "Ready" to "Idle"
    ENDIF
}

cash(x){
    return cash
}
```

Class "VMData"

```
n       // number of cups
p       // price


getN(){
   return n
}

setN(int x){
    n = x
}


getP(){
   return p
}

setP(int x){
    p = x
}
```

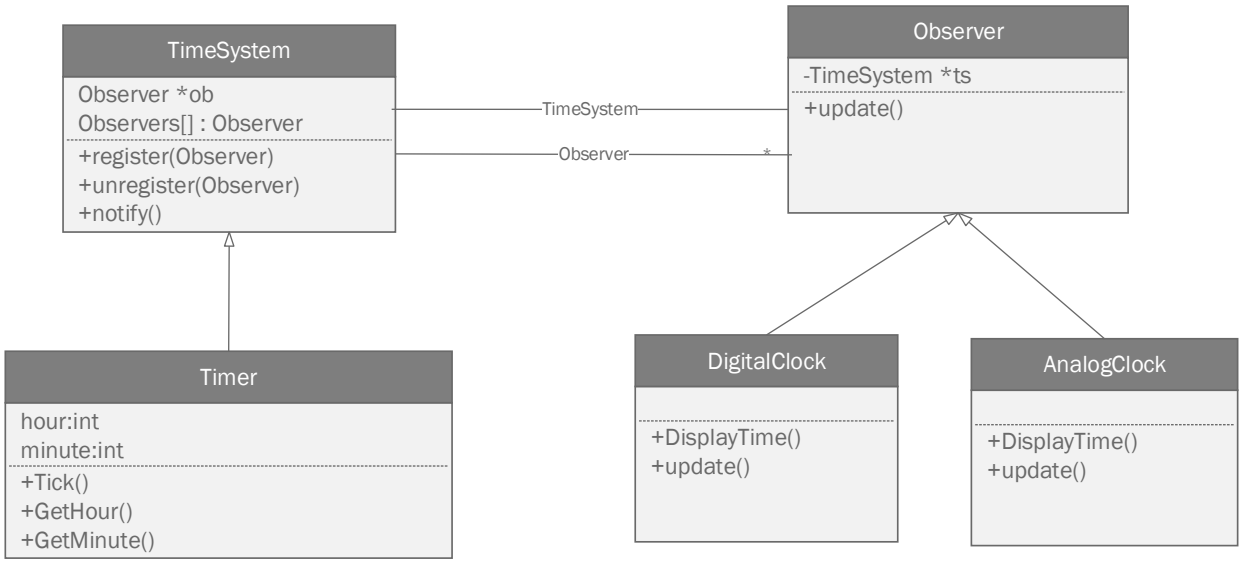
De-centralized Pattern – Sequence Diagram

create(2.5), insert_cups(10), cash(3), coffee()



Problem 2 Solution:

Class Diagram:



Pseudocode:

Class TimeSystem:

Operations:

register(Observer)

Add Observer to Observers[]

unregister(Observer)

Remove Observer from Observers[]

notify()

For each observer ob in Observers[]

ob.update()

Class Timer:

Operations:

Tick()

```
{
                second++;
                if (second == 60 ) {
                        second = 0;
                        minute++;
                        if (minute == 60) {
                                minute =0
                                if(hour == 23)
                                        hour =0
                                else
                                        hour++;
                        }
                        notify();
                }
    }
```

GetHour()

return hour in 24 hr format

GetMinute()

return minute

Class Observer: DigitalClock/AnalogClock

Operation

Update()

if observer is interested in new time then

call DisplayTime()

DisplayTime()

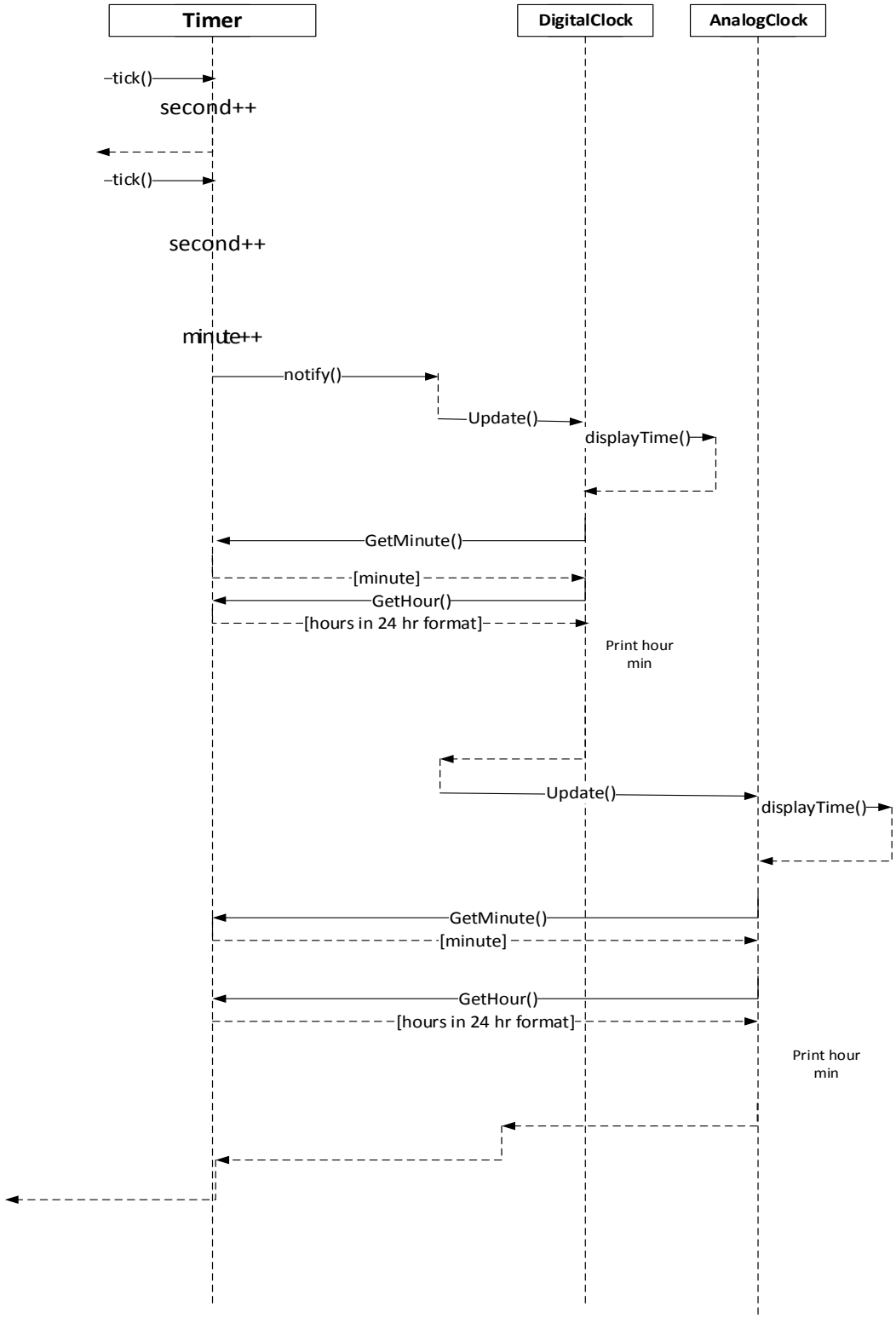
ts->getMinute()

ts->getHour()

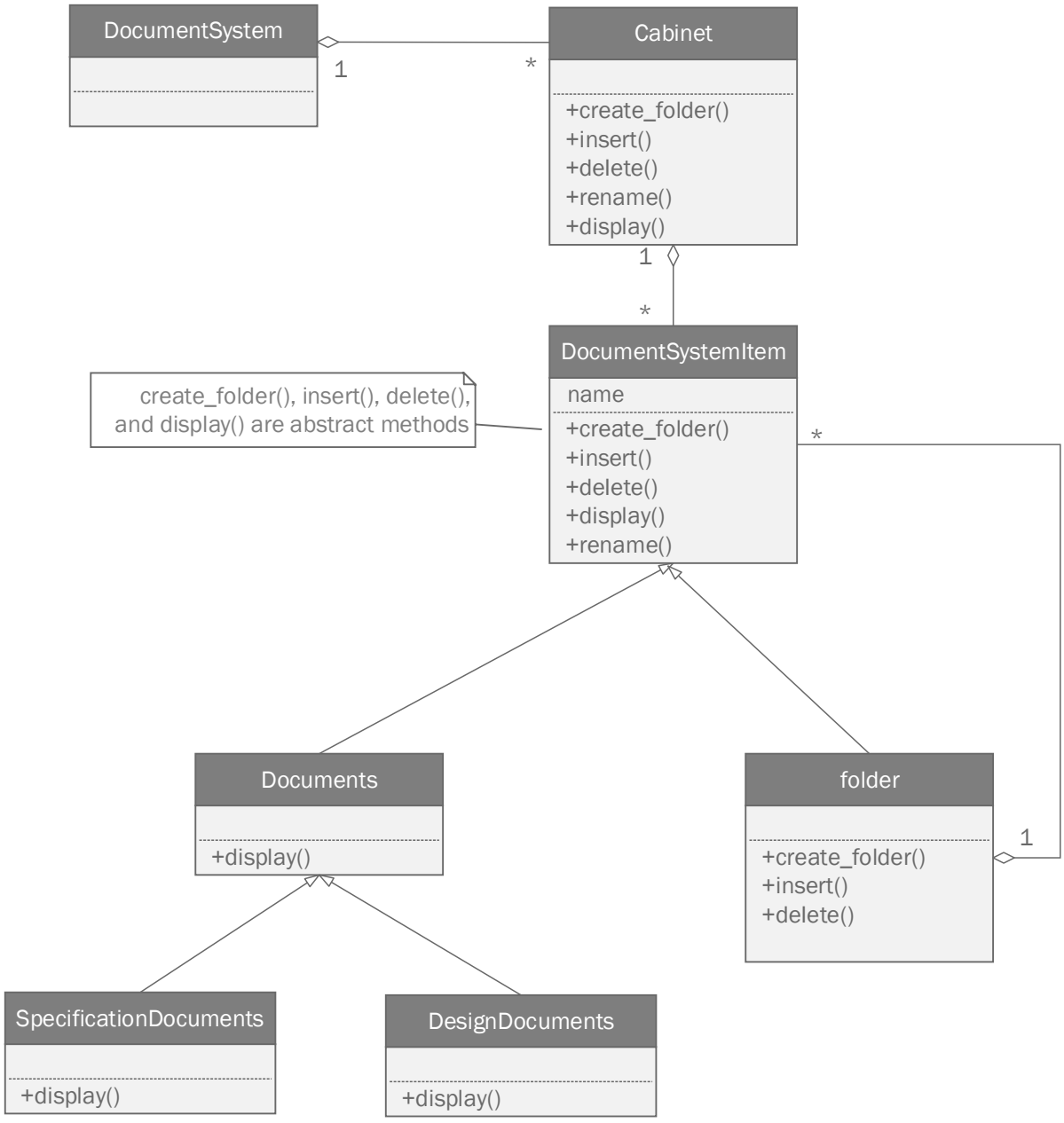
print hour min

Sequence Diagram:

Notifying the registered clock components about the time change:



Problem #3


DocumentSystem
1
*
Cabinet
+create_folder()
+insert()
+delete()
+rename()
+display()
1
*
DocumentSystemItem
name
+create_folder()
+insert()
+delete()
+display()
+rename()
create_folder(), insert(), delete(), and display() are abstract methods
*
Documents
+display()
folder
+create_folder()
+insert()
+delete()
1
SpecificationDocuments
+display()
DesignDocuments
+display()