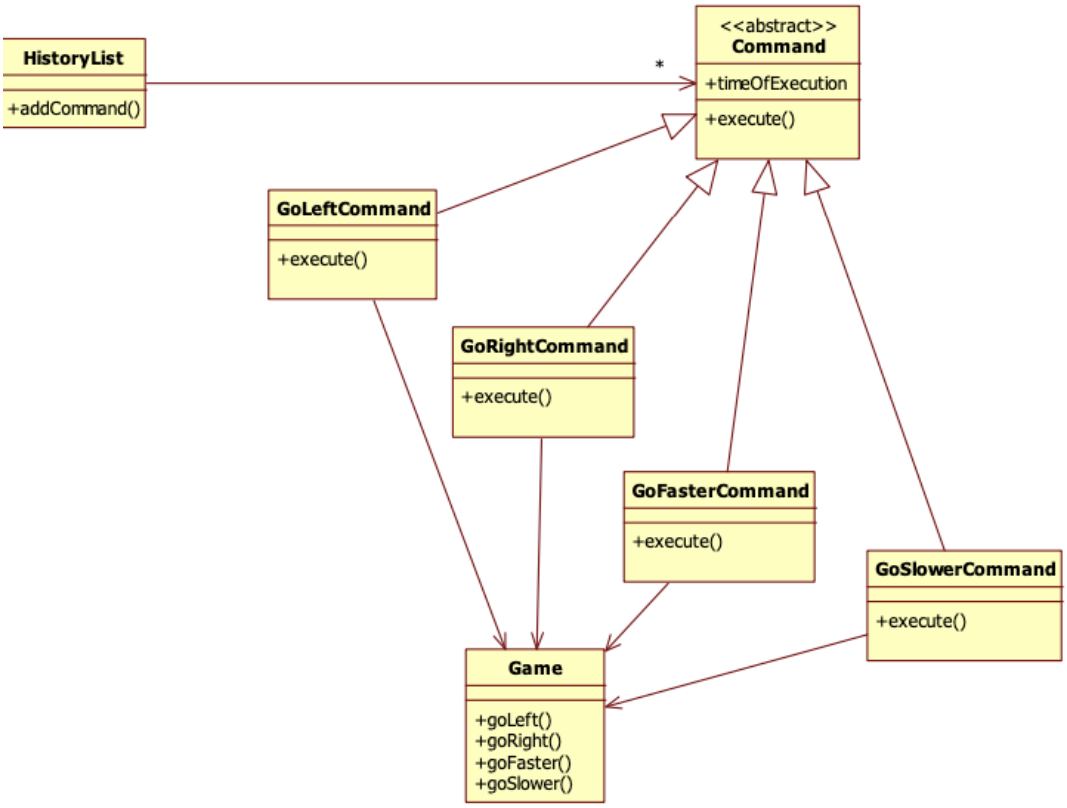
Question 1 [15 points] {15 minutes}

Suppose we have a game that allows us to race with a car. The user can perform the following actions: go faster, go slower, go left and go right. We want to add functionality to record a whole race such that we can replay the recorded race.

a. Which pattern do you choose to implement this functionality?
b. Draw the partial class diagram that shows how your design works. Do NOT draw the whole class diagram of the race game, but only the class diagram that shows how your design works. Make sure your class diagram contains all the important information to communicate your design.

RESULT 1

a. Command pattern
b.



Question 2 [15 points] {15 minutes}

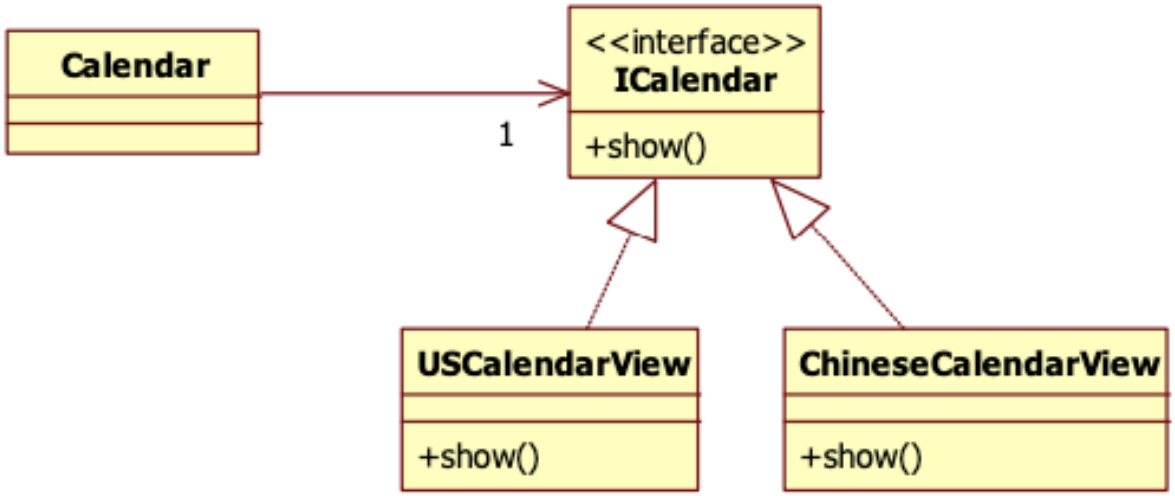
Suppose that you have to write a program that displays calendars. One of the requirements for the program is that it be able to display holidays celebrated by different nations and different religious groups. The user must be able to specify which sets of holidays to display

a. Which pattern do you choose to implement this functionality?
b. Draw the partial class diagram that shows how your design works. Do NOT draw the whole class diagram of the calendar application, but only the class diagram that shows how your design works. Make sure your class diagram contains all the important information to communicate your design.

RESULT 2

a. Strategy pattern

b.



Question 3 [15 points] {15 minutes}

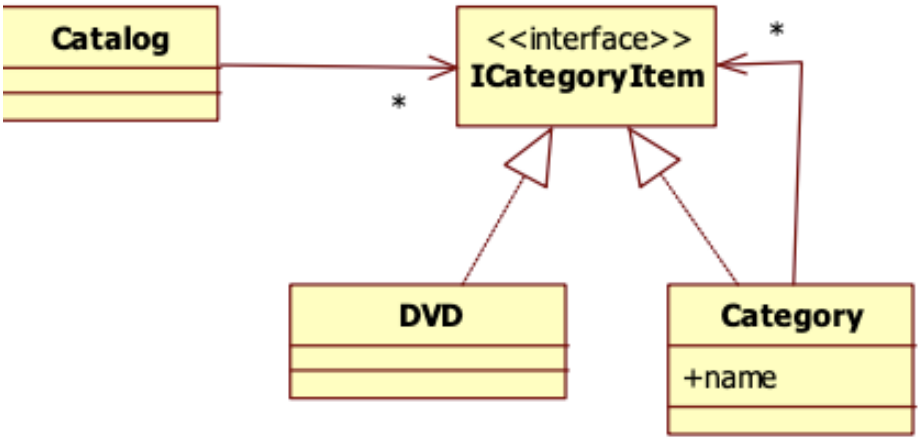
Suppose you have to implement an online DVD rental application. The application allows the user to browse the DVD catalog. The DVD catalog consists of different categories. We have for example the categories "new releases", "drama", "Sports", etc. Categories can also have sub-categories, and sub-categories can have other sub-categories. For example, within the category drama, we can have sub-categories like "new releases", "2012", "2011", "2005-2010", etc. The same DVD can also be part of multiple categories.

a. Which pattern do you choose to implement this functionality?
b. Draw the partial class diagram that shows how your design works. Do NOT draw the whole class diagram of the DVD rental application, but only the class diagram that shows how your design works. Make sure your class diagram contains all the important information to communicate your design.

RESULT 3

a. Composite pattern

b.



Question 4 [25 points] {40 minutes}

Suppose you have to maintain a garage door controller application. The garage door controller application has the following requirements:
We have a remote control with only one button. Pressing this button results in the following action:

Current door state	Pressing the button results in the following actions
closed	The door is opening
open	The door is closing
closing	The door is opening
opening	The door is closing

We also have a door sensor that signals our application if the door is completely closed, or opened.
We also have a flashing alarm light with the following behavior:

Current door state	Behavior of the flashing alarm light
closed	Light is off
open	Light is off
closing	Light flashes yellow
opening	Light flashes orange

Someone else wrote the following program:

```
public class Light {
	public void flash(String color){
		System.out.print(", Light: flash "+color);
	}
	public void stop(){
		System.out.print(", Light: stop ");
	}
}

public class Remote {
	private Door door;

	public Remote(Door door) {
		this.door = door;
	}
	public void pressButton(){
		door.clickRemoteButton();
	}
}

public class Sensor {
	private Door door;

	public Sensor(Door door) {
		this.door = door;
	}
	public void signal(String signal){
		door.sensorSignal(signal);
	}
}
```

```
public class Door {
	private String state = "closed";
	private Light light;

	public Door(Light light) {
		this.light = light;
	}

	public void clickRemoteButton() {
		System.out.print("old state = " + state);
		if (state.equals("closed") || state.equals("closing")) {
			state = "opening";
			light.flash("orange");
		} else if (state.equals("open") || state.equals("opening")) {
			state = "closing";
			light.flash("yellow");
		}
		System.out.println(",  new state = " + state);
	}

	public void sensorSignal(String signal) {
		System.out.print("old state = " + state);
		if (signal.equals("close")) {
			state = "closed";
		} else if (signal.equals("open")) {
			state = "open";
		}
		light.stop();
		System.out.println(",  new state = " + state);
	}
}

public class Application {
	public static void main(String[] args) {
		Light light = new Light();
		Door door = new Door(light);
		Remote remote = new Remote(door);
		Sensor sensor = new Sensor(door);

		remote.pressButton(); //opening
		sensor.signal("open"); //door is open
		remote.pressButton(); // closing
		sensor.signal("close"); // door is close
		remote.pressButton(); //opening
		remote.pressButton(); //closing
		remote.pressButton(); //opening
		sensor.signal("open"); //door is open
	}
}
```

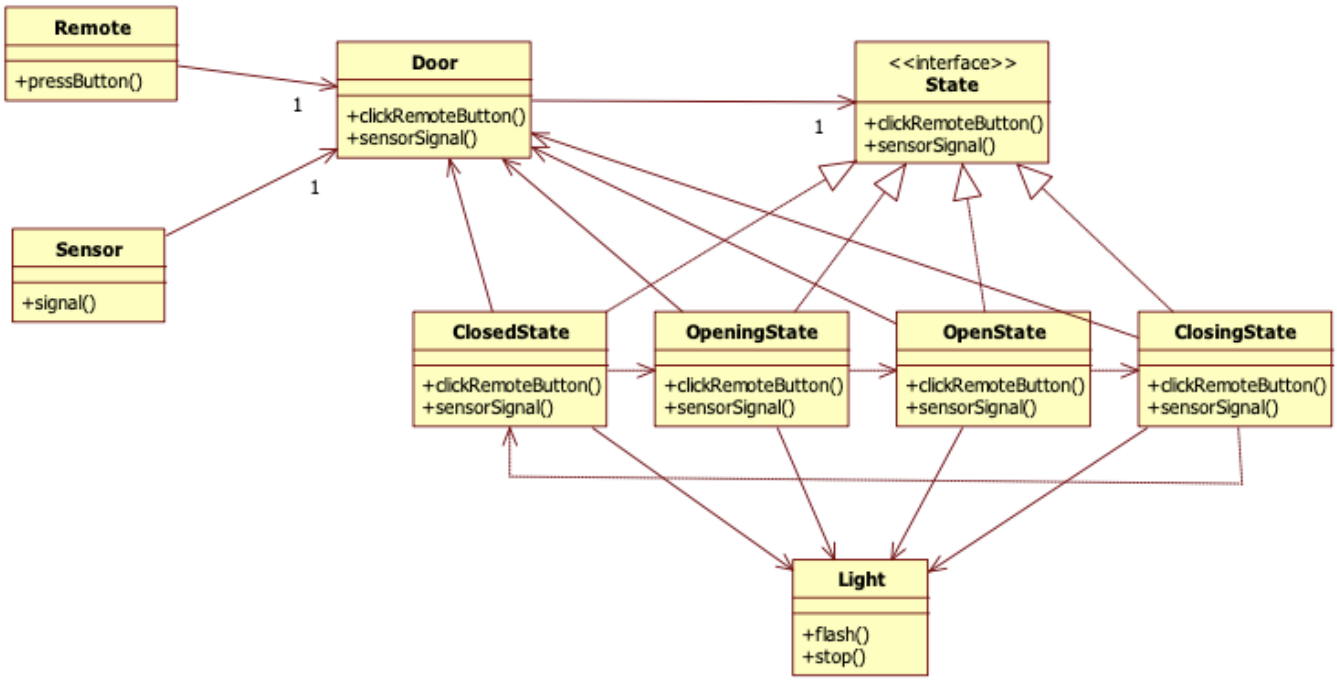
Running this application gives the following output:

```
old state = closed, Light: flash orange,  new state = opening
old state = opening, Light: stop ,  new state = open
old state = open, Light: flash yellow,  new state = closing
old state = closing, Light: stop ,  new state = closed
old state = closed, Light: flash orange,  new state = opening
old state = opening, Light: flash yellow,  new state = closing
old state = closing, Light: flash orange,  new state = opening
old state = opening, Light: stop ,  new state = open
```

The problem with this application is that all logic is in the Door class. Suppose we add another state to the door. Suppose we want to add the state "**75%open**" where the door is only for 75% open, which is open enough for children, or for a normal car to enter or leave the garage. Then we have to modify all the complex **if-then** structures in the Door class. You have learned the state pattern in this course, which makes it easier to add new states. Redesign this application such that it is easier to add new states.

a. Draw the class diagram that shows how your design works. Make sure your class diagram contains all the important information to communicate your design. You are not allowed to make changes the following classes: Application, Light, Sensor and Remote.
b. Write the code of the following 2 classes
    1. The Door class
    2. One implementation of a State class (only one!)

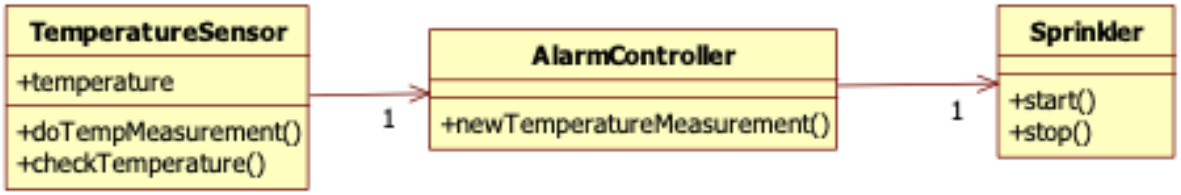
RESULT 4

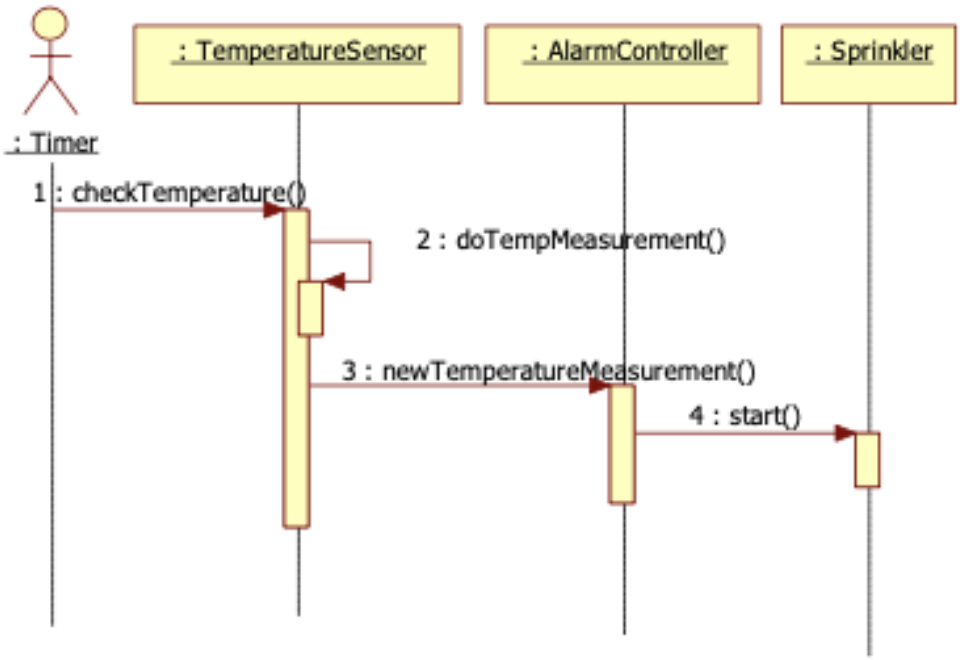

Remote
+pressButton()
Door
+clickRemoteButton()
+sensorSignal()
<<interface>>
State
+clickRemoteButton()
+sensorSignal()
Sensor
+signal()
1
1
1
ClosedState
+clickRemoteButton()
+sensorSignal()
OpeningState
+clickRemoteButton()
+sensorSignal()
OpenState
+clickRemoteButton()
+sensorSignal()
ClosingState
+clickRemoteButton()
+sensorSignal()
Light
+flash()
+stop()

```
public class Door {
	private State state;
	private Light light;

	public Door(Light light) {
		this.light = light;
		state = new ClosedState(this,light);
	}
	public void clickRemoteButton() {
		state.clickRemoteButton();
	}
	public void sensorSignal(String signal) {
		state.sensorSignal(signal);

	}
	public void setState(State state) {
		this.state = state;
	}
}

public class ClosingState implements State{
	private Door door;
	private Light light;

	public ClosingState(Door door, Light light) {
		this.door = door;
		this.light=light;
		light.flash("yellow");
	       System.out.println(", new state = closing");
	}

	    public void clickRemoteButton() {
		System.out.print("old state = closing ");
		door.setState(new OpeningState(door,light));
	    }

	    public void sensorSignal(String signal) {
		System.out.print("old state = closing ");
		if (signal.equals("close"))
			door.setState(new ClosedState(door,light));
	    }
}
```

Question 5 [25 points] {35 minutes}

Suppose we implemented an alarm system in the following way:





Here are some code snippets of the implementation:

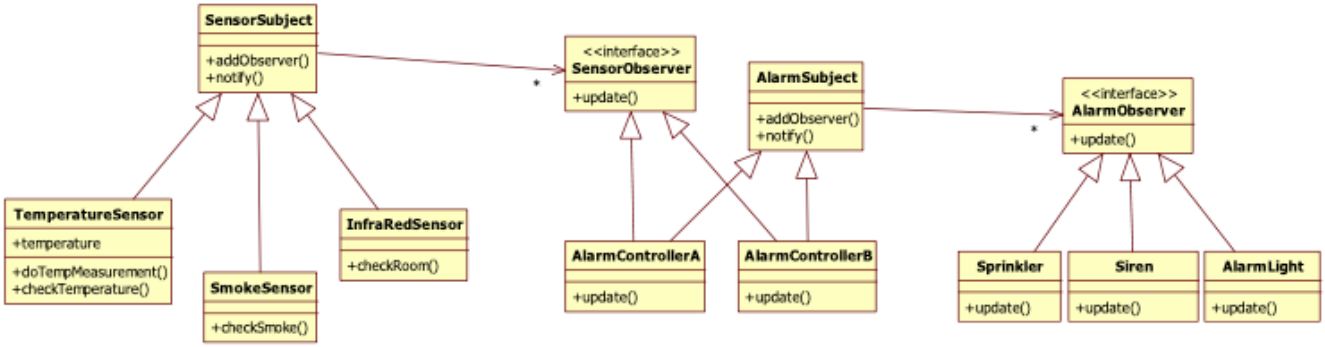
```
public class TemperatureSensor {
        private int temperature;
        private AlarmController alarmController;

        public void checkTemperature(){
                temperature=doTempMeasurement();
                alarmController.newTemperatureMeasurement(temperature);
        }

        public int doTempMeasurement(){
                // logic to do the measurement is omitted
                return 178;
        }

...

public class AlarmController {
        private Sprinkler sprinkler;


        public void newTemperatureMeasurement(int temperature){
                if (temperature > 160){
                        sprinkler.start();
                }
        }
...
```

Now we are asked to extend our alarm system with the following requirements:

1. The alarm system should support **multiple sensors**, like Infra-red sensors, smoke sensors, sound sensors, etc.
2. The current TemperatureSensor knows about the AlarmController, so we can use this TemperatureSensor only together with this AlarmController. In our new design we want all **our sensors to be independent of any AlarmController**, so we can reuse these sensors for any other possible system
3. We should be able to connect a sensor to multiple Alarm controllers.
4. The alarm system should support **multiple alarm signaling devices**, like a siren, alarm lights, SMS senders that send an SMS in case of an alarm, sprinklers, etc.
5. The current AlarmController knows about the Sprinkler. This is not desirable because we want to be able to use multiple signaling devices without changing the AlarmController. In our new design **the AlarmController should be independent of any alarm signaling device**.

Draw the class diagram of your design. Make sure your class diagram contains all the important information to communicate your design.

RESULT 5


SensorSubject
+addObserver()
+notify()
*
<<interface>>
SensorObserver
+update()
AlarmSubject
+addObserver()
+notify()
*
<<interface>>
AlarmObserver
+update()
TemperatureSensor
+temperature
+doTempMeasurement()
+checkTemperature()
SmokeSensor
+checkSmoke()
InfraRedSensor
+checkRoom()
AlarmControllerA
+update()
AlarmControllerB
+update()
Sprinkler
+update()
Siren
+update()
AlarmLight
+update()