Façade Pattern

Design Patterns

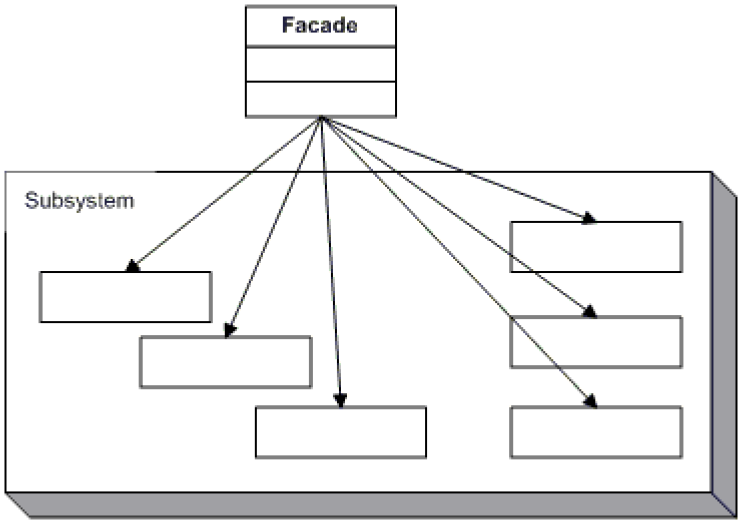
Christopher Doseck

9/14/2016

Introduction

This assignment is an application that I created that shows how the Façade Pattern works. In this application I am using a façade class to hold all of the other classes in it, in order to use them. Each of these classes represents a part of my dorm room, and the façade class represents a universal remote that changes the properties of these parts of my dorm room.

The UML Diagram for Façade



The UML diagram for the façade pattern, shown on the right, shows the classes that are needed to have the requirements. I had the universal remote class, which was the façade class, and I had many subsystems for it. The table below shows how each of the classes were used.

Form	This form was used to show that the façade pattern was actually working.
Room Remote	This class was the Façade class of the UML diagram. It contained an object of each of the rest of the classes that are shown in this table.
Alarm	This class was used to demonstrate an alarm, which would be a subsystem of the room. It could be given a time, turned on and off, checked if it was on, and check what time it was set for.
Light	This class was used to demonstrate lights, which would be a subsystem of the room. This could be checked if it was on, or turned on and off.

Temperature Gauge	This class was used to demonstrate a temperature gauge, which would be a subsystem of the room. This could be set to a temperature, including off, and could get the temperature that it is set at.
TV	This class was used to demonstrate a TV, which would be a subsystem of the room. This could be turned on and off, checked if it was on, given and input, and checked for which input it was on.

Narrative

Alarm

```
public class Alarm
{
    bool isAlarmOn = false;
    private double alarmTime;

    public Alarm()
    {
    }

    public void turnOff()
    {
        isAlarmOn = false;
    }

    public void setAlarm(double time)
    {
        isAlarmOn = true;
        alarmTime = time;
    }

    public bool isOn()
    {
        return isAlarmOn;
    }

    public double getTime()
    {
        return alarmTime;
    }
}
```

This alarm class is one of the subsystems of the RoomRemote class. This class symbolizes an alarm. It can be set by calling setAlarm(double time) where time is the time that you want to go off, and when this is called it also tells the alarm to turn on. This class can also turn the alarm off, and get whether it is on, and return the time that it is set for.

Lights

```
public class Lights
{
    private bool lightsOn = false;
    public Lights()
    {
    }

    public bool areOn()
    {
        return lightsOn;
    }
```

This alarm class is one of the subsystems of the RoomRemote class. This class symbolizes lights. These lights can be turned on and off, and can be checked if they are on.

```
    public void turnOn()
    {
        lightsOn = true;
    }

    public void turnOff()
    {
        lightsOn = false;
    }
}
```

TemperatureGauge

```
public class TemperatureGauge
{
    private string tempSetting = "off";

    public TemperatureGauge()
    {
    }

    public string getTemp()
    {
        return tempSetting;
    }

    public void setTemp(string temp)
    {
        tempSetting = temp;
    }
}
```

This alarm class is one of the subsystems of the RoomRemote class. This class symbolizes a temperature gauge. In this class, the temperature setting, tempSetting, is stored as a string, and can be retrieved through getTemp(), and changed through setTemp().

TV

```
public class TV
{
    private bool tvOn = false;
    private string input = "cable";

    public TV()
    {
    }

    public void setInput(string input)
    {
        this.input = input;
    }

    public string getInput()
    {
        return input;
    }

    public void turnOn()
    {
        tvOn = true;
    }
```

This alarm class is one of the subsystems of the RoomRemote class. This class symbolizes a TV. This class stores the input of the TV in a string, and allows you to change it through setInput(string input), and check it through getInput(). It also allows you to turn the TV on and off, and check if it is on.

```
    public void turnOff()
    {
        tvOn = false;
    }

    public bool isOn()
    {
        return tvOn;
    }
}
```

Room Remote

```
public class RoomRemote
{
    public Alarm alarm = new Alarm();
    public Lights bedroomLights = new Lights();
    public Lights livingRoomLights = new Lights();
    public TemperatureGauge temperatureGauge = new TemperatureGauge();
    public TV tv = new TV();

    public RoomRemote()
    {
    }

    public void wakeUp()
    {
        alarm.turnOff();
        bedroomLights.turnOn();
        temperatureGauge.setTemp("Off");
    }

    public void leaveRoom()
    {
        temperatureGauge.setTemp("Off");
        bedroomLights.turnOff();
        livingRoomLights.turnOff();
        tv.turnOff();
    }

    public void watchTV()
    {
        livingRoomLights.turnOn();
        bedroomLights.turnOff();
        temperatureGauge.setTemp("Off");
        tv.turnOn();
        tv.setInput("Cable");
    }

    public void study()
    {
        livingRoomLights.turnOff();
        bedroomLights.turnOn();
        temperatureGauge.setTemp("Low Warm");
        tv.turnOff();
    }

    public void watchMovie()
```

This class is the Façade class of the Façade Pattern. It represents a remote that controls the objects in the room. The first thing that happens in this class is to create all of the objects that are in the room. Next in the code is all of the method that get called to do something such as watching TV. This will change the properties of the objects that should be changed when you want to do that action. In the watching TV example, it turns on living room lights, turns off bedroom lights, turns off the temperature gauge, turns the TV on, and sets the TV's input to cable. This shows how all of these subsystems are controlled by one class, just as in the UML diagram of the Façade Pattern.

```
    {
        livingRoomLights.turnOff();
        bedroomLights.turnOff();
        temperatureGauge.setTemp("Off");
        tv.turnOn();
        tv.setInput("Xbox");
    }

    public void goToBed(double time)
    {
        livingRoomLights.turnOff();
        bedroomLights.turnOff();
        alarm.setAlarm(time);
        temperatureGauge.setTemp("Cold");
        tv.turnOff();
    }
}
```

Form1

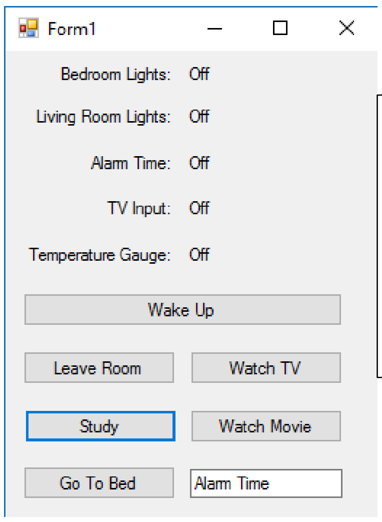
```
public partial class Form1 : Form
{
    RoomRemote remote;
    public Form1()
    {
        InitializeComponent();
        remote = new RoomRemote();
    }

    private void btnWakeUp_Click(object sender, EventArgs e)
    {
        remote.wakeUp();
        updateInfo();

        btnLeaveRoom.Enabled = true;
        btnStudy.Enabled = true;
        btnWatchMovie.Enabled = true;
        btnWatchTV.Enabled = true;
    }

    private void updateInfo()
    {
        if (remote.bedroomLights.areOn())
            bedroomLightsLabel.Text = "On";
        else
            bedroomLightsLabel.Text = "Off";

        if (remote.livingRoomLights.areOn())
            livingRoomLightsLabel.Text = "On";
        else
            livingRoomLightsLabel.Text = "Off";

        if (remote.alarm.isOn())
            alarmLabel.Text = remote.alarm.getTime().ToString();
        else
            alarmLabel.Text = "Off";

        if (remote.tv.isOn())
            tvLabel.Text = remote.tv.getInput();
```

This class is the GUI for the remote. The first thing that happens in this code is that a remote is created. When one of the buttons is clicked, it calls the corresponding method in the remote is called. Then it calls the update info button. In the wake up button method, it sets all of the buttons to be enabled.

The update info method gets all of the information from the remote, which gets all of the information from the objects in it, to see what all of the properties are, so they can be updated on the GUI.

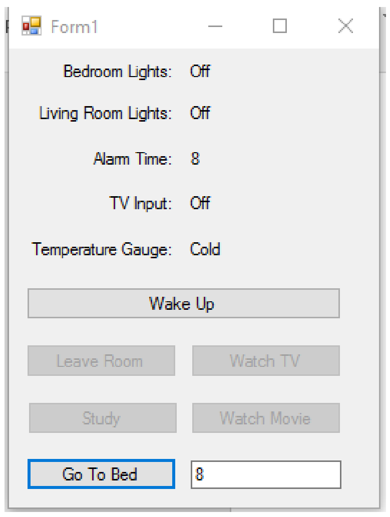
```
            else
                tvLabel.Text = "Off";

            temperatureLabel.Text = remote.temperatureGauge.getTemp();
        }

        private void btnLeaveRoom_Click(object sender, EventArgs e)
        {
            remote.leaveRoom();
            updateInfo();
        }

        private void btnWatchTV_Click(object sender, EventArgs e)
        {
            remote.watchTV();
            updateInfo();
        }

        private void btnStudy_Click(object sender, EventArgs e)
        {
            remote.study();
            updateInfo();
        }

        private void btnWatchMovie_Click(object sender, EventArgs e)
        {
            remote.watchMovie();
            updateInfo();
        }

        private void btnGoToBed_Click(object sender, EventArgs e)
        {
            remote.goToBed(Double.Parse(tbAlarmTime.Text));
            updateInfo();
            btnLeaveRoom.Enabled = false;
            btnStudy.Enabled = false;
            btnWatchMovie.Enabled = false;
            btnWatchTV.Enabled = false;
        }
    }
```

This method is what is called when you click go to bed. It gets the time from the alarm time textbox to pass to the remote, and sets all of the buttons to disabled, because you can't do any of these things between going to bed and waking up.



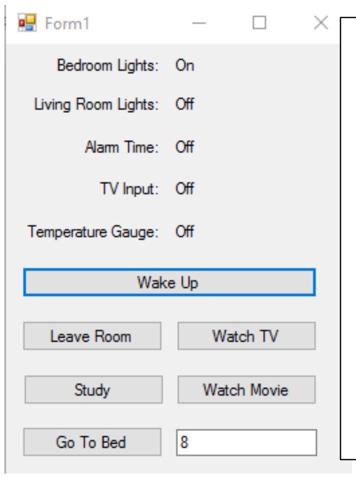
Bedroom Lights: Off
Living Room Lights: Off
Alarm Time: Off
TV Input: Off
Temperature Gauge: Off
Alarm Time

This is the initial setup of the GUI, before any buttons have been clicked.



Bedroom Lights: Off
Living Room Lights: Off
Alarm Time: 8
TV Input: Off
Temperature Gauge: Cold
8

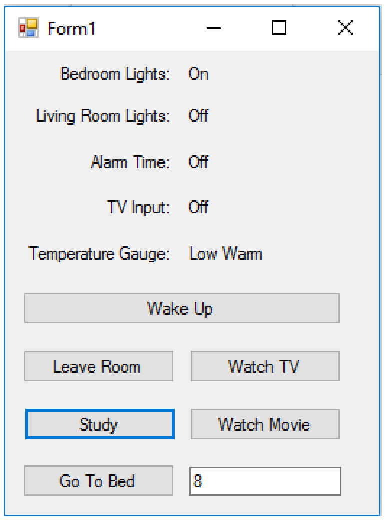
This is after go to bed has been clicked, and the time has been entered at 8. Notice how all of the buttons are disabled.



Form1

Bedroom Lights:	On
Living Room Lights:	Off
Alarm Time:	Off
TV Input:	Off
Temperature Gauge:	Off

After the wake up button has been pressed, all buttons are enabled again.



Form1

Bedroom Lights:	On
Living Room Lights:	Off
Alarm Time:	Off
TV Input:	Off
Temperature Gauge:	Low Warm

The study button has been pressed so you can see the changes.

Observations

This was a rather easy design pattern to do. There were no parts that I really struggled with. This pattern seems like it is really useful when using a bunch of similar classes that all can fall under one subsystem.