Jack Raney

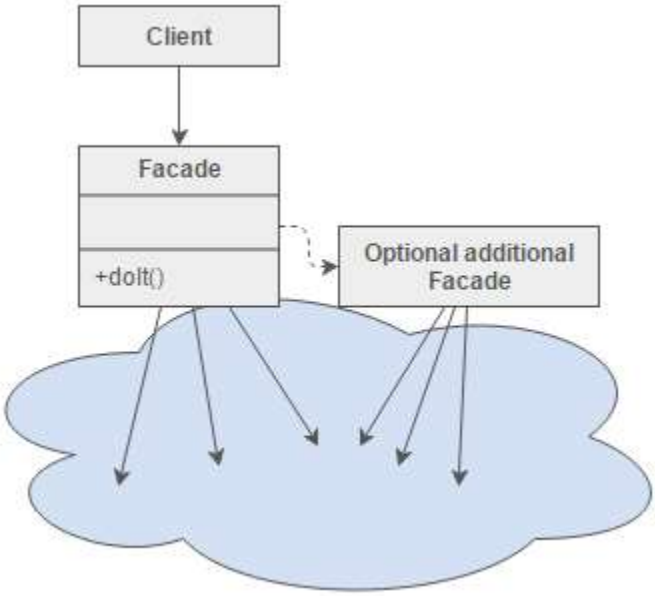
Design Patterns

9/11/2016

The Façade Pattern

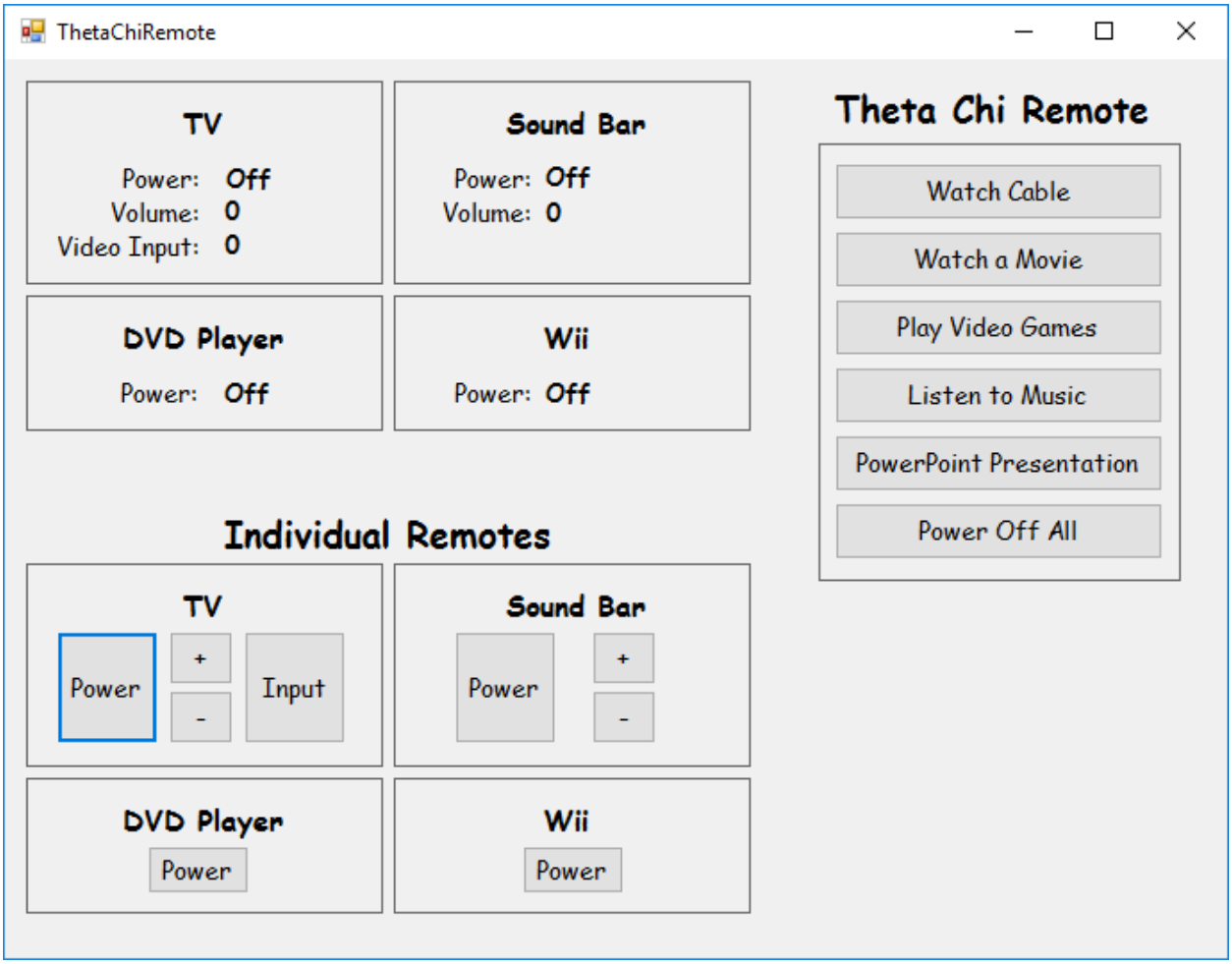
This assignment is to exemplify the Façade Pattern, a design pattern that, according to sourcemaking.com, is used to "provide a unified interface to a set of interfaces in a subsystem" and "wrap a complicated subsystem with a simpler interface". What this means is that this pattern takes a group of different objects and creates a "façade" that lets a user manipulate the objects in one simple interface, rather than using an interface for each object.

Sourcemaking.com also provides a UML diagram for the Façade pattern:



The cloud the arrows are pointing to is the subsystems the façade interacts with.

To showcase the Façade Pattern, I created the program "ThetaChiRemote", a simple simulation of the many devices used in the Theta Chi Hut's lobby:



The four objects (the TV, Sound Bar, DVD Player, and Wii) are of the Device class, which has the properties of a Boolean for if it is turned on and two functions that turn it on and off.

```
public class Device
    {
        public bool isOn = false;

        public void powerOn()
        {
            isOn = true;
        }

        public void powerOff()
        {
            isOn = false;
        }
    }
```

The TV and Sound Bar are part of the AudioDevice subclass, which adds a volume integer to the device, minimum and maximum values for the volume and a function to alter the volume.

```
public class AudioDevice : Device
    {
        public int volume = 0;
        public const int MIN_VOL = 0;
        public const int MAX_VOL = 100;

        public void adjustVolume(int newVol)
        {
            if (newVol < MIN_VOL)
            {
                newVol = MIN_VOL;
            }

            if (newVol > MAX_VOL)
            {
                newVol = MAX_VOL;
            }
            volume = newVol;
        }
    }
```

Because the TV also has a video input integer (for switching different inputs to watch movies, etc.) it has its own Television class that adds that.

```
public class Television : AudioDevice
    {
        public int vidInput = 0;

        public void adjustVidInput()
        {
            vidInput++;
            if (vidInput >= 3){
                vidInput = 0;
            }
        }
    }
```

The Remote class is the Façade class for this example. This class contains the four devices, 6 different functions to manipulate the devices for different occasions (more on them to come), and four preset volumes to be used in those functions.

```
public class Remote    //This is the Facade class
    {
        public Television tv = new Television();
        public AudioDevice soundBar = new AudioDevice();
        public Device dvdPlayer = new Device();
        public Device wii = new Device();

        const int GAME_VOL  = 25;
        const int CABLE_VOL = 30;
        const int MUSIC_VOL = 40;
        const int MOVIE_VOL = 50;
```

The watchCable function, for example, turns on the TV and the sound bar, turns off the DVD player and the Wii, mutes the TV because audio will be coming the sound bar, and sets the sound bar's volume to 30 (the CABLE_VOL value). The 5 other functions similarly adjust the devices for particular uses that would otherwise require more than one action by the user (multiple buttons and remotes).

```
public void watchCable()
        {
            tv.powerOn();
            soundBar.powerOn();
            dvdPlayer.powerOff();
            wii.powerOff();

            tv.adjustVolume(AudioDevice.MIN_VOL);
            while (!(tv.vidInput == 0))
            {
                tv.adjustVidInput();
            }
            soundBar.adjustVolume(CABLE_VOL);
        }
```

The program itself instantiates a Remote object, which comes with the four devices we need. The program also contains an onLabel and updateDevices function. updateDevices updates the labels in the form to the devices' values for each property; it uses onLabel to convert the Booleans for power into On/Off strings that the power labels can display.

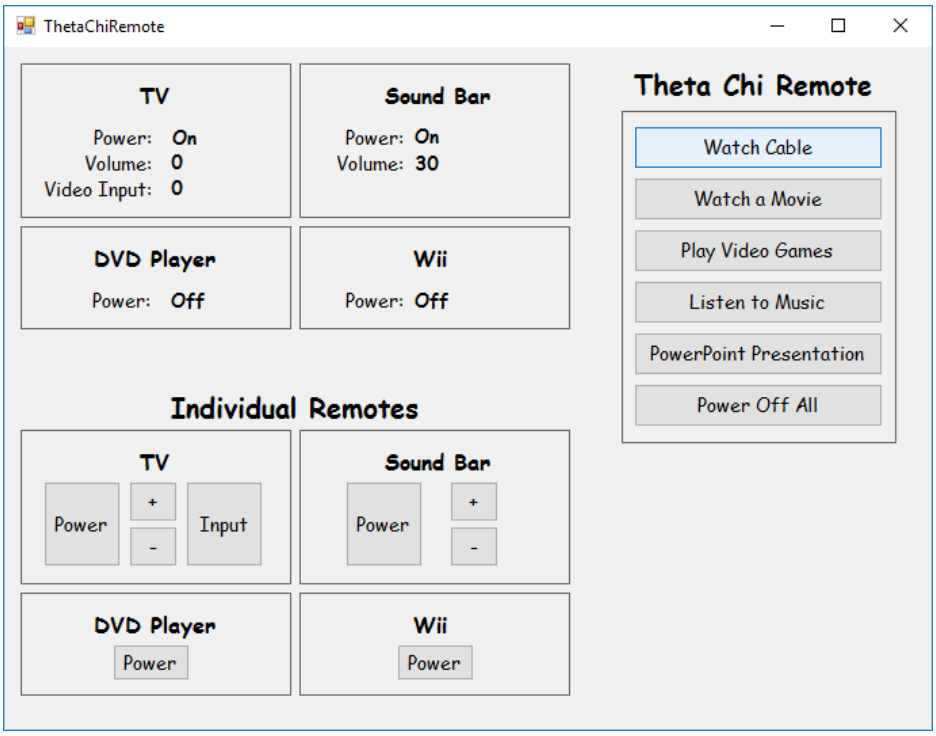
```
public partial class Form1 : Form
    {
        Remote remote = new Remote();

        public Form1()
        {
            InitializeComponent();
            updateDevices();
        }

        private string onLabel(Boolean isOn)
        {
            if (isOn)
                return "On";
            return "Off";
        }

        private void updateDevices()
        {
            tvPowerlabel.Text = onLabel(remote.tv.isOn);
            tvVolLabel.Text   = "" + remote.tv.volume;
            tvInputLabel.Text = "" + remote.tv.vidInput;

            soundBarPowerLabel.Text = onLabel(remote.soundBar.isOn);
            soundBarVolLabel.Text   = "" + remote.soundBar.volume;

            dvdPowerLabel.Text = onLabel(remote.dvdPlayer.isOn);
            wiiPowerLabel.Text = onLabel(remote.wii.isOn);
        }
```

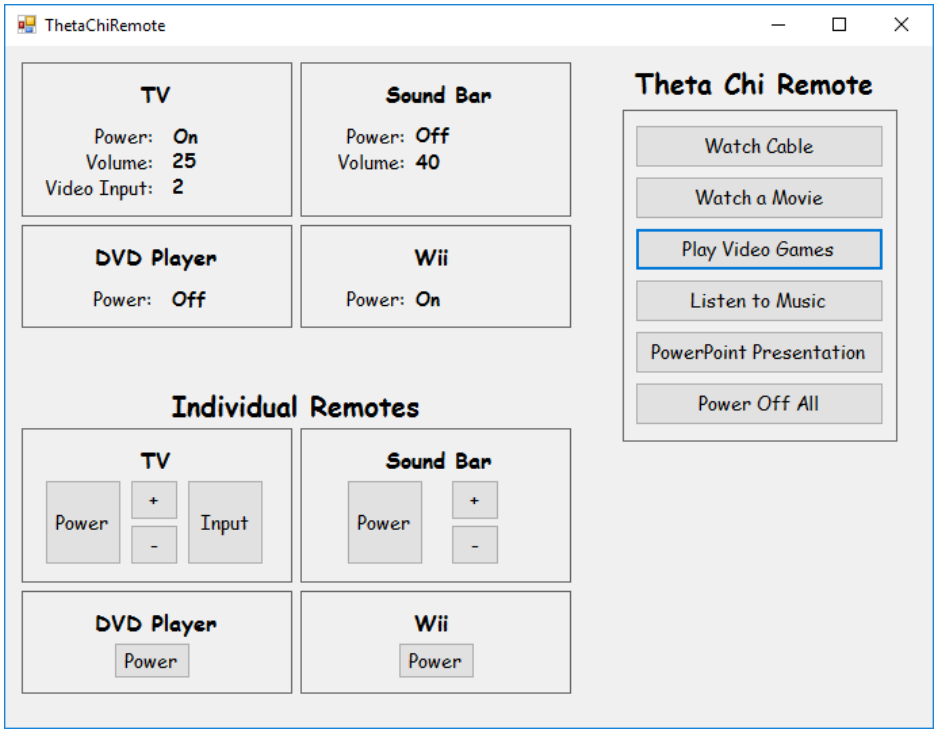
The 6 buttons on the side correlate to the 6 functions used to manipulate the devices and simply call the respective function and updateDevices.

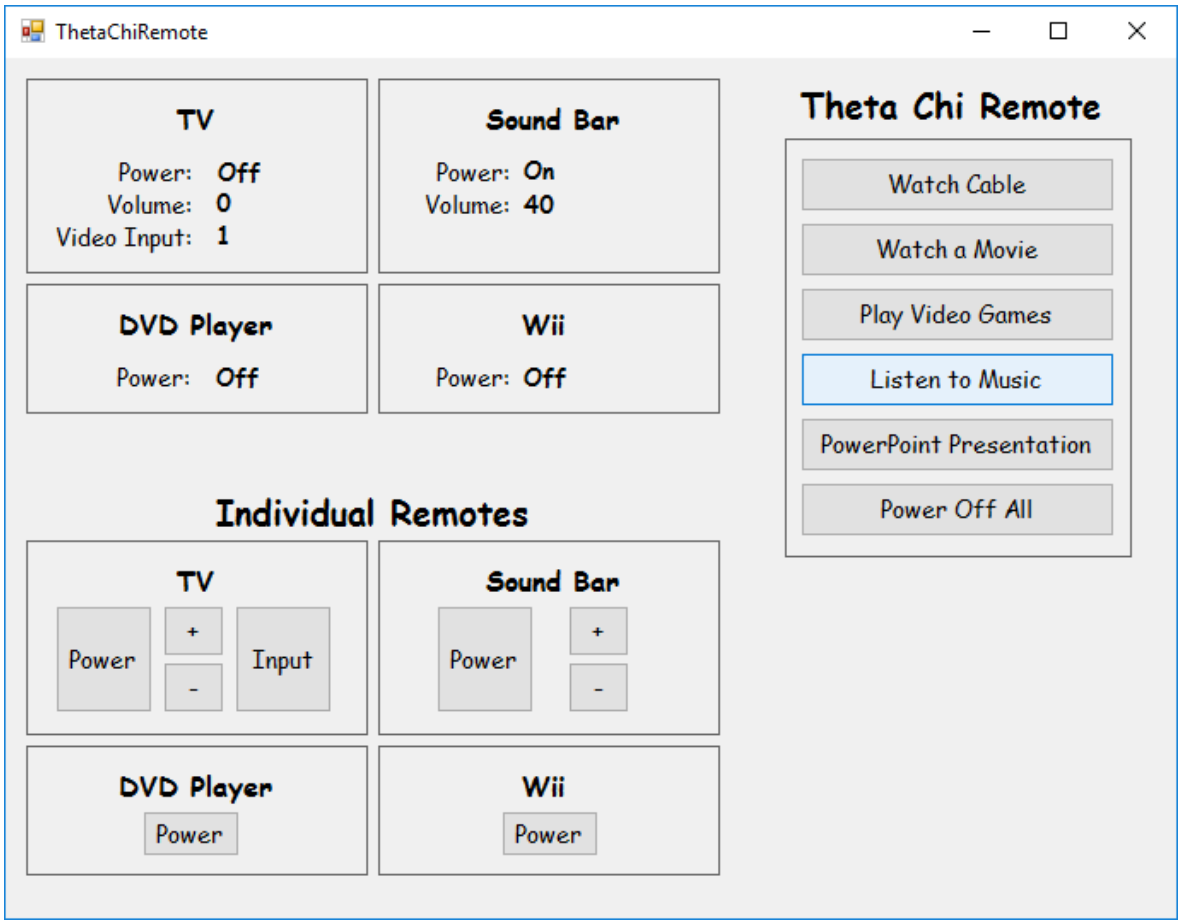
```
private void watchCableButton_Click(object sender, EventArgs e)
        {
            remote.watchCable();
            updateDevices();
        }

        private void watchMovieButton_Click(object sender, EventArgs e)
        {
            remote.watchMovie();
            updateDevices();
        }

        private void playGamesButton_Click(object sender, EventArgs e)
        {
            remote.playVideoGames();
            updateDevices();
        }

        private void musicButton_Click(object sender, EventArgs e)
        {
            remote.listenToMusic();
            updateDevices();
        }

        private void PresentationButton_Click(object sender, EventArgs e)
        {
            remote.watchPresentation();
            updateDevices();
        }

        private void OffButton_Click(object sender, EventArgs e)
        {
            remote.powerOffAll();
            updateDevices();
        }
```

The Individual Remotes section of the form manipulate the individual properties of the devices and is not part of the Façade Pattern. Therefore, they will not be given further detail; I will note, though, that comparing the usage of both the individual remotes and the Façade remote exemplifies very well why the Façade Pattern exists: it is much easier to use the Façade remote to adjust the values.

Screenshots of a few of the functions in action:






ThetaChiRemote
TV
Power: Off
Volume: 0
Video Input: 1
Sound Bar
Power: On
Volume: 40
DVD Player
Power: Off
Wii
Power: Off
Theta Chi Remote
Watch Cable
Watch a Movie
Play Video Games
Listen to Music
PowerPoint Presentation
Power Off All
Individual Remotes
TV
Power
+
-
Input
Sound Bar
Power
+
-
DVD Player
Power
Wii
Power

Reflection

This was a very simple pattern to execute, though the program's creation was a bit time consuming (especially the form). Because of this exercise I believe I have a good understanding of the Façade Pattern, which what really matters from this assignment anyway.