Cory Heitkamp

Façade Pattern

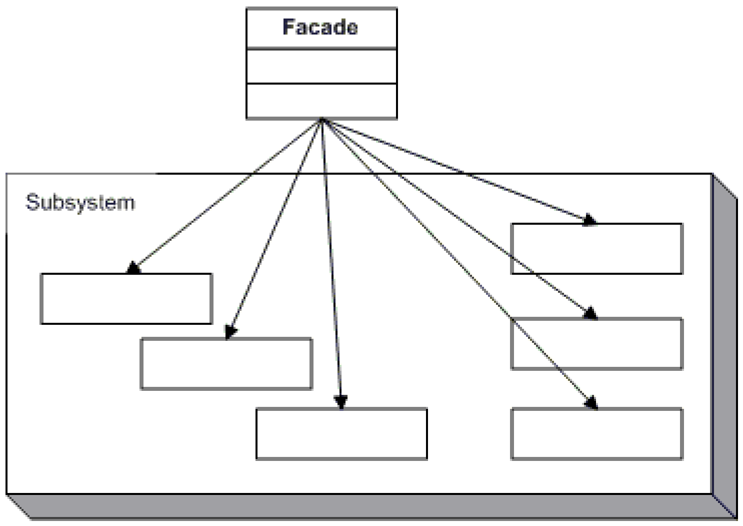
Design Patterns

9/13/16

Introduction:

For this the second design pattern of the year, we were tasked with designing an application that utilizes the Façade pattern. The Façade provides an interface for controlling a larger group of subsets. My application is a defense remote control, which controls the different positions of a football defense, and tells them what to do for different defenses.

UML Diagram: This UML Diagram seen below shows just how the Façade contains all of the subsystems that are being used, which in turn creates an interface to interact with those smaller pieces.



This table below summarizes the classes that I've created and used to make the facade pattern.

Ends	Defines the defensive end objects, and says what they are doing on each defense.
Tackles	Creates the defensive tackles objects, and says what they do on each defense.
Corners	Creates the corner players, and says their assignment on each defense.
Linebackers	Defines the linebacker position, and says what they do on each defense.
Safeties	Defines the safety position, and says what they don on each defense.
Façade	The Façade class contains objects of all of the above classes, and is used as the interface to use these classes and functions together in a uniform fashion.
Form	My windows form application.

Narrative:

When I began creating my program for the Facad pattern, I decided that the best place to begin was with the classes for each position, because that was the portion that I was the most confident in. The end class defines the position of defensive end. It contains two boolean values and functions, that determine if the player is rushing the quarterback, or dropping into coverage.

```
        public bool rush = false;
        public bool cover = false;

        public void de_rush()
        {
            rush = true;
            cover = false;
        }

        public void de_cover()
        {
            cover = true;
            rush = false;
        }
```

Lastly there is a function called deState, and it simply returns a string that says what the defensive end is doing. This string will later be used for display on the GUI.

```
    public string deState()
        {
            if (cover)
            {
                return "Drop Into Coverage";
            }
            if (rush)
            {
                return "Rush";
            }
            else return "Unassigned";
        }
```

The other position classes follow the same conventions as the end class. The defensive tackle class is very similar to the end class, the only difference will be in when they rush and when they cover.

```
    class Tackles
    {
        public bool rush = false;
        public bool cover = false;
        public void dt_rush()
        {
            rush = true;
            cover = false;
        }

        public void dt_cover()
        {
            cover = true;
            rush = false;
        }
        public string dtState()
        {
            if (cover)
            {
                return "Drop Into Coverage";
            }
            if (rush)
            {
                return "Rush";
            }
            else return "Unassigned";
        }
```

The linebacker class similarly has only two options, but has to choose between blitzing and covering, which is still essentially the same as the first two positions.

```
    public bool blitz = false;
    public bool cover = false;

    public void lb_blitz()
        {
            blitz = true;
            cover = false;
        }
```

With the corner class, things get slightly more elaborate, as the cornerback has many more coverage responsibilities than the first three.

```
        public bool man = false;
        public bool press = false;
        public bool blitz = false;
        public bool zone3 = false;
        public bool zone2 = false;
```

The conventions in the class are still the same, just a little bit longer.

```
        public void cb_zone2()
        {
            zone2 = true;
            man = false;
            press = false;
            blitz = false;
            zone3 = false;
        }
        public string cornerState()
        {
            if (man)
            {
                return "Man";
            }
            if (press)
            {
                return "Press";
            }
            if (blitz)
            {
                return "Blitz";
            }
            if (zone2)
            {
                return "Zone 2";
            }
            if (zone3)
            {
                return "Zone 3";
            }
            else return "Unassigned";
```

The safety class is very similar to the corner class, with just one more different kind of coverage being added. That would be the cover 1 option.

```
            blitz = true;
            man = false;
            zone3 = false;
            press = false;
            zone2 = false;
            zone1 = false;
```

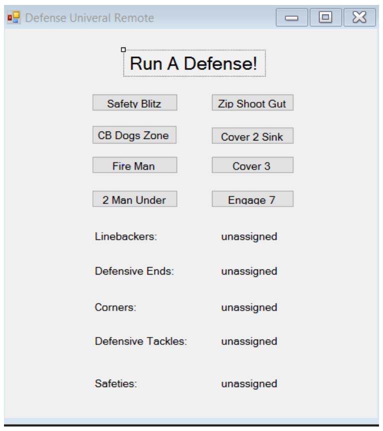
The next class is the Façade class. This class contains all of the other objects that have been defined.

```
class Facade
    {
        // Creates all the player objects in the Facade
        public Safeties safety = new Safeties();
        public Linebackers backers = new Linebackers();
        public Ends end = new Ends();
        public Tackles tackles = new Tackles();
        public Corners corner = new Corners();
```

The Façade class also contains the functions of all eight of the defenses that my remote can control. Each defense function contains the necessary functions for each player to be doing the correct assignment. It effectively creates an interface for all of the classes.

```
   public void SafetyBlitz()
        {
            safety.safety_blitz();
            backers.lb_cover();
            end.de_rush();
            tackles.dt_rush();
            corner.cb_man();
        }
  public void CBDogsZone()
        {
            safety.safety_zone3();
            backers.lb_cover();
            end.de_rush();
            tackles.dt_cover();
            corner.cb_blitz();
        }
```

Lastly, there is the form itself. I designed the GUI to have the available defenses on top, and display the results below.



In the form code, a new Façade called myFacade is made when the components are initialized.

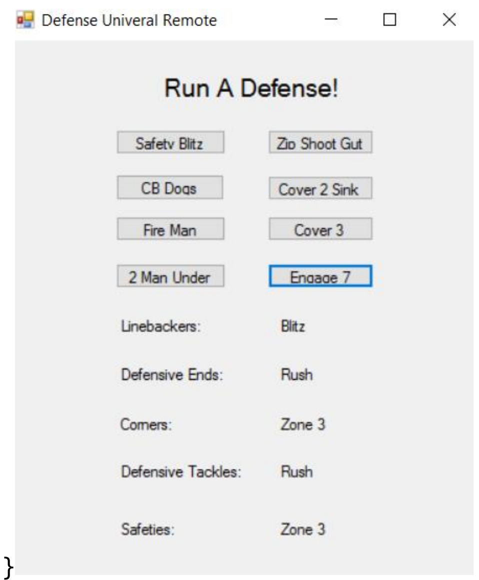
```
   public Form1()
        {
            InitializeComponent();
             myFacade = new Facade();

        }
```

I then made a function called displayPlay, which uses the string from the state functions in the player classes to display what each player is doing on each play. This prevents unnecessarily doing this step for each play.

```
void displayPlay()
        {
            s_assign_label.Text = myFacade.safety.safety_task();
            cb_assign_label.Text = myFacade.corner.cornerState();
            dt_assign_label.Text = myFacade.tackles.dtState();
            de_assign_label.Text = myFacade.end.deState();
            lb_assign_label.Text = myFacade.backers.lbState();
        }
```

Lastly, I set that for each button clicked action event, the corresponding play is ran by each position. It is also displayed by the display play function. This is done for every play button.

```
        private void engage7_button_Click(object sender, EventArgs e)
        {
            myFacade.Engage7();
            displayPlay();
```



```
Linebackers:         Blitz
Defensive Ends:      Rush
Corners:             Zone 3
Defensive Tackles:   Rush
Safeties:            Zone 3
```

}

And thus, my program runs as intended!

Observations:

For this program, I struggled to know where to begin. It took me a while to understand what the final product that was expected actually was. Once I figured that out, it took off and was an enjoyable project to create. I believe my program does a good job of displaying the Façade pattern, and I am very happy with how it turned out.