Composite Pattern

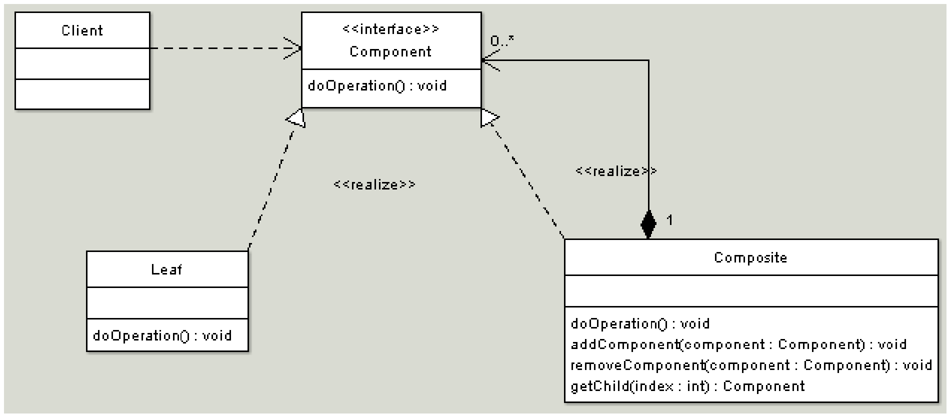
Design Patterns

Christopher Doseck

10/16/2016

Introduction

This assignment is an application that I created to show how the composite pattern works. In this application I am using Divisions to represent the Composite, Football for the component, and Team for the leaf.



The UML Diagram for Composite

The UML Diagram for the composite pattern, shown on the right, shows the classes that are needed to have the requirements. I had the Football abstract class as the component interface, the Division class for the Composite class, and the Team class for the Leaf class. The table below shows how all of the classes were used.

Form1	This is the client in the diagram that uses the component interface Football.
Football	This is the abstract class that fulfills the component in the UML diagram.
Composite	This is the Composite class from the UML diagram, it has the ability to hold Component objects inside of it.
Team	This represents the Leaf diagram from the UML diagram above, it represents a team.

Narrative

```
public abstract class Football
{
    public abstract string getName();
    public abstract int getWins();
    public abstract int getLosses();
    public abstract int getNumChildren();
    public abstract Football getChild(int index);
    public string getRecord()
    {
        return " " + getWins() + " - " + getLosses();
    }
}
```

This is the abstract class Football that represents the Component interface in the UML Diagram. The methods getName(), getWins(), getLosses(), and getRecord() are all methods that represent the doOperation() method in the UML diagram. The getRecord() method is already defined, because regardless of the class, the way to get the record will be the same.

I included the methods to get the number of children of the class, and the method to get the child with a certain index number to the abstract class because it was necessary to do so in order for the client to treat individual objects and compositions of objects uniformly.

```
public class Division : Football
{
    private string name;
    private List<Football> footballs = new List<Football>();

    public Division(string name)
    {
        this.name = name;
    }

    public override int getLosses()
    {
        int losses = 0;
        foreach(Football children in footballs)
        {
            losses += children.getLosses();
        }
        return losses;
    }

    public override string getName()
    {
        return name;
    }

    public override int getWins()
    {
        int wins = 0;
        foreach (Football children in footballs)
        {
            wins += children.getWins();
        }
        return wins;
    }

    public override int getNumChildren()
    {
        return footballs.Count;
```

This is the Division class that represents the Composite class in the UML Diagram. It inherits from the Football abstract class. The constructor has it pass the name of the division in. This class contains a list of all of the children of the class.

It overrides the getWins() and getLosses() methods that have their signature in the Football class. It sets the initial wins or losses to 0, and goes through each of the children and adds the wins or losses from them, and returns the total.

The getName() method returns the name that was passed in.

The getNumChildren() method returns the number of children by returning the number of elements in the array that contains the children.

```
    }

    public override Football getChild(int index)
    {
        return footballs[index];
    }

    public void add(Football football)
    {
        footballs.Add(football);
    }

    public void remove(Football football)
    {
        footballs.Remove(football);
    }
}
```

The getChild() method return the child at the element number index.

The add and remove functions add and remove objects of one of the classes that inherit from Football from the list of children.

```
public class Team : Football
{
    private string name;
    private string mascot;
    private int wins;
    private int losses;

    public Team(string name, string mascot, int wins, int losses)
    {
        this.name = name;
        this.mascot = mascot;
        this.wins = wins;
        this.losses = losses;
    }

    public override string getName()
    {
        return name + " " + mascot;
    }

    public override int getWins()
    {
        return wins;
    }

    public override int getLosses()
    {
        return losses;
    }

    public override int getNumChildren()
    {
        return 0;
    }

    public override Football getChild(int index)
    {
        return null;
    }
}
```

This is the Team class that represents the Leaf class in the UML Diagram. It inherits from the Football abstract class. It gets the name, mascot, wins, and losses passed to it in its constructor.

The getName() method returns the name and nickname.

The getWins() method returns the number of wins.

The getLosses() method returns the number of losses.

The getNumChildren() method returns 0 since this is a leaf, which means that it would not have any children.

The getChild() method return null, since there are no children to return, because it is a leaf.

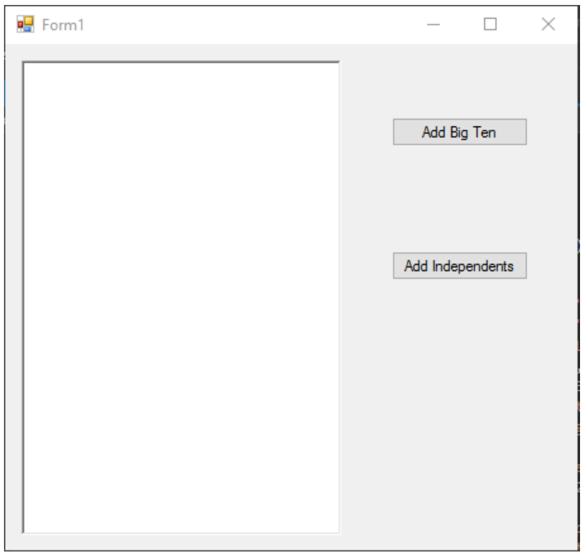
```
public partial class Form1 : Form
{
    Division bigTen = new Division("Big Ten");
    Division bigTenEast = new Division("Big Ten East");
    Division bigTenWest = new Division("Big Ten West");
    Division independents = new Division("Independents");
    public Form1()
    {
        bigTenEast.add(new Team("Michigan", "Wolverines", 6, 0));
        bigTenEast.add(new Team("Ohio State", "Buckeyes", 6, 0));
        bigTenEast.add(new Team("Penn State", "Nittany Lions", 4, 2));
        bigTenEast.add(new Team("Maryland", "Terrapins", 4, 2));
        bigTenEast.add(new Team("Indiana", "Hoosiers", 3, 3));
        bigTenEast.add(new Team("Michigan State", "Spartans", 2, 4));
        bigTenEast.add(new Team("Rutgers", "Scarlet Knights", 2, 5));

        bigTenWest.add(new Team("Nebraska", "Cornhuskers", 6, 0));
        bigTenWest.add(new Team("Iowa", "Hawkeyes", 5, 2));
        bigTenWest.add(new Team("Wisconsin", "Badgers", 4, 2));
        bigTenWest.add(new Team("Minnesota", "Golden Gophers", 4, 2));
        bigTenWest.add(new Team("Northwestern", "Wildcats", 3, 3));
        bigTenWest.add(new Team("Purdue", "Boilermakers", 3, 3));
        bigTenWest.add(new Team("Illinois", "Fighting Illini", 2, 4));

        independents.add(new Team("Army", "Black Knights", 4, 2));
        independents.add(new Team("BYU", "Cougars", 4, 3));
        independents.add(new Team("Notre Dame", "Fighting Irish", 2, 5));
        independents.add(new Team("UMass", "Minutemen", 1, 6));
        InitializeComponent();
    }

    private void display(Football football, string indent)
    {
        tbFootball.Text += "\n" + indent + football.getName() + football.getRecord();
        for(int i = 0; i < football.getNumChildren(); i++)
        {
            display(football.getChild(i), indent + "    ");
        }
    }

    private void btnAddIndependents_Click(object sender, EventArgs e)
    {
        display(independents, "");
    }

    private void btnAddBigTen_Click(object sender, EventArgs e)
    {
        bigTen.add(bigTenEast);
        bigTen.add(bigTenWest);
        display(bigTen, "");
    }
}
```

This is the Form that is used to display the composite pattern. It starts off with adding four divisions, and adding teams to three of those divisions. The division it does not add teams to is the bigTen, because it is composed of the bigTenEast and the bigTenWest.

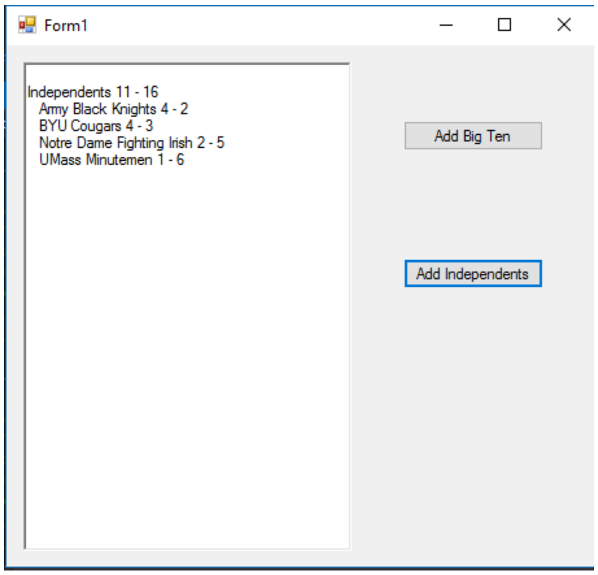
The display function displays an object of a class that inherits from the Football class, and all of its children.

When the Add Independents button is clicked, it passes the independens object to the display method.

When the Add Big Ten button is clicked, it adds bigTenEast and bigTenWest to the bigTen. It then calls the diplay method and passes it the bigTen object.



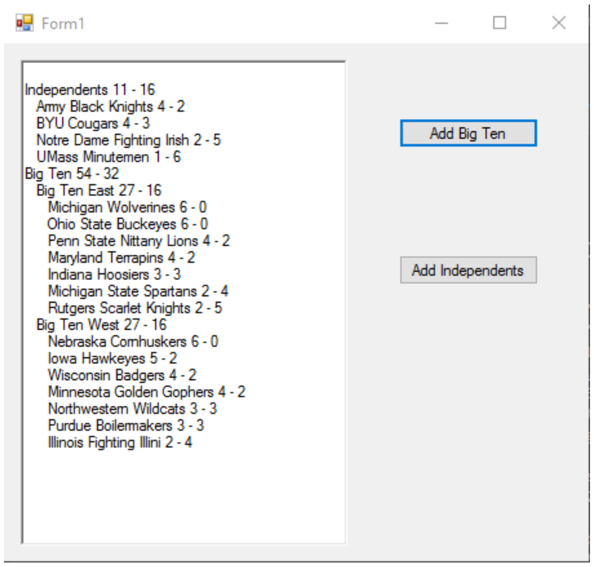
Initial setup



```
Independents 11 - 16
  Army Black Knights 4 - 2
  BYU Cougars 4 - 3
  Notre Dame Fighting Irish 2 - 5
  UMass Minutemen 1 - 6
```

Independents Added

Big Ten Added



```
Independents 11 - 16
  Army Black Knights 4 - 2
  BYU Cougars 4 - 3
  Notre Dame Fighting Irish 2 - 5
  UMass Minutemen 1 - 6
Big Ten 54 - 32
  Big Ten East 27 - 16
    Michigan Wolverines 6 - 0
    Ohio State Buckeyes 6 - 0
    Penn State Nittany Lions 4 - 2
    Maryland Terrapins 4 - 2
    Indiana Hoosiers 3 - 3
    Michigan State Spartans 2 - 4
    Rutgers Scarlet Knights 2 - 5
  Big Ten West 27 - 16
    Nebraska Cornhuskers 6 - 0
    Iowa Hawkeyes 5 - 2
    Wisconsin Badgers 4 - 2
    Minnesota Golden Gophers 4 - 2
    Northwestern Wildcats 3 - 3
    Purdue Boilermakers 3 - 3
    Illinois Fighting Illini 2 - 4
```

Conclusion

I enjoyed making this program. This design pattern is one of my favorites. I felt like I got a good understanding of it. I really liked making the recursive function take care of the tabbing in the program.