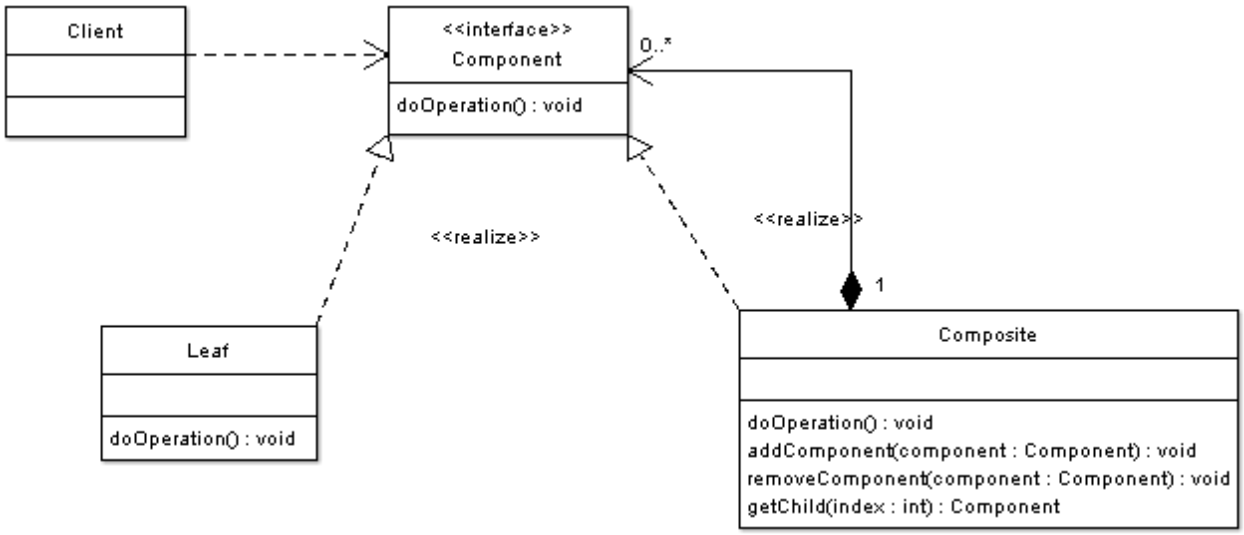
Jack Raney

Design Patterns

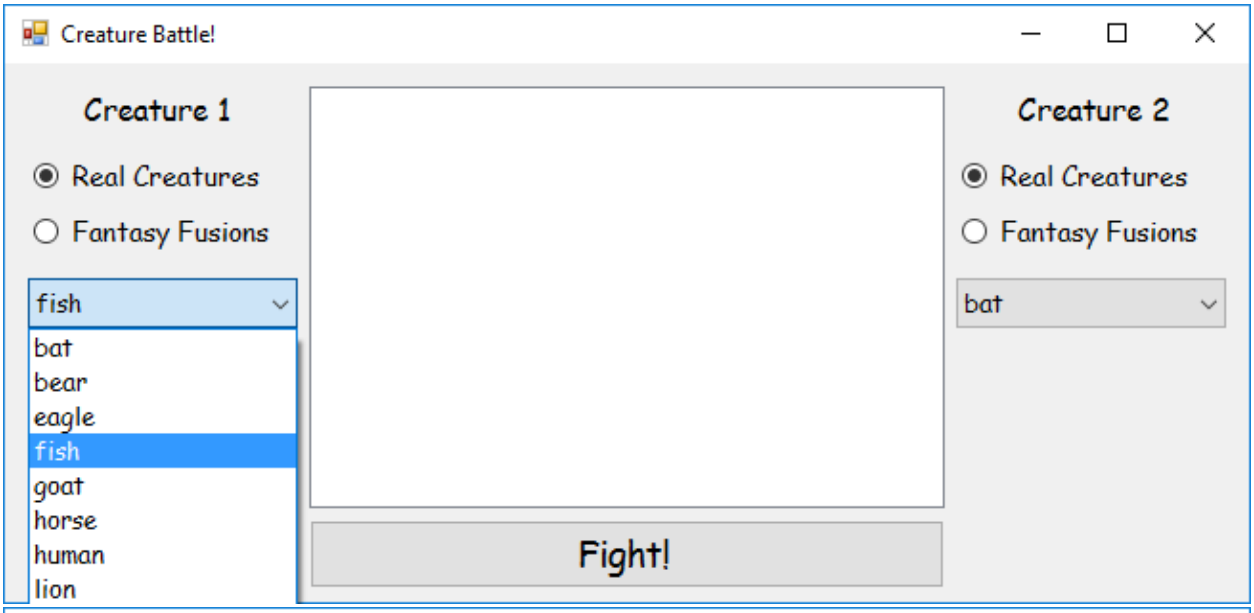
10/15/2016

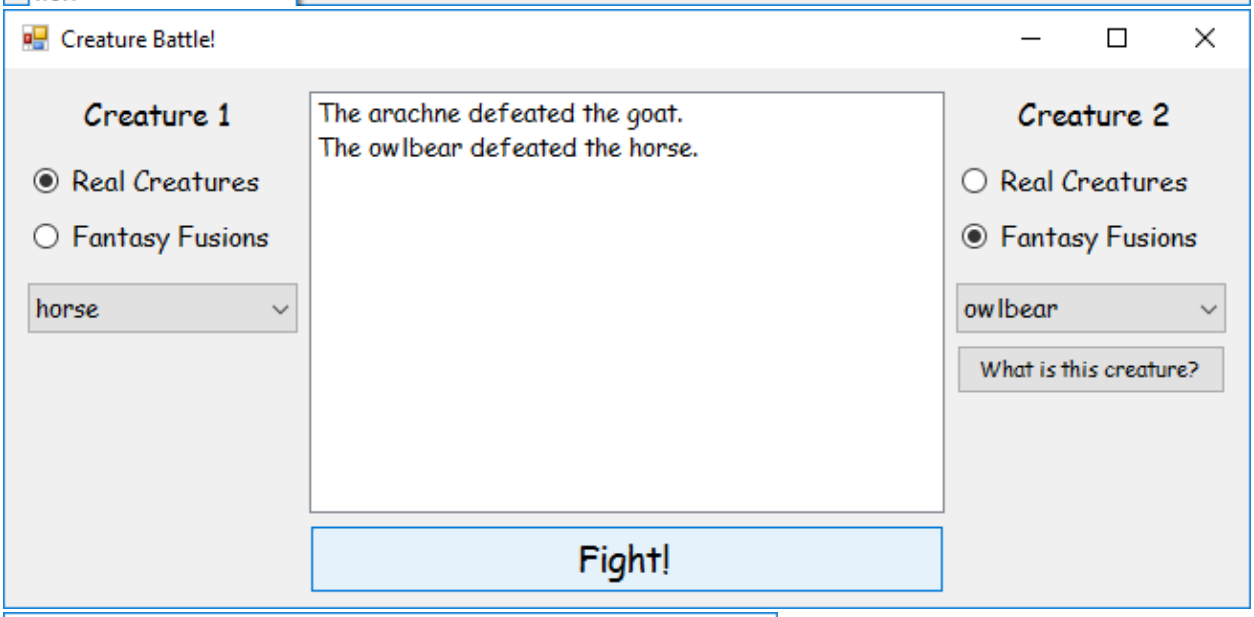
The Composite Pattern

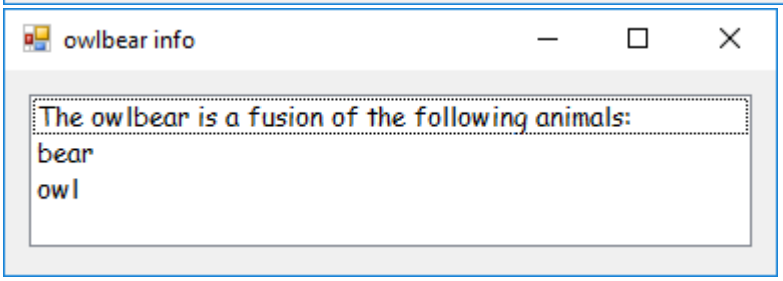
This paper has been created to show an example of the Composite Pattern. This design pattern's intent, according to oodesign, is to "compose objects into tree structures to represent part-whole hierarchies" and that it "lets clients treat individual objects and compositions of objects uniformly". The UML oodesign provides is shown below.



The example program I created for this pattern is CreatureBattle.exe.







Code for CreatureBattle.exe

Creature.cs:

```
public interface Creature //This is the Component interface
    {
        int getPower();
        string ToString();
    }
```

getPower is analogous to doOperation from the UML diagram

RealCreature.cs:

```
public class RealCreature : Creature   //This is the Leaf class
    {
        private string name;

        private int power;

        public RealCreature(string theName, int thePower)
        {
            name  = theName;
            power = thePower;
        }

        public int getPower()
        {
            return power;
        }

        public override string ToString()
        {
            return name;
        }
    }
```

Real creatures have a name and power associated with them that are assigned in the constructor.

Names are used for ToString's override, and power will be used in calculating who wins battles.

Here, getPower simply returns the value of the power variable.

Fusion.cs:

```
public class Fusion : Creature //This is the Composite class
    {
        private string name;

        public int numComps;

        private List<Creature> components = new List<Creature>();

        public Fusion(string theName, Creature c1, Creature c2)
        {
            name = theName;
            numComps = 2;
            addCreature(c1);
            addCreature(c2);
        }

        public Fusion(string theName, Creature c1, Creature c2, Creature c3)
        {
            name = theName;
            numComps = 3;
            addCreature(c1);
            addCreature(c2);
            addCreature(c3);
        }
```

Fusion creatures have a list of what component creatures are part of the fusion along with a name.

The constructor for Fusion varies depending on whether the creatures is derived from 2 or 3 real creatures.

```
        public int getPower()
        {
            int totalPower = 0;

            foreach (Creature c in components)
            {
                totalPower += c.getPower();
            }

            return totalPower;
        }

        public void addCreature(Creature c)
        {
            components.Add(c);
        }

        public void removeCreature(Creature c)
        {
            components.Remove(c);
        }

        public Creature getComponent(int index)
        {
            return components[index];
        }

        public override string ToString()
        {
            return name;
        }
    }
```

Here, getPower returns the total values of the component creatures.

addCreature, removeCreature, and getComponent are analogous to the UML diagram's addComponent, removeComponent, and getChild.

Form1.cs:

```
public partial class Form1 : Form
    {
        Random rand = new Random();

        static List<Creature> realList = new List<Creature>()
        {
            new RealCreature("bat"     , 2),    //0
            new RealCreature("bear"    , 8),    //1
            new RealCreature("eagle"   , 3),    //2
            new RealCreature("fish"    , 1),    //3
            new RealCreature("goat"    , 4),    //4
            new RealCreature("horse"   , 6),    //5
            new RealCreature("human"   , 5),    //6
            new RealCreature("lion"    , 9),    //7
            new RealCreature("owl"     , 3),    //8
            new RealCreature("scorpion", 2),    //9
            new RealCreature("snake"   , 4),    //10
            new RealCreature("spider"  , 2),    //11
        };

        static List<Creature> fuseList = new List<Creature>()
        {
            new Fusion("arachne"   , realList[6], realList[11]),
            new Fusion("chimera"   , realList[4], realList[7]  , realList[10]),
            new Fusion("griffin"   , realList[2], realList[7]) ,
            new Fusion("harpy"     , realList[2], realList[6]) ,
            new Fusion("hippogriff", realList[2], realList[5]) ,
            new Fusion("kelpie"    , realList[3], realList[5]) ,
            new Fusion("lamia"     , realList[6], realList[10]),
            new Fusion("manticore" , realList[0], realList[7]  , realList[9]),
            new Fusion("mermaid"   , realList[3], realList[6]) ,
            new Fusion("owlbear"   , realList[1], realList[8]) ,
            new Fusion("satyr"     , realList[4], realList[6]) ,
            new Fusion("sphinx"    , realList[6], realList[7]) ,
        };
```

The form contains the full lists of creatures used in this program, separated by whether or not they are a fusion.

A random object is also present for battles (which will be detailed later).

The comment numbers are their index numbers within realList (this made entering the values into fuseList easier).

```
public Form1()
{
    InitializeComponent();

    leftReal.Checked  = true;
    rightReal.Checked = true;

    leftInfo.Visible = false;
    rightInfo.Visible = false;

    populate(leftCreatures, realList);
    populate(rightCreatures, realList);
}

private void populate(ComboBox cb, List<Creature> l)
{
    cb.Items.Clear();
    foreach (Creature c in l)
        cb.Items.Add(c);
    cb.SelectedItem = l[0];
}

private void leftReal_CheckedChanged(object sender, EventArgs e)
{
    populate(leftCreatures, realList);
    leftInfo.Visible = false;
}

private void leftFuse_CheckedChanged(object sender, EventArgs e)
{
    populate(leftCreatures, fuseList);
    leftInfo.Visible = true;
}

private void rightReal_CheckedChanged(object sender, EventArgs e)
{
    populate(rightCreatures, realList);
    rightInfo.Visible = false;
}

private void rightFuse_CheckedChanged(object sender, EventArgs e)
{
    populate(rightCreatures, fuseList);
    rightInfo.Visible = true;
}

private void fightButton_Click(object sender, EventArgs e)
{
    Creature left = (Creature) leftCreatures.SelectedItem;
    Creature right = (Creature)rightCreatures.SelectedItem;

    int randPool = left.getPower() + right.getPower();
    int decider = rand.Next() % randPool + 1;

    if (left.getPower() < decider)
        fightList.Items.Add("The " + right + " defeated the " + left + ".");
    else
        fightList.Items.Add("The " + left + " defeated the " + right + ".");
}
```

The program begins with real creatures selected in the form, so the comboBoxes are populated with real creatures and the buttons for creature information are not visible.

populate adds the items of a selected list into a comboBox. This is called when the radio buttons are selected.

The info buttons are only present when fusion creatures are selected.

Battles are decided by random chance influenced by the power levels of the battling creatures. This means that, for example, while a fish can defeat a lion, it is very unlikely.

The information buttons create a Form2 object that tells the user what real animals combine to create the specified fusion creature.

```
    private void leftInfo_Click(object sender, EventArgs e)
    {
        Fusion left = (Fusion) leftCreatures.SelectedItem;
        Form2 f2 = new Form2();
        f2.Text = left + " info";
        f2.infoBox.Items.Add("The " + left + " is a fusion of the following animals:");
        for (int i = 0; i < left.numComps; i++)
            f2.infoBox.Items.Add(left.getComponent(i));
        f2.Show();
    }

    private void rightInfo_Click(object sender, EventArgs e)
    {
        Fusion right = (Fusion)rightCreatures.SelectedItem;
        Form2 f2 = new Form2();
        f2.Text = right + " info";
        f2.infoBox.Items.Add("The " + right + " is a fusion of the following animals:");
        for (int i = 0; i < right.numComps; i++)
            f2.infoBox.Items.Add(right.getComponent(i));
        f2.Show();
    }
}
```

Form2 has no code associated with it save for the code automatically created by visual studio for it and the listBox within it.

Reflection

I enjoyed creating this program; it does exactly what I originally wanted it to do (there was no compromise due to various constraints) and I never ran into big problems with implementation. I also enjoy talking about mythical creatures and the timeless question, "Who would win in a fight between __ and __?"