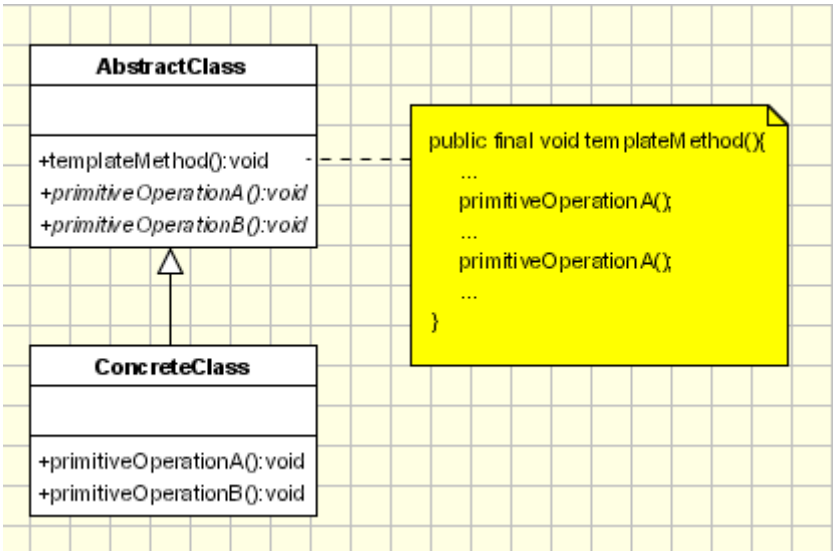
Jack Raney

Design Patterns

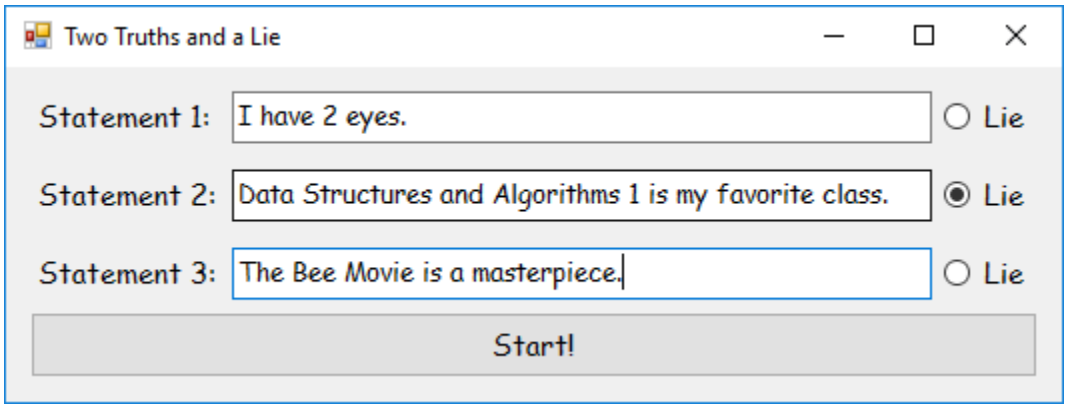
12/1/16

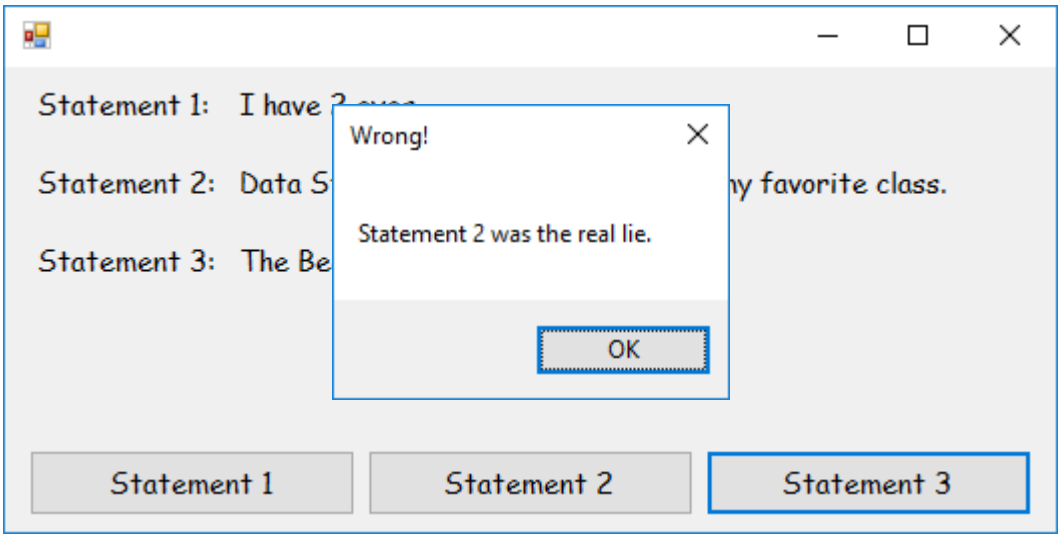
The Template Method Pattern

This paper details and shows an example of the Template Method Pattern, a pattern that's purpose, according to oodesign, is to "define the skeleton of an algorithm in an operation, deferring some steps to subclasses. [The] Template Method lets subclasses redefine certain steps of an algorithm without letting them… change the algorithm's structure". This means that a method that is constant in every subclass of a Template Method class calls other methods within the class that vary between subclasses. The UML class diagram for this Template Method Pattern is shown below:



To demonstrate the Template Method Pattern, I created the program "TwoTruthsAndALie":





The Code

TwoTruthsOneLie.cs

```
public abstract class TwoTruthOneLie    //AbstractClass
{
    public int getLie()  //templateMethod()
    {
        if (!statement1())
            return 1;
        if (!statement2())
            return 2;
        return 3;
    }

    public abstract bool statement1();  //primitiveOperation()
    public abstract bool statement2();  //primitiveOperation()
}
```

TwoTruthsOneLie objects use getLie() to determine which statement is a lie using statement1() and statement2(). Its subclasses, FirstLie, SecondLie, and ThirdLie change the return result of statement1 and statement2.

FirstLie returns statement1() as false, signifying that the first statement is a lie.

SecondLie does the same, but with statement2() as the lie.

ThirdLie returns both as true, which means that the third statement has to be the lie.

FirstLie.cs

```
public class FirstLie : TwoTruthOneLie  //ConcreteClass
{
    public override bool statement1()   //primitiveOperation()
    {
        return false;
    }

    public override bool statement2()   //primitiveOperation()
    {
        return true;
    }
}
```

Form1.cs

In the first form, statements are typed into the form and a radio button is clicked to signify which statement is the lie. This radio button selection changes the TwoTruthsOneLie object into its respective subclass object via polymorphism.

When the start button is clicked, the statements and TwoTruthOneLie object are passed into the second form through its constructor and the first form closes.

```
public partial class Form1 : Form
{
    TwoTruthOneLie ttol;

    public Form1()
    {
        InitializeComponent();

        statement1Radio.Checked = true;
        ttol = new FirstLie();
    }

    private void startButton_Click(object sender, EventArgs e)
    {
        string statement1 = statement1Box.Text;
        string statement2 = statement2Box.Text;
        string statement3 = statement3Box.Text;

        Form2 f = new Form2(statement1, statement2, statement3, ttol);
        f.Show();
        Hide();
    }

    private void statement1Radio_CheckedChanged(object sender, EventArgs e)
    {
        ttol = new FirstLie();
    }
}
```

Form2.cs

In the second form, there is a choice of three different buttons to press to try to guess which statement is the lie using the data from the first form. Each button calls the getAnswer() method with an integer respective to its associated statement. This method checks the guess against the returns integer from the TwoTruthOneLie object's getLie() method.

```
public partial class Form2 : Form
{
    TwoTruthOneLie ttol;

    public Form2(string s1, string s2, string s3, TwoTruthOneLie t)
    {
        InitializeComponent();

        statement1Label.Text = s1;
        statement2Label.Text = s2;
        statement3Label.Text = s3;
        ttol = t;
    }

    private void getAnswer(int lie)
    {
        if (ttol.getLie() == lie)
            MessageBox.Show("You are correct!", "Good Job!");
        else
            MessageBox.Show("Statement " + ttol.getLie() + " was the real lie.", "Wrong!");

         Application.Exit();
    }

    private void statement1Button_Click(object sender, EventArgs e)
    {
        getAnswer(1);
    }
}
```

Reflection

This was a very simple program made for a very simple design pattern. Simple does not mean useful, however; while I can see the use for this pattern, it is very situational and therefore not a pattern I see myself using often.

I would like to thank Alex Lang for pioneering the production of incredibly simple design pattern examples.