Activity 5

Ian M. McConihay

College of Science, Engineering and Technology, Grand Canyon University

CST-150: C# Programming I

Mark Smithers

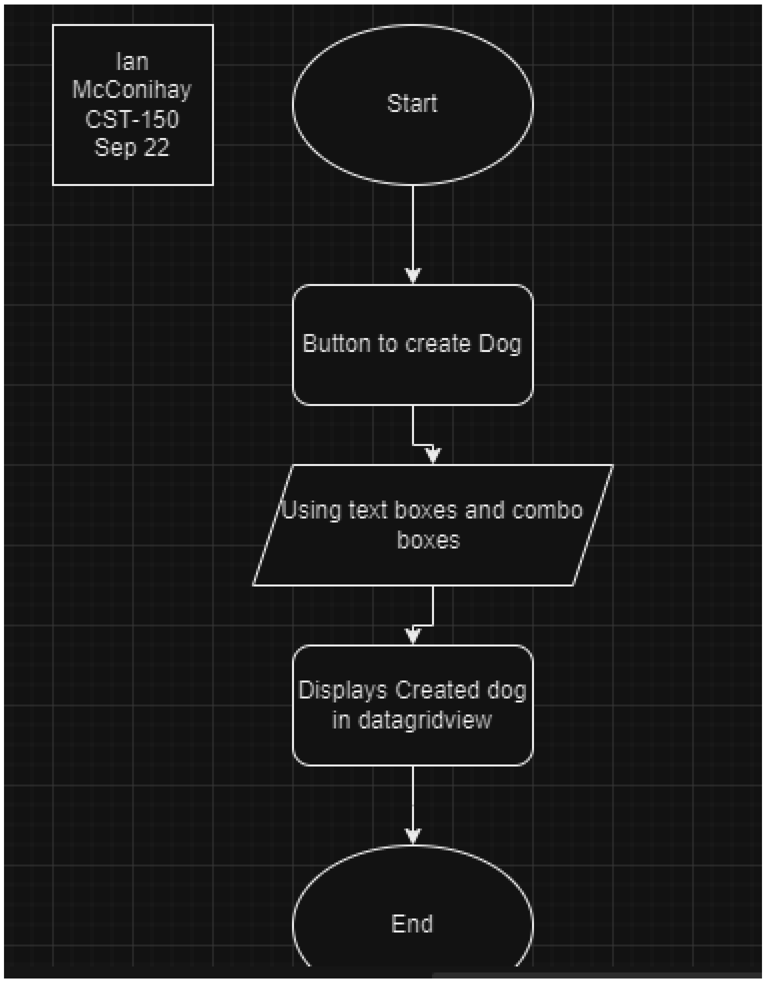
September 22, 2024

Video Link:

https://www.loom.com/share/95358dd1aa7b44ada7b56a3d7798b179?sid=cbc77d0b-fc3d-4c91-b284-9add5a7bff95

Github: https://github.com/Ian-McConihay/CST-150

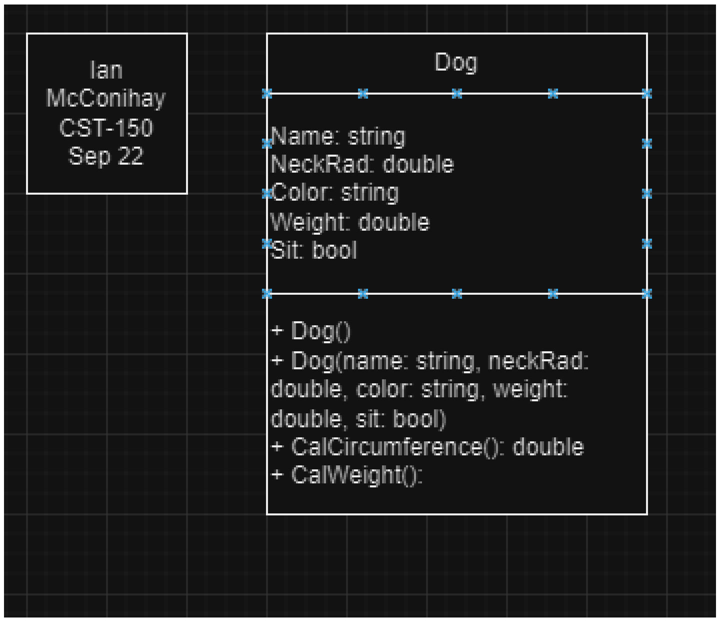
Flowchart


Ian McConihay CST-150 Sep 22
Start
Button to create Dog
Using text boxes and combo boxes
Displays Created dog in datagridview
End

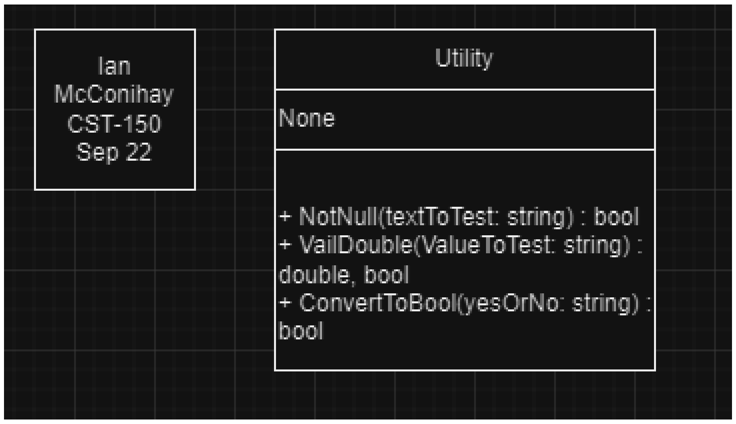
Activity 5 has a button that takes in textbox text to persist into a datagridview. This application allows the user to create a Dog using a series of components. We also need to make sure the class has the appropriate layers of logic.

UML

Figure 1: Dog

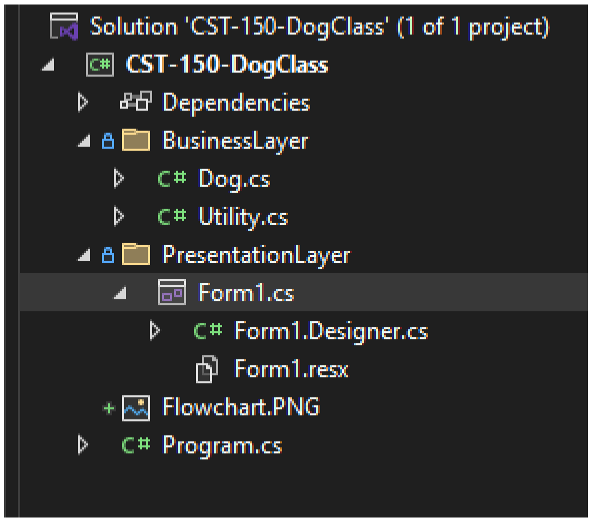


The UML Dog object is the main object of the application. Five properties for setting various parts of the dog. Also, four methods to include an empty constructor and a set constructor.
Figure 2: Utility


Ian
McConihay
CST-150
Sep 22
Utility
None
+ NotNull(textToTest: string) : bool
+ VailDouble(ValueToTest: string) : double, bool
+ ConvertToBool(yesOrNo: string) : bool

The Utility object class is for miscellaneous functionality methods. ConvertToBool is my favorite of these methods for the simplicity of yes or no functionality. The other methods are for testing.

N-Layer

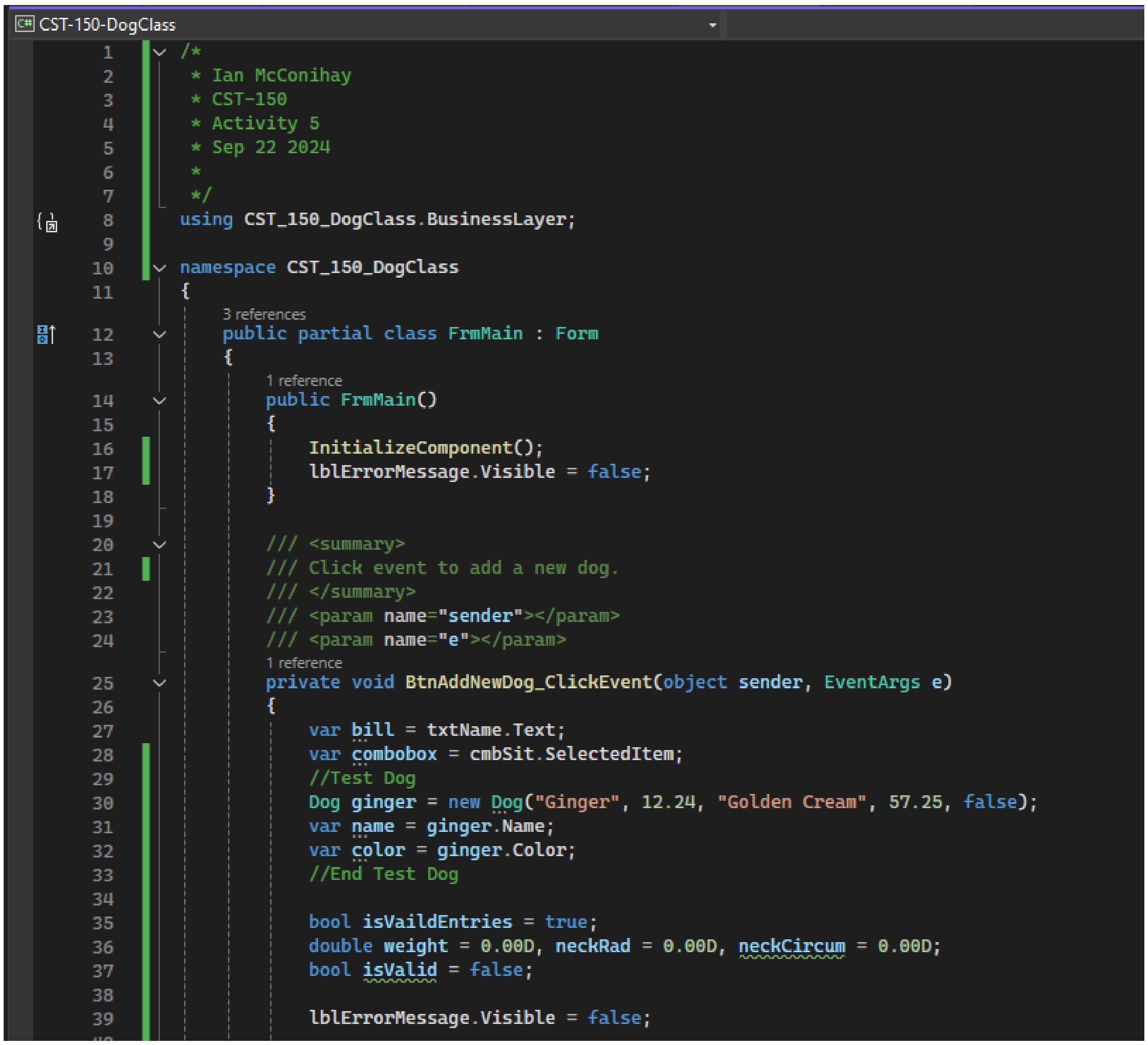


```
Solution 'CST-150-DogClass' (1 of 1 project)
  CST-150-DogClass
    Dependencies
    BusinessLayer
      Dog.cs
      Utility.cs
    PresentationLayer
      Form1.cs
        Form1.Designer.cs
        Form1.resx
    Flowchart.PNG
    Program.cs
```

Here is a screenshot of the file structure for the application. N-layer was required and demostraded in the assignment. Dog and Utility have been moved to the BusinessLayer and The PresentationLayer contains the Main form for design.

Application Screenshots

Figure 1: Code

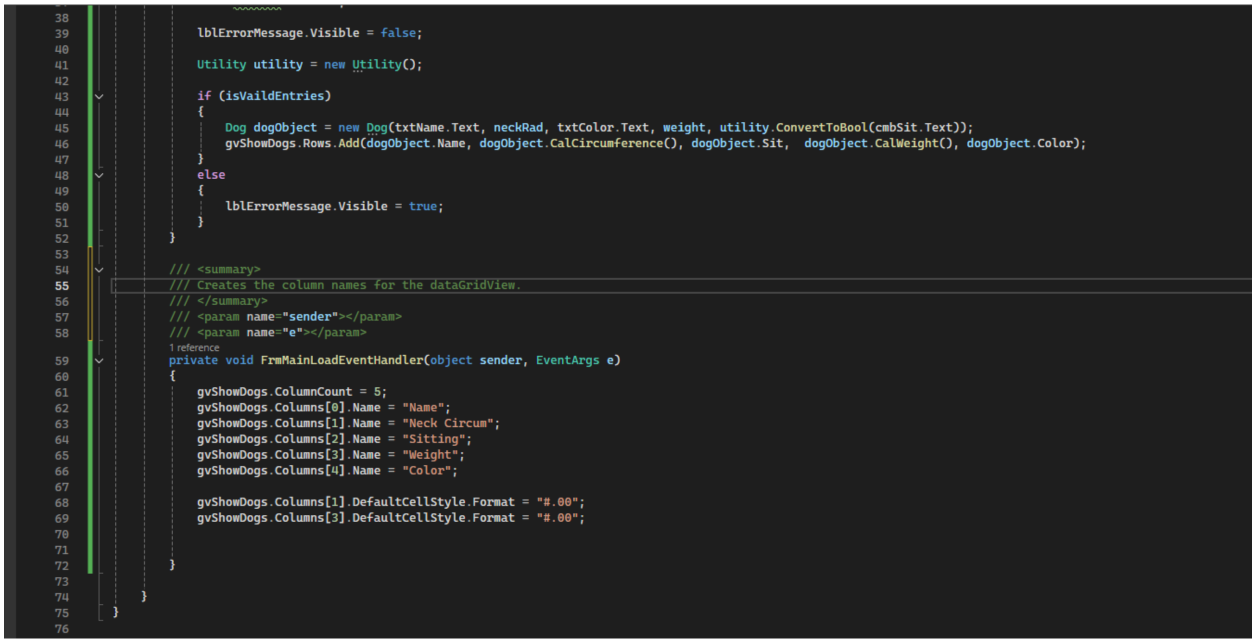


CST-150-DogClass

```
/*
 * Ian McConihay
 * CST-150
 * Activity 5
 * Sep 22 2024
 *
 */
using CST_150_DogClass.BusinessLayer;

namespace CST_150_DogClass
{
    3 references
    public partial class FrmMain : Form
    {
        1 reference
        public FrmMain()
        {
            InitializeComponent();
            lblErrorMessage.Visible = false;
        }

        /// <summary>
        /// Click event to add a new dog.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        1 reference
        private void BtnAddNewDog_ClickEvent(object sender, EventArgs e)
        {
            var bill = txtName.Text;
            var combobox = cmbSit.SelectedItem;
            //Test Dog
            Dog ginger = new Dog("Ginger", 12.24, "Golden Cream", 57.25, false);
            var name = ginger.Name;
            var color = ginger.Color;
            //End Test Dog

            bool isVaildEntries = true;
            double weight = 0.00D, neckRad = 0.00D, neckCircum = 0.00D;
            bool isValid = false;

            lblErrorMessage.Visible = false;
```

Starting with the citation at the top of the class. BtnAddNewDog_ClickEvent is the event method that fires off the creation of a Dog object. We had testing earlier in the activity but here I have also demonstrated that the constuctor of creating a new Dog object works.

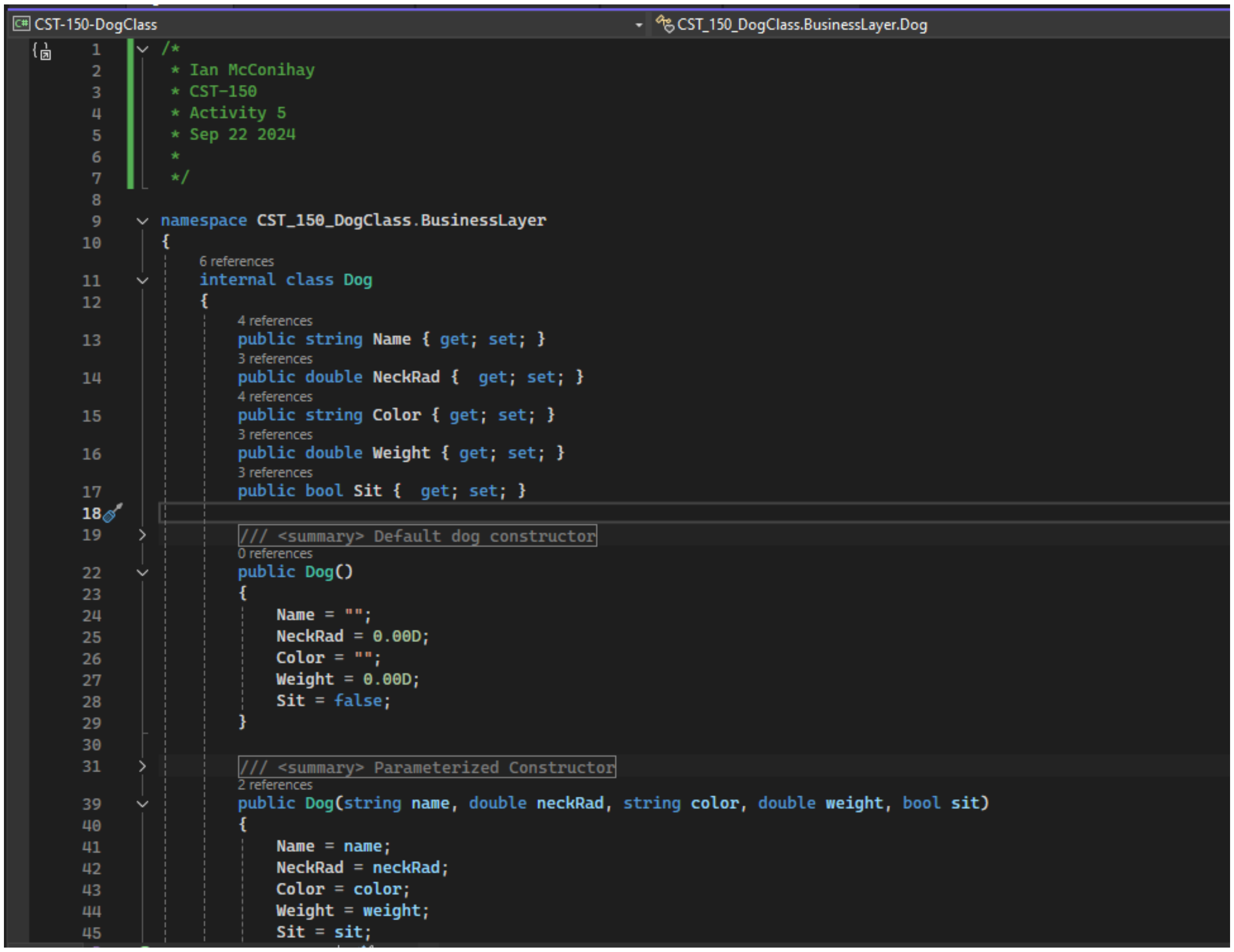
Figure 2: Code



```
            lblErrorMessage.Visible = false;

            Utility utility = new Utility();

            if (isVaildEntries)
            {
                Dog dogObject = new Dog(txtName.Text, neckRad, txtColor.Text, weight, utility.ConvertToBool(cmbSit.Text));
                gvShowDogs.Rows.Add(dogObject.Name, dogObject.CalCircumference(), dogObject.Sit,  dogObject.CalWeight(), dogObject.Color);
            }
            else
            {
                lblErrorMessage.Visible = true;
            }
        }

        /// <summary>
        /// Creates the column names for the dataGridView.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        1 reference
        private void FrmMainLoadEventHandler(object sender, EventArgs e)
        {
            gvShowDogs.ColumnCount = 5;
            gvShowDogs.Columns[0].Name = "Name";
            gvShowDogs.Columns[1].Name = "Neck Circum";
            gvShowDogs.Columns[2].Name = "Sitting";
            gvShowDogs.Columns[3].Name = "Weight";
            gvShowDogs.Columns[4].Name = "Color";

            gvShowDogs.Columns[1].DefaultCellStyle.Format = "#.00";
            gvShowDogs.Columns[3].DefaultCellStyle.Format = "#.00";


        }

    }
}
```

In this screenshot we continue with the click event instantiating the Utility object to use for the sitting combo box. An if statement will check if the valid entries are true then it will create the object, else show the error message.

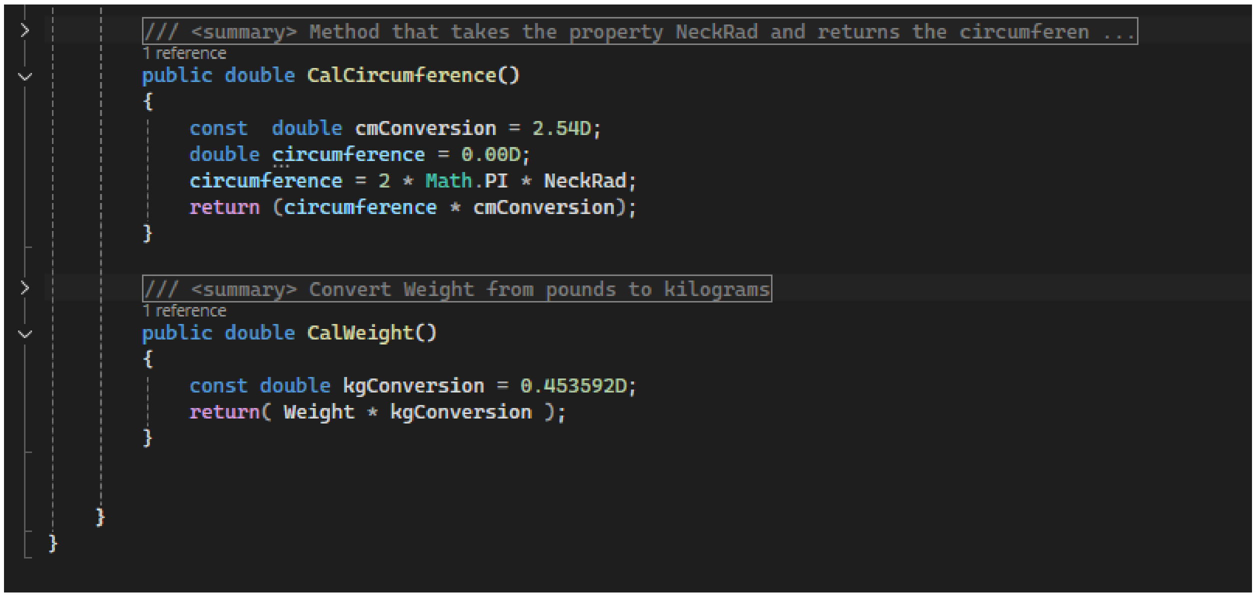
Figure 3: Dog Code



```csharp
/*
 * Ian McConihay
 * CST-150
 * Activity 5
 * Sep 22 2024
 *
 */

namespace CST_150_DogClass.BusinessLayer
{
    6 references
    internal class Dog
    {
        4 references
        public string Name { get; set; }
        3 references
        public double NeckRad {  get; set; }
        4 references
        public string Color { get; set; }
        3 references
        public double Weight { get; set; }
        3 references
        public bool Sit {  get; set; }

        /// <summary> Default dog constructor
        0 references
        public Dog()
        {
            Name = "";
            NeckRad = 0.00D;
            Color = "";
            Weight = 0.00D;
            Sit = false;
        }

        /// <summary> Parameterized Constructor
        2 references
        public Dog(string name, double neckRad, string color, double weight, bool sit)
        {
            Name = name;
            NeckRad = neckRad;
            Color = color;
            Weight = weight;
            Sit = sit;
```

In this screenshot we have the getters and setter for the Dog object. An empty constructor allows the creation of an object without needing to provide any parameters. Then we have a parametrized constructor.

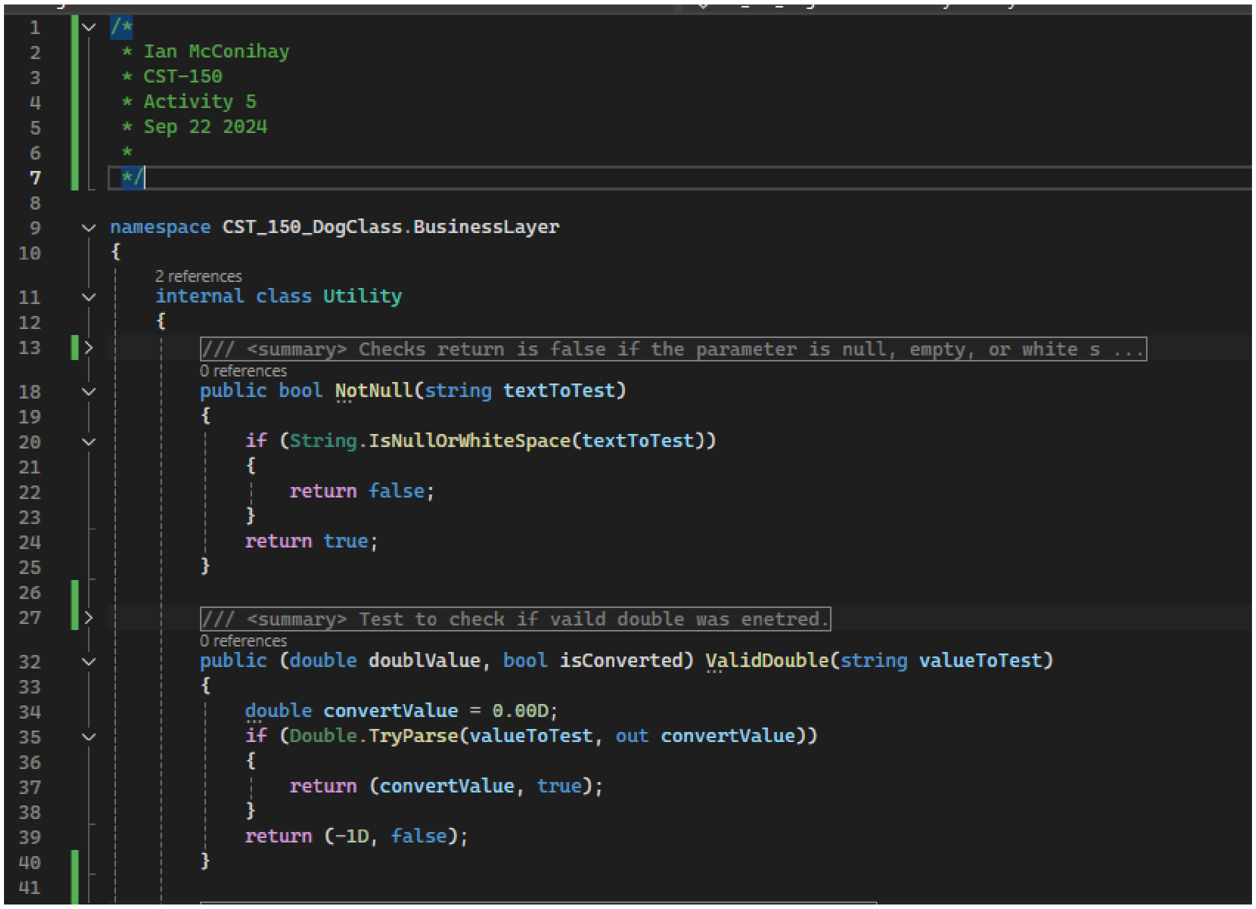
Figure 4: Dog Code



```
        /// <summary> Method that takes the property NeckRad and returns the circumferen ...
        1 reference
        public double CalCircumference()
        {
            const  double cmConversion = 2.54D;
            double circumference = 0.00D;
            circumference = 2 * Math.PI * NeckRad;
            return (circumference * cmConversion);
        }

        /// <summary> Convert Weight from pounds to kilograms
        1 reference
        public double CalWeight()
        {
            const double kgConversion = 0.453592D;
            return( Weight * kgConversion );
        }


    }
}
```

In this screenshot we have the GetQty grab the qty from the row selected in the combo box. This also uses a try catch to cover format exceptions. Next we have the IncDisplayQty method to save the new qty update to the text file.

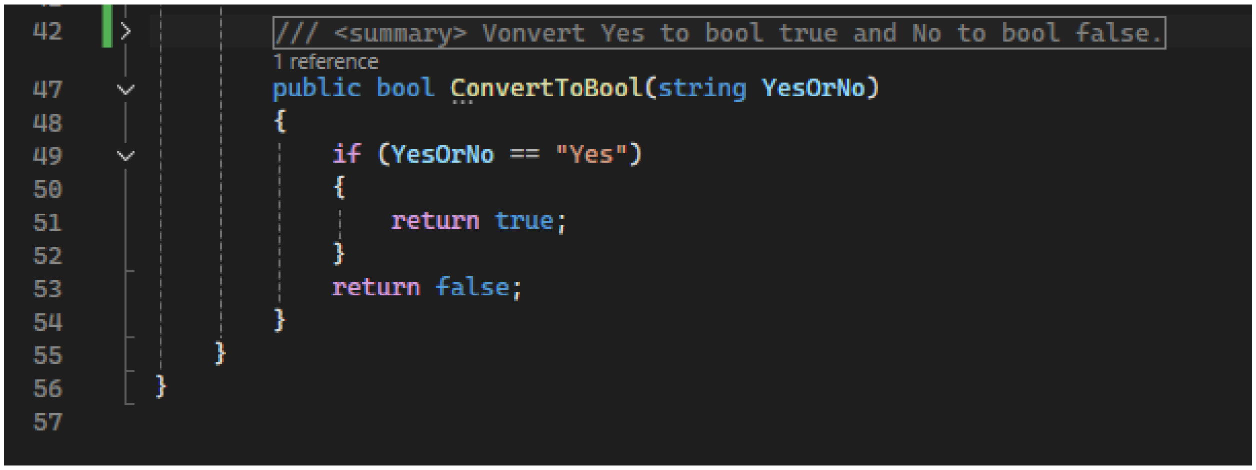
Figure 5: Utility Code



```
/*
 * Ian McConihay
 * CST-150
 * Activity 5
 * Sep 22 2024
 *
 */

namespace CST_150_DogClass.BusinessLayer
{
    2 references
    internal class Utility
    {
        /// <summary> Checks return is false if the parameter is null, empty, or white s ...
        0 references
        public bool NotNull(string textToTest)
        {
            if (String.IsNullOrWhiteSpace(textToTest))
            {
                return false;
            }
            return true;
        }

        /// <summary> Test to check if vaild double was enetred.
        0 references
        public (double doublValue, bool isConverted) ValidDouble(string valueToTest)
        {
            double convertValue = 0.00D;
            if (Double.TryParse(valueToTest, out convertValue))
            {
                return (convertValue, true);
            }
            return (-1D, false);
        }

```

In this screenshot we have the GetQty grab the qty from the row selected in the combo box. This also uses a try catch to cover format exceptions. Next we have the IncDisplayQty method to save the new qty update to the text file.

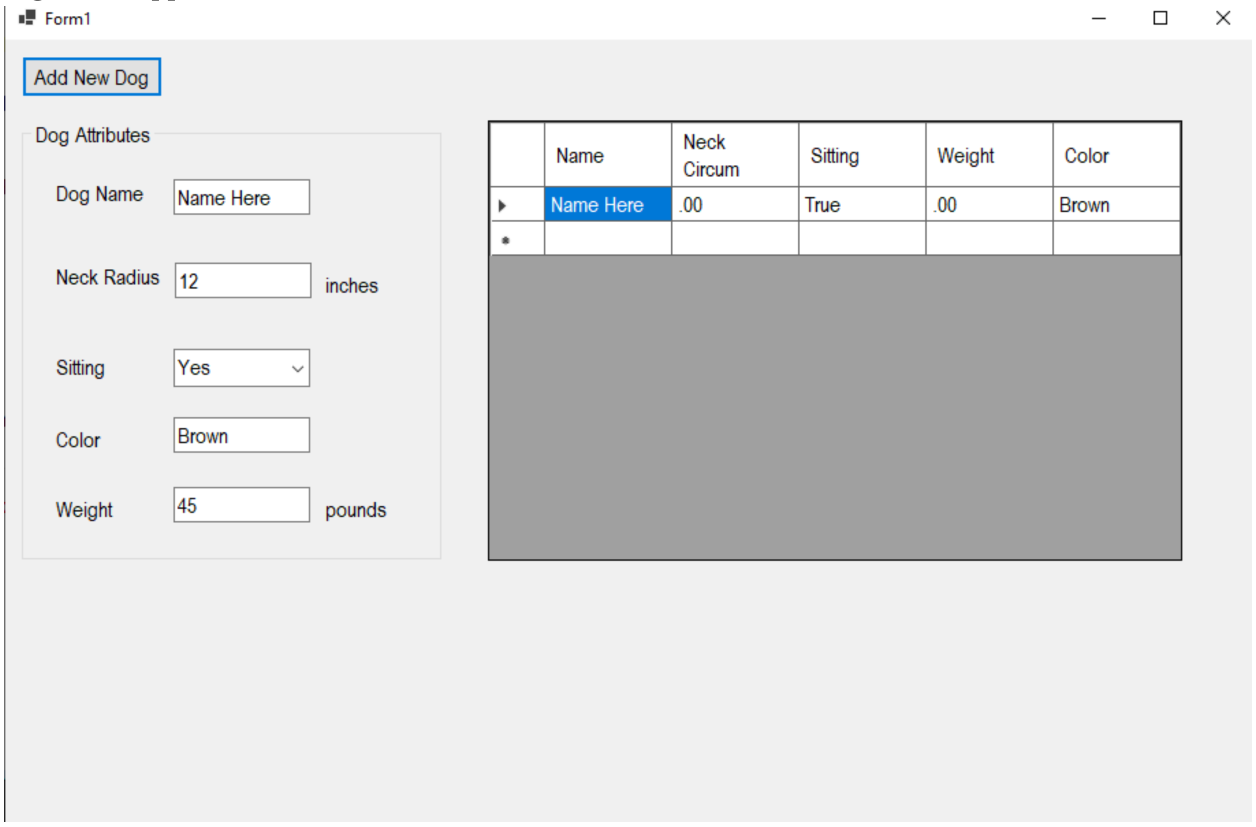
Figure 6: Utility Code



```
        /// <summary> Vonvert Yes to bool true and No to bool false.
        1 reference
        public bool ConvertToBool(string YesOrNo)
        {
            if (YesOrNo == "Yes")
            {
                return true;
            }
            return false;
        }
    }
}
```

In this screenshot we have the GetQty grab the qty from the row selected in the combo box. This also uses a try catch to cover format exceptions. Next we have the IncDisplayQty method to save the new qty update to the text file.

Figure 7: Application



Dog Name: Name Here

Neck Radius: 12 inches

Sitting: Yes

Color: Brown

Weight: 45 pounds

	Name	Neck Circum	Sitting	Weight	Color
▸	Name Here	.00	True	.00	Brown
*					

Here I have the application running. We have only the green Read File button ready for the user to select the text file. We can also see the name for the application is changed to MainForm. All of the columns are lower case. Also, you can see the select row.

1. What was challenging?

The activity worksheet bounced around a bit, so I was lost with some of the methods.

2. What did you learn?
   I learned about N-Layer architecture.

3. How would you improve on the project?
   I would create something to upload a picture for the Dog inventory.

4. How can you use what you learned on the job?

N-Layer architecture can be used for readability and organization for any application set up.