Eric Gathinji

Programming in C# CST-150-0500

Grand Canyon University

13th JULY 2025

Activity 5 part 1

Github link: https://github.com/Ericgathinji444/GCU

Video link: https://youtu.be/jJb364Ny-cQ?si=AB_D-Frpn4WmYmmU

FLOWCHART OF APPLICATION.

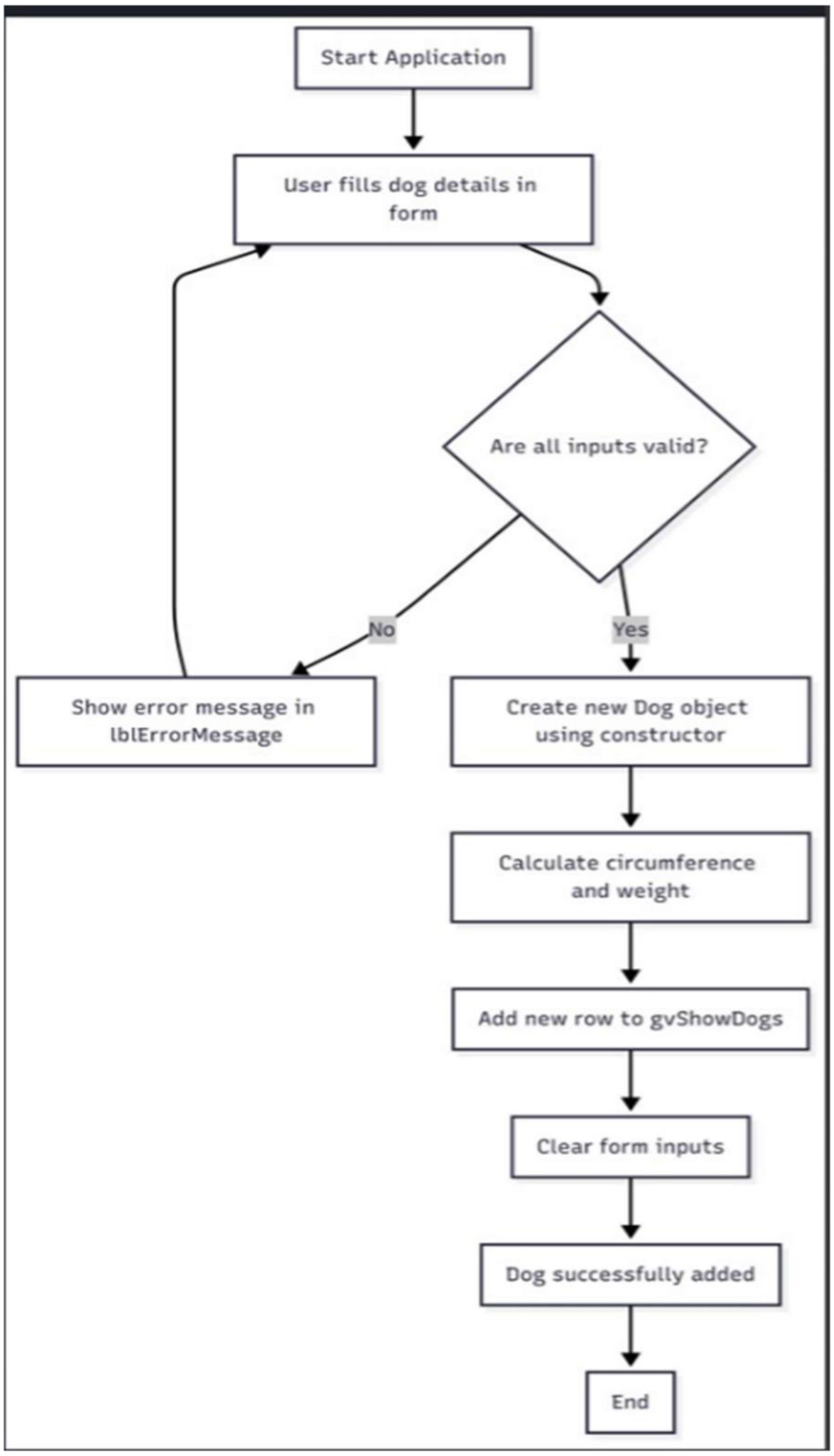

Start Application
User fills dog details in form
Are all inputs valid?
No
Yes
Show error message in lblErrorMessage
Create new Dog object using constructor
Calculate circumference and weight
Add new row to gvShowDogs
Clear form inputs
Dog successfully added
End

Figure 1: Flowchart of application
Figure 1 explanation:

Image of the flowchart of the application.

The flowchart illustrates the logical steps followed when the user interacts with the form

It starts with the user filling in dog attributes.

The program checks whether all fields are valid.

If invalid, an error message appears.

If valid, a new Dog object is created and its circumference and weight are calculated.

The result is displayed in the DataGridView, and the form is cleared for the next entry.

UML CLASS DIAGRAM

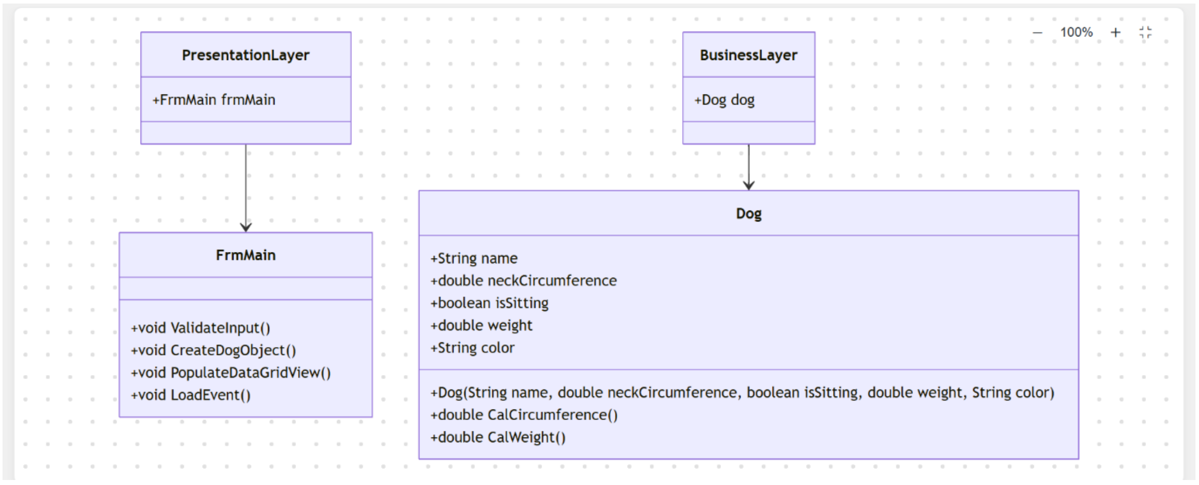
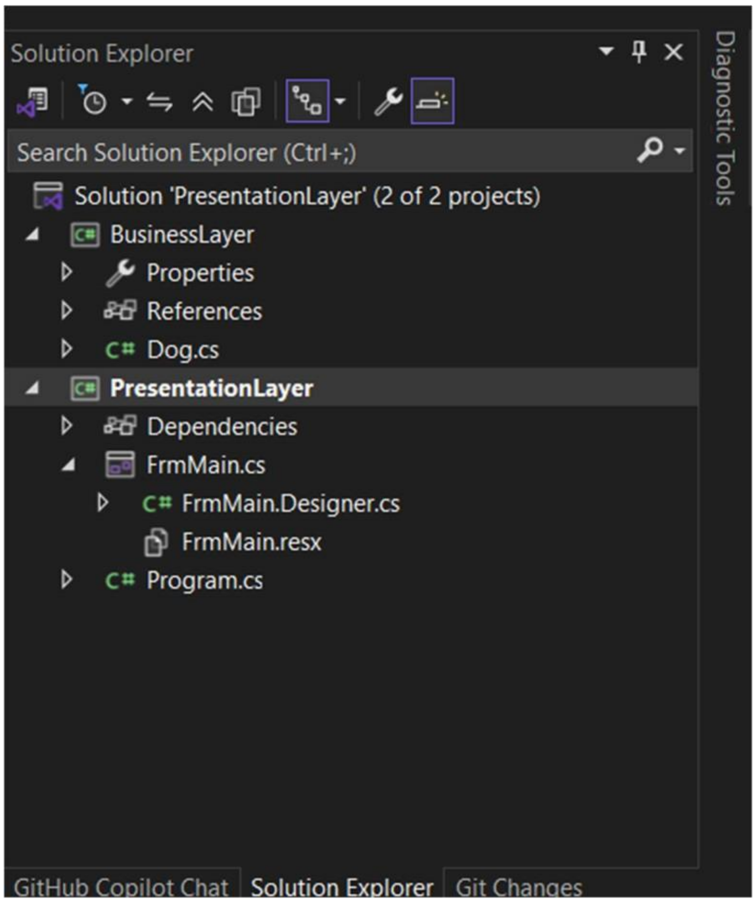


Figure 2: UML Class Diagram for Dog Management Application.

The UML diagram shows the structure of the dog management Application, highlighting the Dog class, with its properties and methods for managing dog data. It is a layered design; the FrmMain class makes it easier to communicate with users and show data. The application's maintainability and concern separation are enhanced by this arrangement.

SCREENSHOT OF THE SOLUTION EXPLORER WITH N-LAYER

Expanded, showing the form and class.



```
Solution 'PresentationLayer' (2 of 2 projects)
  BusinessLayer
    Properties
    References
    Dog.cs
  PresentationLayer
    Dependencies
    FrmMain.cs
      FrmMain.Designer.cs
      FrmMain.resx
    Program.cs
```

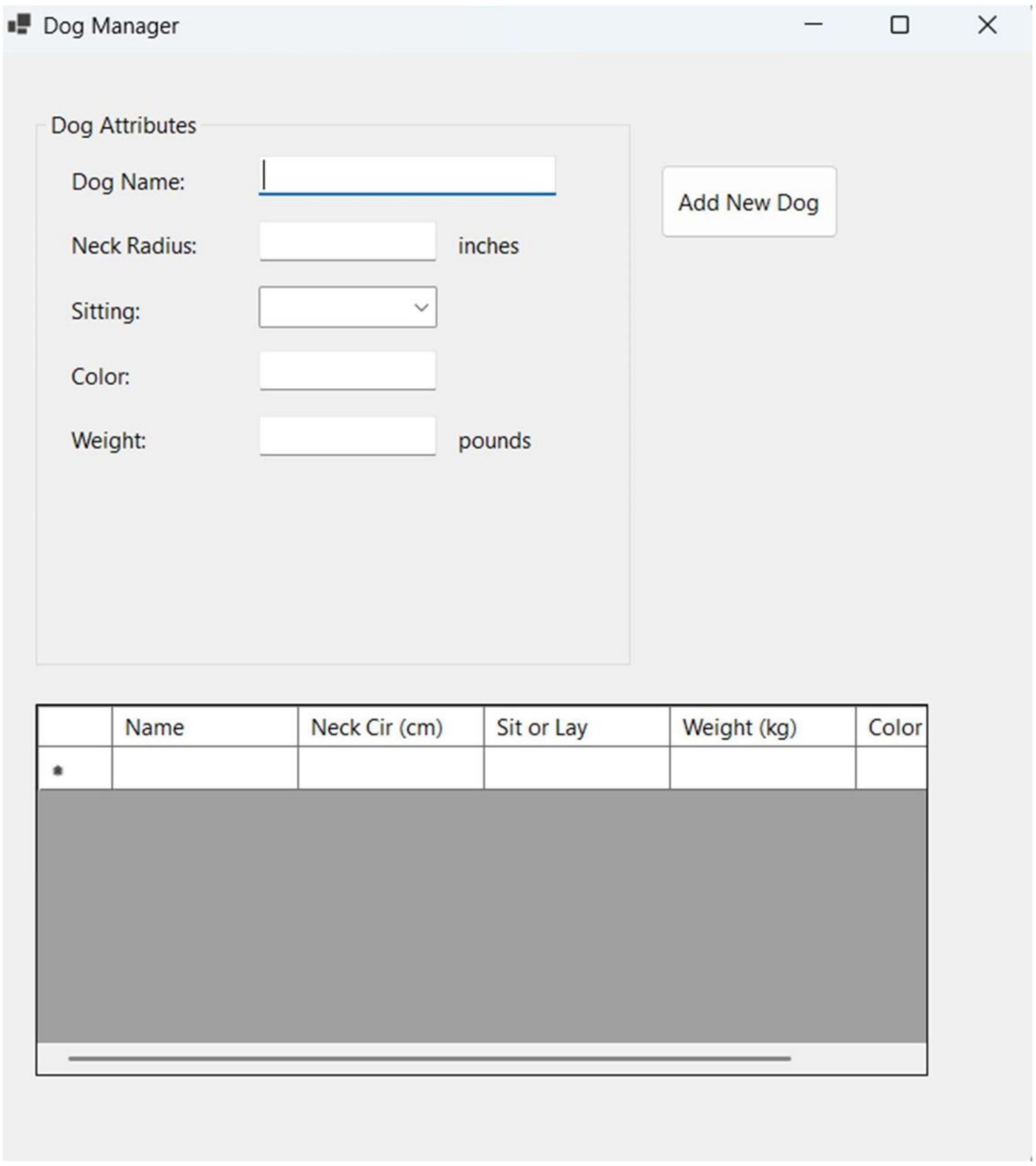
Figure 2:This screenshot shows the use of a multi-layered architecture, where:

Presenta onLayer holds the form and user interface logic.

BusinessLayer holds the Dog class and logic calculations.

This organization supports code maintainability and follows n-layer application on design principles.

Screenshot of the form before populated.



	Name	Neck Cir (cm)	Sit or Lay	Weight (kg)	Color
*					

Figure 3:initial state when it loads

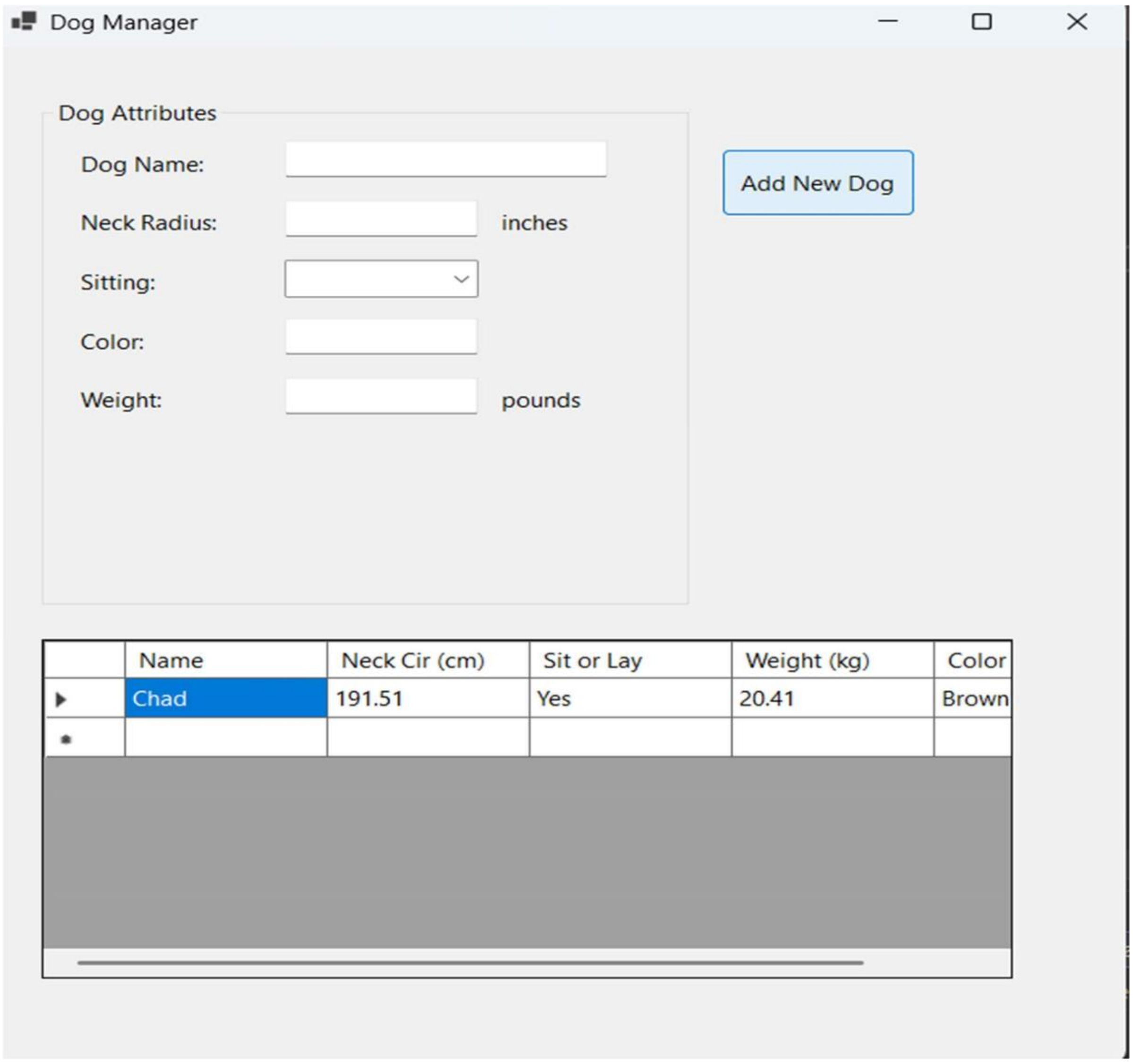
This shows the initial state of the form when it loads.

The text boxes are empty.

The combo box offers two choices: “Yes” or “No.”

The error label is hidden, and the DataGridView is empty.

Screenshot of the form after populated.

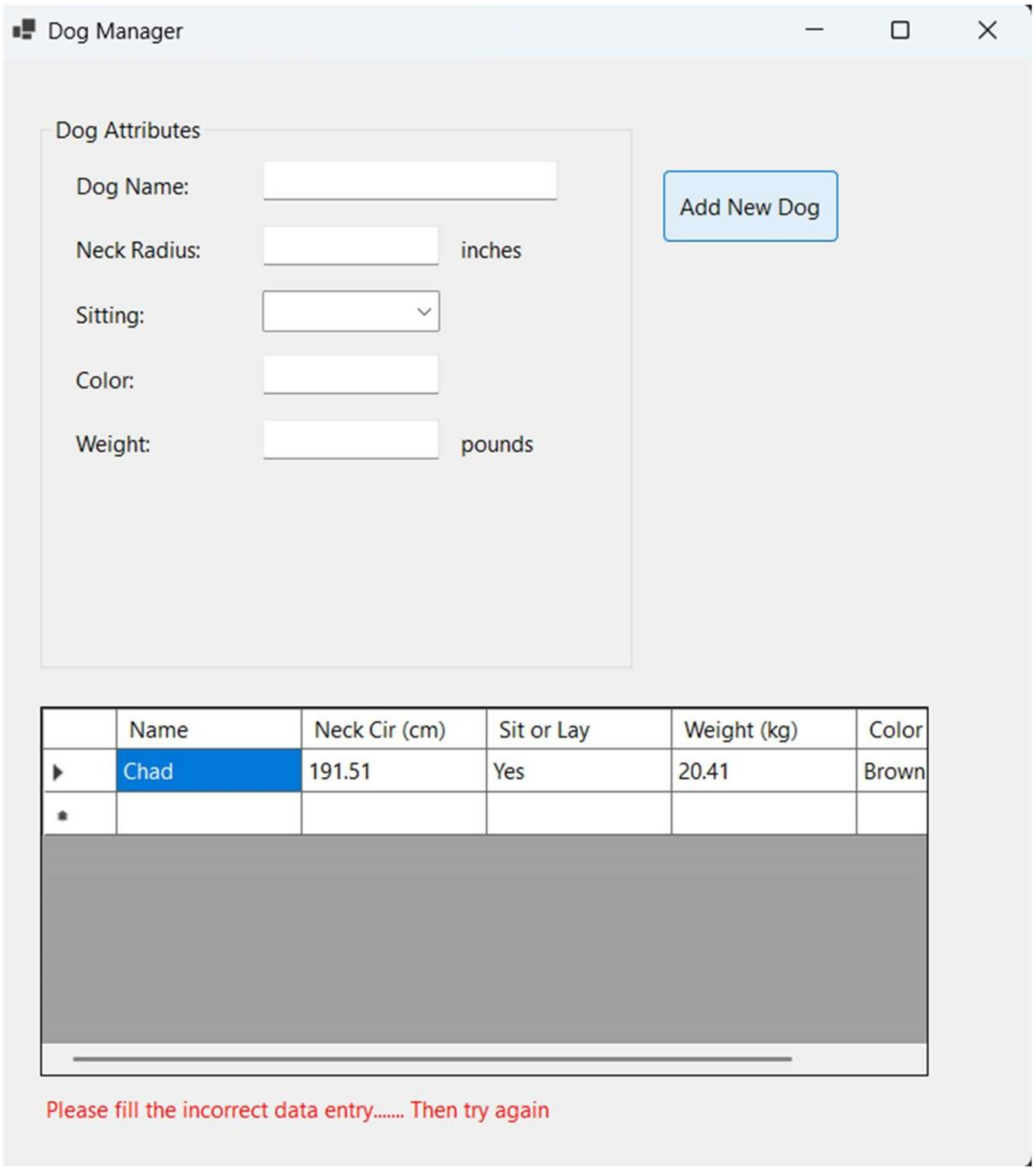


Dog Name:		
Neck Radius:		inches
Sitting:		
Color:		
Weight:		pounds

	Name	Neck Cir (cm)	Sit or Lay	Weight (kg)	Color
▶	Chad	191.51	Yes	20.41	Brown
*					

Figure 4: Populated form of Dog management Application.

After adding a dog successfully, the Dog Management Application interface, as shown in the screenshot, includes a DataGridView filled with information such as the dog's name, weight, and neck circumference. The application's functionality and user-friendly design are confirmed when input fields are cleared for clean new entries.



Dog Manager

Dog Attributes

Dog Name:	
Neck Radius:	inches
Sitting:	
Color:	
Weight:	pounds

	Name	Neck Cir (cm)	Sit or Lay	Weight (kg)	Color
▶	Chad	191.51	Yes	20.41	Brown
*					

Please fill the incorrect data entry....... Then try again

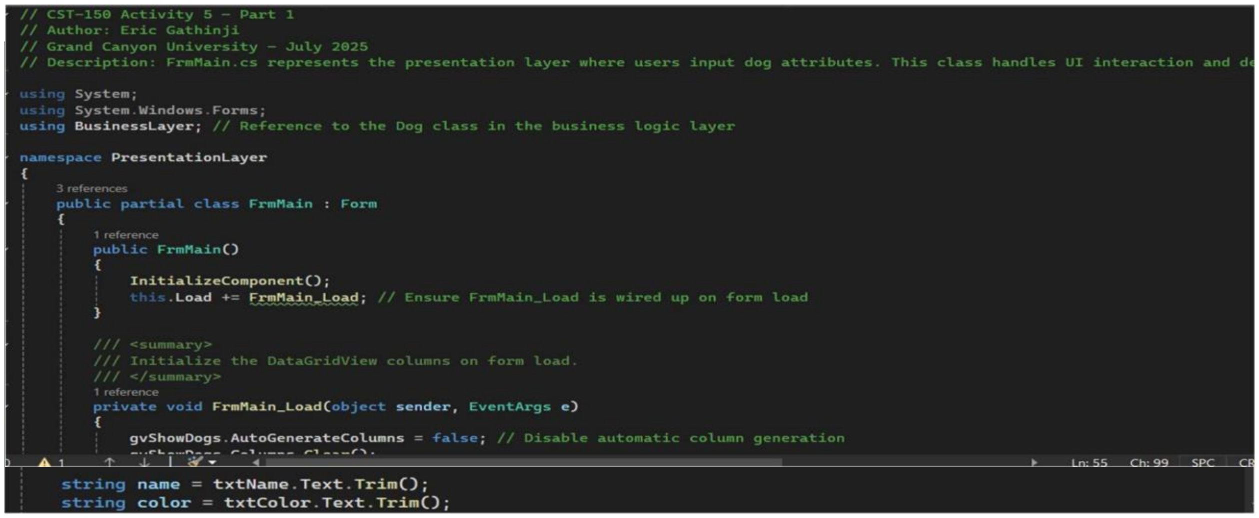
Figure 5: This shows what the form looks like after adding a dog or when there is an error while entering information on the page.

The dog's name, neck circumference, sitting/lying status, weight in kg, and color appear in the table.

Input fields are cleared after the addition.

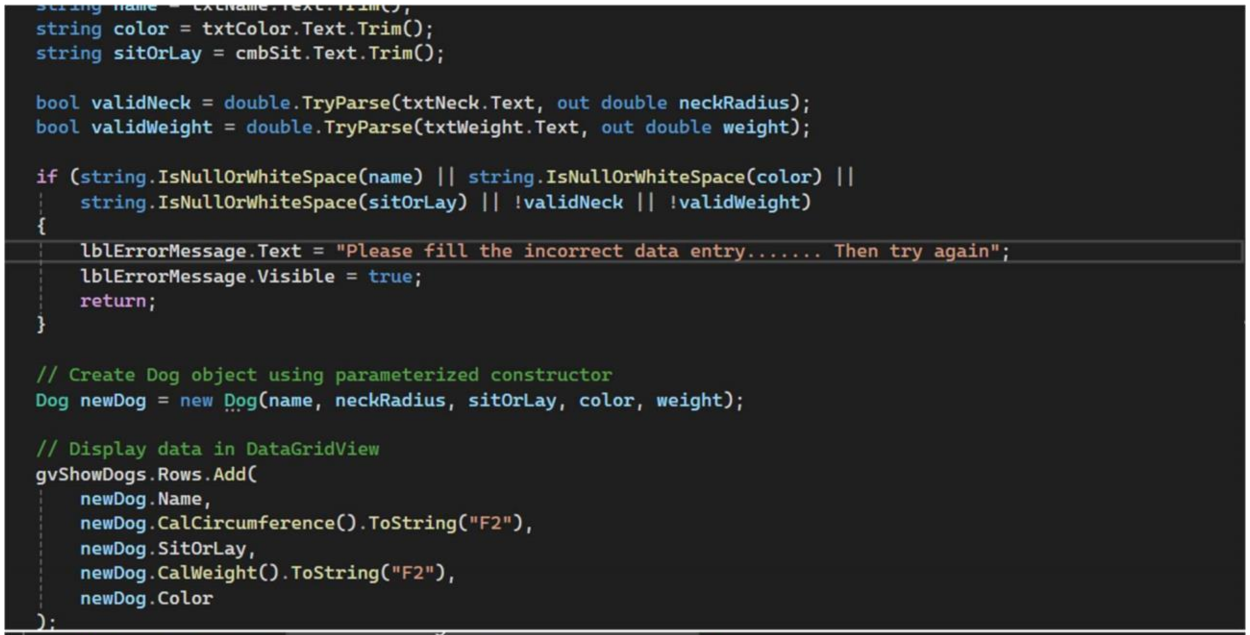
This confirms the logic of object creation, data display, and field reset is function.

SCREENSHOT(S) OF THE CODE BEHIND FrmMain.

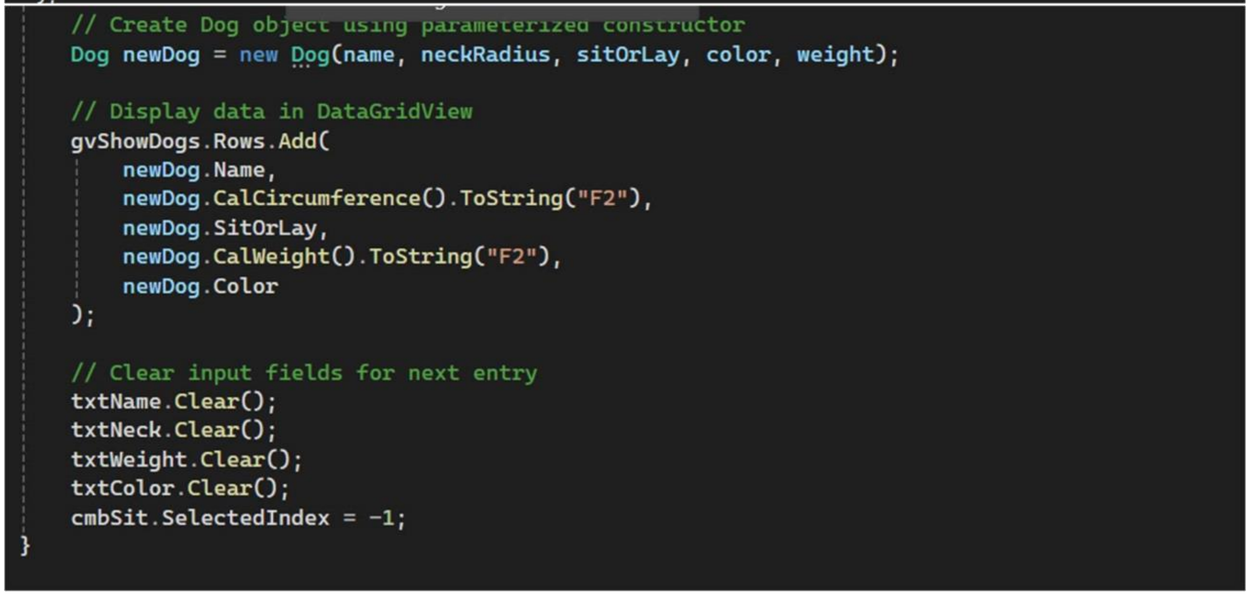


```
// CST-150 Activity 5 - Part 1
// Author: Eric Gathinji
// Grand Canyon University - July 2025
// Description: FrmMain.cs represents the presentation layer where users input dog attributes. This class handles UI interaction and de

using System;
using System.Windows.Forms;
using BusinessLayer; // Reference to the Dog class in the business logic layer

namespace PresentationLayer
{
    3 references
    public partial class FrmMain : Form
    {
        1 reference
        public FrmMain()
        {
            InitializeComponent();
            this.Load += FrmMain_Load; // Ensure FrmMain_Load is wired up on form load
        }

        /// <summary>
        /// Initialize the DataGridView columns on form load.
        /// </summary>
        1 reference
        private void FrmMain_Load(object sender, EventArgs e)
        {
            gvShowDogs.AutoGenerateColumns = false; // Disable automatic column generation
```

```
    string name = txtName.Text.Trim();
    string color = txtColor.Text.Trim();
```



```
string name = txtName.Text.Trim();
string color = txtColor.Text.Trim();
string sitOrLay = cmbSit.Text.Trim();

bool validNeck = double.TryParse(txtNeck.Text, out double neckRadius);
bool validWeight = double.TryParse(txtWeight.Text, out double weight);

if (string.IsNullOrWhiteSpace(name) || string.IsNullOrWhiteSpace(color) ||
    string.IsNullOrWhiteSpace(sitOrLay) || !validNeck || !validWeight)
{
    lblErrorMessage.Text = "Please fill the incorrect data entry....... Then try again";
    lblErrorMessage.Visible = true;
    return;
}

// Create Dog object using parameterized constructor
Dog newDog = new Dog(name, neckRadius, sitOrLay, color, weight);

// Display data in DataGridView
gvShowDogs.Rows.Add(
    newDog.Name,
    newDog.CalCircumference().ToString("F2"),
    newDog.SitOrLay,
    newDog.CalWeight().ToString("F2"),
    newDog.Color
);
```



```
    // Create Dog object using parameterized constructor
    Dog newDog = new Dog(name, neckRadius, sitOrLay, color, weight);

    // Display data in DataGridView
    gvShowDogs.Rows.Add(
        newDog.Name,
        newDog.CalCircumference().ToString("F2"),
        newDog.SitOrLay,
        newDog.CalWeight().ToString("F2"),
        newDog.Color
    );

    // Clear input fields for next entry
    txtName.Clear();
    txtNeck.Clear();
    txtWeight.Clear();
    txtColor.Clear();
    cmbSit.SelectedIndex = -1;
}
```

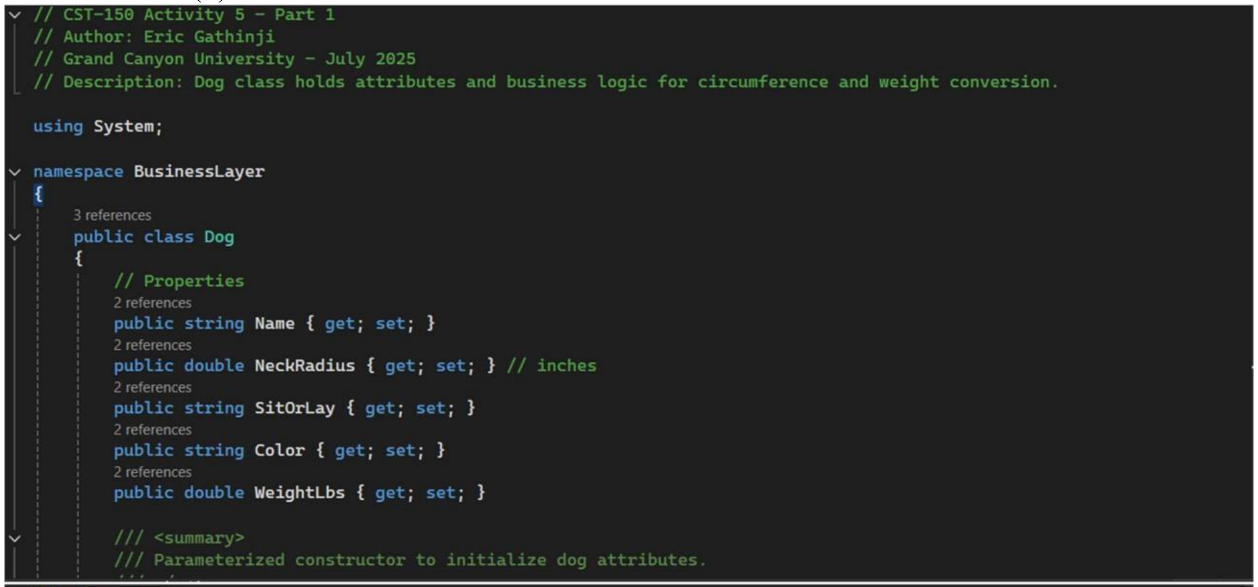
Figure 6: The screenshot shows the FrmMain.cs file:

It contains methods for validating input, creating the dog object, and populating the grid.

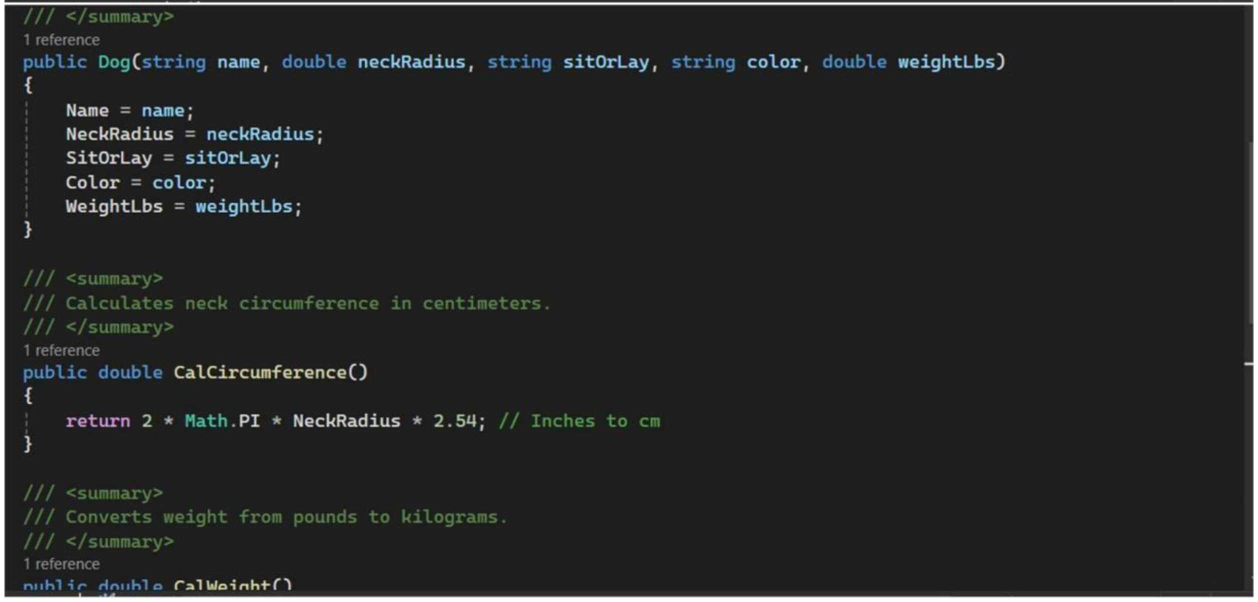
It also contains the Load event that defines columns for the DataGridView.

This supports event-driven programming and keeps logic outside of the Designer.

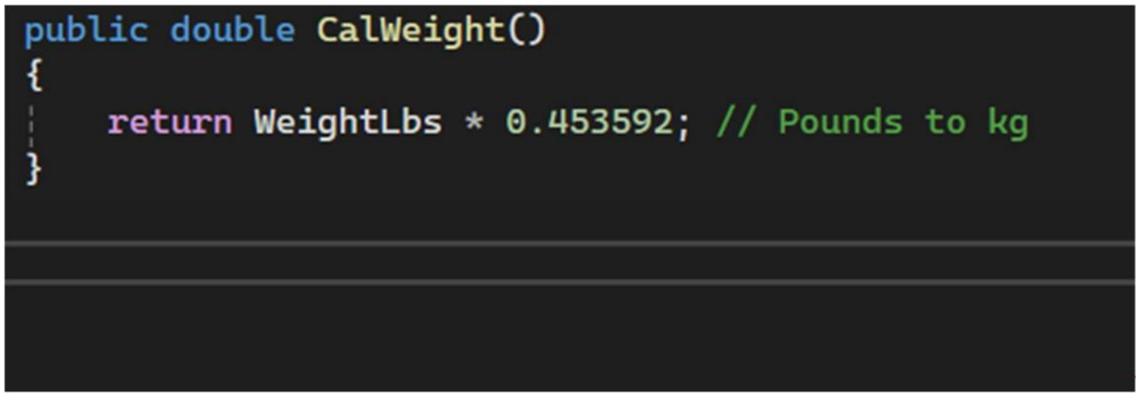
SCREENSHOT(S) OF THE CODE FOR DOG CLASS.



```
// CST-150 Activity 5 - Part 1
// Author: Eric Gathinji
// Grand Canyon University - July 2025
// Description: Dog class holds attributes and business logic for circumference and weight conversion.

using System;

namespace BusinessLayer
{
    3 references
    public class Dog
    {
        // Properties
        2 references
        public string Name { get; set; }
        2 references
        public double NeckRadius { get; set; } // inches
        2 references
        public string SitOrLay { get; set; }
        2 references
        public string Color { get; set; }
        2 references
        public double WeightLbs { get; set; }

        /// <summary>
        /// Parameterized constructor to initialize dog attributes.
```



```
/// </summary>
1 reference
public Dog(string name, double neckRadius, string sitOrLay, string color, double weightLbs)
{
    Name = name;
    NeckRadius = neckRadius;
    SitOrLay = sitOrLay;
    Color = color;
    WeightLbs = weightLbs;
}

/// <summary>
/// Calculates neck circumference in centimeters.
/// </summary>
1 reference
public double CalCircumference()
{
    return 2 * Math.PI * NeckRadius * 2.54; // Inches to cm
}

/// <summary>
/// Converts weight from pounds to kilograms.
/// </summary>
1 reference
public double CalWeight()
```



```
public double CalWeight()
{
    return WeightLbs * 0.453592; // Pounds to kg
}
```

Figure 7: The Code for the Dog class.

The Dog.cs file contains:

A parameterized constructor to initialize proper es.

Two methods: CalCircumference() and CalWeight() to perform required calculations.

These methods separate logic from UI and promote reusability and testability.

Follow-up questions

1. What was challenging?

It was difficult to set up the multi-layered design with distinct boundaries between the PresentationLayer and BusinessLayer, particularly to ensure that the data was correctly sent between the Dog class and the form.

2. What did you learn? I gained knowledge on managing user input validation in a Windows Forms environment, creating and utilizing classes in a structured program, and using Class methods to calculate, like weight and circumference.
3. How would you improve the project?
   Allowing the user to change or remove dog entries straight from the DataGridView and providing more thorough error handling, including indicating invalid fields, are two ways I would improve my project.

4. How can you use what you learned on the job?
   Understanding how to arrange code in multiple layers and classes when creating user-friendly applications.

ADD ON

Monday 7th July

Start: 8:00 am End: 2:00 pm Activity: 5

Tuesday8 th July

Start: 8:00 End: 2:30 pm Activity: 5

Wednesday 9th July

Start:2:00 am End: 4:00 am Activity: 5

Saturday 12th July

Start 12:00 am End: 4:00 am: Activity 5