# Milestone 3

Ian M. McConihay

College of Science, Engineering and Technology, Grand Canyon University

CST-150: C# Programming I

Mark Smithers

September 08, 2024

**Video Link:**

**https://www.loom.com/share/5959767e57bd4f65bad6446d704be2b7?sid=83f1b903-c857-4d99-9844-88188f19ff33**

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

What was challenging?

Understanding the creation of the data tavle to be the source for the DataGridView

What did you learn?

Binding text file data.

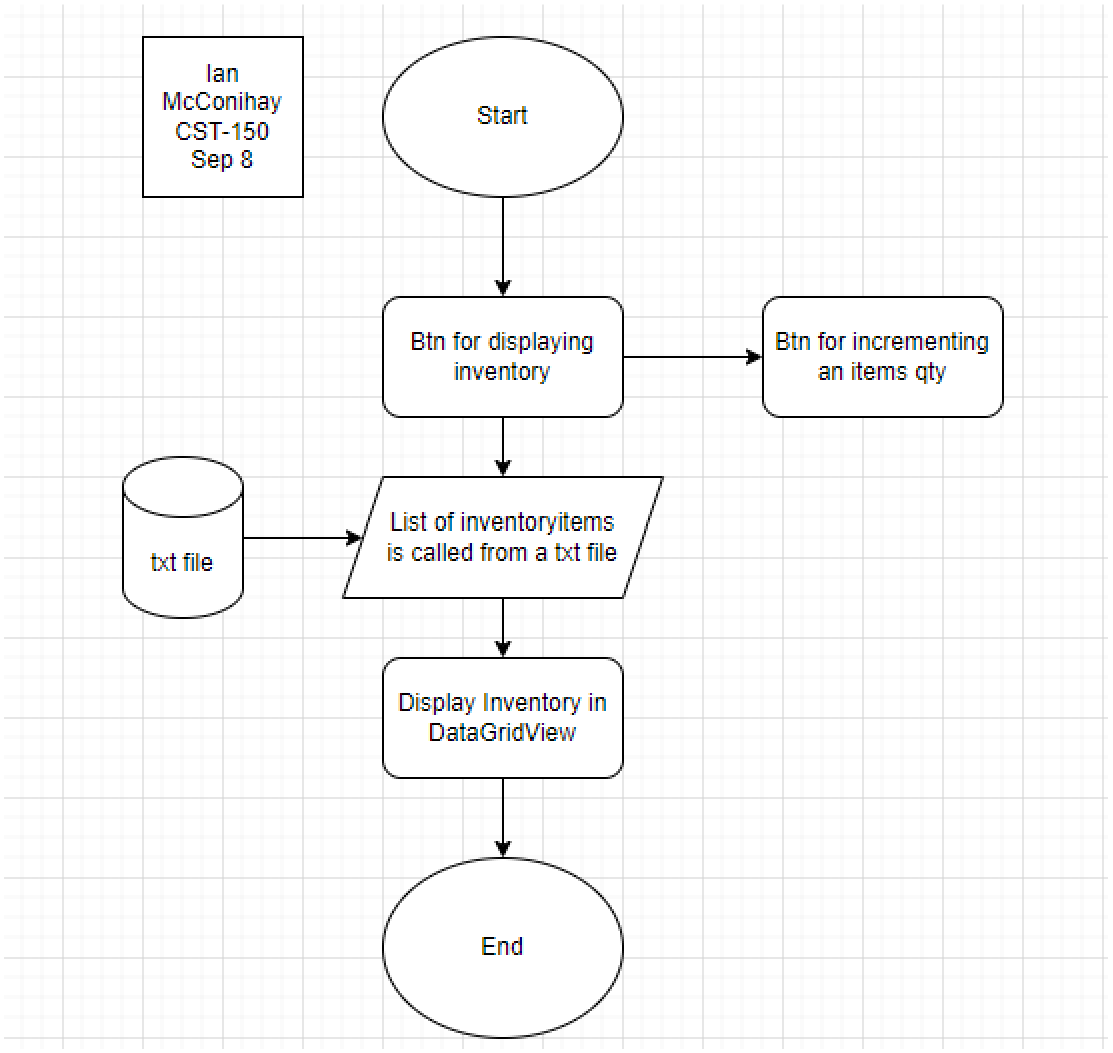
How would you improve on the project?

I would use a different source for data storage.

How can you use what you learned on the job?

Reading and writing text files can be useful for regexing out spesfic information you need.

Figure 1: FlowChart


Ian McConihay CST-150 Sep 8
Start
Btn for displaying inventory
Btn for incrementing an items qty
txt file
List of inventoryitems is called from a txt file
Display Inventory in DataGridView
End

At the start of this application will open to a button for the user to click and persist a grid view of inventory items from a text file. Another button will allow the user to increment one of the items quantity. The datagridview provides table functions for the user to view the inventory.

Figure 2: UML InventoryItem

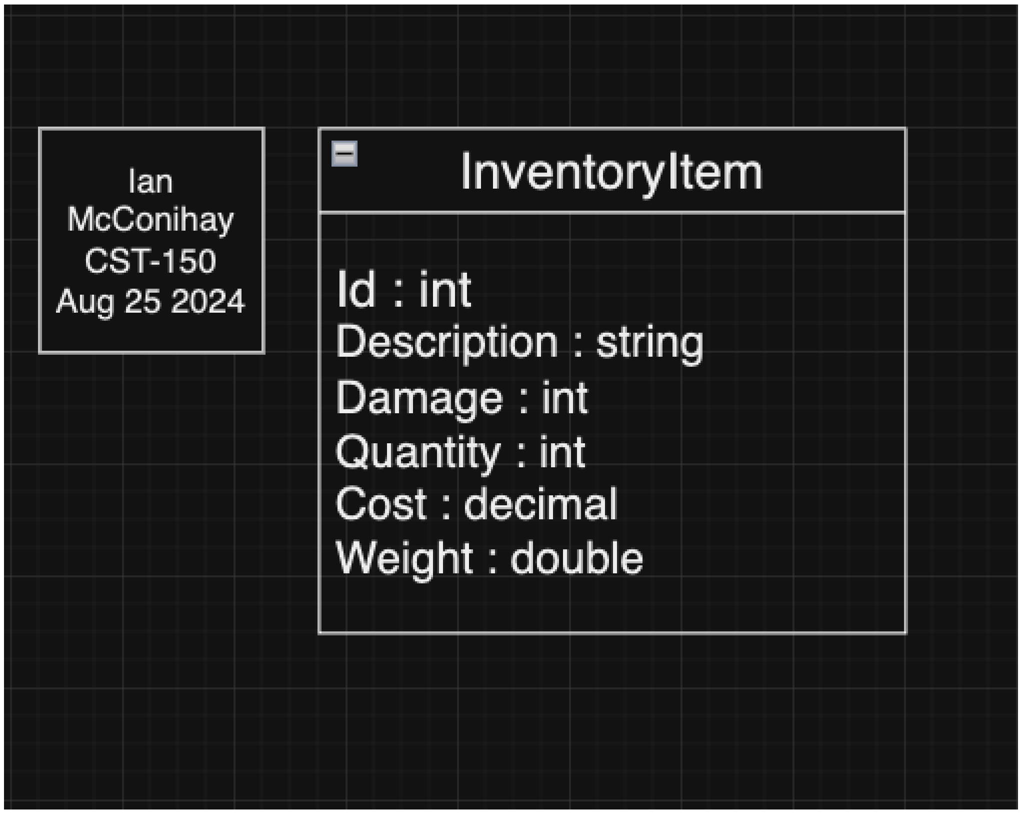
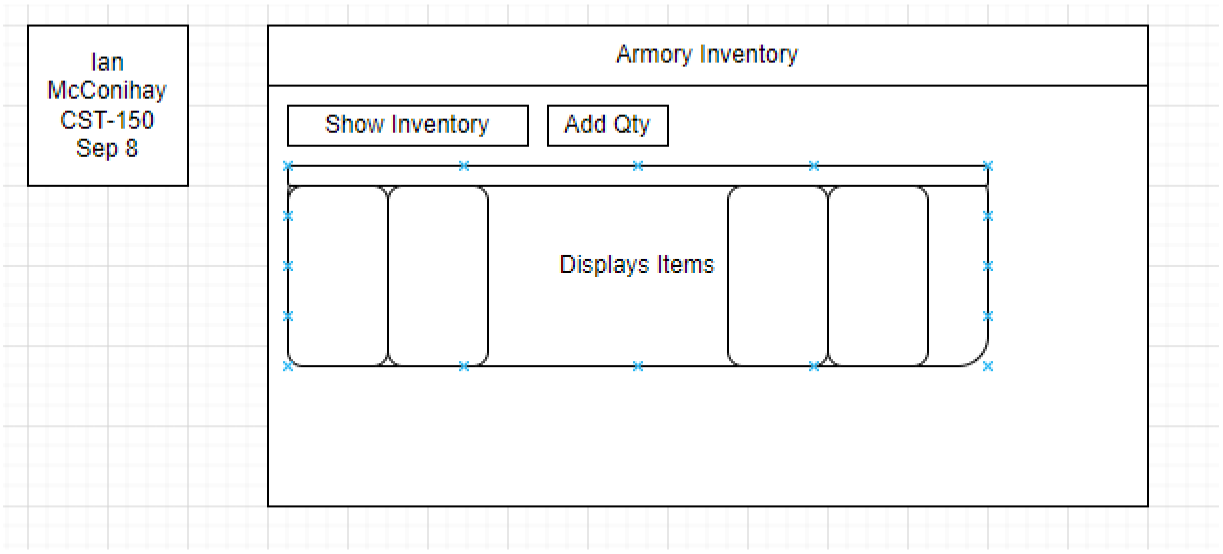


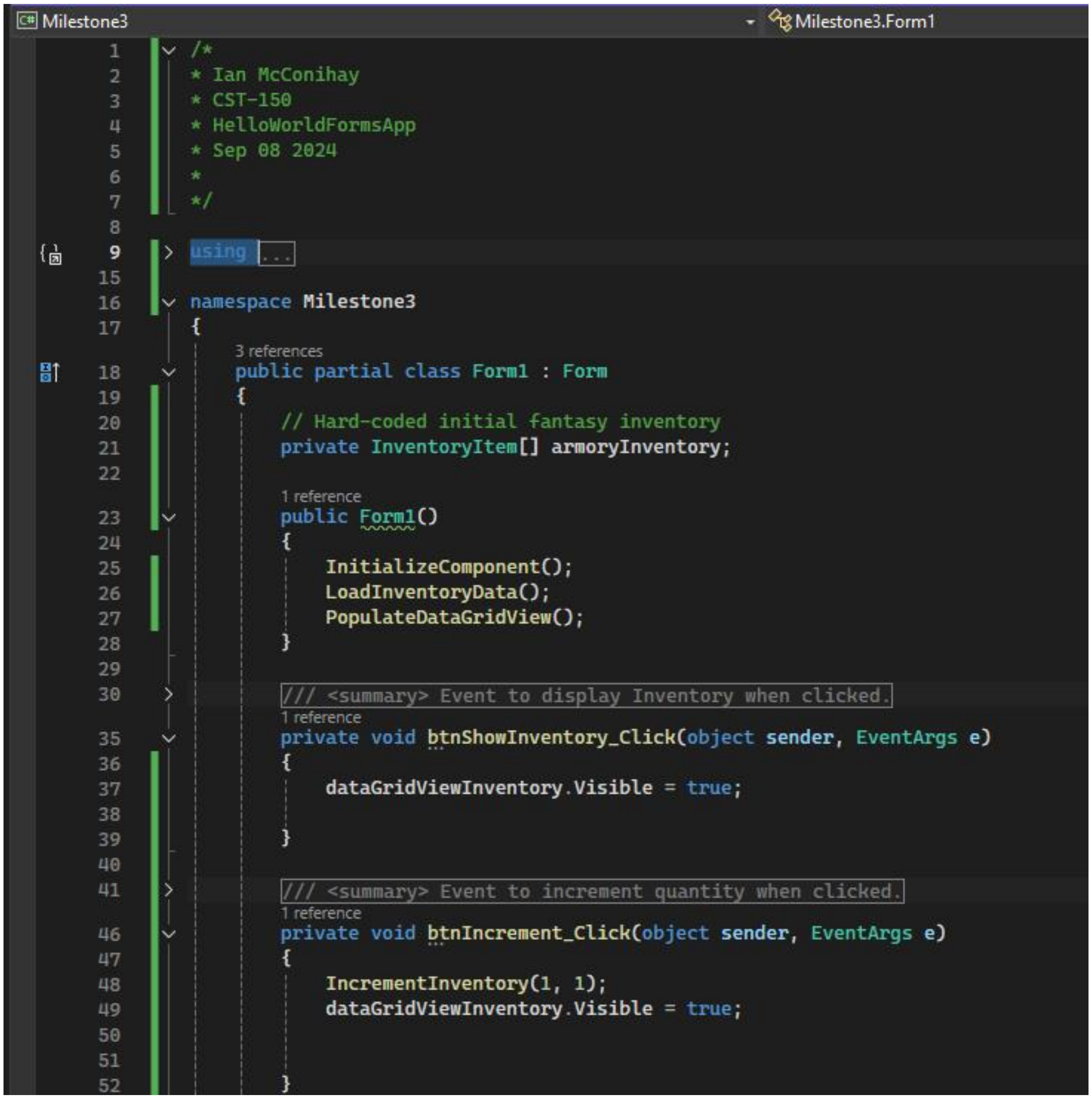
Figure 3: Wireframe



The updated wire frame has a handful of changes. The form has a name when displayed for the user. The button shows inventory is located above where the list will be displayed using a data grid view. The user will have columns to display the information. Aso an Add Qty button for incrementing an items quantity.

## Application Screenshots

Figure 4: Code

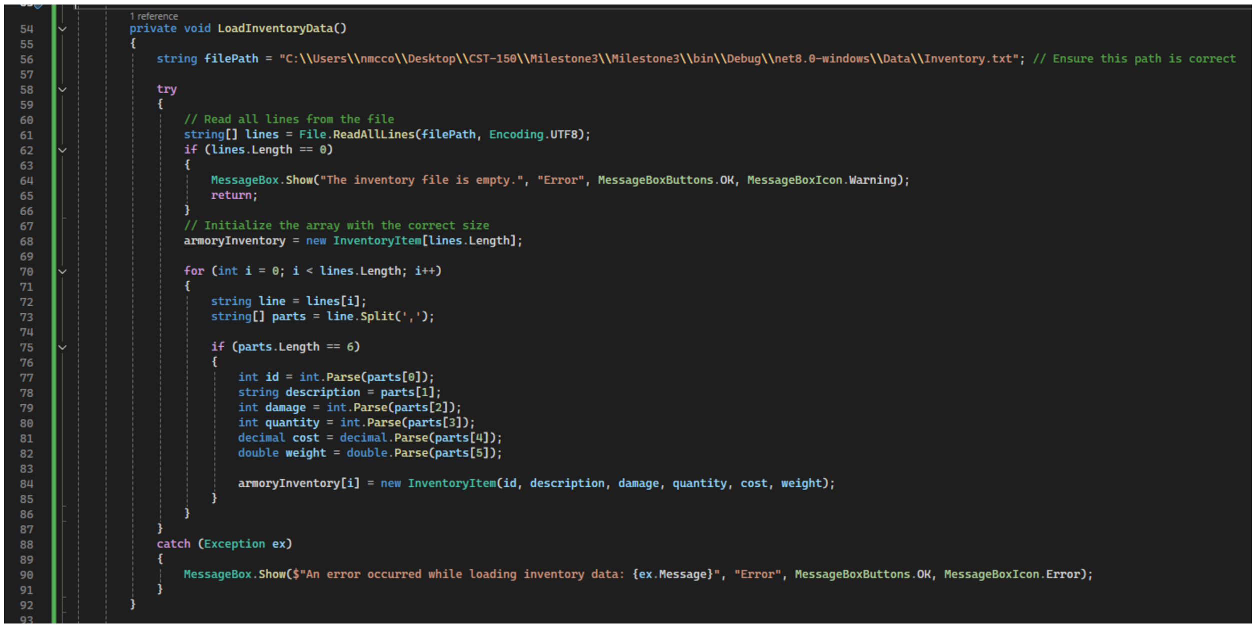


Milestone3 | Milestone3.Form1

```
/*
* Ian McConihay
* CST-150
* HelloWorldFormsApp
* Sep 08 2024
*
*/

using ...

namespace Milestone3
{
    3 references
    public partial class Form1 : Form
    {
        // Hard-coded initial fantasy inventory
        private InventoryItem[] armoryInventory;

        1 reference
        public Form1()
        {
            InitializeComponent();
            LoadInventoryData();
            PopulateDataGridView();
        }

        /// <summary> Event to display Inventory when clicked.
        1 reference
        private void btnShowInventory_Click(object sender, EventArgs e)
        {
            dataGridViewInventory.Visible = true;

        }

        /// <summary> Event to increment quantity when clicked.
        1 reference
        private void btnIncrement_Click(object sender, EventArgs e)
        {
            IncrementInventory(1, 1);
            dataGridViewInventory.Visible = true;


        }
```

For figure 4 we start off with the citation at the top. I have a btnShowInventory_Click method that causes the inventory display even to take place. btnIncrement_Click is a method for Calling incrementInventory.

Figure 5: Code

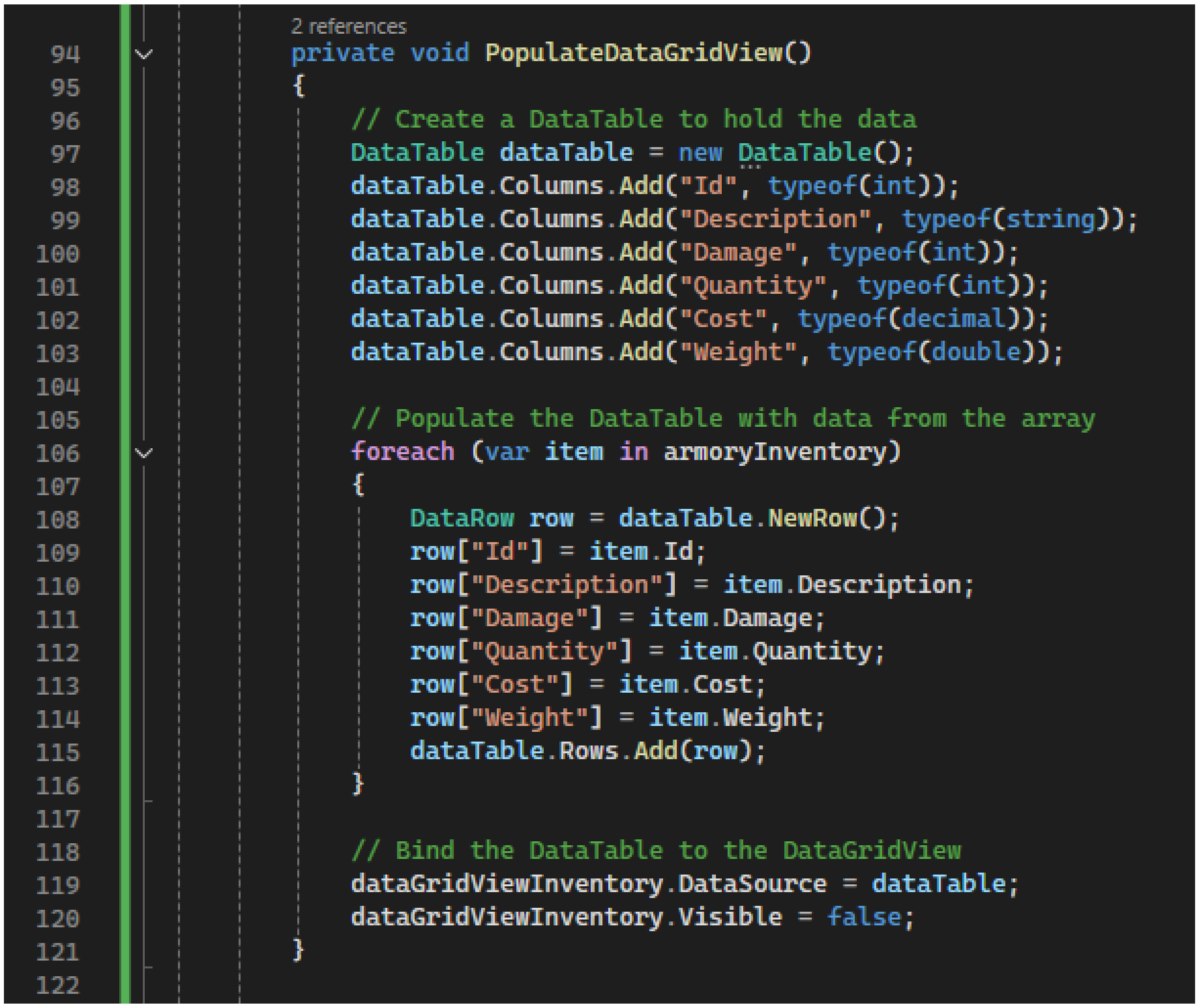


```
1 reference
private void LoadInventoryData()
{
    string filePath = "C:\\Users\\nmcco\\Desktop\\CST-150\\Milestone3\\Milestone3\\bin\\Debug\\net8.0-windows\\Data\\Inventory.txt"; // Ensure this path is correct

    try
    {
        // Read all lines from the file
        string[] lines = File.ReadAllLines(filePath, Encoding.UTF8);
        if (lines.Length == 0)
        {
            MessageBox.Show("The inventory file is empty.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }
        // Initialize the array with the correct size
        armoryInventory = new InventoryItem[lines.Length];

        for (int i = 0; i < lines.Length; i++)
        {
            string line = lines[i];
            string[] parts = line.Split(',');

            if (parts.Length == 6)
            {
                int id = int.Parse(parts[0]);
                string description = parts[1];
                int damage = int.Parse(parts[2]);
                int quantity = int.Parse(parts[3]);
                decimal cost = decimal.Parse(parts[4]);
                double weight = double.Parse(parts[5]);

                armoryInventory[i] = new InventoryItem(id, description, damage, quantity, cost, weight);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred while loading inventory data: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Figure 5 has the LodeInventoryData method that calls to the filePath leading to the txt file.

Using a try catch we read all the lines in the file and break them up using a for loop. Once broke up its added to an armoryInventory ready for use.

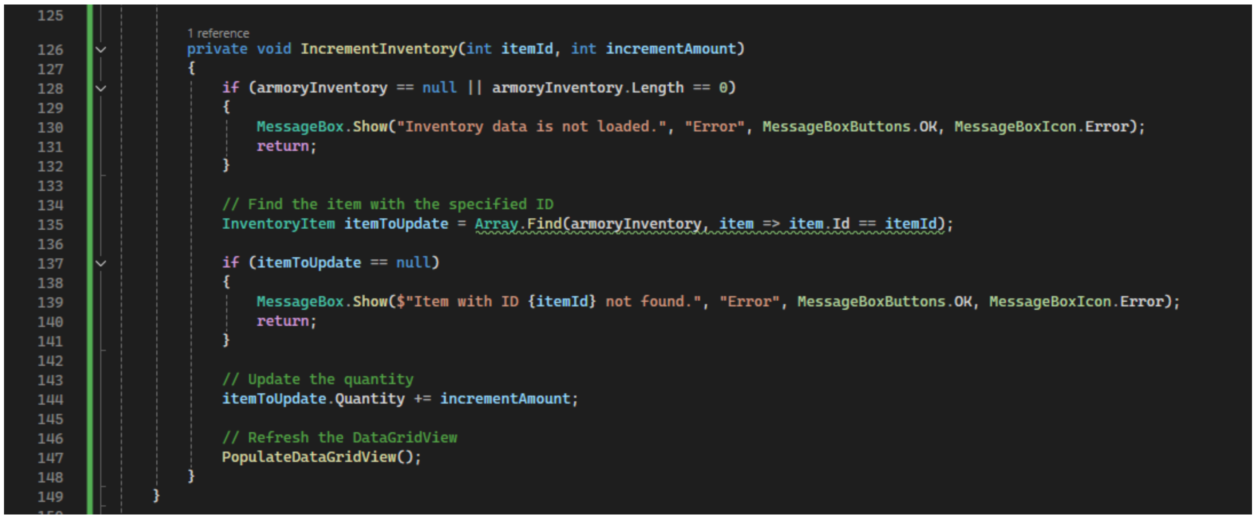
Figure 6: Code



```
2 references
private void PopulateDataGridView()
{
    // Create a DataTable to hold the data
    DataTable dataTable = new DataTable();
    dataTable.Columns.Add("Id", typeof(int));
    dataTable.Columns.Add("Description", typeof(string));
    dataTable.Columns.Add("Damage", typeof(int));
    dataTable.Columns.Add("Quantity", typeof(int));
    dataTable.Columns.Add("Cost", typeof(decimal));
    dataTable.Columns.Add("Weight", typeof(double));

    // Populate the DataTable with data from the array
    foreach (var item in armoryInventory)
    {
        DataRow row = dataTable.NewRow();
        row["Id"] = item.Id;
        row["Description"] = item.Description;
        row["Damage"] = item.Damage;
        row["Quantity"] = item.Quantity;
        row["Cost"] = item.Cost;
        row["Weight"] = item.Weight;
        dataTable.Rows.Add(row);
    }

    // Bind the DataTable to the DataGridView
    dataGridViewInventory.DataSource = dataTable;
    dataGridViewInventory.Visible = false;
}
```

Figure 6 has PopulateDataGrid view. This method creates a DataTable to add columns representing the model InventoryItem. Then populates the broken up pieces in the armoryInventory array and plugs them into the datatable. The dataGridViewInventory is then used to house and display the dataTable.

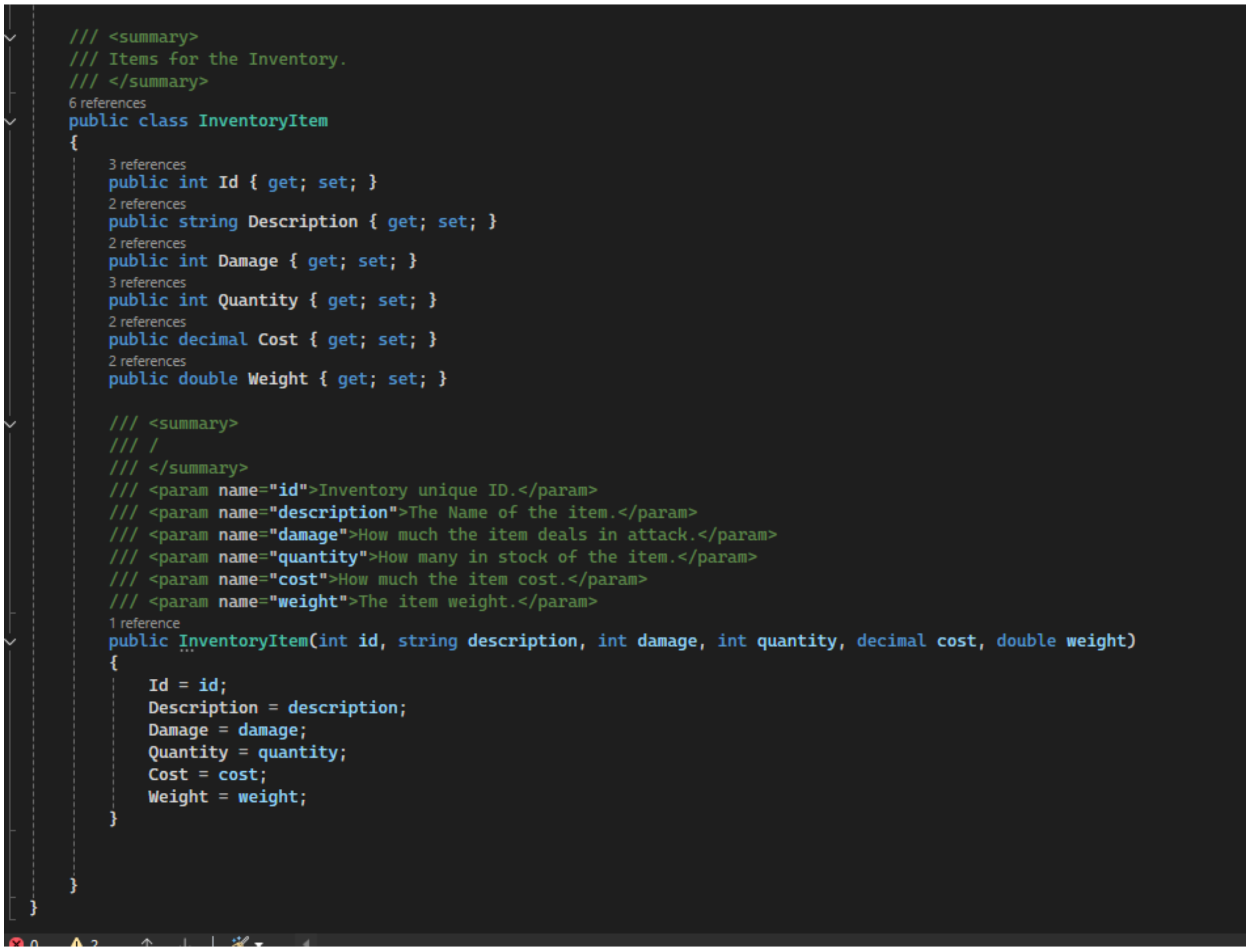
Figure 7: Code



```
        1 reference
        private void IncrementInventory(int itemId, int incrementAmount)
        {
            if (armoryInventory == null || armoryInventory.Length == 0)
            {
                MessageBox.Show("Inventory data is not loaded.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            // Find the item with the specified ID
            InventoryItem itemToUpdate = Array.Find(armoryInventory, item => item.Id == itemId);

            if (itemToUpdate == null)
            {
                MessageBox.Show($"Item with ID {itemId} not found.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            // Update the quantity
            itemToUpdate.Quantity += incrementAmount;

            // Refresh the DataGridView
            PopulateDataGridView();
        }
    }
```

Figure 7 is the IncrementInventory method. We first check to see if the array is null or has a length of 0. Next, we use a lambda expression to find the item by id and add too the quantity depending on the parameters set by the method.

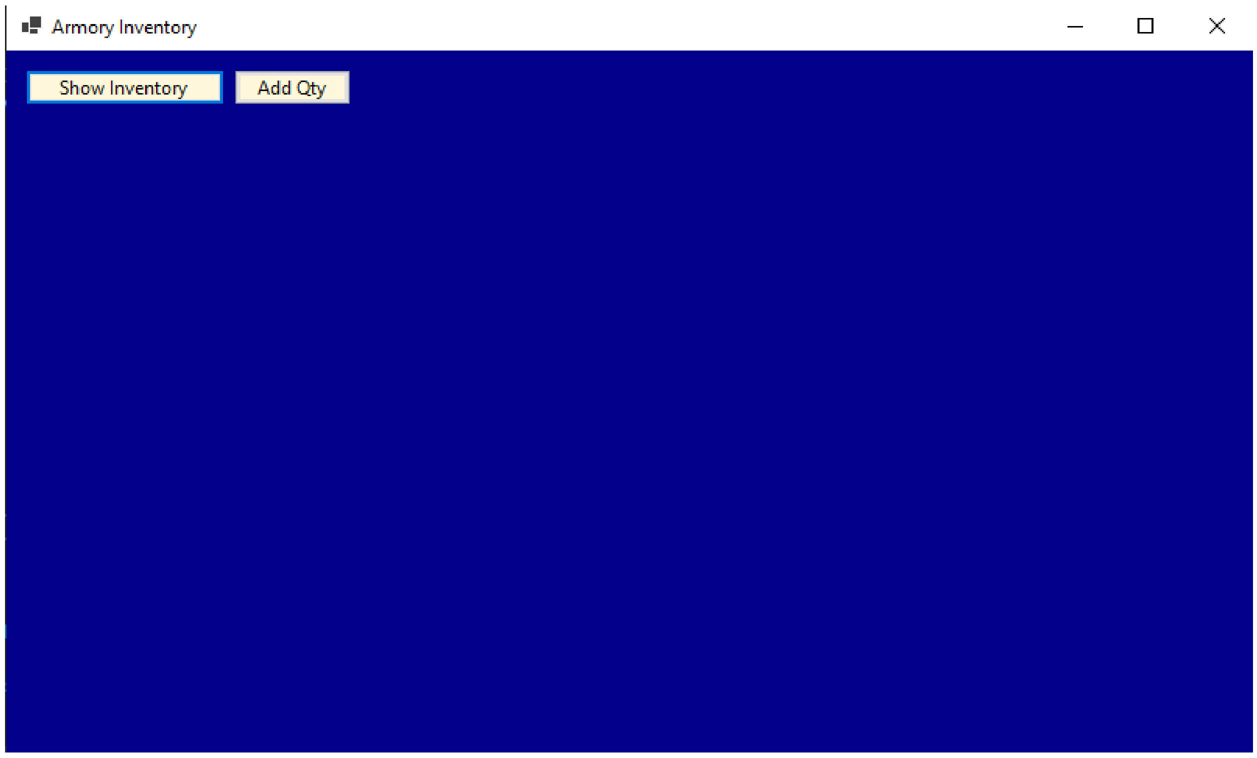
Figure 8: Code



```
    /// <summary>
    /// Items for the Inventory.
    /// </summary>
    6 references
    public class InventoryItem
    {
        3 references
        public int Id { get; set; }
        2 references
        public string Description { get; set; }
        2 references
        public int Damage { get; set; }
        3 references
        public int Quantity { get; set; }
        2 references
        public decimal Cost { get; set; }
        2 references
        public double Weight { get; set; }

        /// <summary>
        /// /
        /// </summary>
        /// <param name="id">Inventory unique ID.</param>
        /// <param name="description">The Name of the item.</param>
        /// <param name="damage">How much the item deals in attack.</param>
        /// <param name="quantity">How many in stock of the item.</param>
        /// <param name="cost">How much the item cost.</param>
        /// <param name="weight">The item weight.</param>
        1 reference
        public InventoryItem(int id, string description, int damage, int quantity, decimal cost, double weight)
        {
            Id = id;
            Description = description;
            Damage = damage;
            Quantity = quantity;
            Cost = cost;
            Weight = weight;
        }


    }
}
```

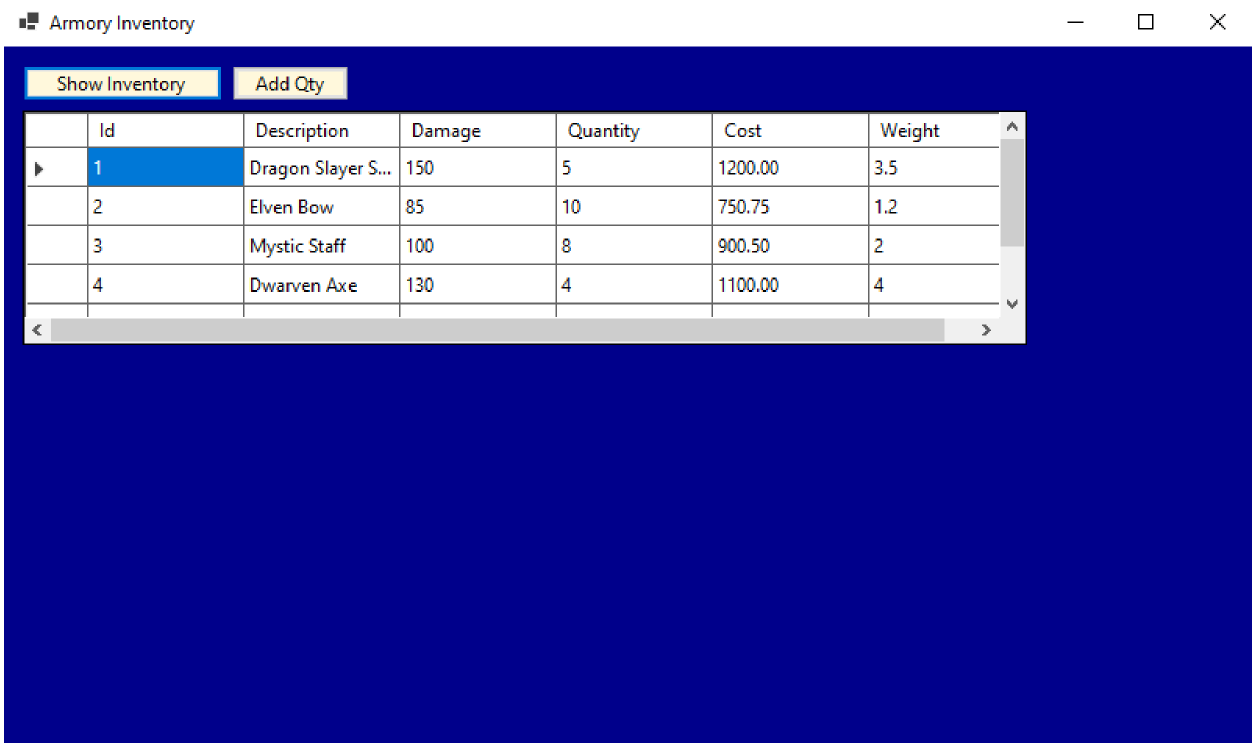
This is the inventory item class. The onlything added was a constructor. The constructor is used in the LoadInventoryData method for taking the array and pluging it into the model.

Figure 9: Application Start



The start of the application displays the show inventory button and add qty button. From here I can also point out the updated color scheme. I will be research the sources posted to figure out a design for the final Milestone.

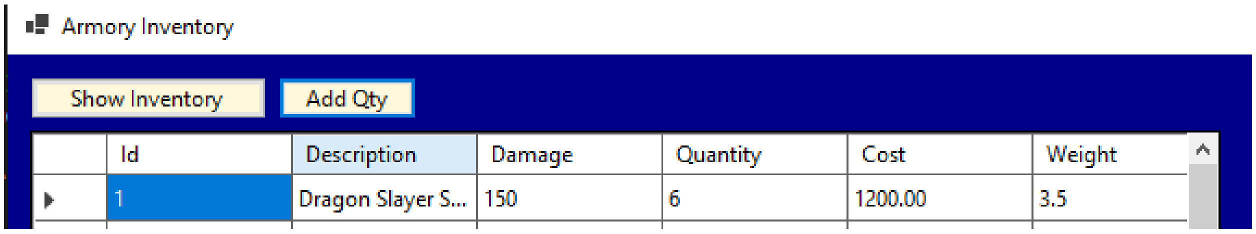
Figure 10: Application Display Inventory



| | Id | Description | Damage | Quantity | Cost | Weight |
|---|---|---|---|---|---|---|
| ▶ | 1 | Dragon Slayer S... | 150 | 5 | 1200.00 | 3.5 |
| | 2 | Elven Bow | 85 | 10 | 750.75 | 1.2 |
| | 3 | Mystic Staff | 100 | 8 | 900.50 | 2 |
| | 4 | Dwarven Axe | 130 | 4 | 1100.00 | 4 |

After clicking the button, the inventory list is displayed in a data grid view. The view allows the user to sort through the columns easier. For the furture I think I will put the buttons below the view.

Figure 11: Application Increment Inventory item



| | Id | Description | Damage | Quantity | Cost | Weight |
|---|---|---|---|---|---|---|
| ▶ | 1 | Dragon Slayer S... | 150 | 6 | 1200.00 | 3.5 |

Figure 11 is post pushing the Add Qty button. As we can see the Quantity went from 5 to 6 for the id 1 item. This is only set to increment id 1.

**Bug Reports**

Bug Report

Class name

Method name :

Steps to reproduce the bug:

Expected results

Actual results

details: N/A for milestone 1

Solution

1. List your computer specs (type of computer, OS, memory, etc)

| | |
|---|---|
| Device name | DESKTOP-IAQ5CCD |
| Processor | Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz   1.80 GHz |
| Installed RAM | 8.00 GB (7.88 GB usable) |
| Device ID | A0AC8D02-4885-4491-B27B-B40F0A0D2E35 |
| Product ID | 00356-02139-31547-AAOEM |
| System type | 64-bit operating system, x64-based processor |
| Pen and touch | Touch support with 10 touch points |

2. Create 3 test cases

Button Click Displays Inventory: Verifies that the button correctly updates the dataGridViewInventory with inventory details.

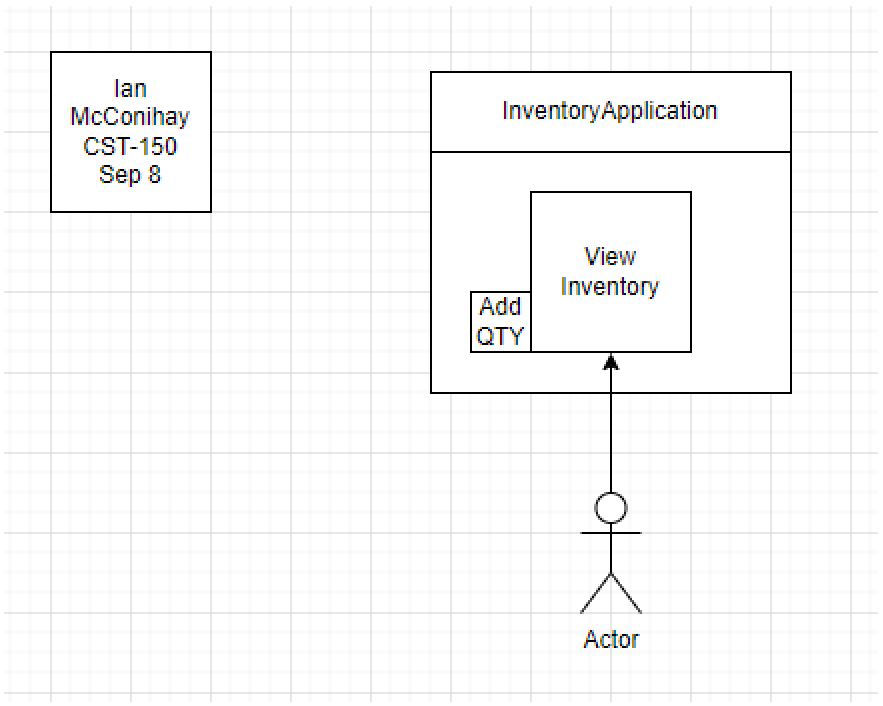
dataGridViewInventory Initialization: Ensures that the dataGridViewInventory starts in the correct initial state before user interaction.

Verify Inventory Content Format: Confirms that the displayed inventory information is formatted correctly.

3. List 3 Programming conventions that will be used all milestones

Naming, Format, and Documentation Conventions

4. Create Use case diagram



System Boundary: Representing the WinForms application.
Use Case: "View Inventory" indicating the functionality provided by the application.
Actor: "User" who interacts with the system to view the inventory.

Monday
Start: 900pm End: 9:30pm Activity: Read announcements
Start: 930pm End: 1030 Activity: DQ1 and DQ 2
Start: 1030pm End: 1100pm Activity: Read Book
Tuesday
Start: 900pm End: 9:30pm Activity: Participation post
Start: 930pm End: 1030 Activity: Activity
Start: 1030pm End: 1100pm Activity: Read Book
Wednesday

Start: End: Activity: N/A
Start: End: Activity: N/A
Start: End: Activity: N/A
Thursday
Start: 900pm End: 9:30pm Activity: Participation post
Start: 930pm End: 1030 Activity: Activity
Start: 1030pm End: 1100pm Activity: Read Book
Friday
Start: 900pm End: 9:30pm Activity: Participation post
Start: 930pm End: 1030 Activity: Milestone
Start: 1030pm End: 1100pm Activity: Read Book
Saturday
Start: 900pm End: 9:30pm Activity: Milestone
Start: 930pm End: 1030 Activity: Milestone
Start: 1030pm End: 1100pm Activity: Read Book
Sunday
Start: 900pm End: 9:30pm Activity: Activity 3
Start: 930pm End: 1030 Activity: Milestone
Start: 1030pm End: 1100pm Activity: Read Book