Name: Eric Gathinji

Programming in C# CST-150-0500

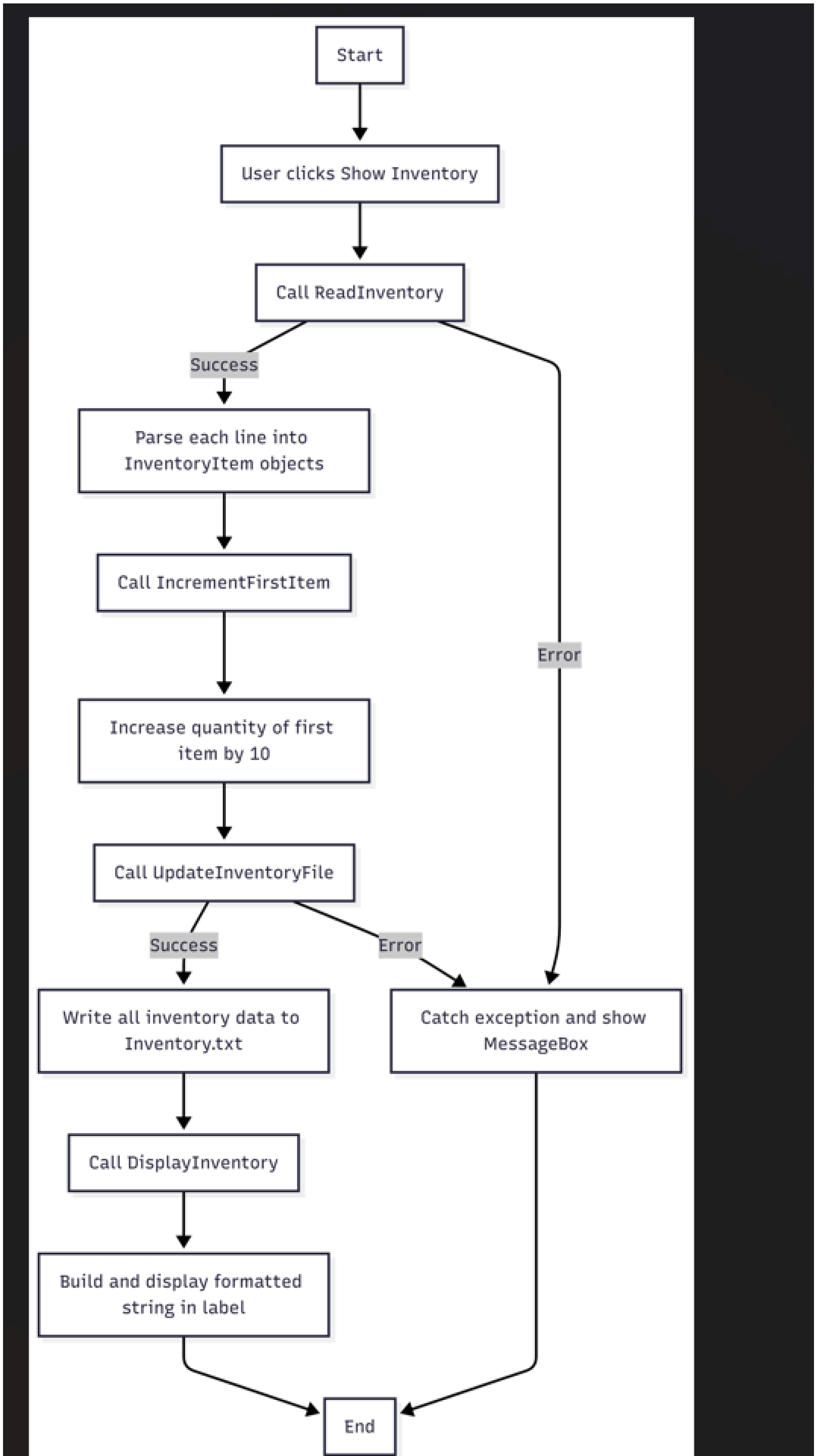
Grand Canyon University

5th July 2025

Milestone 4

LOOM VIDEO LINK

UPDATED FLOWCHART


Start
User clicks Show Inventory
Call ReadInventory
Success
Error
Parse each line into InventoryItem objects
Call IncrementFirstItem
Increase quantity of first item by 10
Call UpdateInventoryFile
Success
Error
Write all inventory data to Inventory.txt
Catch exception and show MessageBox
Call DisplayInventory
Build and display formatted string in label
End

showing the data flow of the application. The process starts when the user clicks the button and ends when the results are displayed.

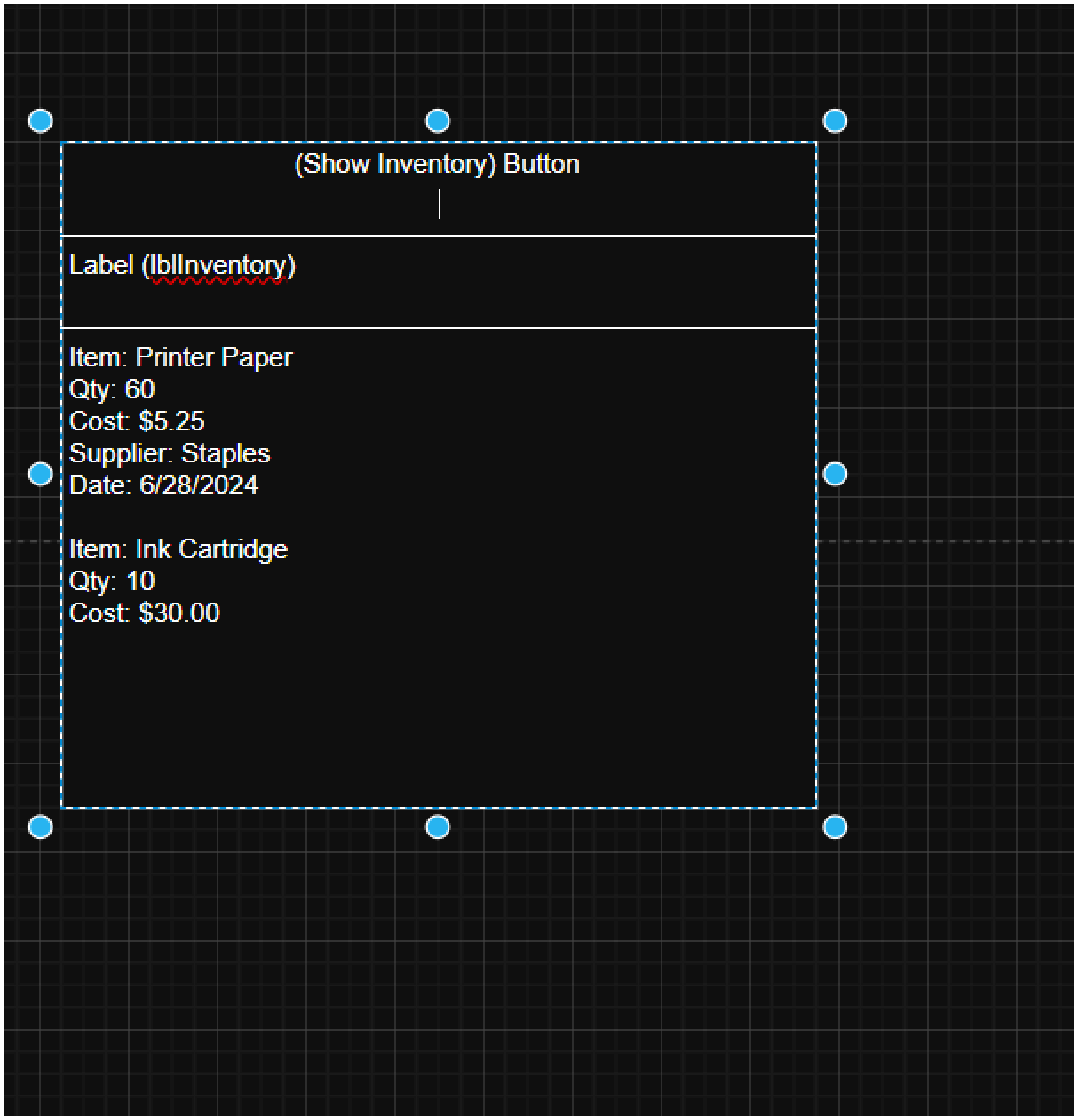
Explanation
The flowchart abstracts internal logic inside method blocks.

Each method becomes a logical "black box," showing what happens, not how it's implemented which reflects object oriented design principles.

This is represented in the flowchart as discrete actions, making it easier to understand and extend.

The flowchart now visually communicates where failures might happen and how they're handled, which improves debugging and robustness.

UPDATED WIREFRAME



```
Item: Printer Paper
Qty: 60
Cost: $5.25
Supplier: Staples
Date: 6/28/2024

Item: Ink Cartridge
Qty: 10
Cost: $30.00
```

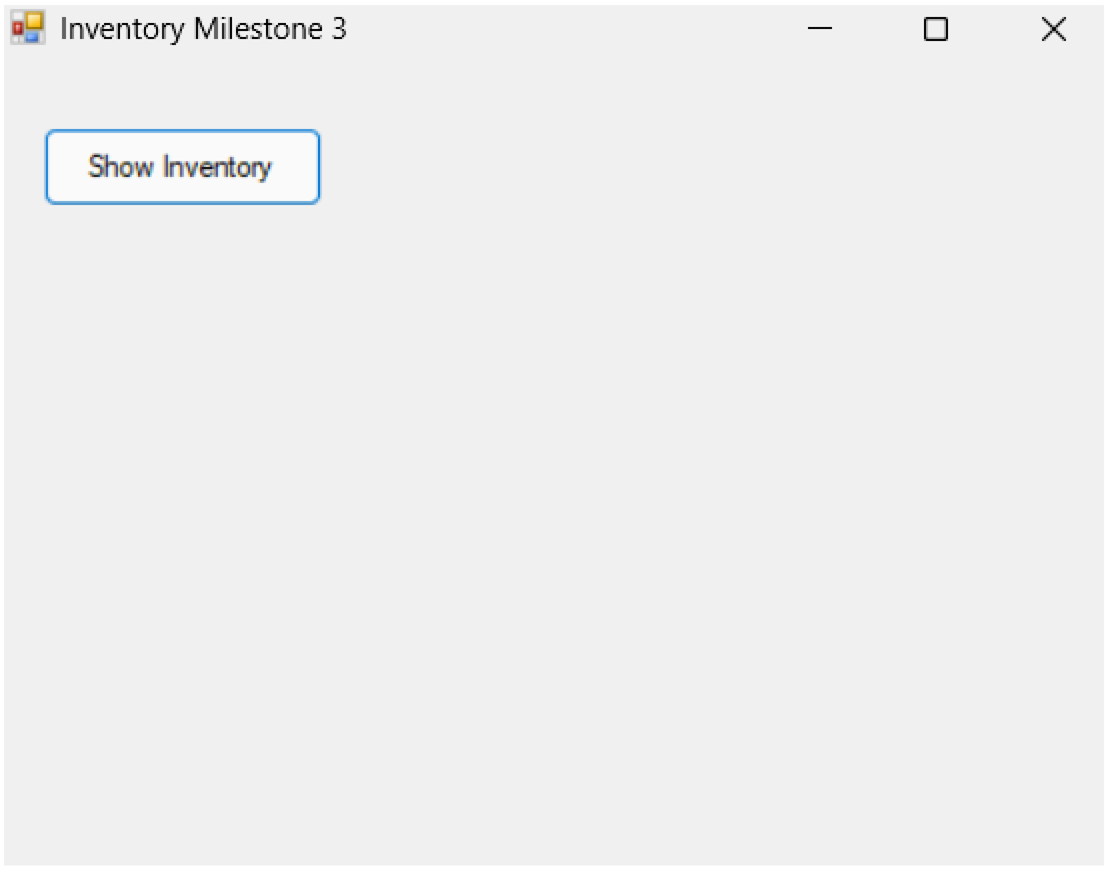
EXPLANATION;

Same layout, but now powered by methods

While the interface itself remains simple, the underlying logic is now structured into reusable methods.

The button click triggers four distinct methods to read, update, and display inventory, making the form easier to maintain and scale.

Screenshot before Form it is populated:

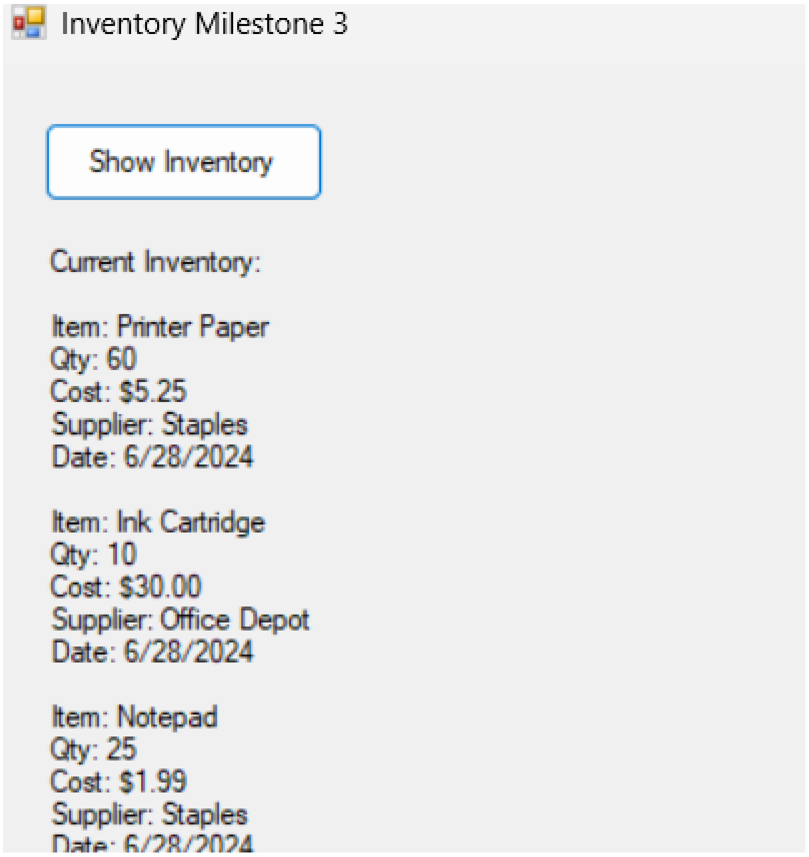


Initial state of the inventory form before button click.

Explanation:
This screenshot shows the form immediately after launching the application. At this point, the inventory label is empty and no data has been loaded from the text file.
The interface is intentionally simple with a single button labeled "Show Inventory."

This design ensures the user must initiate inventory loading manually, helping to isolate UI logic from business logic.

Screenshot after form is populated:

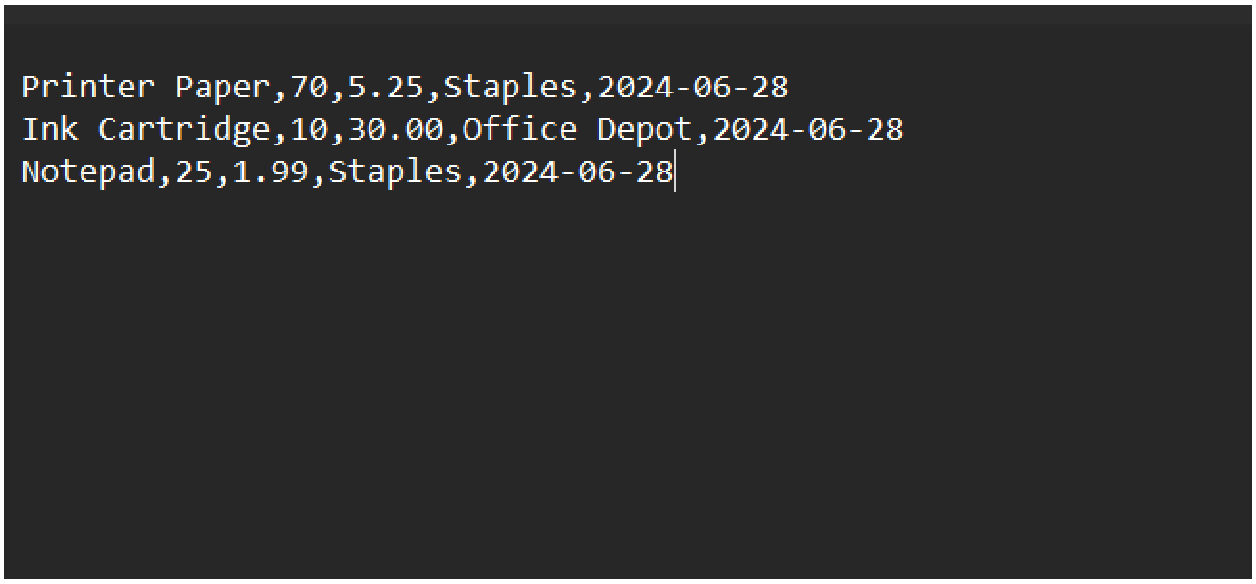


```
Current Inventory:

Item: Printer Paper
Qty: 60
Cost: $5.25
Supplier: Staples
Date: 6/28/2024

Item: Ink Cartridge
Qty: 10
Cost: $30.00
Supplier: Office Depot
Date: 6/28/2024

Item: Notepad
Qty: 25
Cost: $1.99
Supplier: Staples
Date: 6/28/2024
```

Inventory results displayed after business logic execution.

Explanation:
After the user clicks the "Show Inventory" button, the application calls the `ReadInventory()` and `DisplayInventory()` methods.
This screenshot shows how inventory data from `Inventory.txt` is formatted and displayed in the label.
Each item displays all relevant fields, confirming that the file was read and parsed successfully.
This output also confirms that modular method structure is working as intended.

SCREENSHOT OF TEXT FILE AFTER UPDATING QUANTITY



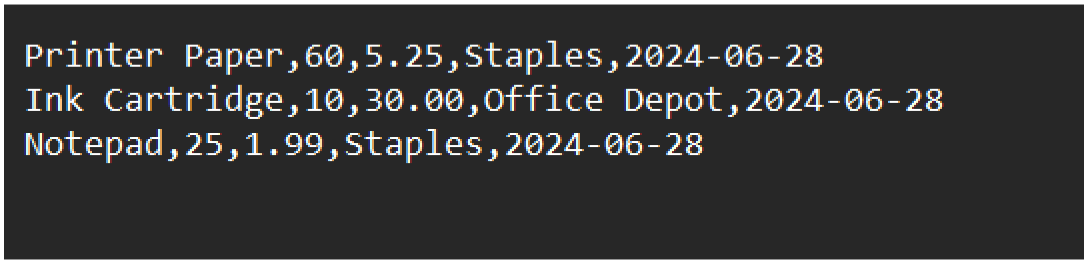
```
Printer Paper,70,5.25,Staples,2024-06-28
Ink Cartridge,10,30.00,Office Depot,2024-06-28
Notepad,25,1.99,Staples,2024-06-28
```

EXPLANATION;

This screenshot shows the updated state of `Inventory.txt` after the program executes `IncrementFirstItem()` and `UpdateInventoryFile()`.
The quantity of the first item has increased by 10, confirming that the program not only reads the file but also writes changes back to it.
This persistent update is a key requirement for Milestone 4 and demonstrates effective file output handling using modular methods.

SCREENSHOT OF ORIGINAL TEXT FILE

S



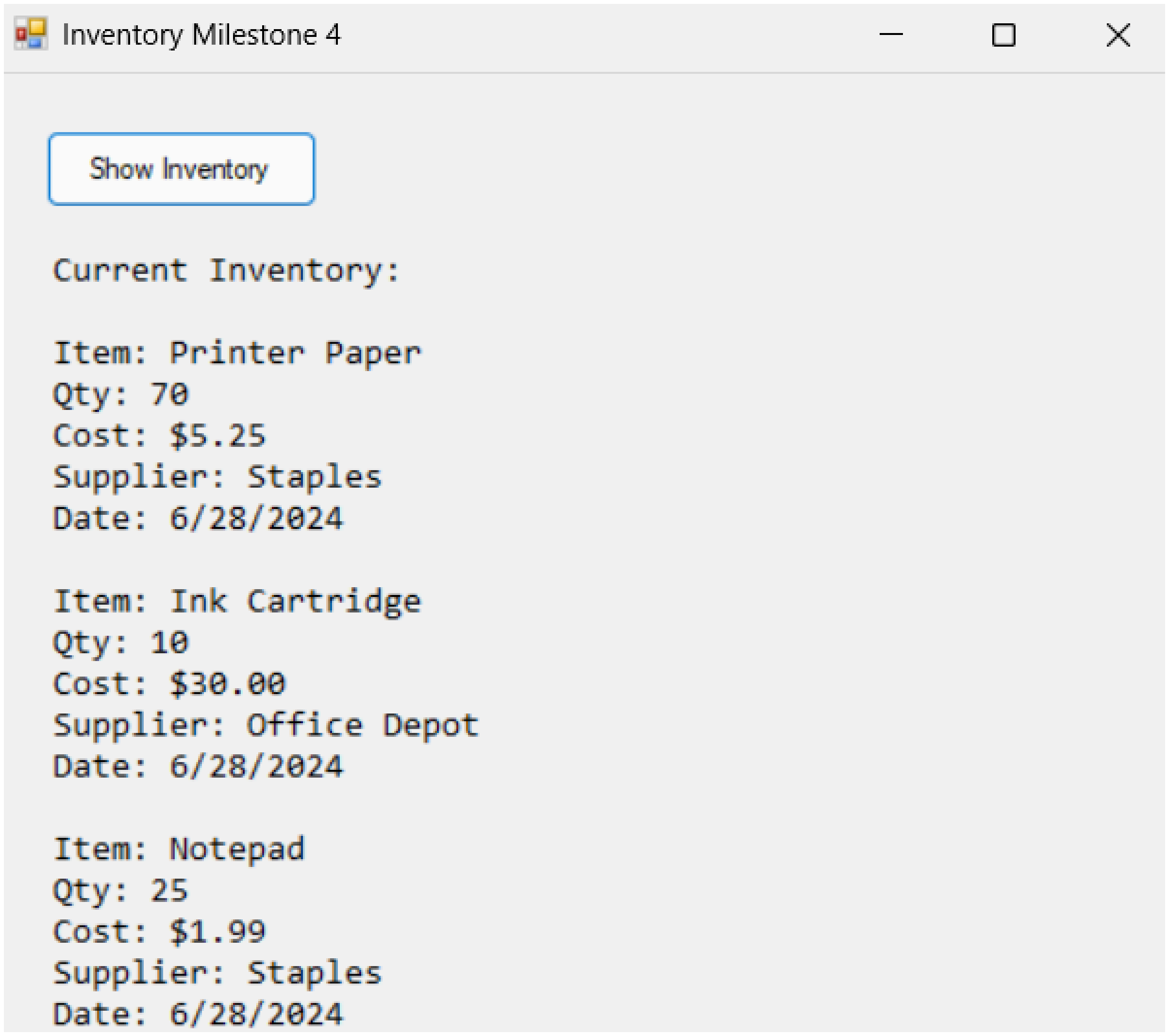
```
Printer Paper,60,5.25,Staples,2024-06-28
Ink Cartridge,10,30.00,Office Depot,2024-06-28
Notepad,25,1.99,Staples,2024-06-28
```

Updated Inventory.txt showing new quantity saved.

EXPLANATION;

This screenshot shows the contents of the inventory file before any changes are made.
It contains a list of items with their description, quantity, cost, supplier, and date. This baseline state is important for verifying that the application's update logic correctly modifies the intended data in this case, the first item's quantity.
The file resides in the `Data` folder and is read dynamically by the application.

SCREENSHOT OF THE FORM AFTER INCREMENTING QUANTITY

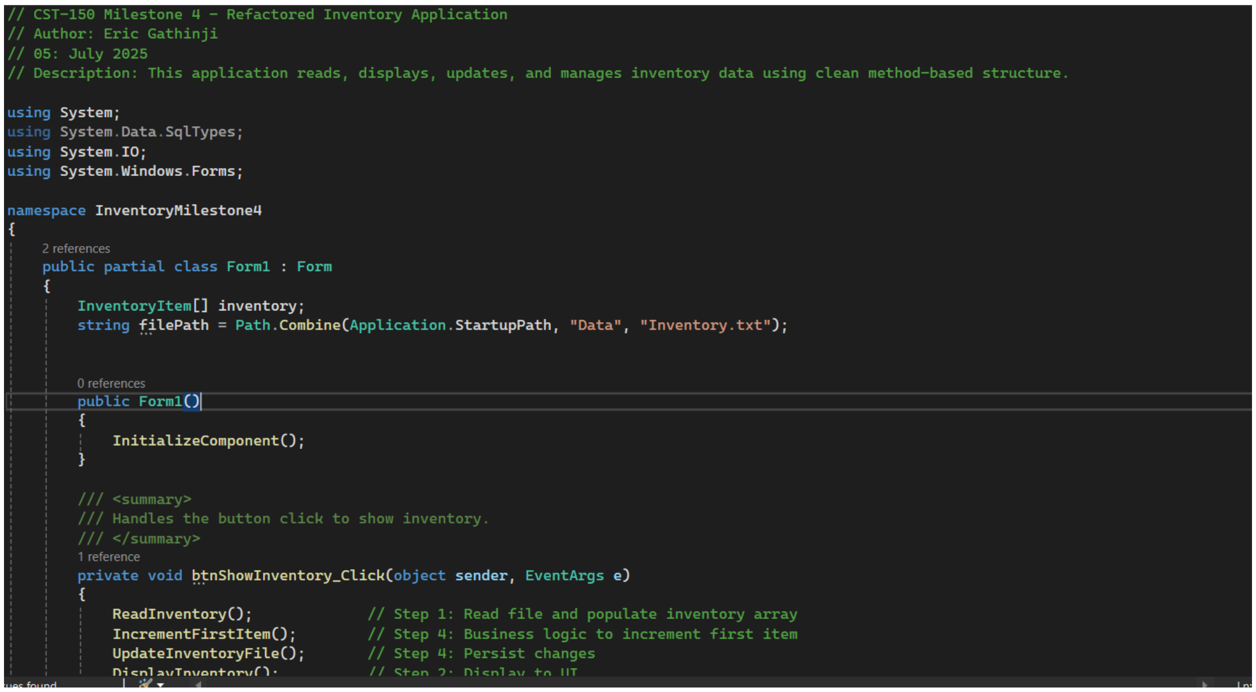


```
Current Inventory:

Item: Printer Paper
Qty: 70
Cost: $5.25
Supplier: Staples
Date: 6/28/2024

Item: Ink Cartridge
Qty: 10
Cost: $30.00
Supplier: Office Depot
Date: 6/28/2024

Item: Notepad
Qty: 25
Cost: $1.99
Supplier: Staples
Date: 6/28/2024
```

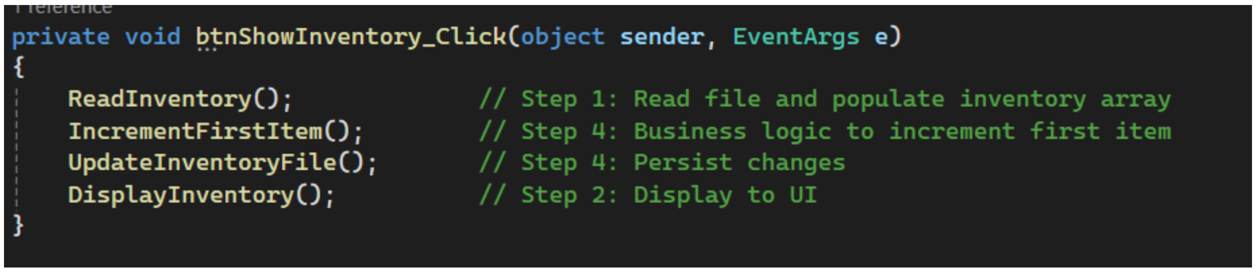
EXPLANATION;

This form displays the result after successfully updating the first inventory item's quantity and then reloading it.
The user clicks the same "Show Inventory" button, but now the label reflects the new value stored in the file.
This visual confirmation of the update completes the feedback loop and validates that all logic is cleanly separated into methods: reading, updating, and displaying inventory.

Screenshot(s) of the code behind

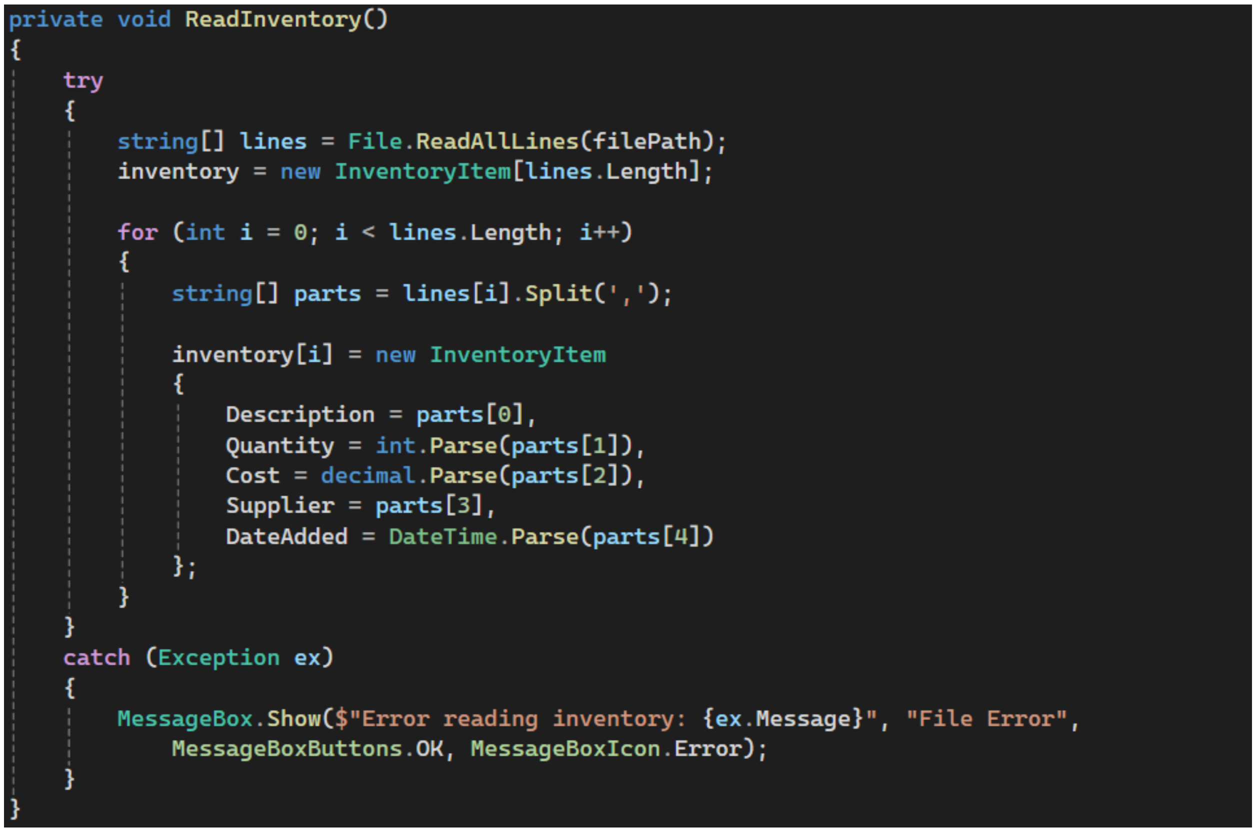


```
// CST-150 Milestone 4 - Refactored Inventory Application
// Author: Eric Gathinji
// 05: July 2025
// Description: This application reads, displays, updates, and manages inventory data using clean method-based structure.

using System;
using System.Data.SqlTypes;
using System.IO;
using System.Windows.Forms;

namespace InventoryMilestone4
{
    2 references
    public partial class Form1 : Form
    {
        InventoryItem[] inventory;
        string filePath = Path.Combine(Application.StartupPath, "Data", "Inventory.txt");


        0 references
        public Form1()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Handles the button click to show inventory.
        /// </summary>
        1 reference
        private void btnShowInventory_Click(object sender, EventArgs e)
        {
            ReadInventory();            // Step 1: Read file and populate inventory array
            IncrementFirstItem();       // Step 4: Business logic to increment first item
            UpdateInventoryFile();      // Step 4: Persist changes
            DisplayInventory();         // Step 2: Display to UI
```



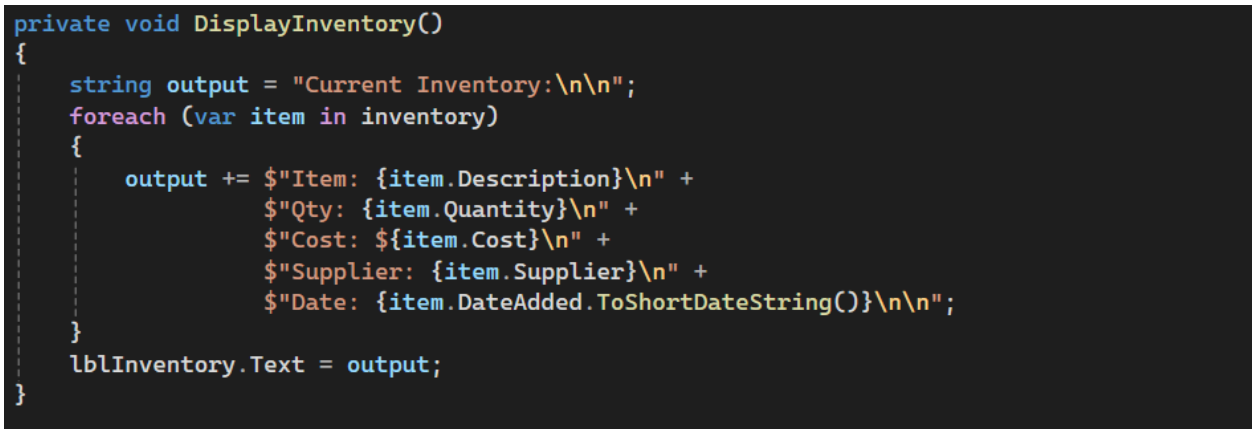
```
private void btnShowInventory_Click(object sender, EventArgs e)
{
    ReadInventory();            // Step 1: Read file and populate inventory array
    IncrementFirstItem();       // Step 4: Business logic to increment first item
    UpdateInventoryFile();      // Step 4: Persist changes
    DisplayInventory();         // Step 2: Display to UI
}
```

Handles the button click to show inventory.



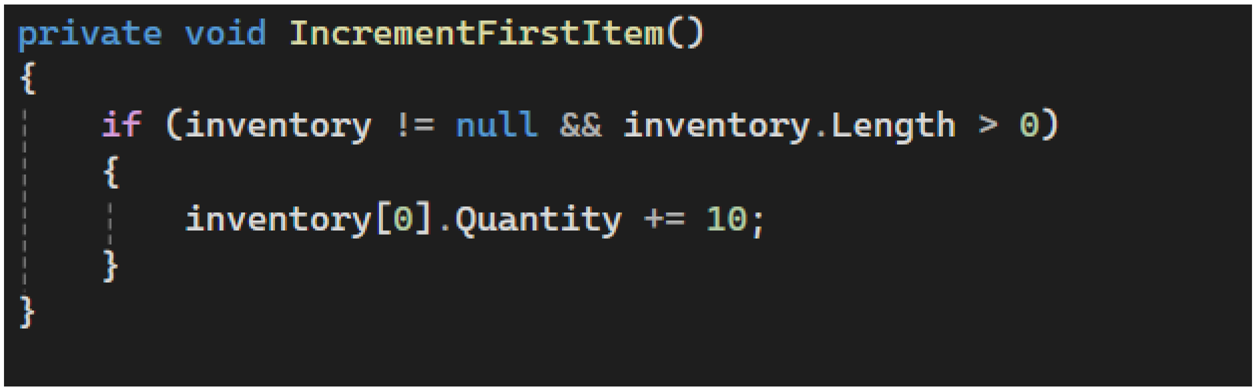
```
private void ReadInventory()
{
    try
    {
        string[] lines = File.ReadAllLines(filePath);
        inventory = new InventoryItem[lines.Length];

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

            inventory[i] = new InventoryItem
            {
                Description = parts[0],
                Quantity = int.Parse(parts[1]),
                Cost = decimal.Parse(parts[2]),
                Supplier = parts[3],
                DateAdded = DateTime.Parse(parts[4])
            };
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error reading inventory: {ex.Message}", "File Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Reads the inventory from the text file and populates the inventory array.



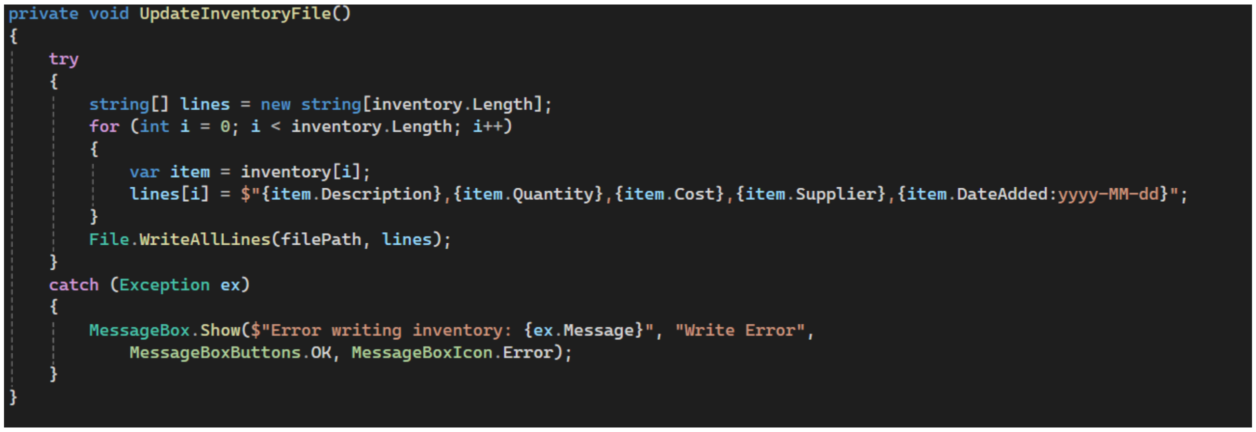
```
private void DisplayInventory()
{
    string output = "Current Inventory:\n\n";
    foreach (var item in inventory)
    {
        output += $"Item: {item.Description}\n" +
                  $"Qty: {item.Quantity}\n" +
                  $"Cost: ${item.Cost}\n" +
                  $"Supplier: {item.Supplier}\n" +
                  $"Date: {item.DateAdded.ToShortDateString()}\n\n";
    }
    lblInventory.Text = output;
}
```

Displays the inventory on the label control.



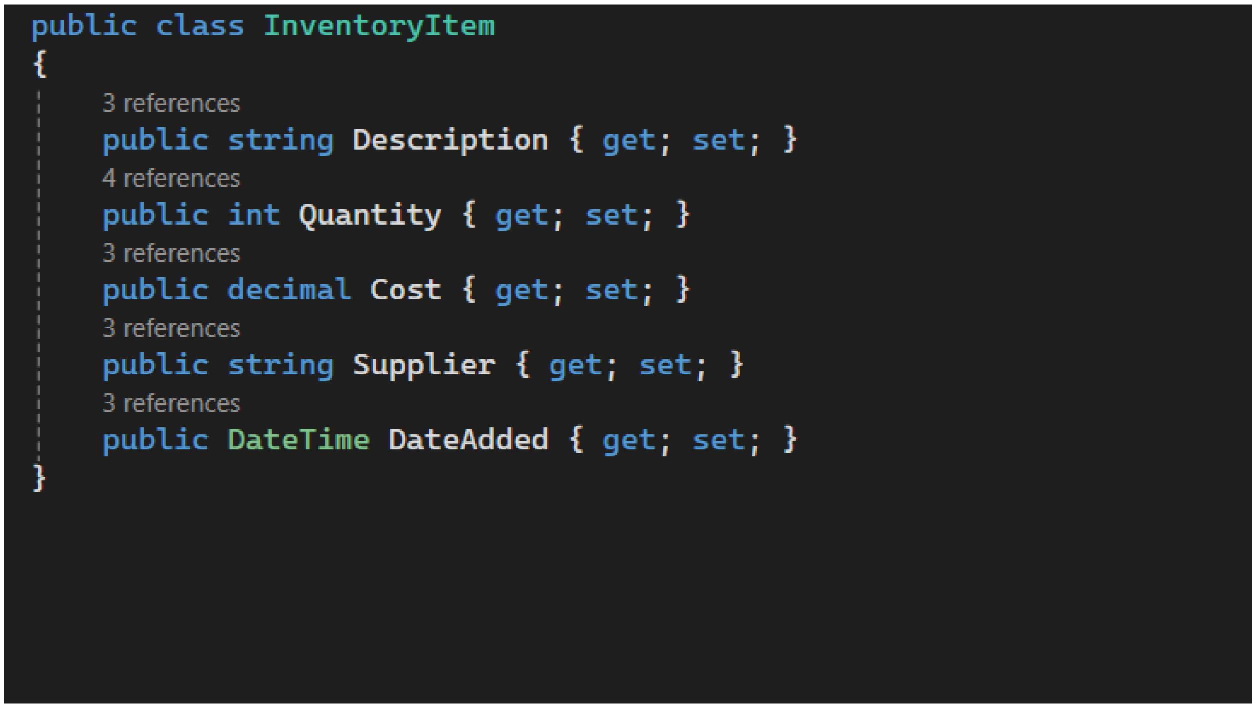
```csharp
private void IncrementFirstItem()
{
    if (inventory != null && inventory.Length > 0)
    {
        inventory[0].Quantity += 10;
    }
}
```

Increments the quantity of the first inventory item.



```csharp
private void UpdateInventoryFile()
{
    try
    {
        string[] lines = new string[inventory.Length];
        for (int i = 0; i < inventory.Length; i++)
        {
            var item = inventory[i];
            lines[i] = $"{item.Description},{item.Quantity},{item.Cost},{item.Supplier},{item.DateAdded:yyyy-MM-dd}";
        }
        File.WriteAllLines(filePath, lines);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error writing inventory: {ex.Message}", "Write Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Writes the updated inventory back to the text file.



```csharp
public class InventoryItem
{
    3 references
    public string Description { get; set; }
    4 references
    public int Quantity { get; set; }
    3 references
    public decimal Cost { get; set; }
    3 references
    public string Supplier { get; set; }
    3 references
    public DateTime DateAdded { get; set; }
}
```

Inventory item model for Milestone 4.