Owen Lindsey

Professor Smithers, Mark

08/20/2023

CST-150

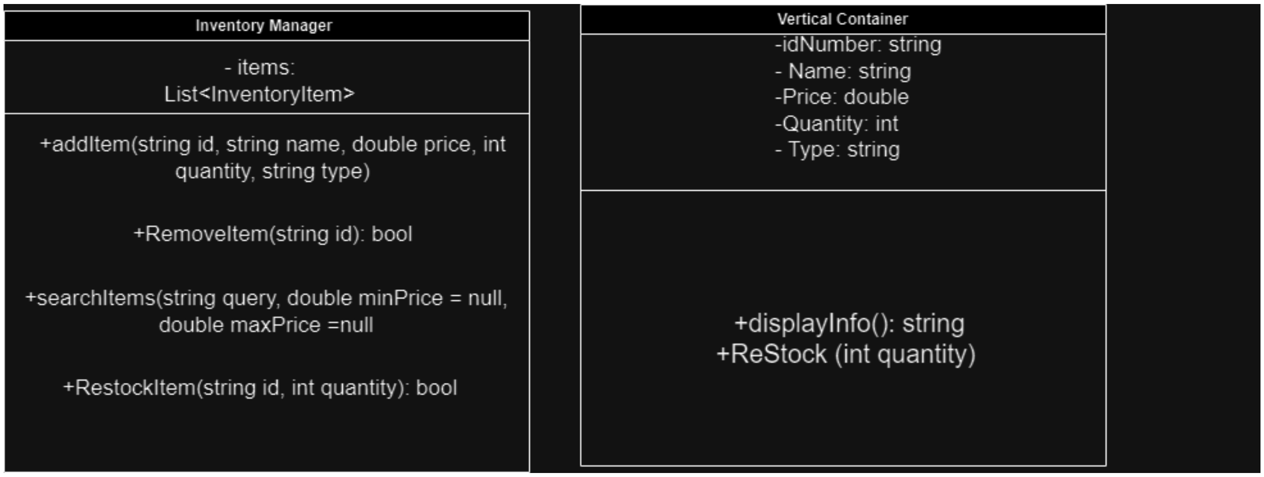
Grand Canyon University

Milestone

Video: https://youtu.be/KErL1S4Bmfc

GitHub: https://github.com/Omni-v/Cst-150-Workspace/tree/master/milestone

UML


Inventory Manager
- items:
List<InventoryItem>
+addItem(string id, string name, double price, int quantity, string type)
+RemoveItem(string id): bool
+searchItems(string query, double minPrice = null, double maxPrice =null
+RestockItem(string id, int quantity): bool
Vertical Container
-idNumber: string
- Name: string
-Price: double
-Quantity: int
- Type: string
+displayInfo(): string
+ReStock (int quantity)

Written discussion

What I Learned:

1. Importance of Iterative Development: Going step by step through each functionality, from system setup to debugging, taught me the power of iterative development. This approach allowed me to adapt to challenges, refine the application continuously, and keep it on the right path.

2. Collaborative Problem Solving: Working collaboratively showed me the strength of different perspectives when tackling complex issues. Having an outside viewpoint allowed me to understand various approaches to solving problems.

Challenges I Faced:

1. Errors and Debugging: Debugging the application was a significant challenge. It tested my problem-solving skills, especially when errors weren't immediately apparent. Handling errors in real-time was a valuable learning experience.

2. Naming Confusions: The overlapping naming conventions I encountered highlighted the importance of clear and distinct naming in coding. This experience emphasized how proper naming can prevent confusion and improve code clarity.

3. Unit Testing: Setting up a testing environment using XUnit brought its own set of challenges. Ensuring that the tests aligned with the actual code and debugging any inconsistencies was a rewarding learning curve.

Future Improvements:

1. Code Clarity: Given more time, I would revisit the code to enhance clarity and maintainability. Clear, organized code is essential for making future updates and improvements efficiently.

2. Database Integration: Transitioning from a list-based system to a database-backed structure is a natural evolution. This enhancement would provide scalability and robustness, crucial for real-world applications.

3. User Experience: If this were a GUI application, I'd concentrate on improving the user interface to enhance intuitiveness and introduce more features.

4. Expand Testing: Expanding the testing suite to address more edge cases and possibly delving into integration testing would provide thorough coverage and ensure a more reliable application.

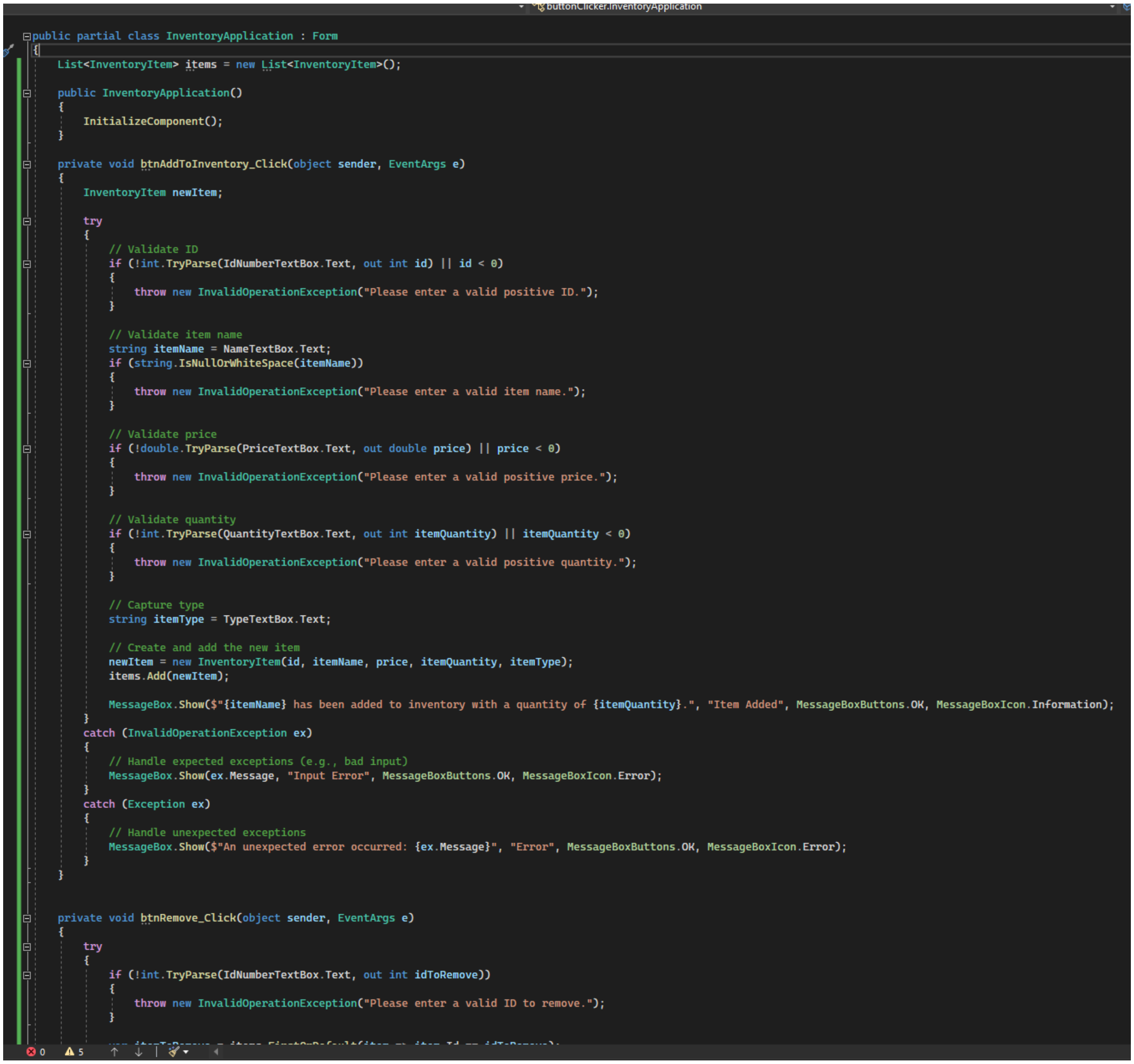
Application to Future Projects:

1. Code Modularity: The project reinforced the significance of writing adaptable and modular code. This principle isn't limited to inventory systems but applies universally, whether I'm developing games, mobile apps, or web services.

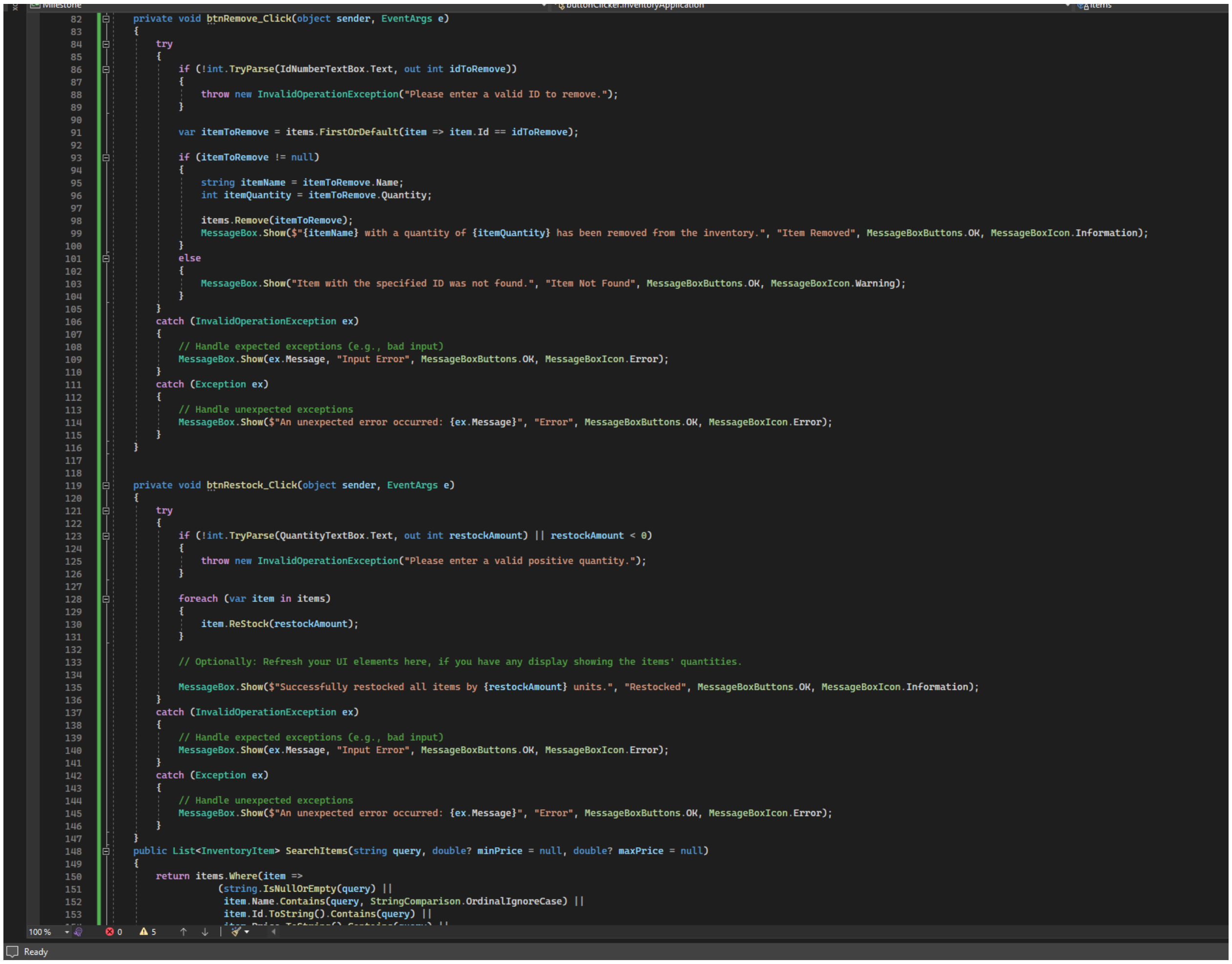
2. Rigorous Error Handling: Robust error handling is crucial across all projects. Ensuring users receive helpful feedback instead of facing unhandled errors is a priority in any development endeavor.

3. Emphasis on Testing: This project underscored the importance of testing. Ensuring that every aspect functions as intended is vital, and I'll carry this lesson into all my future development ventures.

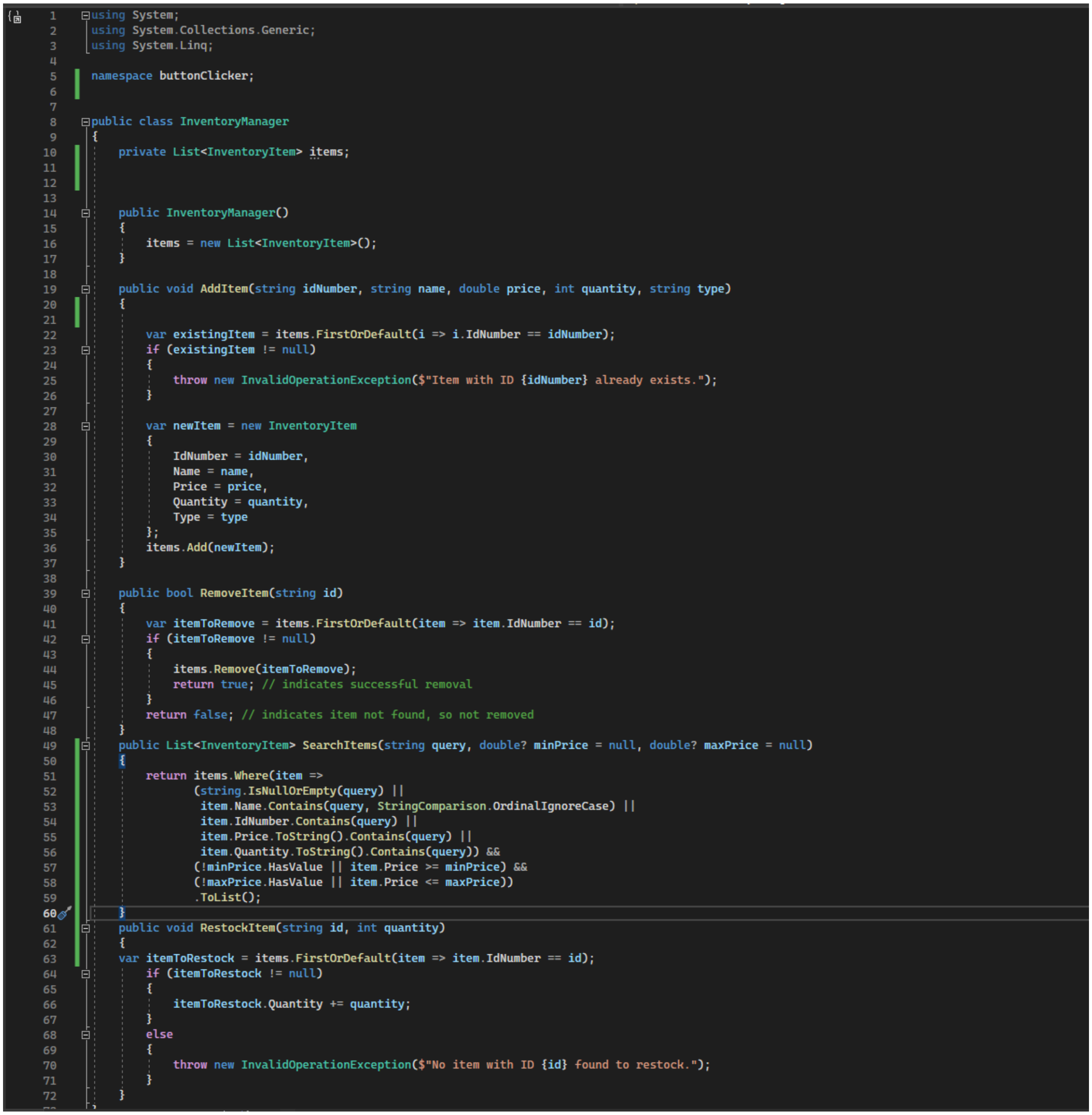
Main Application:



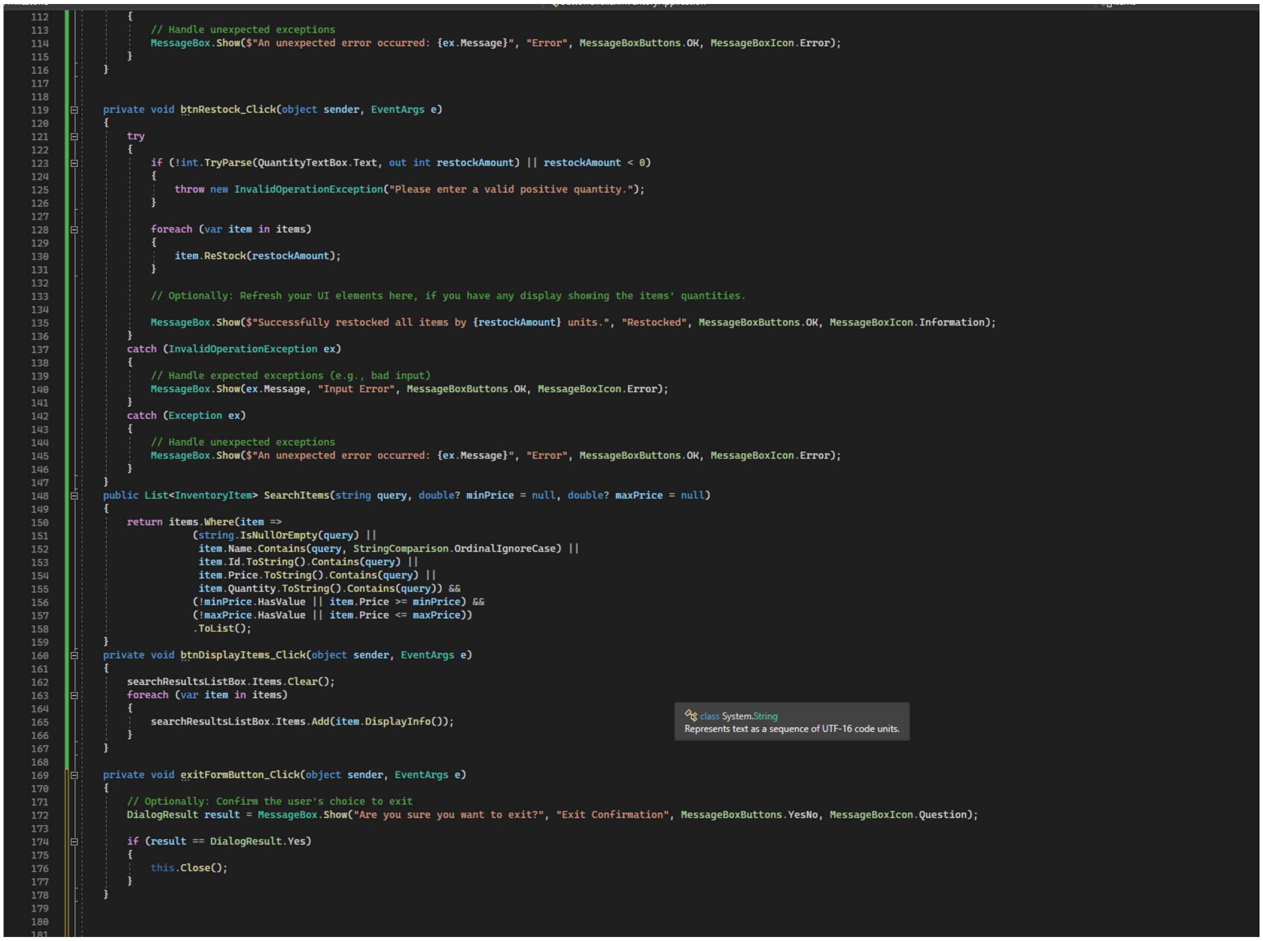
```
public partial class InventoryApplication : Form
{
    List<InventoryItem> items = new List<InventoryItem>();

    public InventoryApplication()
    {
        InitializeComponent();
    }

    private void btnAddToInventory_Click(object sender, EventArgs e)
    {
        InventoryItem newItem;

        try
        {
            // Validate ID
            if (!int.TryParse(IdNumberTextBox.Text, out int id) || id < 0)
            {
                throw new InvalidOperationException("Please enter a valid positive ID.");
            }

            // Validate item name
            string itemName = NameTextBox.Text;
            if (string.IsNullOrWhiteSpace(itemName))
            {
                throw new InvalidOperationException("Please enter a valid item name.");
            }

            // Validate price
            if (!double.TryParse(PriceTextBox.Text, out double price) || price < 0)
            {
                throw new InvalidOperationException("Please enter a valid positive price.");
            }

            // Validate quantity
            if (!int.TryParse(QuantityTextBox.Text, out int itemQuantity) || itemQuantity < 0)
            {
                throw new InvalidOperationException("Please enter a valid positive quantity.");
            }

            // Capture type
            string itemType = TypeTextBox.Text;

            // Create and add the new item
            newItem = new InventoryItem(id, itemName, price, itemQuantity, itemType);
            items.Add(newItem);

            MessageBox.Show($"{itemName} has been added to inventory with a quantity of {itemQuantity}.", "Item Added", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (InvalidOperationException ex)
        {
            // Handle expected exceptions (e.g., bad input)
            MessageBox.Show(ex.Message, "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            // Handle unexpected exceptions
            MessageBox.Show($"An unexpected error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }


    private void btnRemove_Click(object sender, EventArgs e)
    {
        try
        {
            if (!int.TryParse(IdNumberTextBox.Text, out int idToRemove))
            {
                throw new InvalidOperationException("Please enter a valid ID to remove.");
            }

            var itemToRemove = items.FirstOrDefault(item => item.Id == idToRemove);
```



```
        private void btnRemove_Click(object sender, EventArgs e)
        {
            try
            {
                if (!int.TryParse(IdNumberTextBox.Text, out int idToRemove))
                {
                    throw new InvalidOperationException("Please enter a valid ID to remove.");
                }

                var itemToRemove = items.FirstOrDefault(item => item.Id == idToRemove);

                if (itemToRemove != null)
                {
                    string itemName = itemToRemove.Name;
                    int itemQuantity = itemToRemove.Quantity;

                    items.Remove(itemToRemove);
                    MessageBox.Show($"{itemName} with a quantity of {itemQuantity} has been removed from the inventory.", "Item Removed", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else
                {
                    MessageBox.Show("Item with the specified ID was not found.", "Item Not Found", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }
            }
            catch (InvalidOperationException ex)
            {
                // Handle expected exceptions (e.g., bad input)
                MessageBox.Show(ex.Message, "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle unexpected exceptions
                MessageBox.Show($"An unexpected error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }


        private void btnRestock_Click(object sender, EventArgs e)
        {
            try
            {
                if (!int.TryParse(QuantityTextBox.Text, out int restockAmount) || restockAmount < 0)
                {
                    throw new InvalidOperationException("Please enter a valid positive quantity.");
                }

                foreach (var item in items)
                {
                    item.ReStock(restockAmount);
                }

                // Optionally: Refresh your UI elements here, if you have any display showing the items' quantities.

                MessageBox.Show($"Successfully restocked all items by {restockAmount} units.", "Restocked", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (InvalidOperationException ex)
            {
                // Handle expected exceptions (e.g., bad input)
                MessageBox.Show(ex.Message, "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                // Handle unexpected exceptions
                MessageBox.Show($"An unexpected error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        public List<InventoryItem> SearchItems(string query, double? minPrice = null, double? maxPrice = null)
        {
            return items.Where(item =>
                       (string.IsNullOrEmpty(query) ||
                        item.Name.Contains(query, StringComparison.OrdinalIgnoreCase) ||
                        item.Id.ToString().Contains(query) ||
                        item.Price.ToString().Contains(query) ||
```

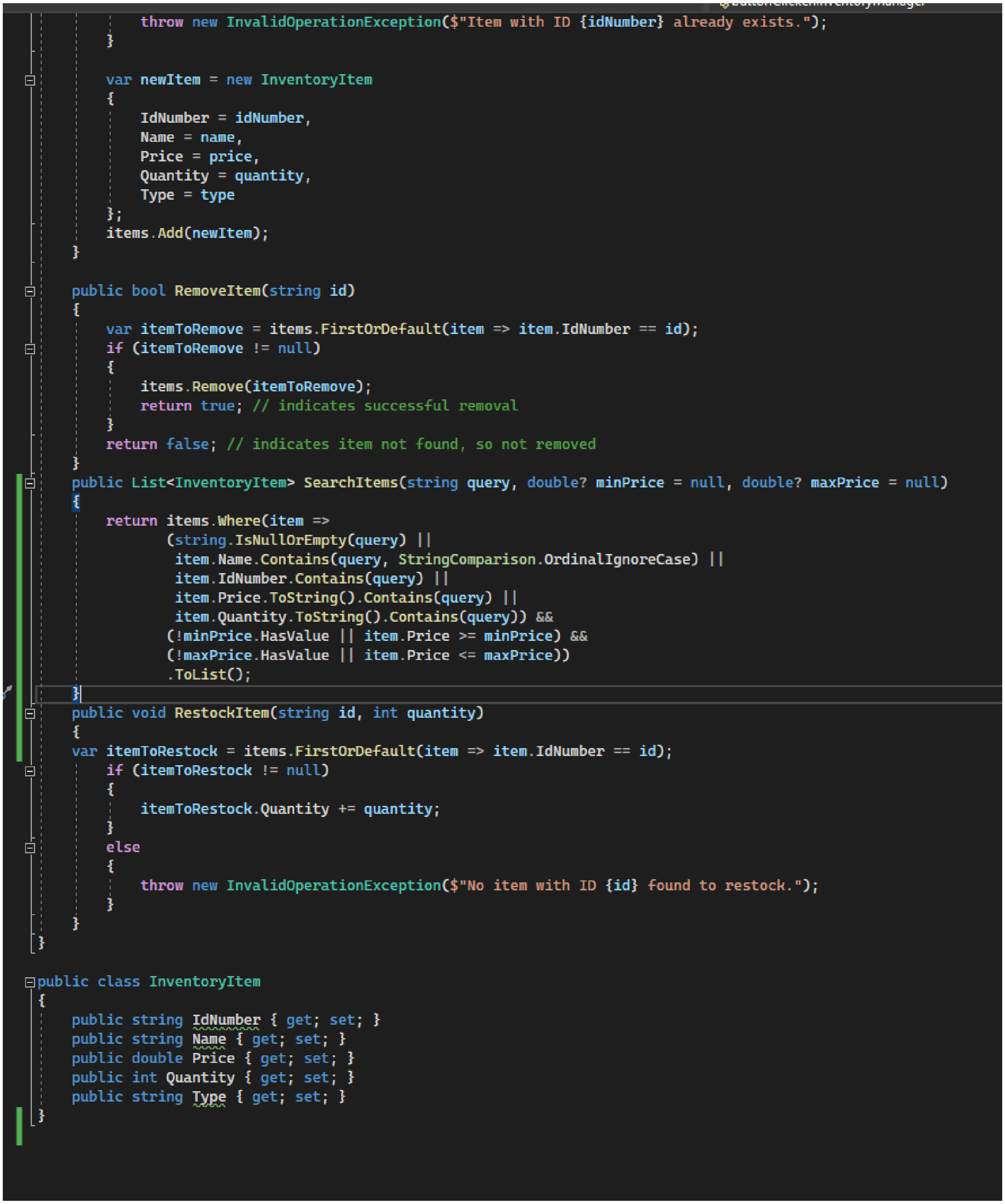


```csharp
using System;
using System.Collections.Generic;
using System.Linq;

namespace buttonClicker;


public class InventoryManager
{
    private List<InventoryItem> items;



    public InventoryManager()
    {
        items = new List<InventoryItem>();
    }

    public void AddItem(string idNumber, string name, double price, int quantity, string type)
    {

        var existingItem = items.FirstOrDefault(i => i.IdNumber == idNumber);
        if (existingItem != null)
        {
            throw new InvalidOperationException($"Item with ID {idNumber} already exists.");
        }

        var newItem = new InventoryItem
        {
            IdNumber = idNumber,
            Name = name,
            Price = price,
            Quantity = quantity,
            Type = type
        };
        items.Add(newItem);
    }

    public bool RemoveItem(string id)
    {
        var itemToRemove = items.FirstOrDefault(item => item.IdNumber == id);
        if (itemToRemove != null)
        {
            items.Remove(itemToRemove);
            return true; // indicates successful removal
        }
        return false; // indicates item not found, so not removed
    }
    public List<InventoryItem> SearchItems(string query, double? minPrice = null, double? maxPrice = null)
    {
        return items.Where(item =>
               (string.IsNullOrEmpty(query) ||
                item.Name.Contains(query, StringComparison.OrdinalIgnoreCase) ||
                item.IdNumber.Contains(query) ||
                item.Price.ToString().Contains(query) ||
                item.Quantity.ToString().Contains(query)) &&
               (!minPrice.HasValue || item.Price >= minPrice) &&
               (!maxPrice.HasValue || item.Price <= maxPrice))
               .ToList();
    }
    public void RestockItem(string id, int quantity)
    {
    var itemToRestock = items.FirstOrDefault(item => item.IdNumber == id);
        if (itemToRestock != null)
        {
            itemToRestock.Quantity += quantity;
        }
        else
        {
            throw new InvalidOperationException($"No item with ID {id} found to restock.");
        }
    }
}
```



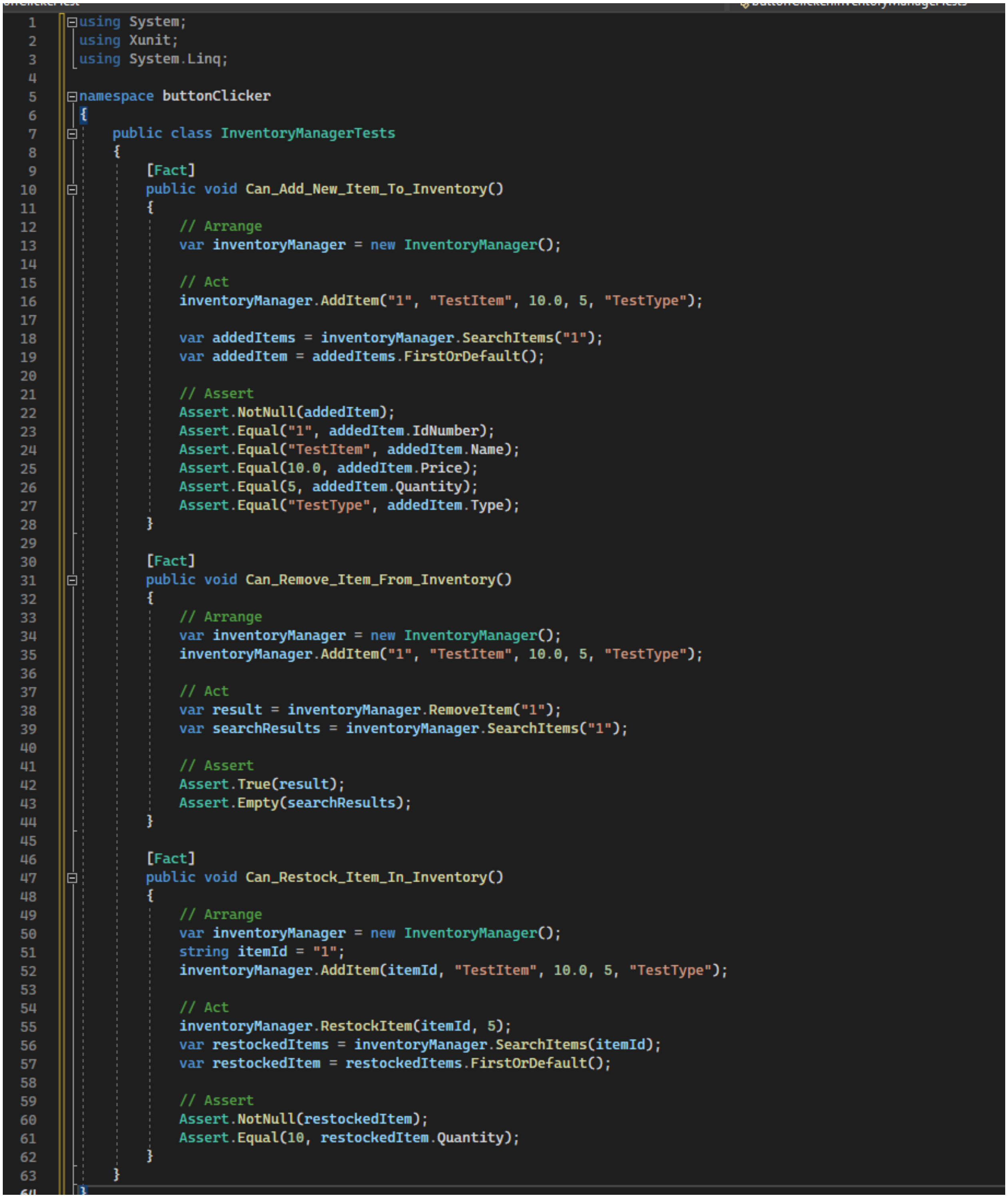
```
    {
        // Handle unexpected exceptions
        MessageBox.Show($"An unexpected error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}


private void btnRestock_Click(object sender, EventArgs e)
{
    try
    {
        if (!int.TryParse(QuantityTextBox.Text, out int restockAmount) || restockAmount < 0)
        {
            throw new InvalidOperationException("Please enter a valid positive quantity.");
        }

        foreach (var item in items)
        {
            item.ReStock(restockAmount);
        }

        // Optionally: Refresh your UI elements here, if you have any display showing the items' quantities.

        MessageBox.Show($"Successfully restocked all items by {restockAmount} units.", "Restocked", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (InvalidOperationException ex)
    {
        // Handle expected exceptions (e.g., bad input)
        MessageBox.Show(ex.Message, "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        // Handle unexpected exceptions
        MessageBox.Show($"An unexpected error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
public List<InventoryItem> SearchItems(string query, double? minPrice = null, double? maxPrice = null)
{
    return items.Where(item =>
                (string.IsNullOrEmpty(query) ||
                 item.Name.Contains(query, StringComparison.OrdinalIgnoreCase) ||
                 item.Id.ToString().Contains(query) ||
                 item.Price.ToString().Contains(query) ||
                 item.Quantity.ToString().Contains(query)) &&
                (!minPrice.HasValue || item.Price >= minPrice) &&
                (!maxPrice.HasValue || item.Price <= maxPrice))
                .ToList();
}
private void btnDisplayItems_Click(object sender, EventArgs e)
{
    searchResultsListBox.Items.Clear();
    foreach (var item in items)
    {
        searchResultsListBox.Items.Add(item.DisplayInfo());
    }
}

private void exitFormButton_Click(object sender, EventArgs e)
{
    // Optionally: Confirm the user's choice to exit
    DialogResult result = MessageBox.Show("Are you sure you want to exit?", "Exit Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (result == DialogResult.Yes)
    {
        this.Close();
    }
}
```

class System.String
Represents text as a sequence of UTF-16 code units.



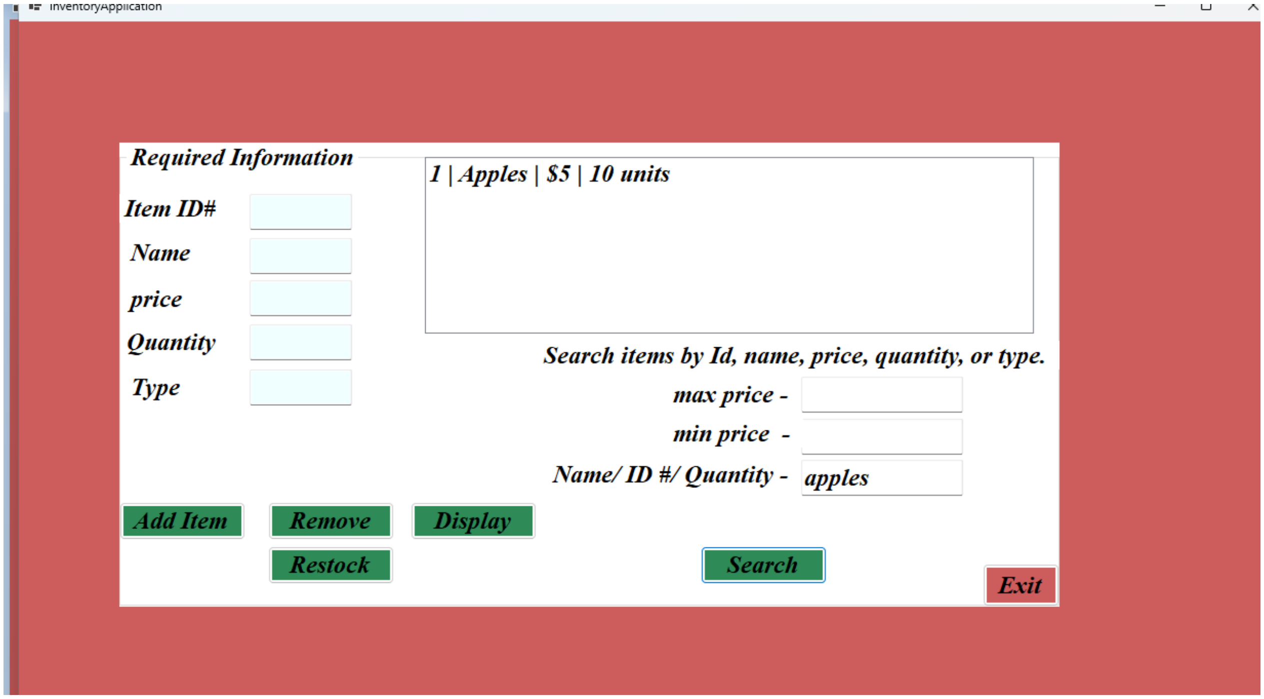
```
            throw new InvalidOperationException($"Item with ID {idNumber} already exists.");
        }

        var newItem = new InventoryItem
        {
            IdNumber = idNumber,
            Name = name,
            Price = price,
            Quantity = quantity,
            Type = type
        };
        items.Add(newItem);
    }

    public bool RemoveItem(string id)
    {
        var itemToRemove = items.FirstOrDefault(item => item.IdNumber == id);
        if (itemToRemove != null)
        {
            items.Remove(itemToRemove);
            return true; // indicates successful removal
        }
        return false; // indicates item not found, so not removed
    }
    public List<InventoryItem> SearchItems(string query, double? minPrice = null, double? maxPrice = null)
    {
        return items.Where(item =>
               (string.IsNullOrEmpty(query) ||
                item.Name.Contains(query, StringComparison.OrdinalIgnoreCase) ||
                item.IdNumber.Contains(query) ||
                item.Price.ToString().Contains(query) ||
                item.Quantity.ToString().Contains(query)) &&
               (!minPrice.HasValue || item.Price >= minPrice) &&
               (!maxPrice.HasValue || item.Price <= maxPrice))
               .ToList();
    }
    public void RestockItem(string id, int quantity)
    {
    var itemToRestock = items.FirstOrDefault(item => item.IdNumber == id);
        if (itemToRestock != null)
        {
            itemToRestock.Quantity += quantity;
        }
        else
        {
            throw new InvalidOperationException($"No item with ID {id} found to restock.");
        }
    }
}

public class InventoryItem
{
    public string IdNumber { get; set; }
    public string Name { get; set; }
    public double Price { get; set; }
    public int Quantity { get; set; }
    public string Type { get; set; }
}
```

Xunit test code



```csharp
using System;
using Xunit;
using System.Linq;

namespace buttonClicker
{
    public class InventoryManagerTests
    {
        [Fact]
        public void Can_Add_New_Item_To_Inventory()
        {
            // Arrange
            var inventoryManager = new InventoryManager();

            // Act
            inventoryManager.AddItem("1", "TestItem", 10.0, 5, "TestType");

            var addedItems = inventoryManager.SearchItems("1");
            var addedItem = addedItems.FirstOrDefault();

            // Assert
            Assert.NotNull(addedItem);
            Assert.Equal("1", addedItem.IdNumber);
            Assert.Equal("TestItem", addedItem.Name);
            Assert.Equal(10.0, addedItem.Price);
            Assert.Equal(5, addedItem.Quantity);
            Assert.Equal("TestType", addedItem.Type);
        }

        [Fact]
        public void Can_Remove_Item_From_Inventory()
        {
            // Arrange
            var inventoryManager = new InventoryManager();
            inventoryManager.AddItem("1", "TestItem", 10.0, 5, "TestType");

            // Act
            var result = inventoryManager.RemoveItem("1");
            var searchResults = inventoryManager.SearchItems("1");

            // Assert
            Assert.True(result);
            Assert.Empty(searchResults);
        }

        [Fact]
        public void Can_Restock_Item_In_Inventory()
        {
            // Arrange
            var inventoryManager = new InventoryManager();
            string itemId = "1";
            inventoryManager.AddItem(itemId, "TestItem", 10.0, 5, "TestType");

            // Act
            inventoryManager.RestockItem(itemId, 5);
            var restockedItems = inventoryManager.SearchItems(itemId);
            var restockedItem = restockedItems.FirstOrDefault();

            // Assert
            Assert.NotNull(restockedItem);
            Assert.Equal(10, restockedItem.Quantity);
        }
    }
}
```

Application running:



inventoryApplication

Required Information

Item ID#

Name

price

Quantity

Type

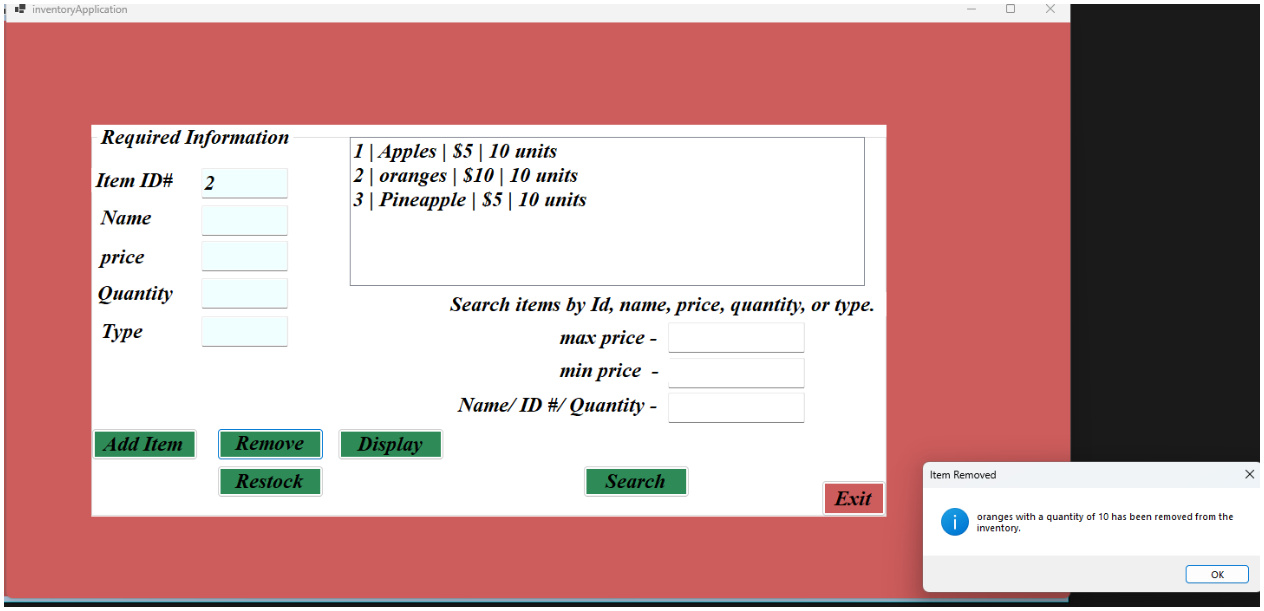
1 | Apples | $5 | 10 units

Search items by Id, name, price, quantity, or type.

max price -

min price -

Name/ ID #/ Quantity - apples



inventoryApplication

Required Information

Item ID# 2

Name

price

Quantity

Type

1 | Apples | $5 | 10 units
2 | oranges | $10 | 10 units
3 | Pineapple | $5 | 10 units

Search items by Id, name, price, quantity, or type.

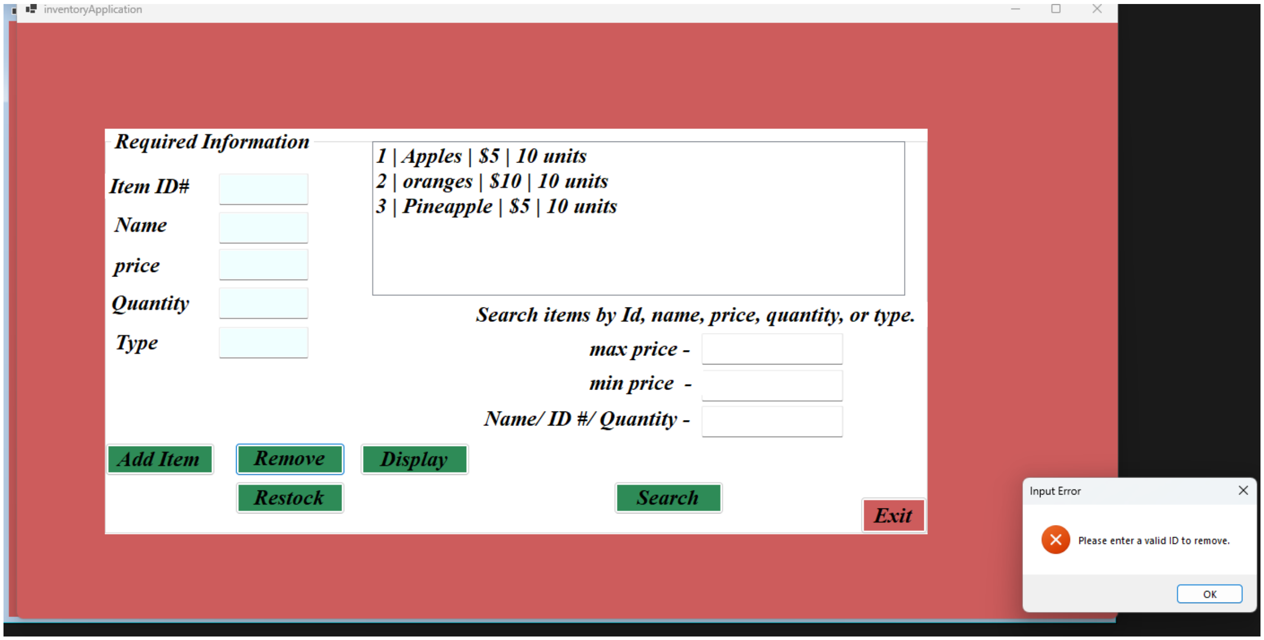
max price -

min price -

Name/ ID #/ Quantity -

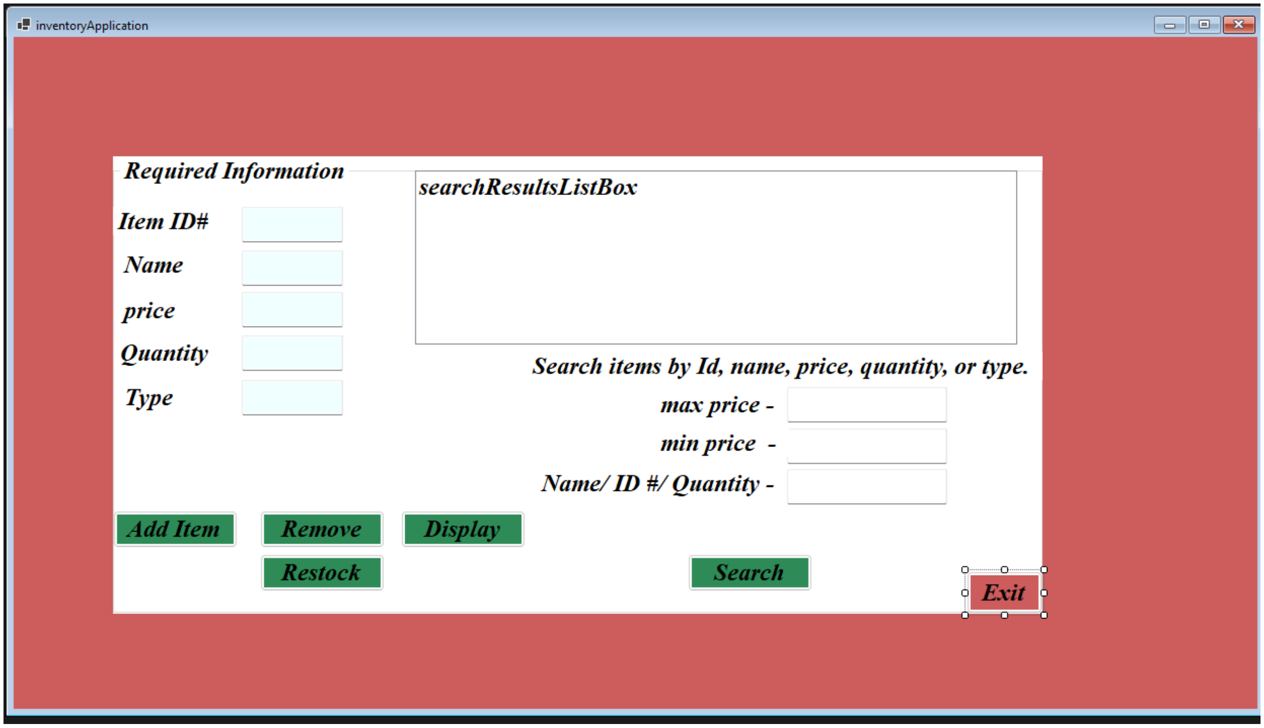
Item Removed

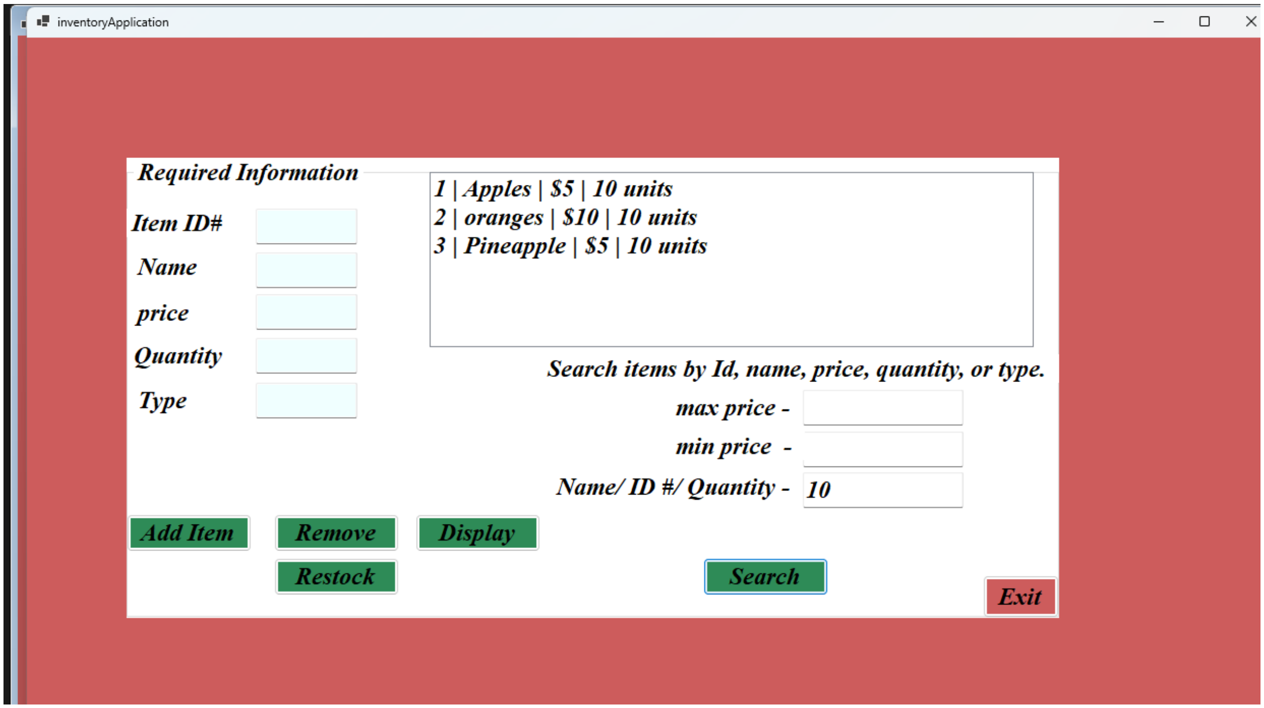
oranges with a quantity of 10 has been removed from the inventory.



1 | Apples | $5 | 10 units
2 | oranges | $10 | 10 units
3 | Pineapple | $5 | 10 units

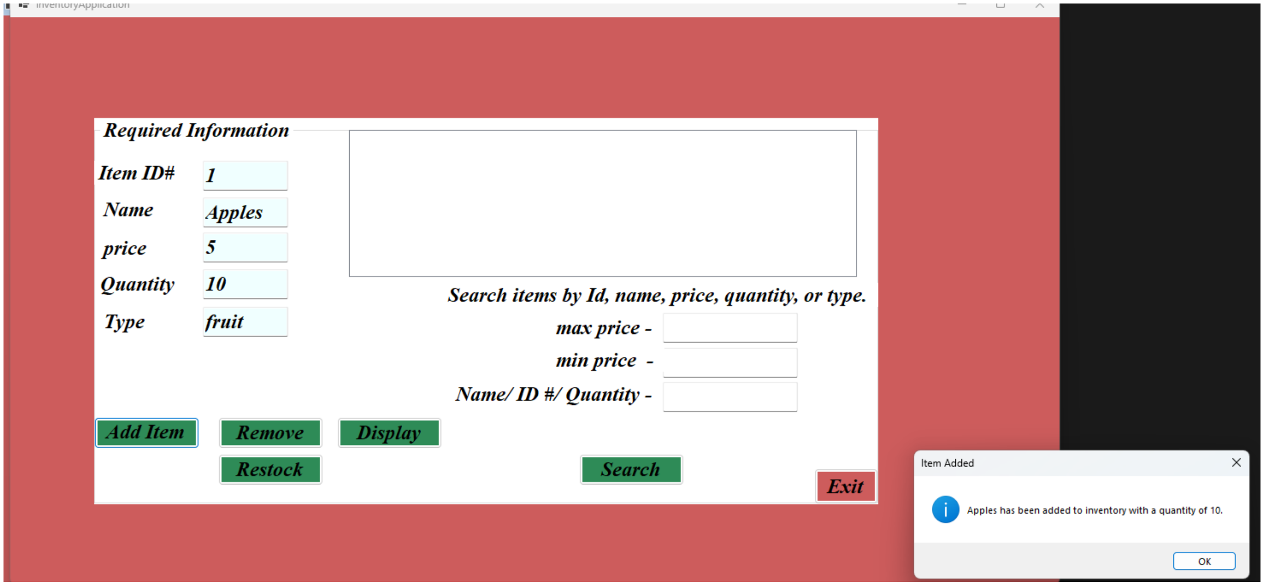
Please enter a valid ID to remove.





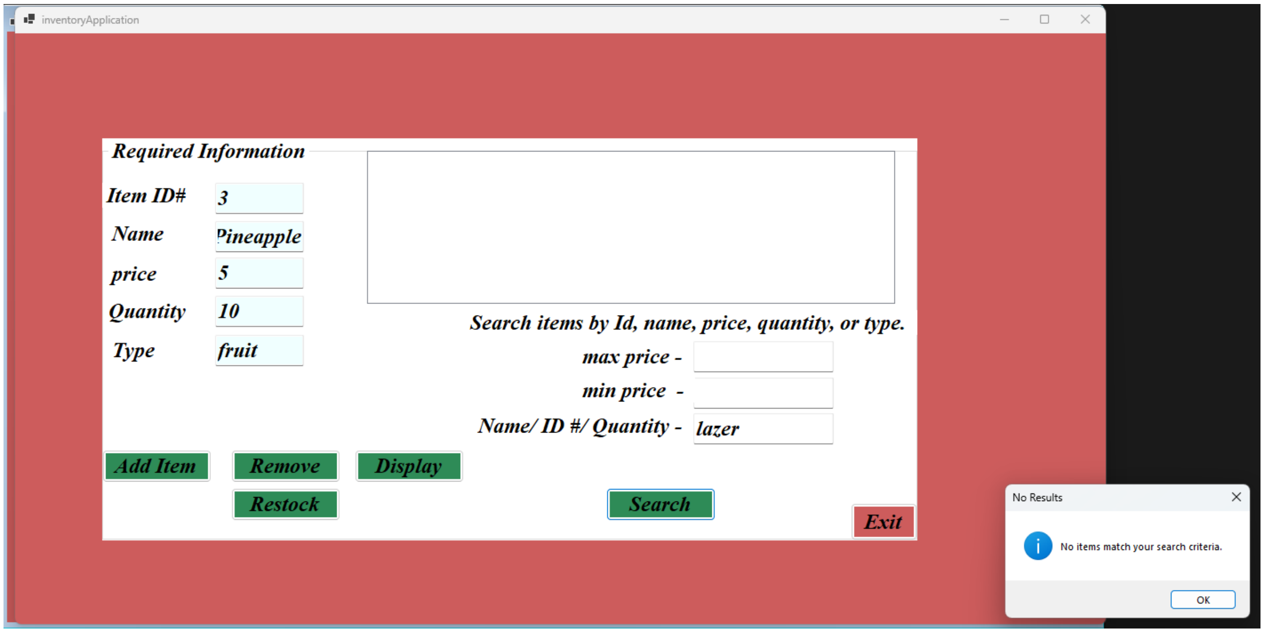
1 | Apples | $5 | 10 units
2 | oranges | $10 | 10 units
3 | Pineapple | $5 | 10 units

Name/ ID #/ Quantity - 10



Item ID# 1
Name Apples
price 5
Quantity 10
Type fruit

Apples has been added to inventory with a quantity of 10.



Required Information

Item ID# 3

Name Pineapple

price 5

Quantity 10

Type fruit

Search items by Id, name, price, quantity, or type.

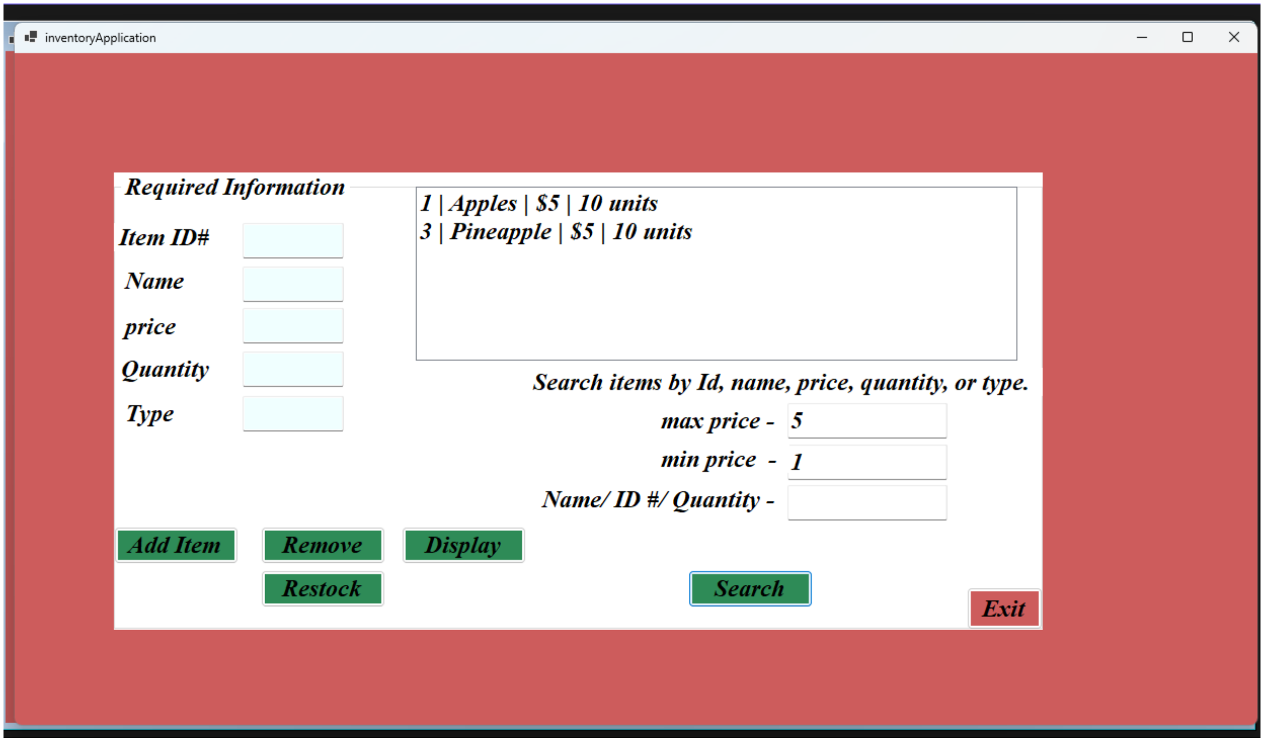
max price -

min price -

Name/ ID #/ Quantity - lazer

No Results

No items match your search criteria.



Required Information

Item ID#

Name

price

Quantity

Type

1 | Apples | $5 | 10 units
3 | Pineapple | $5 | 10 units

Search items by Id, name, price, quantity, or type.

max price - 5

min price - 1

Name/ ID #/ Quantity -