Peer-graded Assignment: Project Submission

Console-Based Inventory Management System in C#

Step 1: Review the design challenges

a. To complete this challenge, you will need to create a console application where users can manage product stock. Users should be able to add new products, update stock, and remove products.

b. Some key features include:

   i. Add new products with name, price, and stock quantity;

   ii. Update stock when products are sold or restocked;

   iii. View all products and their stock levels;

   iv. Remove products from inventory;

Step 2: Define the project requirements and objectives

Intro: you now know the issue that you're trying to solve. Take some time to determine the project requirements and objectives. Remember, requirements fall into two categories: functional and non-functional. Then, set the project objectives- the specific results you aim to achieve. You'll submit these requirements and objectives at the end of this project.

Project Overview

This project is a console-based Inventory Management System. The program will allow users to manage product stock by adding, updating, viewing, and removing products. The system will store product data in memory during runtime and provide a user-friendly menu for interaction.

Functional Requirements (what the system should do)

1. The system must allow users to add new products with the following details:
   - Name (string);
   - Price (decimal);
   - Stock quantity (integer).
2. The system must allow users to view all products, displaying their name, price, and current stock;

3. The system must allow users to update stock levels when items are sold or restocked (positive or negative adjustment);
4. The system must allow users to remove a product from inventory by name;
5. The system must present a menu of options in a loop until the user chooses to exit.

Non-Functional Requirements (how the system should behave)

1. The system must be a console application written in C#;
2. The program should handle user input errors gracefully (e.g., invalid menu option or wrong data type);
3. Code should be modular, using separate methods for each major function (e.g., AddProduct, UpdateStock);
4. The system should be easy to use, with clear instructions and labels.

Project Objectives

1. Design and implement a working inventory system using object-oriented principles;
2. Demonstrate control structures (e.g., if-else, switch) for decision-making logic;
3. Use loop constructs (while, for) to support ongoing interaction;
4. Apply function decomposition by defining and calling custom methods;
5. Practice structured programming and basic user interaction in a C# console environment.

Step 3: Create a design outline

1. Major Tasks and Code Planning:

Task	Description	Code component needed
Display menu	Show user options (Add, View, Update, Remove, Exit)	Switch or if-else loop
Add new product	Get input: name, price, stock. Create and store product.	Method: AddProduct() variables
View inventory	Display all products in list	Method: ViewInventory() foreach
Update product stock	Search product by name adjust stock (add or subtract)	Method: UpdateStock() conditionals

Remove product	Search product by name, remove from list	Method: RemoveProduct(), list methods
Keep program running	Loop through menu until exit is chosen	While or do-while loop
Handle errors	Validate user input, display error messages	Try-catch, if-else

2. **Planned Variables:**

Variable Name	Type	Purpose
inventory	List<Product>	Stores all products in memory
name	string	Holds product name input
price	decimal	Holds product price input
stock	int	Holds quantity input
choice	string	Hold user menu selection

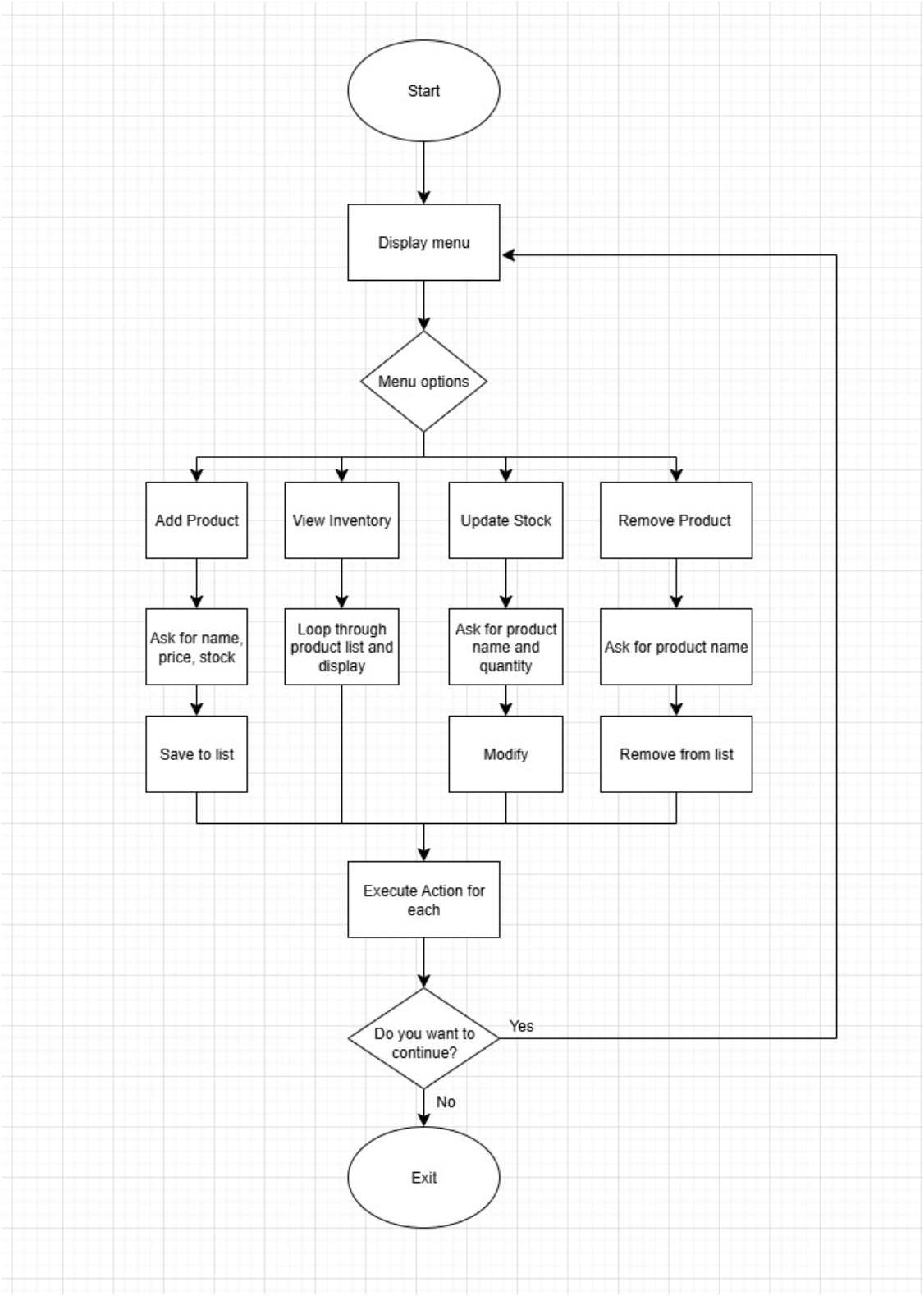
3. **Planned Methods:**
   - AddProduct(): get product input and adds to inventory;
   - ViewInventory(): display all products;
   - UpdateStock(): modifies stock of selected product;
   - RemoveProduct(): deletes product by name.

4. **Control Structures:**
   - Switch-case: for handling user menu choices;
   - If-else: for conditions (e.g., product found or not);
   - While loop: keeps menu running until user exits

5. **Flowchart/Diagram Suggestion**

The flowchart in Picture 1 illustrates the core logic of the Inventory Management System console application. It begins with the program displaying a menu of options, allowing the user to choose between adding a new product, viewing the current inventory, updating stock quantities, or removing a product. Each option leads to a specific set of

actions, such as collecting user input, modifying the product list, or displaying product details. After completing the selected task, the system prompts the user to decide whether to continue using the application or exit. This loop ensures continuous interaction until the user explicitly chooses to terminate the program, providing a structured and user-driven experience.



Picture 1 – Inventory Management System Flowchart

Step 4: Build the application

Intro: with your plan in place, you can begin creating the application using C# in Visual Studio Code. This is the first time you've written code entirely independently, so take your time and review earlier lessons for a refresher. Also, feel free to leverage Copilot to help you code. Copilot can help you write, analyze, debug, and optimize your code as you're going. Remember that all generative AI tools are meant to work in conjunction with you and require your input and knowledge to complete tasks.

To show your understanding of the concepts in this course, make sure your code includes:

- Control structures like if-else and switch statements;
- Loops like for and while;
- Methods that you define and call.

Code:

```
class Product(string name, decimal price, int stock)
{
    public string Name { get; set; } = name;
    public decimal Price { get; set; } = price;
    public int Stock { get; set; } = stock;

    public void Display()
    {
        Console.WriteLine($"Name: {Name}, Price: ${Price}, Stock:
{Stock}");
    }
}

class Program
{
    static readonly List<Product> inventory = [];

    static void Main()
    {
        bool running = true;

        while (running)
        {
            Console.WriteLine("\nInventory Management System");
            Console.WriteLine("1. Add Product");
            Console.WriteLine("2. View Inventory");
            Console.WriteLine("3. Update Stock");
            Console.WriteLine("4. Remove Product");
            Console.WriteLine("5. Exit");
            Console.Write("Choose an option (1-5): ");
            string choice = Console.ReadLine() ?? string.Empty;

            switch (choice)
            {
```

```
                case "1":
                    AddProduct();
                    break;
                case "2":
                    ViewInventory();
                    break;
                case "3":
                    UpdateStock();
                    break;
                case "4":
                    RemoveProduct();
                    break;
                case "5":
                    running = false;
                    break;
                default:
                    Console.WriteLine("Invalid option. Please choose
between 1 and 5.");
                    break;
            }
        }

        Console.WriteLine("Exiting the program. Goodbye!");
    }

    static void AddProduct()
    {
        Console.Write("Enter product name: ");
        string name = Console.ReadLine() ?? string.Empty;

        Console.Write("Enter price: ");
        if (!decimal.TryParse(Console.ReadLine(), out decimal price))
        {
            Console.WriteLine("Invalid price input.");
            return;
        }

        Console.Write("Enter stock quantity: ");
        if (!int.TryParse(Console.ReadLine(), out int stock))
        {
            Console.WriteLine("Invalid stock input.");
            return;
        }

        inventory.Add(new Product(name, price, stock));
        Console.WriteLine("Product added successfully.");
    }

    static void ViewInventory()
    {
        if (inventory.Count == 0)
        {
            Console.WriteLine("Inventory is empty.");
        }
        else
        {
            Console.WriteLine("\nCurrent Inventory:");
            foreach (var product in inventory)
            {
```

```
                product.Display();
            }
        }
    }

    static void UpdateStock()
    {
        Console.Write("Enter product name to update: ");
        string name = Console.ReadLine() ?? string.Empty;

        Product? found = inventory.Find(p => p.Name.Equals(name,
StringComparison.OrdinalIgnoreCase));
        if (found != null)
        {
            Console.Write("Enter quantity to add (use negative number to
subtract): ");
            if (!int.TryParse(Console.ReadLine(), out int
quantityChange))
            {
                Console.WriteLine("Invalid input.");
                return;
            }

            found.Stock += quantityChange;
            Console.WriteLine("Stock updated successfully.");
        }
        else
        {
            Console.WriteLine("Product not found.");
        }
    }

    static void RemoveProduct()
    {
        Console.Write("Enter product name to remove: ");
        string name = Console.ReadLine() ?? string.Empty;

        Product? found = inventory.Find(p => p.Name.Equals(name,
StringComparison.OrdinalIgnoreCase));
        if (found != null)
        {
            inventory.Remove(found);
            Console.WriteLine("Product removed successfully.");
        }
        else
        {
            Console.WriteLine("Product not found.");
        }
    }
}
```

Step 5: Save and submit for review

Review your project outline and code. Make sure that you have completed the tasks required for this project. Save your completed project outline and code. When completed, copy and paste the appropriate sections for each of the submission prompts. Great work!