

**Project Title: "E-commerce Shopping Cart Management System**"

**Team Members :**

* **Faseeh -233683**
* **Dua Nadeem- 233609**
* **Fatima – 233519**

**VISUAL PROGRAMMING(CS284)**

**C# PROGRAMMING PROJECT**

**BSCS-A – FALL 23**

P

Submitted to : Mam Atka Ali

Submission Date:10/25/2024

**Objective**  
The objective of this project is to create a fully functional shopping cart system that allows users to efficiently manage their purchases by adding, removing, and viewing products in the cart. The system will calculate the total cost, apply discounts, manage quantities, and recommend products based on user interactions. This project aims to simulate a real-world shopping experience, showcasing proficiency in C# and data structures like lists and dictionaries for product management.

**Team Members' Contribution**

**1. Faseeh:** Worked on User Interface and Adding , Removing Selected Products Functionality and developed vegetable section .

**2. Dua :** Worked and developed on the Meat and Sweets Section and implementation of button and textbox display controlling .

**3.Fatima:** Worked on the functionality of Cart View, Total Bill Calculation, Product Recommendations, a Discount Management , and developed fruit section .

**Project Initialization and UI Setup in Form1 Constructor**

This section outlines the foundational elements and structure of the shopping cart system.

**1. Variable Declarations and Initialization**

The class begins with multiple variable declarations to manage the functionality of various sections in the cart (Fruits, Vegetables, Meat, Sweets, etc.). Each section includes variables for price, quantity, boolean flags to check if the item is selected, and specific item bills. These variables play a crucial role in tracking items, quantities, and pricing in the cart.

**2.Index and Flags:**

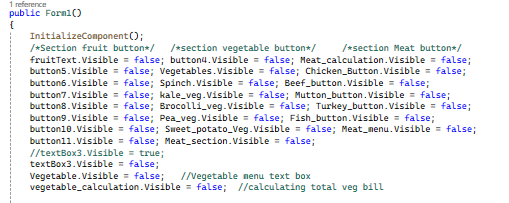
* index, reset, remove\_item, and click are initialized to manage user actions like resetting the cart, removing items, tracking button clicks,
* recomended\_item, add, show\_cart, Clear\_items, discount, and view\_total\_bill are boolean flags to track different cart operations, such as adding items, viewing the cart, applying discounts, and displaying the total bill.

**3.Item-Specific Variables:**

* Each item within each section (Fruits, Vegetables, Meat, Sweets) has a dedicated variable for price, quantity, selection flag, and calculated bill.
* Example: For Mangoes, Price\_mango holds the item price, Quantity\_mango tracks the quantity, mango is a flag indicating if mango is added, and Mango\_bill stores the cumulative bill for mangoes.

**4.Total Price Variables:**

total\_fruit\_price, total\_Veg\_price, total\_meat\_price, and total\_sweet\_price maintain the total cost per section, making it easier to calculate the overall bill.

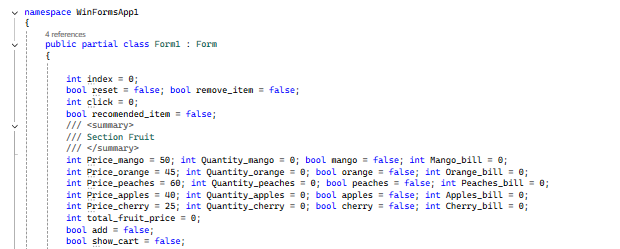


**Constructor Form1(): User Interface (UI) Initialization**

The Form1 constructor serves to initialize the components of the shopping cart’s Graphical User Interface (GUI). It sets the visibility of various UI elements and organizes them into clear sections. Each section is tailored to handle specific items within the cart, and only relevant elements are displayed based on the user’s actions.

* **Section: Fruits**

The Fruit section contains buttons and images for each type of fruit (e.g., Mango, Orange, Peaches), as well as calculations related to the fruit selection.

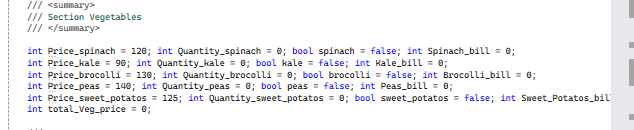


* **Buttons & Labels:**

fruitText, button4, and Meat\_calculation are set to invisible initially to avoid clutter on the main screen.They will be made visible only when the Fruit section is selected by the user.

* **2.Section: Vegetables**

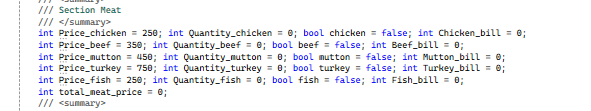
Vegetable Menu:



Elements like Vegetable, Spinch, kale\_veg, and vegetable\_calculation are also initially hidden.These components include buttons to add specific vegetables and a text field for the vegetable menu.

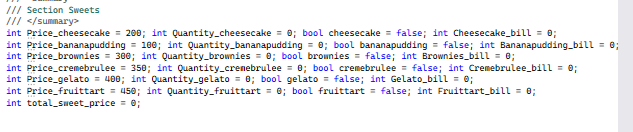
* **3.Section: Meat**

The Meat section includes UI controls for selecting various types of meat, each item associated with an image and calculation button.

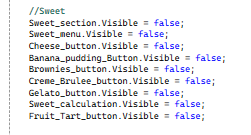


* ***Meat Selection:***
* Buttons such as Chicken\_Button, Beef\_button, and Mutton\_button are hidden initially.
* Meat\_menu and Meat\_section display the overall meat menu.
* **4.Section: Sweets**

Sweets are another category in the shopping cart, with UI components for items like cheesecake, brownies, and gelato.

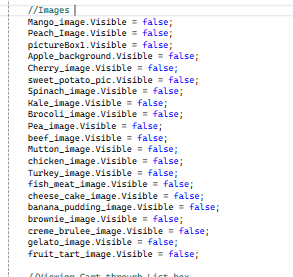


* ***Buttons &images:***



* Sweet\_section, Sweet\_menu, and specific buttons (e.g., Cheese\_button, Brownies\_button) remain hidden.
* When the Sweets section is selected, these buttons and related images will appear, allowing the user to interact with them.
* **Images and Icons Setup**

In the shopping cart system, each item has an associated image that visually represents the product in the cart. These images are linked to specific buttons and displayed based on the user’s selection.



* **Image Control:**
* All images, such as Mango\_image, Peach\_Image, Apple\_background, etc., are set to invisible by default.
* Each image will be displayed when a corresponding item is added to the cart, giving the user visual confirmation.
* **5. Cart Viewing and List Management**

To manage the items in the cart effectively, the ListView control is used for a detailed display of items currently in the cart.

* **ListBox and TextBox:**
* listView1, which will display the list of items, is set to invisible until the cart view is activated.
* textBox2 and textBox3 serve as text fields for additional information or interaction, like showing the item list or displaying total costs.
* **Key Features Introduced in this Section**

The UI design within this constructor focuses on user experience and clear functionality. The initial hidden state of many components is a deliberate choice to prevent information overload for users. Only the necessary buttons, images, and sections will be visible based on user interaction. This makes the shopping cart easy to navigate and visually organized.

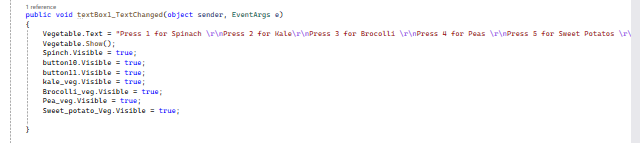
**1.1 Recommended Item Button :**

* **Functionality: This button sets the recomended\_item boolean to true, signaling that a recommended item can be displayed when triggered.**
* **Workflow:** When this button is clicked, it activates the recommended item feature by setting the **boolean flag**. If the flag is true, the program can later display the message with a recommended item (e.g., "Mango") as shown in other parts of the code.



**1.2 Vegetable Selection Text Box :**

* **Functionality**: This text box displays a list of available vegetable options, enabling the user to choose by number.
* **Workflow:** When the text changes, a message appears in **Vegetable.Text**, showing numbered options for each vegetable. The selection-related buttons (like Spinch, kale\_veg, etc.) become visible for the user to add items.



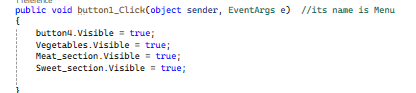
**1.3 Reset Cart Button :**

* **Functionality:** This button clears the cart by removing all items from **listView1.**
* **Workflow**: Clicking this button clears all items in the list view (listView1), effectively resetting the shopping cart.



**1.4 Menu Button:**

* **Functionality:** Shows main sections of the menu: Fruits, Vegetables, Meat, and Sweets.
* **Workflow:** Clicking **button1** reveals buttons for each main menu section, allowing users to select specific categories to explore. For example, button4 opens fruits, Vegetables shows the vegetable section, etc.

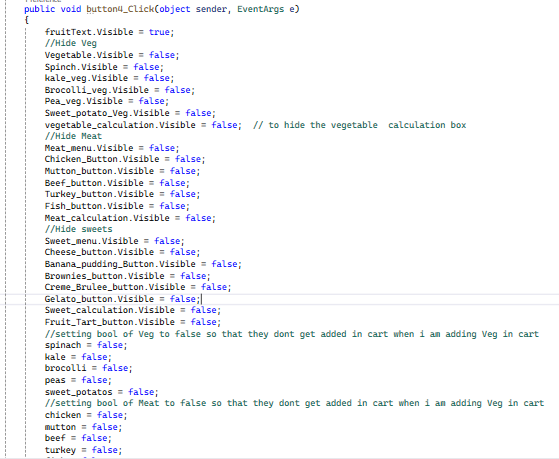


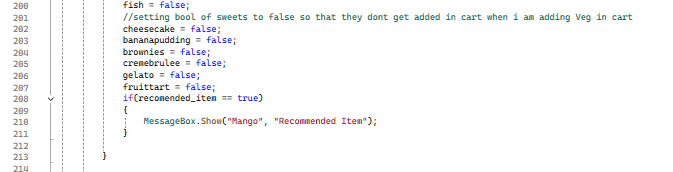
**1.5 Fruit Section Button :**

**Functionality:** Displays fruit options while hiding all other sections and items.

**Workflow:**

* When **button4** is clicked, it sets the fruitText to visible, which contains instructions for selecting fruits.
* All other section-related UI elements (vegetables, meat, sweets) are hidden, ensuring only fruit-related items appear on the screen.
* If recomended\_item is set to true, a message box shows the recommended item (e.g., "Mango").



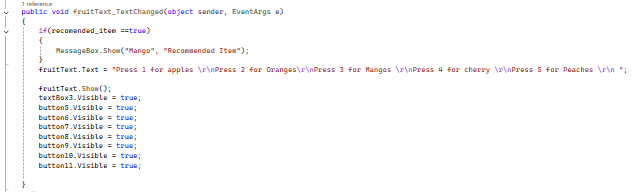


* **Fruit Selection Text Box**

**Functionality:** Provides a numbered list of fruit options to guide the user in selecting fruit items.

**Workflow:**

* When the text changes in **fruitText**, it displays a list of numbered fruit choices (e.g., "Press 1 for apples").
* This setup also shows fruit selection buttons (button5 through button9) and a visible textBox3 where additional user input can be processed.

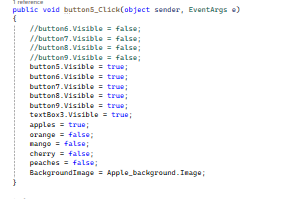


**1.6 Apple Selection Button :**

**Functionality:** Selects apples and displays an apple-related image as the background.

**Workflow:**

* When **button5** is clicked, it sets apples to true and resets other fruits’ boolean values to false.
* The background image changes to Apple\_background.Image to visually indicate the selected fruit.

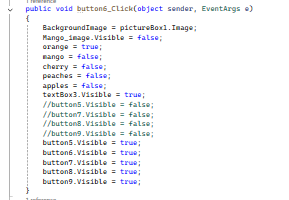


**1.7 Orange Selection Button :**

**Functionality:** Selects oranges, displaying the corresponding image in the background.

**Workflow:**

* Clicking **button6** activates the orange boolean and deactivates others like mango and apples.
* The background image updates to pictureBox1.Image, representing oranges visually for the user.

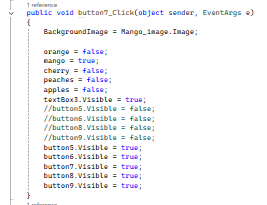


**1.8** **Mango Selection Button :**

**Functionality:** Selects mangoes, changing the background to the mango image.

**Workflow:**

* Clicking **button7** sets mango to true while resetting other fruit flags.
* The BackgroundImage changes to Mango\_image.Image, providing a visual cue for the selection of mangoes.

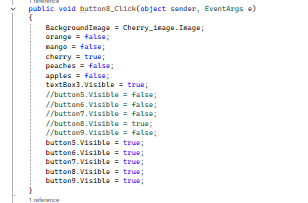


**1.9** **Cherry Selection Button :**

**Functionality:** Selects cherries and updates the background accordingly.

**Workflow:**

* When **button8** is clicked, it activates the cherry boolean and deactivates other fruit booleans like mango and apples.
* The background updates to Cherry\_image.Image to represent cherry selection.

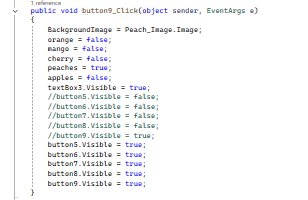


**1.10 Peach Selection Button:**

**Functionality:** Selects peaches, setting the peach image as the background.

**Workflow:**

* Clicking **button9** activates peaches and deactivates other fruits.
* The BackgroundImage changes to Peach\_Image.Image, providing a clear visual representation of peach selection.



* **textBox3\_TextChanged**:
* Calculates the total price of selected fruits based on user input in textBox3.
* If adding, it updates the quantity and total cost for the selected fruit; if removing, it reduces the quantity and recalculates the total.
* Each fruit (apple, mango, orange, cherry, peach) has its price updated separately based on a quantity \* price formula.
* The total\_fruit\_price sums up all individual fruit bills.
* **button10\_Click (Remove Button)**:
* Activates removal mode by setting remove\_item to true and add to false.
* **button11\_Click (Add Button)**:
* Activates add mode by setting add to true and remove\_item to false.
* **Vegetables\_Click**:
  + - Displays vegetable options and hides other sections (Fruit, Meat, Sweets) along with their calculations.
    - Sets all other sections' selection flags to false, ensuring only the vegetable items are active.
* **Spinch\_Click, kale\_veg\_Click, Brocolli\_veg\_Click, Pea\_veg\_Click, Sweet\_potato\_Veg\_Click**:
  + - Each button click sets the background image to the selected vegetable and enables only the specific vegetable's flag (e.g., spinach = true for Spinach).
    - Disables all other vegetable selection flags to prevent overlapping selections.

**Similar Functionalities:**

**Fruit Quantity Addition:**

* **Button11 (Add Button):** Sets add to true and remove\_item to false. It enables the addition of fruit quantities based on user input.
* **textBox3\_TextChanged:** Uses the add flag to determine if the quantity of selected fruit should be increased when valid input is provided.

**Fruit Quantity Removal:**

* **Button10 (Remove Button**): Sets remove\_item to true and add to false. It allows for the removal of fruit quantities based on user input.
* **textBox3\_TextChanged:** Uses the remove\_item flag to determine if the quantity of the selected fruit should be decreased when valid input is provided.

**Differences:**

**Button Functionalities:**

* **Button10 and Button11:** These buttons have opposite functionalities. Button10 initiates the removal process, while Button11 initiates the addition process.
* **textBox3\_TextChanged:** This method handles both adding and removing items based on the state set by Button10 or Button11, which changes the behavior of the method dynamically.

**Vegetable Selection Buttons:**

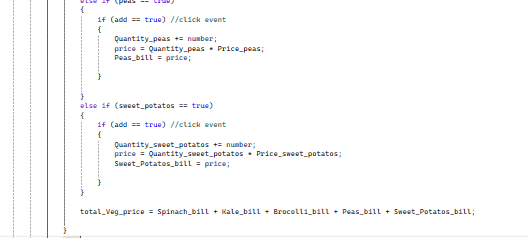
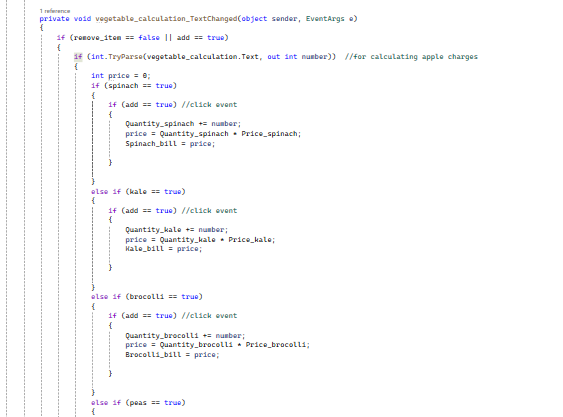
* **Spinach\_Click, kale\_veg\_Click, Brocolli\_veg\_Click, Pea\_veg\_Click, Sweet\_potato\_Veg\_Click:** Each of these buttons updates the background image and sets a specific vegetable flag to true, while ensuring that all other vegetable flags are set to false. Although they all perform similar tasks (updating the UI and managing state), they are distinct in which vegetable is being selected.

**Overview**

The **vegetable\_calculation\_TextChanged** method handles the addition and removal of vegetable quantities based on user input in a text box (vegetable\_calculation). It updates the corresponding quantity and bill amount for each vegetable type and calculates the total price accordingly.

**Key Points:**

* **State Handling:**
* The method checks if items are being added or removed by evaluating the remove\_item and add boolean flags.
* If remove\_item is false or add is true, it indicates the user wants to add vegetables. Otherwise, it processes removal.
* **Adding Vegetables:**
* The method first attempts to **parse the input** from the text box as an integer (number).
* For each vegetable type (spinach, kale, broccoli, peas, sweet potatoes), it **checks if the corresponding boolean flag is true.**
* If the add flag **is true**, it **increments** the respective vegetable quantity and calculates the bill:
* **Quantity\_vegetable += number;**
* **price = Quantity\_vegetable \* Price\_vegetable;**
* **Updates** the respective bill variable (e.g., Spinach\_bill).
* **Total Price Calculation:**
* After updating the individual vegetable bills, it calculates the total vegetable price by summing up all the individual vegetable bills:
* total\_Veg\_price = Spinach\_bill + Kale\_bill + Brocolli\_bill + Peas\_bill + Sweet\_Potatos\_bill;
* **Removing Vegetables:**
* If **remove\_item is true**, the method again attempts to parse the input as an integer (R\_Number).
* Similar to the addition process, it checks which vegetable is being removed based on the corresponding **boolean flag.**
* It **calculates** the actual removal and adjusts the quantities and bills accordingly:
* **If the requested removal quantity exceeds the available quantity**, it caps the removal quantity to the available amount.
* It then **updates** the quantity and bill and subtracts the removed value from the total\_Veg\_price.
* **Handling Edge Cases:**
* The code includes logic to handle cases where the input is invalid (not numeric) but does not specify any action in the else blocks.

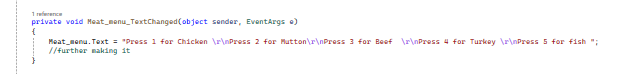


* **Meat\_menu\_TextChanged**
* **Overview**

This code manages the **selection of different types of meat in a shopping application**, allowing users to choose from **chicken, mutton, beef, turkey, and fish**. It *updates the user interface* accordingly and hides other food sections (fruits, vegetables, and sweets) when the meat section is selected.

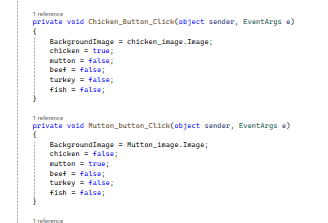
* **Key Functions**

1. **Meat\_menu\_TextChanged:**

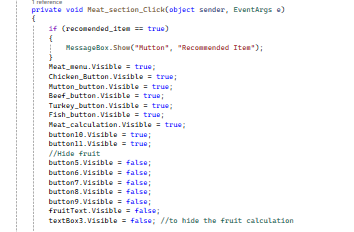
* This method sets the text of the Meat\_menu control to display options for different types of meat.
* It presents the user with choices (chicken, mutton, beef, turkey, fish) when the meat menu is accessed.
  + 

1. **Meat Selection Buttons:**
   * + **Chicken\_Button\_Click, Mutton\_button\_Click, Beef\_button\_Click, Turkey\_button\_Click, Fish\_button\_Click:**
     + Each button corresponds to a specific meat type. When a button is clicked, it updates the background image of the form to visually represent the selected meat.
     + The boolean flags (chicken, mutton, beef, turkey, fish) are set to true for the selected meat, while all others are set to false. This ensures only one type of meat can be selected at a time.

Similar Functionality: The meat selection buttons have the same basic structure and functionality, differing only in the specific meat type being handled.

* + 

1. **Meat\_section\_Click:**
   * **This method is triggered when the meat section is clicked. It checks if a recommended item is indicated (recomended\_item == true) and shows a message box recommending "Mutton".**
   * **It then makes the meat selection buttons and the meat calculation visible while hiding the fruit, vegetable, and sweet sections:**
     + Sets the visibility of the respective buttons and calculation fields to true or false based on the meat selection.
     + Additionally, it resets boolean flags for fruits, vegetables, and sweets to ensure they do not get added to the cart while selecting meat.



* **Summary of Similarities**
* **Button Click Methods:** The structure of the button click methods for each type of meat is consistent, with each method updating the corresponding boolean flag and the background image. The only difference lies in the meat type being processed.
* **Visibility Management:** The Meat\_section\_Click method functions similarly to the section management methods for fruits and vegetables, where visibility is toggled for different sections based on user selection**.**
* **Highlighting Differences**
* Each button click method updates a specific image and sets only its corresponding boolean flag to true, while ensuring all others are false. This is crucial for allowing only one meat selection at a time.
* The Meat\_section\_Click method includes additional logic for hiding other sections and managing the state of various boolean flags, which is unique to the context of selecting the meat section.
* **Overview**

The Meat\_calculation\_TextChanged method is responsible for managing the pricing calculations of the selected meat items based on user input. It differentiates between adding and removing items and updates the total price accordingly.

* **Key Functionality**

1. **Adding Items:**
   * **if (remove\_item == false || add == true):**
     + *This block checks whether an item is being added (i.e., add == true) or if the removal flag is not set (remove\_item == false).*
     + *If the user inputs a valid integer quantity, it determines which meat type is selected and updates the corresponding quantity and bill.*
   * **Meat Type Logic:**
     + **For each type of meat (chicken, mutton, beef, turkey, fish):**
       - It checks if the corresponding boolean flag is true.
       - If add is true, it updates the quantity (e.g., Quantity\_chicken += number;) and calculates the total price for that meat (e.g., price\_ = Quantity\_chicken \* Price\_chicken;).
       - It stores the calculated price in the respective bill variable (e.g., Chicken\_bill, Mutton\_bill, etc.).
2. **Calculating Total Price:**
   * **The method sums up the prices of all selected meats to get total\_meat\_price:**

total\_meat\_price = Chicken\_bill + Beef\_bill + Mutton\_bill + Turkey\_bill + Fish\_bill;

1. **Removing Items:**
   * **else block:**
     + **This section is executed if the user intends to remove items.**
     + It checks if the input quantity is valid and processes the removal based on the selected meat type.
     + For each meat type, it checks if the input quantity exceeds the available quantity. If it does, it adjusts the removal quantity to the available amount:

***if (r\_Number > Quantity\_chicken) { r\_Number = Quantity\_chicken; }***

* + **Price Adjustment:**
    - It calculates the cut\_value based on the removal and updates the corresponding quantity and bill:

**Quantity\_chicken -= r\_Number;**

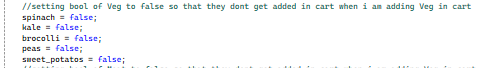
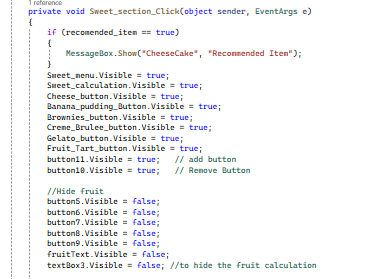
**Chicken\_bill = Quantity\_chicken \* Price\_chicken;**

* + **Updating Total Price:**
    - **The total\_meat\_price is updated by subtracting the cut\_value of the removed items:**

**total\_meat\_price = total\_meat\_price - cut\_value;**

* **Summary of Similarities**
* **Add and Remove Logic:** Both the adding and removing functionalities follow a similar structure, involving checks for valid integer inputs and determining the selected meat type. Each section updates the respective quantity and bill based on user actions.
* **Meat Type Handling**: The logic for handling each meat type is consistently structured in both adding and removing sections, making it easy to understand how each type is processed. The only change is whether it adds to or subtracts from the total quantity and price.
* **Highlighting Differences**
* **Conditional Logic:** The main difference is the condition that determines whether the quantity is being added or removed, leading to different operations (addition vs. subtraction of quantities and price calculations).
* **Price Calculation:** The addition logic involves multiplying the quantity by the price to compute the bill, while the removal logic involves checking the quantity available and updating the total price accordingly.
* **Sweet\_menu\_TextChanged:**
* **Overview**

The methods in this section manage the user interface and behavior for selecting sweet items. The Sweet\_menu\_TextChanged method updates the menu text, while the Sweet\_section\_Click method displays the relevant buttons and hides other categories when the sweets section is activated.



* **Key Functionality**

1. **Sweet Menu Text Changed:**

This method updates the text in the Sweet\_menu to display options for various sweet items (Cheesecake, Banana Pudding, Brownies, etc.):

**Sweet\_menu.Text = "Press 1 for Cheesescake \r\nPress 2 for Banana Pudding\r\nPress 3 for Brownies \r\nPress 4 for Creme Brulee \r\nPress 5 for Gelato \r\nPress 6 for Fruit tart\r\n ";**

*It then shows the Sweet\_menu.*

1. **Sweet Section Click:**

This method is triggered when the sweets section is clicked. It checks if a recommended item is set (recomended\_item == true) and displays a message box recommending Cheesecake.

It makes various buttons and controls visible (related to sweets) and hides others related to fruits, vegetables, and meats:

**Sweet\_menu.Visible = true;**

**Sweet\_calculation.Visible = true;**

**Cheese\_button.Visible = true;**

***Similar to previous sections, it resets the boolean flags for fruits, vegetables, and meats to false, ensuring that those categories do not interfere when sweets are added to the cart.***

* **Individual Sweet Button Click Handlers:**
* **Cheese Button:**

**private void Cheese\_button\_Click(object sender, EventArgs e)**

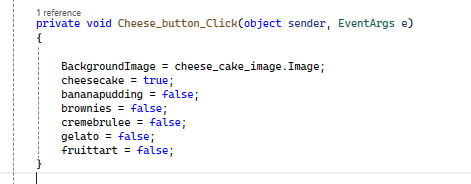
**{**

**BackgroundImage = cheese\_cake\_image.Image;**

**cheesecake = true;**

**// Other sweet flags are set to false...**

**}**



*This handler sets the background image to the Cheesecake image and updates the boolean flag for Cheesecake to true, while setting all others (bananapudding, brownies, etc.) to false.*

**Similar logic is followed for the other sweet buttons (Banana\_pudding\_Button\_Click, Brownies\_button\_Click, etc.), where each handler updates the background image and the respective boolean flag.**

* **Summary of Similarities:**
* **Add and Hide Logic:** Both the Sweet\_section\_Click and individual sweet button click handlers follow a similar structure to manage visibility and state:
* **Button Handler** :Each sweet button handler adjusts the background image and manages the boolean states for sweet items in a consistent manner.
* **Boolean State Management:** All button handlers set their corresponding boolean flags to true while resetting others to false, maintaining a clear single selection for sweet items.
* **Highlighting Differences**
* **Menu Display vs. Individual Selection:** *The main difference lies in the functionality:*

1. *The Sweet\_section\_Click method controls the overall visibility of sections and elements when entering the sweets category.*
2. *Individual button handlers focus on specific sweet selections and update the UI (background image and boolean states) for each sweet type.*

* **Overview**

The Sweet\_calculation\_TextChanged method handles the calculations related to sweet items based on user input. It determines whether to add or remove quantities of sweets, updating the total price accordingly.

* **Key Functionality**

The method starts by checking if the item is being removed (remove\_item == false) or added (add == true):

**if (remove\_item == false || add == true)**

* **Adding Sweet Items:**

1. *if (int.TryParse(Sweet\_calculation.Text, out int number\_s))*
2. *It tries to parse the input from the Sweet\_calculation text box into an integer (number\_s).*
3. *Then, based on which sweet item is selected (checked through boolean flags), it updates the quantity and calculates the price:*

**if (cheesecake == true)**

**{**

**if (add == true)**

**{**

**Quantity\_cheesecake += number\_s;**

**price\_s = Quantity\_cheesecake \* Price\_cheesecake;**

**Cheesecake\_bill = price\_s;**

**}**

**}**

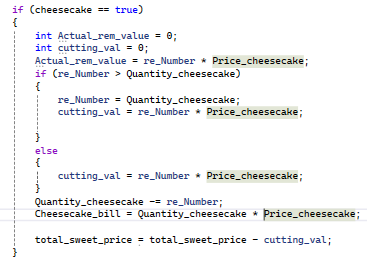
***This pattern is repeated for all sweet types (Banana Pudding, Brownies, etc.), adjusting the corresponding quantities and calculating the total bill for each item*.**

* **Calculating Total Price:**

*At the end of the addition block, it sums the individual bills to get the total price of all*

total\_sweet\_price = Cheesecake\_bill + Bananapudding\_bill + Brownies\_bill + Gelato\_bill + Fruittart\_bill;

*Similar to adding items, it checks the selected sweet item and calculates how much should be removed based on the input quantity:*



***This pattern is also repeated for all sweet types, ensuring that quantities and prices are adjusted correctly when items are removed.***

* **Summary of Similarities**
* **Boolean State Management:** Like previous methods, this method utilizes boolean flags to determine which sweet item is currently selected, ensuring that only the appropriate calculations are performed.
* **Input Parsing:** Both this method and the earlier methods use int.TryParse to validate and convert user input into usable integers.
* **Visibility Control:** While the previous methods controlled visibility of buttons and sections, this method manages quantity and price calculations.
* **Highlighting Differences**
* **Functionality**: The main difference is that this method focuses on the calculation of quantities and prices based on user input, whereas the previous methods handled the visibility and state management of the UI elements.
* **Add/Remove Logic:** This method explicitly differentiates between adding and removing items, with distinct logic for each scenario. In contrast, earlier methods were primarily concerned with displaying available options and toggling visibility.
* **Method: button2\_Click**

***This method is called when the button (presumably labeled "Show Cart") is clicked.***

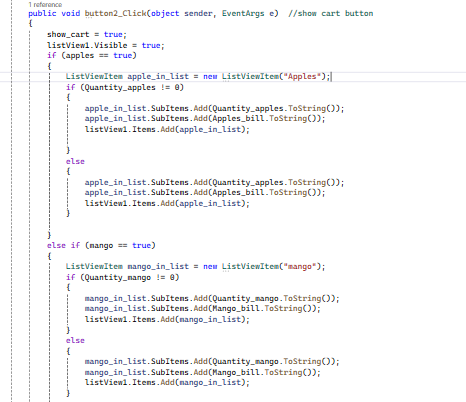
* **Functionality**: The method is responsible for displaying items in a shopping cart using a ListView control. It checks the quantity and availability of different items (fruits, vegetables, meat, and sweets) and adds them to the cart if they are selected and their quantities are greater than zero.
* **Code Breakdown:**

**Variable Initialization:**

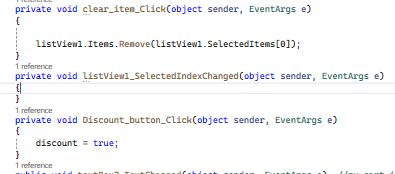
* *show\_cart is set to true, which likely indicates that the cart should be visible.*
* *listView1.Visible is set to true, making the ListView control visible on the form.*
* *Checking Item Conditions:* The method uses a series of if-else statements to check which items have been selected based on boolean flags (e.g., apples, mango, orange, etc.).
* Each if condition checks whether a particular item is selected. If true, it creates a ListViewItem representing that item.

**Adding Items to ListView:**

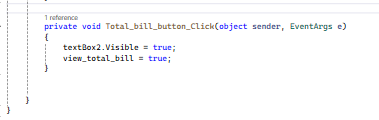
For each item, it checks the quantity. If the quantity is not zero, it adds the item name, quantity, and total bill to the ListView. If the quantity is zero, it still adds the item but with a quantity of zero.



* **Purpose of Each If-Else Condition:**
* **Item Selection:** Each if checks for a specific item (e.g., apples, mango, orange, etc.) to determine if it was selected by the user.
* **Quantity Check:** Inside each condition, another check is made to determine if the quantity for the selected item is greater than zero.
* If greater than zero, the item is added to the ListView with its name, quantity, and bill amount.
* If the quantity is zero, it still adds the item to the ListView but indicates that no quantity is available (which may be useful for UI feedback).



* **Clear Item Functionality:**
* This method is triggered when the user clicks the button associated with clearing an item from the shopping cart.
* It removes the selected item from the listView1, which presumably displays the items in the cart.
* **ListView Selection Change**:
* This method is defined but currently does not perform any action. It can be utilized in the future for additional functionality, such as updating item details when a selection is changed.
* **Discount Activation**:
  + This method is invoked when the user clicks the discount button.
* It sets the discount flag to true, indicating that a discount will be applied to the total bill.
* **Displaying Total Bill:**
* This method is executed when the text in textBox2 changes, primarily for displaying the total bill.
* It checks if view\_total\_bill is true. If so, it calculates the total amount by summing the prices of various items (total\_fruit\_price, total\_meat\_price, etc.).
* If the discount is applied (discount == true), it calculates a 15% discount (d\_amount) and deducts it from the total amount.
* The total amounts are then displayed in textBox2, which shows the breakdown of each category, the discount amount (if applicable), and the grand total.
* **Total Bill Button Functionality**:
  + This method makes textBox2 visible when the user clicks the button to view the total bill.
* It sets view\_total\_bill to true, which triggers the calculations in the textBox2\_TextChanged method the next time it runs.



* **Summary of the Program**

This C# program serves as a shopping cart system within a graphical user interface (GUI) environment, developed to enhance user experience while managing purchases. The application allows users to add various items, such as fruits, vegetables, meat, and sweets, to their cart, providing an interactive way to view and manage their shopping list.

* **Key Functionalities**:
* **Item Management**: Users can add items to their cart, remove selected items, and view the current contents of their shopping cart in a ListView.
* **Total Calculation**: The program dynamically calculates the total amount of the items in the cart, considering various categories of products, which enhances the clarity of expenditures.
* **Discount Application**: Users can apply a discount, which is automatically calculated and reflected in the final amount. The 15% discount feature encourages users to make larger purchases by providing a monetary incentive.
* **User Feedback**: Through the textBox, the system displays detailed information about the purchases, including individual category totals and the grand total, improving user understanding of their spending.
* **Purpose**: The primary goal of this program is to provide a straightforward and efficient way for users to manage their shopping activities, from item selection to final bill calculation.
* **Key Points**:
* The separation of functionalities through different methods enhances code readability and maintainability.
* The use of flags (view\_total\_bill, discount) simplifies condition checks for displaying the correct information in the GUI.
* The program employs event-driven programming principles, allowing for real-time updates based on user interactions.

*In conclusion, this shopping cart application exemplifies a practical use of C# and GUI design principles, effectively meeting the needs of users by providing a functional and intuitive shopping experience. The design encourages continuous improvement, such as adding more features or optimizing existing functionalities, paving the way for a robust retail management solution.*