Custom Program Report: Inventory Management System

Name: Khang Vo
Student ID: 104220556

Summary of Program

My program simulates a small inventory management system at a store. It provides a 9-slot inventory and users can add or remove products from that inventory. Users can choose between several types of products, and each type of product will provide different information. For example, a book product will provide information about the author, publisher, year, while an electronic product will provide information about the model, brand, warranty, etc.

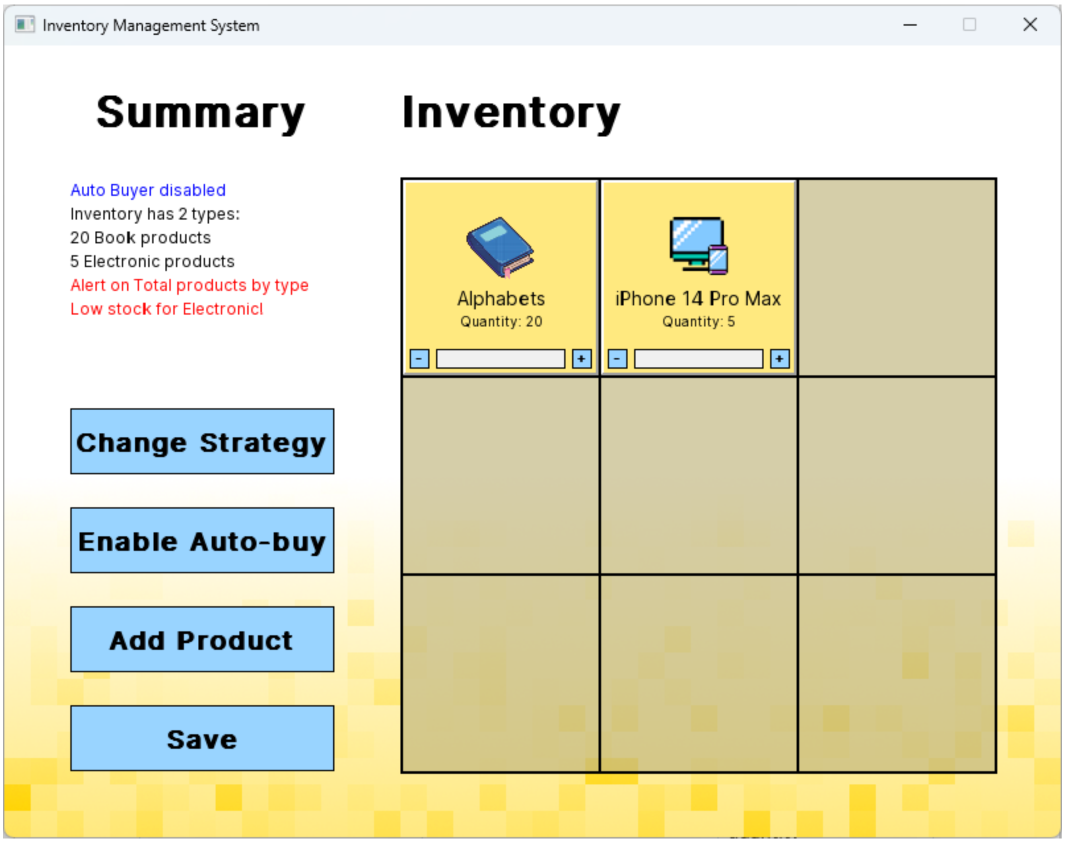
Moreover, the inventory can also alert the user when some products are low in stock. Users can use the Change Strategy button to rotate between several summary strategies in the program, and each strategy will handle the summary and alert differently.

Meanwhile, the Auto-buy feature will simulate users randomly buying products in the inventory. When enabled, it will make the products in the inventory deplete over time, making the manager to increase the product quantity again.

Furthermore, users can add new product to the inventory. In the add product page, manager can enter the new product's details. There is a Change Type button to cycle around all Product Types, a Save button to save the product to the inventory, and a Remove button to remove the product from the inventory.

Finally, there is a Save button in the main inventory page to save all products' information in the inventory to a single text file.

Screenshots of the program:



Summary

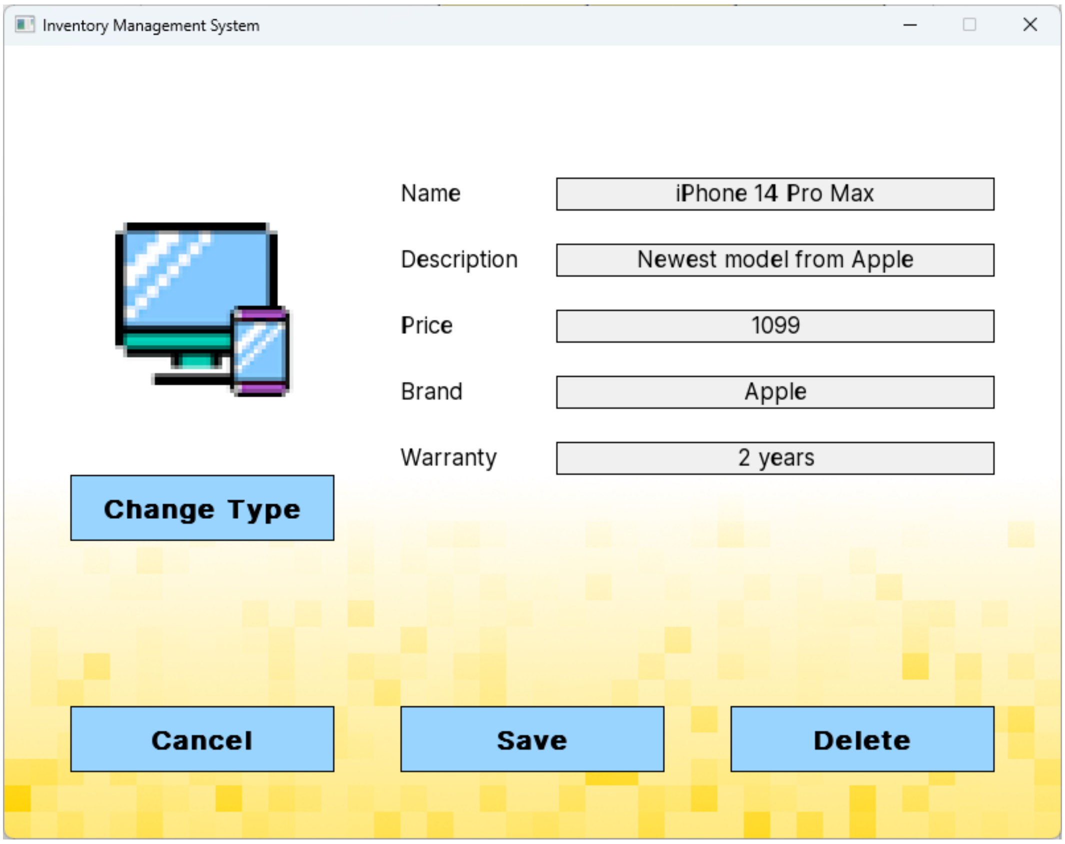
Auto Buyer disabled
Inventory has 2 types:
20 Book products
5 Electronic products
Alert on Total products by type
Low stock for Electronicl

Inventory

Alphabets
Quantity: 20

iPhone 14 Pro Max
Quantity: 5

Figure 1. Inventory main page.



Name	iPhone 14 Pro Max
Description	Newest model from Apple
Price	1099
Brand	Apple
Warranty	2 years

Figure 2. Product detail page.

Required Roles

In this custom program, I created the following classes and methods:

- ***class*** **Program**
  - *The Main method of the program.*
    ```
    void Main()
    ```
- ***class*** **AutoBuyer**: Simulate customers buying products.
  - *Randomly reduce product quantity in Inventory over time.*
    ```
    void Update()
    ```

- ***interface*** **IObserver**: For the Observer design pattern.
  - *Update object according to the Inventory updated product.*
    ```
    void Update (KeyValuePair<Product, int>? updatedProduct = null)
    ```
- ***class*** **Inventory**: Represent an inventory with storage and methods to modify products.
  - *Attach and Detach observers to Inventory, for Observer pattern.*
    ```
    void Attach (IObserver observer)
    void Detach (IObserver observer)
    ```
  - *Modify products and their quantity in the inventory.*
    ```
    void AddProduct      (Product product, int quantity)
    void SubtractProduct (Product product, int quantity)
    void RemoveProduct   (Product product)
    void ReplaceProduct  (Product oldProduct, Product newProduct)
    bool HasProduct      (Product product)
    ```
  - *Save Inventory content to a text file.*
    ```
    void Save (string filename)
    ```

- ***abstract class*** **Product**: Represent a product with necessary information.
  - *Get product information as Dictionary (key-value pair).*
    ```
    Dictionary<string, string> Describe()
    ```
  - *Set product information.*
    ```
    void SetProperty(string key, string value)
    ```
  - *Get string representation of the product to save to text file.*
    ```
    string ToString()
    ```
- ***class*** **BookProduct**: Represent a Book product.
  - *Override Describe and SetProperty to represent a book product.*
- ***class*** **ElectronicProduct**: Represent an Electronic product.
  - *Override Describe and SetProperty to represent an electronic product.*

- ***class*** **TextInputHandler**: A Class to communicate between multiple TextInput and Button objects, like Mediator pattern.
  - *For text input to start receiving input.*
    ```
    void RequestStartInput(TextInput textInput)
    ```
  - *For text input handler to stop active input from receiving input.*
    ```
    void RequestStopActiveInput()
    ```
  - *For button to stop and get text from a particular text input.*
    ```
    string RequestStopInputAndGetText(TextInput textInput)
    ```
  - *Update the active text input and handle unexpected termination.*
    ```
    void Update()
    ```
- ***class*** **GUI**: Handle Font loading, State management, and program loop
  - *Run the GUI*

```
void Run()
```

  - *Change the UI State*
    ```
    void ChangeState(UIState newState)
    ```

- ***abstract class* UIComponent**: Represent a UI component on the screen.
  - *Draw component to the screen.*
    ```
    void Draw()
    void DrawTexture()
    ```
  - *Handle user inputs.*
    ```
    void HandleInput()
    bool IsMouseHover()
    bool IsMouseClick()
    ```
- ***class* BitmapDisplay**: a component to display bitmap.
  - *Override Draw method to draw bitmap to screen.*
- ***class* Button**: an interactive button.
  - *Override Draw and DrawTexture to draw different size of button to screen, override HandleInput method to perform action when being clicked.*
- ***class* InventoryUI**: a component to display Inventory.
  - *Override Draw, DrawTexture and HandleInput to draw InventoryUI and its components to the screen.*
  - *Generate UI components for the products.*
    ```
    void GenerateProductUIs()
    ```
  - *Update the products when the Inventory is updated.*
    ```
    void Update(KeyValuePair<Product, int>? updatedProduct = null)
    ```
- ***class* ProductDetailBox**: a component to display key-pair value of product information.
  - *Override Draw and HandleInput to draw product details and labels to the screen.*
- ***class* ProductUI**: a component to display simple product card in the inventory UI.
  - *Override Draw, DrawTexture and HandleInput to draw and update the product UI and its components.*
- ***class* SummaryBox**: a component to display inventory summary.
  - *Override Draw method to draw summary to the screen.*
  - *Change to the next strategy.*
    ```
    void NextStrategy()
    ```
  - *Update the summary when the Inventory is updated.*
    ```
    void Update(KeyValuePair<Product, int>? updatedProduct = null)
    ```
  - *Generate UI components to display texts.*
    ```
    void GenerateTextDisplay()
    ```
- ***class* TextDisplay**: a component to display text.
  - *Override Draw method to draw text to screen.*
  - *Update object used in the formatted text string.*
    ```
    void UpdateFormatObject(object? formatObject)
    ```
- ***class* TextInput**: a component for users to input text.
  - *Override Draw, DrawTexture and HandleInput to draw and update the text input.*
  - *Start/Stop reading input and accept the text input.*
    ```
    void StartInput()
    void StopInput()
    void AcceptInput()
    ```

- ***abstract class* UIState**: Represent a UI state.
    - *Handle inputs, update, and draw the state components to the screen.*

      ```
      void HandleInput()
      void Update()
      void Draw()
      ```
- ***class* MainUIState**: Represent the UI state for the main inventory page.
    - *Initialize all UI Components in the state.*

      ```
      MainUIState()
      ```
    - *Override HandleInput, Update, and Draw methods to manage all UI Components.*
- ***class* ProductUIState**: Represent the UI state for the product details page.
    - *Initialize all UI Components in the state based on the information of the given editing product.*

      ```
      ProductUIState(Product? editingProduct = null,
                     Product? originProduct = null)
      ```
    - *Override HandleInput, Update, and Draw methods to manage all UI Components.*

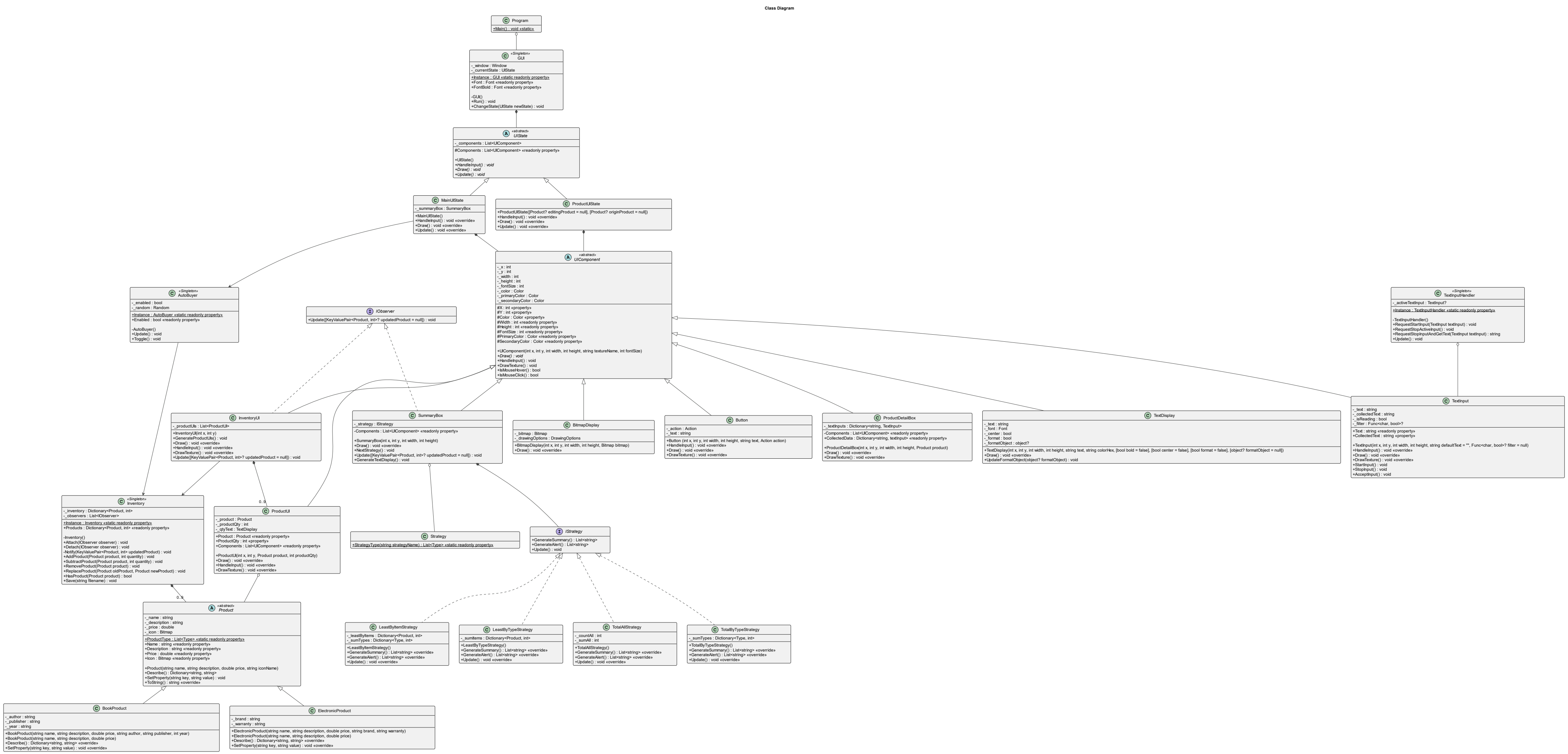
- ***interface* IStrategy**: interface for Strategy classes.
    - *Return necessary information about the Inventory.*

      ```
      List<string> GenerateSummary()
      List<string> GenerateAlert()
      ```
    - *Update the information over time.*

      ```
      void Update()
      ```
- ***static class* Strategy**: class to hold a list of strategy types.
    - *The list of strategy types*

      ```
      List<Type> StrategyType { get; }
      ```
- ***class* LeastByItemStrategy**: Strategy to generate a summary of the inventory by the least number of items of each type.
    - *Override Generate methods and Update method to provide suitable information.*
- ***class* LeastByTypeStrategy**: Strategy to generate a summary of the inventory by product type with the least quantity.
    - *Override Generate methods and Update method to provide suitable information.*
- ***class* TotalAllStrategy**: Strategy to generate summary and alert on total of all products.
    - *Override Generate methods and Update method to provide suitable information.*
- ***class* TotalByTypeStrategy**: Strategy to generate a summary of the total number of products by type.
    - *Override Generate methods and Update method to provide suitable information.*

Class Diagram


Class Diagram
Program
+Main() : void «static»
«Singleton»
GUI
-_window : Window
-_currentState : UIState
+Instance : GUI «static readonly property»
+Font : Font «readonly property»
+FontBold : Font «readonly property»
-GUI()
+Run() : void
+ChangeState(UIState newState) : void
«abstract»
UIState
-_components : List<UIComponent>
#Components : List<UIComponent> «readonly property»
+UIState()
+HandleInput() : void
+Draw() : void
+Update() : void
MainUIState
-_summaryBox : SummaryBox
+MainUIState()
+HandleInput() : void «override»
+Draw() : void «override»
+Update() : void «override»
ProductUIState
+ProductUIState([Product? editingProduct = null], [Product? originProduct = null])
+HandleInput() : void «override»
+Draw() : void «override»
+Update() : void «override»
«abstract»
UIComponent
-_x : int
-_y : int
-_width : int
-_height : int
-_fontSize : int
-_color : Color
-_primaryColor : Color
-_secondaryColor : Color
#X : int «property»
#Y : int «property»
#Color : Color «property»
#Width : int «readonly property»
#Height : int «readonly property»
#FontSize : int «readonly property»
#PrimaryColor : Color «readonly property»
#SecondaryColor : Color «readonly property»
+UIComponent(int x, int y, int width, int height, string textureName, int fontSize)
+Draw() : void
+HandleInput() : void
+DrawTexture() : void
+IsMouseHover() : bool
+IsMouseClick() : bool
«Singleton»
AutoBuyer
-_enabled : bool
-_random : Random
+Instance : AutoBuyer «static readonly property»
+Enabled : bool «readonly property»
-AutoBuyer()
+Update() : void
+Toggle() : void
IObserver
+Update([KeyValuePair<Product, int>? updatedProduct = null]) : void
«Singleton»
TextInputHandler
-_activeTextInput : TextInput?
+Instance : TextInputHandler «static readonly property»
-TextInputHandler()
+RequestStartInput(TextInput textInput) : void
+RequestStopActiveInput() : void
+RequestStopInputAndGetText(TextInput textInput) : string
+Update() : void
InventoryUI
-_productUIs : List<ProductUI>
+InventoryUI(int x, int y)
+GenerateProductUIs() : void
+Draw() : void «override»
+HandleInput() : void «override»
+DrawTexture() : void «override»
+Update([KeyValuePair<Product, int>? updatedProduct = null]) : void
SummaryBox
-_strategy : IStrategy
-Components : List<UIComponent> «readonly property»
+SummaryBox(int x, int y, int width, int height)
+Draw() : void «override»
+NextStrategy() : void
+Update([KeyValuePair<Product, int>? updatedProduct = null]) : void
+GenerateTextDisplay() : void
BitmapDisplay
-_bitmap : Bitmap
-_drawingOptions : DrawingOptions
+BitmapDisplay(int x, int y, int width, int height, Bitmap bitmap)
+Draw() : void «override»
Button
-_action : Action
-_text : string
+Button (int x, int y, int width, int height, string text, Action action)
+HandleInput() : void «override»
+Draw() : void «override»
+DrawTexture() : void «override»
ProductDetailBox
-_textInputs : Dictionary<string, TextInput>
-Components : List<UIComponent> «readonly property»
+CollectedData : Dictionary<string, textInput> «readonly property»
+ProductDetailBox(int x, int y, int width, int height, Product product)
+Draw() : void «override»
+DrawTexture() : void «override»
TextDisplay
-_text : string
-_font : Font
-_center : bool
-_format : bool
-_formatObject : object?
+TextDisplay(int x, int y, int width, int height, string text, string colorHex, [bool bold = false], [bool center = false], [bool format = false], [object? formatObject = null])
+Draw() : void «override»
+UpdateFormatObject(object? formatObject) : void
TextInput
-_text : string
-_collectedText : string
-_isReading : bool
-_filter : Func<char, bool>?
+Text : string «readonly property»
+CollectedText : string «property»
+TextInput(int x, int y, int width, int height, string defaultText = "", Func<char, bool>? filter = null)
+HandleInput() : void «override»
+Draw() : void «override»
+DrawTexture() : void «override»
+StartInput() : void
+StopInput() : void
+AcceptInput() : void
«Singleton»
Inventory
-_inventory : Dictionary<Product, int>
-_observers : List<IObserver>
+Instance : Inventory «static readonly property»
+Products : Dictionary<Product, int> «readonly property»
-Inventory()
+Attach(IObserver observer) : void
+Detach(IObserver observer) : void
-Notify(KeyValuePair<Product, int> updatedProduct) : void
+AddProduct(Product product, int quantity) : void
+SubtractProduct(Product product, int quantity) : void
+RemoveProduct(Product product) : void
+ReplaceProduct(Product oldProduct, Product newProduct) : void
+HasProduct(Product product) : bool
+Save(string filename) : void
0..9
ProductUI
-_product : Product
-_productQty : int
-_qtyText : TextDisplay
+Product : Product «readonly property»
+ProductQty : int «property»
+Components : List<UIComponent> «readonly property»
+ProductUI(int x, int y, Product product, int productQty)
+Draw() : void «override»
+HandleInput() : void «override»
+DrawTexture() : void «override»
Strategy
+StrategyType(string strategyName) : List<Type> «static readonly property»
IStrategy
+GenerateSummary() : List<string>
+GenerateAlert() : List<string>
+Update() : void
0..9
«abstract»
Product
-_name : string
-_description : string
-_price : double
-_icon : Bitmap
+ProductType : List<Type> «static readonly property»
+Name : string «readonly property»
+Description : string «readonly property»
+Price : double «readonly property»
+Icon : Bitmap «readonly property»
+Product(string name, string description, double price, string iconName)
+Describe() : Dictionary<string, string>
+SetProperty(string key, string value) : void
+ToString() : string «override»
LeastByItemStrategy
-_leastByItems : Dictionary<Product, int>
-_sumTypes : Dictionary<Type, int>
+LeastByItemStrategy()
+GenerateSummary() : List<string> «override»
+GenerateAlert() : List<string> «override»
+Update() : void «override»
LeastByTypeStrategy
-_sumItems : Dictionary<Product, int>
+LeastByTypeStrategy()
+GenerateSummary() : List<string> «override»
+GenerateAlert() : List<string> «override»
+Update() : void «override»
TotalAllStrategy
-_countAll : int
-_sumAll : int
+TotalAllStrategy()
+GenerateSummary() : List<string> «override»
+GenerateAlert() : List<string> «override»
+Update() : void «override»
TotalByTypeStrategy
-_sumTypes : Dictionary<Type, int>
+TotalByTypeStrategy()
+GenerateSummary() : List<string> «override»
+GenerateAlert() : List<string> «override»
+Update() : void «override»
BookProduct
-_author : string
-_publisher : string
-_year : string
+BookProduct(string name, string description, double price, string author, string publisher, int year)
+BookProduct(string name, string description, double price)
+Describe() : Dictionary<string, string> «override»
+SetProperty(string key, string value) : void «override»
ElectronicProduct
-_brand : string
-_warranty : string
+ElectronicProduct(string name, string description, double price, string brand, string warranty)
+ElectronicProduct(string name, string description, double price)
+Describe() : Dictionary<string, string> «override»
+SetProperty(string key, string value) : void «override»