Inventory Management System Documentation
Programming Project

Mihyar Al Hariri
Berke Palamutcu
Alexandru Belascu

June 11, 2023

Contents

1 Introduction

The Inventory Management System is a software application designed to track and manage the inventories. It provides functionalities to add, update, sorting, and removing items, as well as generate reports on inventory status. This documentation outlines the design, features, and usage of the system.

2 System Architecture

The Inventory Management System is developed in C++ programming language. It follows a modular design with the following components:

- **Inventory Management Module:** Responsible for managing the inventory, including adding, updating, and removing items.
- **Reporting Module:** Generates various reports on the inventory, such as item lists, stock levels, and sales reports.
- **User Interface Module:** Provides an interface for users to interact with the system, including inputting commands and viewing outputs.
- **Data Persistence Module:** Handles the storage and retrieval of data, such as item information and transaction history, using a file-based approach.

3 Features

The Inventory Management System offers the following features:

- **Add Item:** Allows users to add new items to the inventory by providing details such as item name, quantity, and price.
- **Update Item:** Enables users to update the information of existing items, such as modifying the quantity or price.
- **Remove Item:** Provides the functionality to remove items from the inventory based on the item ID.
- **Generate Reports:** Allows users to generate various reports, including the list of all items, items with low stock levels, and sales reports.
- **Save Reports:** Allows users to save reports to txt extended files
- **Load Reports:** Allows users to load reports from txt extended files
- **Sort Reports:** Allows users to sort reports by comparing the items' prices or alphabetical order.

4 Usage

4.1 Installation

To install and run the Inventory Management System, follow these steps:

1. Download the source code from the repository.
2. Compile the source code using a C++ compiler.
3. Please use C++ compiler with C++11 standard at least and more
4. Please make sure that you give access and authorization for the relevant extracted folder to be able to run the program successfully, it may be needed in some cases.
5. Execute the compiled binary file to start the application.

4.2 User Interface

The system provides a command-line interface for users to interact with. Users can input commands and view the outputs in the terminal. The available commands include:

- `add` - Adds a new item to the inventory.
- `update` - Updates the information of an existing item.
- `remove` - Removes an item from the inventory.
- `report` - Generates various reports on the inventory.
- `exit` - Exits the application.

4.3 Example Usage

Here's an example usage scenario:

```
> add
Enter item name: Laptop
Enter quantity: 10
Enter price: 1200

> list

    Item Name      Quantity     Price
-------------------------------------
    Laptop         10           100
```

```
> update
Enter item name: 1
Enter new quantity: 5

Item updated successfully.

> remove
Enter item name: Laptop

Item removed successfully.

> sort_price

    Item Name     Quantity    Price
-------------------------------------
    Laptop         10          1000
     Mouse          2            10
    Keyboard        5             5

sorts the items comparing their prices from high to low

> sort_name

    Item Name     Quantity    Price
-------------------------------------
    Keyboard        5             5
    Laptop         10          1000
     Mouse          2            10

sorts the items comparing them alphabetically

> save

Saves the list to a txt file.

> load

Loads the list contents to the application.

> exit
Exiting the application...
```

5 Architecture Explanation

The application consists of 2 blocks of .cpp extended program and 3 blocks of .h extended header files.

5.1 Handlers Module

The `handlers.h` file contains the declaration of various functions that handle different commands in the inventory management system. This module provides the functionality to add, remove, update, and sort items in the inventory, as well as print inventory information and handle file operations.

5.1.1 Enum: Command

The `Command` enum defines different commands that can be executed in the system. Each command represents a specific action, such as adding an item, removing an item, listing items, etc. The available commands are:

- `ADD` - Add a new item to the inventory.
- `REMOVE` - Remove an item from the inventory.
- `LIST` - Print the inventory information.
- `SAVE` - Save the inventory to a file.
- `LOAD` - Load the inventory from a file.
- `EXIT` - Exit the application.
- `UPDATE` - Update the information of an existing item.
- `SORT_PRICE` - Sort the items by price.
- `SORT_NAME` - Sort the items by name.
- `INVALID` - Invalid command.

5.1.2 Functions

The `handlers.h` file declares several functions that handle different commands:

- `handleAdd(Inventory& inventory)` - Handles the `ADD` command by adding a new item to the inventory.
- `handleRemove(Inventory& inventory)` - Handles the `REMOVE` command by removing an item from the inventory.
- `handleUpdate(Inventory& inventory)` - Handles the `UPDATE` command by updating the information of an existing item.

- `handleSave(Inventory& inventory)` - Handles the `SAVE` command by saving the inventory to a file.
- `handleLoad(Inventory& inventory)` - Handles the `LOAD` command by loading the inventory from a file.
- `handleSort(Inventory& inventory, const std::string& by)` - Handles the `SORT_PRICE` and `SORT_NAME` commands by sorting the items in the inventory based on the given criteria.
- `printInventory(const Inventory& inventory)` - Prints the inventory information.
- `printItems(const std::vector<Item>& items)` - Prints a vector of items.
- `runInventory()` - Runs the inventory management system by accepting user commands and executing the corresponding handlers.

5.1.3 Utility Functions

The `handlers.h` file also declares two utility functions used for item comparison:

- `compareItemsByName(const Item& item1, const Item& item2)` - Compares two items based on their names.
- `compareItemsByPrice(const Item& item1, const Item& item2)` - Compares two items based on their prices.

These utility functions are used by the `handleSort()` function to perform sorting

5.2 Inventory Module

The `inventory.h` file contains the implementation of the `Inventory` class, which is responsible for managing the items in the inventory.

5.2.1 Class: Inventory

The `Inventory` class provides the following member functions:

- `void addItem(const Item& item)` - Adds a new item to the inventory.
- `void removeItem(const std::string& itemName)` - Removes an item from the inventory based on the item name.
- `std::map<std::string, Item> getItems() const` - Returns a map of all items in the inventory.
- `std::vector<Item> sortByPrice() const` - Returns a vector of items sorted by price in ascending order.

- `std::vector<Item> sortByName() const` - Returns a vector of items sorted by name in alphabetical order.
- `bool updateItem(const std::string& itemName, int quantity, double price)` - Updates the quantity and price of an existing item.
- `void saveToFile(const std::string& filename) const` - Saves the inventory to a file.
- `bool loadFromFile(const std::string& filename)` - Loads the inventory from a file.

The `Inventory` class also defines two private utility functions:

- `bool compareItemsByName(const Item& a, const Item& b)` - Compares two items based on their names.
- `bool compareItemsByPrice(const Item& a, const Item& b)` - Compares two items based on their prices.

These utility functions are used for sorting the items in the `sortByPrice()` and `sortByName()` member functions.

5.2.2 Member Functions

- `void addItem(const Item& item)` - Adds a new item to the inventory by inserting it into the `_items` map using the item name as the key.
- `void removeItem(const std::string& itemName)` - Removes an item from the inventory by erasing it from the `_items` map based on the item name.
- `std::map<std::string, Item> getItems() const` - Returns a copy of the `_items` map, which contains all items in the inventory.
- `std::vector<Item> sortByPrice() const` - Creates a vector of items from the `_items` map and sorts it in ascending order based on the item prices using the `compareItemsByPrice` utility function. The sorted vector is then returned.
- `std::vector<Item> sortByName() const` - Creates a vector of items from the `_items` map and sorts it in alphabetical order based on the item names using the `compareItemsByName` utility function. The sorted vector is then returned.
- `bool updateItem(const std::string& itemName, int quantity, double price)` - Updates the quantity and price of an existing item in the inventory. It searches for the item in the `_items` map based on the item name and updates its quantity and price if found. Returns `true` if the item was found and updated, `false` otherwise.

- `void saveToFile(const std::string& filename) const` - Opens a file with the specified `filename` and writes the item details (name, quantity, and price) for each item in the `_items` map to the file. The file is then closed.
- `bool loadFromFile(const std::string& filename)` - Opens a file with the specified `filename` and reads the item details (name, quantity, and price) from the file. It adds each item to the `_items` map. Returns `true` if the file was successfully opened and the items were loaded, `false` otherwise.

5.2.3 Private Utility Functions

- `bool compareItemsByName(const Item& a, const Item& b)` - Compares two items based on their names. This function is used by the `sortByName()` member function for sorting the items by name.
- `bool compareItemsByPrice(const Item& a, const Item& b)` - Compares two items based on their prices. This function is used by the `sortByPrice()` member function for sorting the items by price.

These utility functions provide the comparison logic for the sorting operations in the `Inventory` class.

5.3 Item Module

The `item.h` file contains the implementation of the `Item` class, which represents an item in the inventory.

5.3.1 Class: Item

The `Item` class provides the following member functions:

- `Item(const std::string& name = "", int quantity = 0, double price = 0.0)` - Constructs a new `Item` object with the specified `name`, `quantity`, and `price` values. The default values are an empty string for `name`, 0 for `quantity`, and 0.0 for `price`.
- `std::string getName() const` - Returns the name of the item.
- `int getQuantity() const` - Returns the quantity of the item.
- `double getPrice() const` - Returns the price of the item.
- `void setQuantity(int quantity)` - Sets the quantity of the item to the specified `quantity` value.
- `void setPrice(double price)` - Sets the price of the item to the specified `price` value.

5.3.2 Member Functions

- `Item(const std::string& name = "", int quantity = 0, double price = 0.0)` - Constructs a new `Item` object with the specified `name`, `quantity`, and `price` values. The default values are an empty string for `name`, 0 for `quantity`, and 0.0 for `price`.
- `std::string getName() const` - Returns the name of the item.
- `int getQuantity() const` - Returns the quantity of the item.
- `double getPrice() const` - Returns the price of the item.
- `void setQuantity(int quantity)` - Sets the quantity of the item to the specified `quantity` value.
- `void setPrice(double price)` - Sets the price of the item to the specified `price` value.

These member functions provide access to the attributes of the `Item` class, such as the name, quantity, and price.

5.3.3 Attributes

- `std::string _name` - The name of the item.
- `int _quantity` - The quantity of the item.
- `double _price` - The price of the item.

These attributes store the information related to the item.

5.4 Handlers Module

The `handlers.cpp` file contains the implementation of various functions that handle different commands and operations in the Inventory Management System.

5.4.1 Function: resolveCommand

The `resolveCommand` function takes a command string as input and returns the corresponding `Command` enum value. It compares the command string with predefined command values and returns the appropriate enum value. The available commands include `add`, `remove`, `list`, `save`, `load`, `exit`, `update`, `sort_price`, `sort_name`, and `invalid`.

5.4.2 Function: printInventory

The `printInventory` function takes an `Inventory` object as input and prints the details of each item in the inventory. It retrieves the items using the `getItems` function of the `Inventory` class and iterates over the items, printing their name, quantity, and price.

5.4.3 Function: printItems

The `printItems` function takes a vector of `Item` objects as input and prints the details of each item. It iterates over the items in the vector and prints their name, quantity, and price.

5.4.4 Function: handleAdd

The `handleAdd` function takes an `Inventory` object as input and handles the `add` command. It prompts the user to enter the item name, quantity, and price, and then adds the item to the inventory using the `addItem` function of the `Inventory` class.

5.4.5 Function: handleRemove

The `handleRemove` function takes an `Inventory` object as input and handles the `remove` command. It prompts the user to enter the item name and then removes the item from the inventory using the `removeItem` function of the `Inventory` class.

5.4.6 Function: handleUpdate

The `handleUpdate` function takes an `Inventory` object as input and handles the `update` command. It prompts the user to enter the item name, new quantity, and new price, and then updates the item in the inventory using the `updateItem` function of the `Inventory` class.

5.4.7 Function: handleSave

The `handleSave` function takes an `Inventory` object as input and handles the `save` command. It prompts the user to enter the filename to save the inventory to, and then saves the inventory to a file using the `saveToFile` function of the `Inventory` class.

5.4.8 Function: handleLoad

The `handleLoad` function takes an `Inventory` object as input and handles the `load` command. It prompts the user to enter the filename to load the inventory from, and then loads the inventory from a file using the `loadFromFile` function of the `Inventory` class.

5.4.9 Function: handleSort

The `handleSort` function takes an `Inventory` object and a sort type string as input and handles the `sort_price` and `sort_name` commands. It checks the sort type and calls the `sortByPrice` or `sortByName` function of the `Inventory` class to obtain a sorted vector of items. It then calls the `printItems` function to print the sorted items.

5.4.10 Function: runInventory

The `runInventory` function is the main function that runs the Inventory Management System. It creates an `Inventory` object and a `running` flag to control the execution of the system. Inside a `while` loop, it prompts the user to enter a command and uses a `switch` statement to execute the corresponding function based on the resolved command value. The loop continues until the user enters the `exit` command.

These functions work together to provide the command handling and interaction with the inventory system.

5.5 Main Program

The `main.cpp` file contains the entry point of the Inventory Management System program. It includes the necessary header files and defines the `main` function.

5.5.1 Function: main

The `main` function is the entry point of the program. It creates an `Inventory` object and then calls the `runInventory` function to start the inventory management system. Finally, it returns 0 to indicate successful program execution.

This function is responsible for starting the program and initiating the user interaction with the inventory management system.

The code snippet in the `main` function demonstrates how to run the inventory management system by calling the `runInventory` function.

```
int main() {
    runInventory();
    return 0;
}
```

By executing the `runInventory` function, the program will prompt the user to enter commands and perform various operations on the inventory, such as adding, updating, removing items, generating reports, and more.

The main program acts as the driver for the entire inventory management system, providing a user-friendly interface for interacting with the system and performing inventory-related tasks.

6 Conclusion

The system's user-friendly command-line interface allows users to interact with the system easily. Users can input commands and view the outputs directly in the terminal, making it intuitive and straightforward to perform inventory-related tasks.

One of the key advantages of the Inventory Management System is its ability to save and load inventory data from files. This feature ensures data persistence and allows users to store and retrieve inventory information conveniently. It also enables users to share inventory data across different sessions or with other team members.

The system's sorting functionality is another valuable feature. Users can sort the inventory items based on prices or alphabetical order, providing a clear view of the inventory and facilitating efficient decision-making.