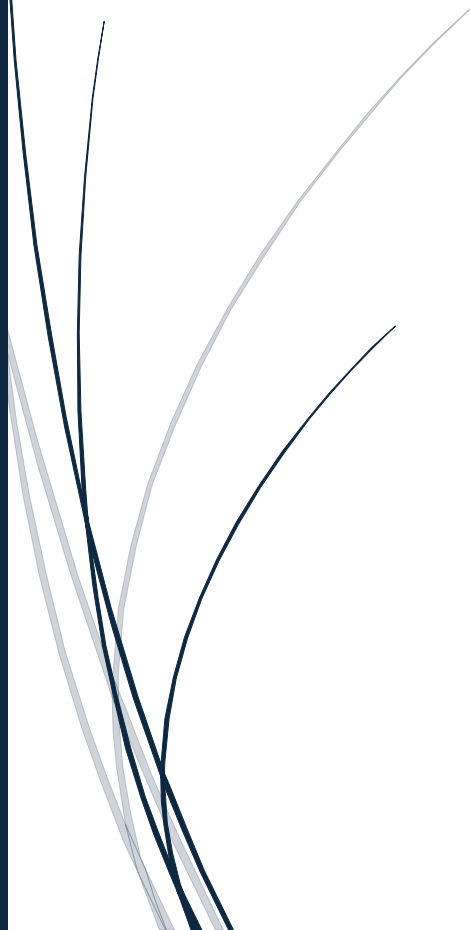




4/18/2024

Supply Chain System

By Unknown Systems



Fabian Galasel – M00860030 (Project Manager)

Ovie Golde – M00851996 (Developer)

Jagjit Bhogal – M00812982 (Developer)

Anikhil Patel – M00861390 (Tester)

Saeed Nawaz – M00862519 (Secretary)

Contents

Introduction – brief description of the project

Design – justification of data structures and algorithms

Testing – testing approach used and test cases

Conclusion – summary of work done

References – references used

Introduction

In this report, I will be explaining how we have achieved the software, designed it, tested it, in between 5 people.

In the project, we had to create a software for supply chain system. The main requirements were:

- File reading.
- Order creating.
- Order fulfilment.
- Order status tracking
- Cancelling orders.
- Displaying stock.
- Exiting the software.

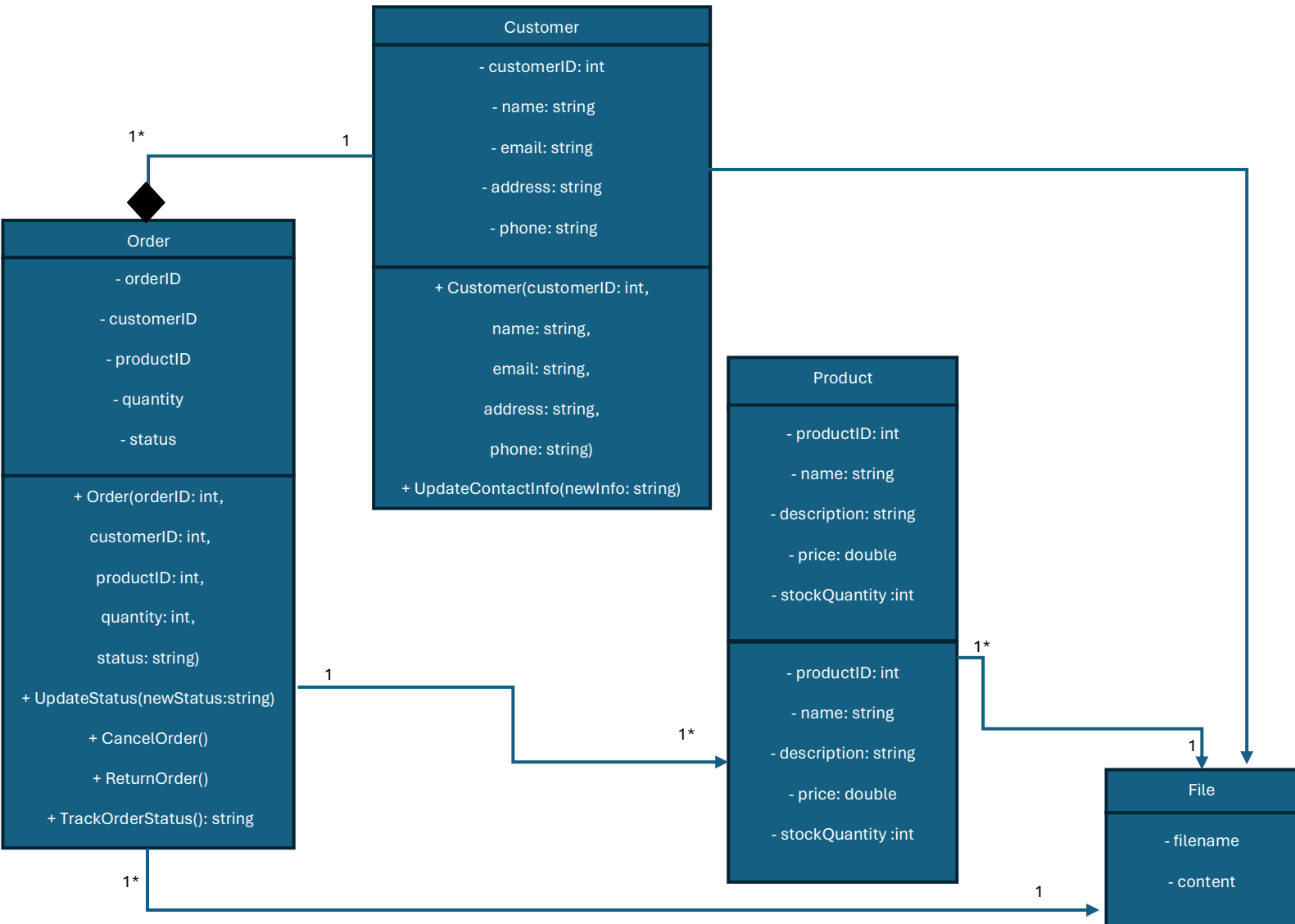
We were tasked with developing a software for a supply chain system. So we developed a software for a game-selling chain, which sells games to other shops and restores their stock.

We had to use file reading, order management, order cancellation, and status tracking.

To meet the requirements, our team collaborated to design and implement a robust software solution that effectively manages the entire supply chain process. Each team member contributed their expertise in different areas, such as file handling, order management, and user interface design, to ensure the successful development of the software.

Design

UML Diagrams



Justification of selected data structure and algorithms:

The selection of appropriate data structures and algorithms was crucial for the efficient functioning of any software system. In our supply chain management software, we carefully considered the following justifications for the chosen data structures and algorithms:

1. **Vector for Orders and Products:** We chose the `std::vector` data structure to store orders and products because it provides dynamic resizing, efficient random access, and straightforward iteration. These characteristics are essential for managing a variable number of orders and products efficiently.

2. Hash Table for Customer Information: We utilized a hash table (e.g., `std::unordered_map`) to store customer information, such as customer IDs and names. Hash tables offer constant-time average complexity for insertion, deletion, and retrieval operations, making them suitable for quick access to customer data.
3. String for Product and Order Status: String data types were employed to represent product names and order statuses. Strings provide flexibility in handling variable-length textual data and support various operations required for managing product names and order statuses effectively.
4. Linear Search Algorithm for Order Status Update: Since the order status update operation involves iterating through the orders vector to find the specific order by its ID, a linear search algorithm was chosen. While not the most efficient for large datasets, it offers simplicity and adequacy for our current requirements.
5. Remove-If Algorithm for Order Deletion: When removing an order after it has been delivered, the `std::remove_if` algorithm combined with the `erase` method of vectors provides an efficient way to delete elements that meet a certain condition. This algorithm offers linear time complexity and avoids unnecessary shifting of elements.

Analysis of key functionality:

1. AddOrder Function:

- Adds an order to the `orderList`.
- Constant time complexity.

2. RemoveOrder Function:

- Removes an order by ID from the `orderList`.
- Linear time complexity (typically efficient).

3. SearchProductByID Function:

- Retrieves a product by ID from the `productMap`.
- Constant time complexity (efficient hash table lookup).

4. SearchCustomerByID Function:

- Retrieves a customer by ID from the `customerMap`.
- Constant time complexity (efficient hash table lookup).

5. ReadSampleDataFile Function:

- Reads and parses data from a file, creates objects, and adds them to data structures.
- Linear time complexity relative to the file size.

6. Main Function:

- Calls ReadSampleDataFile and tests removal of an order.
- Complexity depends on ReadSampleDataFile and RemoveOrder functions, both typically linear.

Testing

We decided to choose unit testing because it provides a reliable approach for making sure code is accurate and robust. We are able to identify and fix possible problems early in the development process by thoroughly testing each part separately. Essentially, unit testing gives us the ability to provide our users with reliable, error-free code, which promotes confidence in our services.

Test Cases based on Classes and Design:

Test Case ID	Test Case Description	Prerequisites	Steps	Expected Result
TC01	Creating a Customer	None	1. Create a customer with valid details.	Customer object is created successfully.
			2. Check if the customer object is successfully created.	
TC02	Updating Contact Info	A valid customer exists.	1. Update the customer's email address.	Email address is updated successfully.
TC03	Creating an Order	Valid customer and product IDs available.	1. Create an order with valid customer and product IDs.	Order is created successfully.

TC04	Updating Order Status	An existing order with a valid status.	1. Change the order status (e.g., from “Processing” to “Shipped”).	Order status is updated successfully.
TC05	Tracking Order	An existing order.	1. Retrieve the status of an existing order.	Order status is accurately tracked.
TC06	Creating a Product	None	1. Create a product with valid details (e.g., name, description, price).	Product is created successfully.
TC07	File Handling (Assumption)	None	1. Create a new file.	File operations are performed correctly.
			2. Read data from the file.	
			3. Update the file content.	
			4. Delete the file.	

Test Cases for Software:

Class/Function Test	Test Case Description	Prerequisites	Expected Result
CustomerInfo class test	- Verify getters return correct values	Customer object exists	CustomerInfo object returns correct customer information
	- Verify UpdateName updates the name correctly	Customer object exists	Customer name is successfully updated
readGamesFromTXT function test	- Verify correct number of games read from file	games.txt file exists	Correct number of games is read from the file

	- Verify correct game information is read	games.txt file exists	Game information is read accurately from the file
OrderItem class test	- Verify getters return correct values	Valid customer and product IDs	OrderItem object returns correct order information
	- Verify UpdateStatus updates the status correctly	OrderItem object exists	Order status is updated successfully
ProductItem class test	- Verify getters return correct values	ProductItem object exists	ProductItem object returns correct product information
	- Verify UpdateStock updates stock quantity correctly	ProductItem object exists	Stock quantity is updated correctly
	- Verify AddStock increases stock quantity correctly	ProductItem object exists	Stock quantity is increased correctly
	- Verify RemoveStock decreases stock quantity correctly	ProductItem object exists	Stock quantity is decreased correctly

Class/Function Test	Test Case Description	Prerequisites	Expected Result
---------------------	-----------------------	---------------	-----------------

updateOrderStatus function test	- Verify order status is updated correctly	Existing orders	Order status is updated accurately
	- Verify order is removed if status is 'Delivered'	Existing orders	Order is successfully removed if status is 'Delivered'
displayMenu function test	- Verify menu display function	None	Menu is displayed correctly
displayGames function test	- Verify games display function	Existing games	Games are displayed correctly in the expected format
getIntegerInput function test	- Verify integer input function	None	Correct integer input is received and processed

Conclusion

In summary, we successfully developed a supply chain management software tailored for a game-selling chain, encompassing essential functionalities such as order creation, fulfilment, cancellation, status tracking, and stock management. Through collaborative effort, we designed and implemented classes for orders, products, and customers, utilizing data structures like lists and maps for efficient storage and retrieval of information.

Additionally, we incorporated file reading capabilities to facilitate the import of sample data.

Limitations and Critical Reflection:

Despite our achievements, several limitations and areas for improvement exist. One notable limitation was the lack of extensive DevOps practices, resulting in fewer meetings and potentially hindering communication and coordination among team

members. Additionally, while we successfully implemented core functionalities, the software may lack certain advanced features or optimizations, potentially limiting its scalability and performance in larger-scale scenarios. Furthermore, the absence of comprehensive testing procedures may leave the software vulnerable to bugs or errors, necessitating a more robust testing framework.

Another thing we could have done is to add more games to the games.txt file, and to make the software write to it, so that the stock could actually be modified depending on user input in the user interface.

How Would You Change Your Approach on a Similar Task in the Future:

In future projects of similar tasks, we would adopt a more comprehensive DevOps approach, emphasizing frequent communication, iteration, and collaboration among team members. This would involve regular meetings to discuss progress, address challenges, and plan next steps, fostering a more cohesive and efficient development process.

Additionally, we would prioritize the implementation of automated testing procedures, including unit tests, integration tests, and end-to-end tests, to ensure the reliability, stability, and quality of the software. Furthermore, we would explore opportunities for incorporating additional features and optimizations to enhance the software's functionality, scalability, and user experience, leveraging best practices and emerging technologies in software development.

References

(No date) C++ classes and objects. Available at:
https://www.w3schools.com/cpp/cpp_classes.asp (Accessed: 18 April 2024).

Reading files in C++, Stack Overflow. Available at:
<https://stackoverflow.com/questions/49395141/reading-files-in-c> (Accessed: 18 April 2024).

Creating an inventory using classes, Code Review Stack Exchange. Available at:
<https://codereview.stackexchange.com/questions/142394/creating-an-inventory-using-classes> (Accessed: 18 April 2024).