EX NO:1	Identify a software system that needs to be developed- STOCK MAINTENANCE SYSTEM
DATE:	

AIM:

To identify a software system that needs to be developed – Stock Maintenance System.

INTRODUCTION:

In the dynamic landscape of business operations, effective management of stocks and inventory is crucial for ensuring smooth operations and meeting customer demands. The Stock Maintenance System is a comprehensive software solution designed to streamline the management of stocks and inventory for businesses of all sizes.

FEATURES OF STOCK MAINTENANCE SYSTEM:

Stock Inventory Management:

- **Add/Update/Delete Stock Items:** Users can add new stock items to the inventory with details such as name, description, quantity, price, and location. They should also be able to update or delete existing stock items as needed.
- **Track Stock Levels:** The system allows users to track the current stock levels of each item in real-time, providing visibility into available stock quantities.
- **Categorization and Tagging:** Users can categorize stock items into different categories or groups for better organization. Additionally, they can assign tags or labels to items for easy identification.

Order Management:

- **Create Orders:** Users can create new orders, specifying the items, quantities, and delivery details. The system should support various order types, such as sales orders, purchase orders, and transfer orders.
- **Order Processing:** The system facilitates order processing, including order verification, payment processing, and order confirmation.
- **Order Tracking:** Users can track the status of orders from creation to fulfillment, including order picking, packing, shipping, and delivery.

- **Returns and Refunds:** The system allows users to manage returns and process refunds for orders, handling return authorization, restocking, and refund processing.

Reporting and Analytics:

- **Generate Reports:** The system provides predefined and customizable reports on various aspects of stock management, including stock levels, sales performance, order history, and inventory turnover.
- **Analytics Tools:** Users can analyze stock and sales data using built-in analytical tools, enabling them to identify trends, patterns, and opportunities for optimization.
- **Forecasting:** The system may include forecasting capabilities to predict future stock requirements based on historical data and demand patterns.

User Management:

- **User Roles and Permissions:** Administrators can define user roles and permissions to control access to system functionalities. Common roles may include administrators, stock managers, order processors, and reporting analysts.
- **User Authentication:** The system requires users to authenticate their identity before accessing sensitive functionalities or data, ensuring security and accountability.
- **Audit Trail:** The system maintains an audit trail of user actions and system activities, providing a record of who did what and when for accountability and compliance purposes.

Integration and Compatibility:

- **Integration with ERP/CRM Systems:** The system integrates seamlessly with other business systems such as Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems for data exchange and process automation.
- **Compatibility with Devices:** The system is compatible with various devices and platforms, including desktop computers, laptops, tablets, and mobile phones, allowing users to access the system from anywhere, anytime.

Security and Compliance:

- **Data Encryption:** The system encrypts sensitive data such as customer information, financial transactions, and inventory details to protect against unauthorized access and data breaches.

- **Access Control:** Role-based access control (RBAC) mechanisms ensure that users can only access the functionalities and data relevant to their roles and responsibilities.
- **Compliance:** The system complies with relevant industry standards and regulations, such as GDPR, HIPAA, or PCI-DSS, to ensure data privacy, security, and regulatory compliance.

RESULT:

Thus, the software system that is need to be developed for Stock Maintenance System was executed successfully.

EX NO:2	DOCUMENT THE SOFTWARE REQUIREMENTS SPECIFICATION (SRS) FOR THE IDENTIFIED SYSTEM
DATE:	

AIM:

To create a system to perform the Stock maintenance.

(I)PROBLEM STATEMENT

The stock maintenance system must take care of sales information of the company and must analyze the potential of the trade. It maintains the number of items that are added or removed. The sales person initiates this Use case. The sales person is allowed to update information and view the database.

(II) SOFTWARE REQUIREMENT SPECIFICATION

1.0 INTRODUCTION

Stock maintenance is an interface between the customer and the sales person. It aims at improving the efficiency in maintaining the stocks.

1.1 PURPOSE

The entire process of Stock maintenance is done in a manual manner Considering the fact that the number of customers for purchase is increasing every year, a maintenance system is essential to meet the demand. So, this system uses several programming and database techniques to elucidate the work involved in this process.

1.2 SCOPE

- The System provides an interface to the customer where they can fill in orders for the item needed.
- The sales person is concerned with the issue of items and can use this system.
- Provide a communication platform between the customer and the sales person.

1.3 DEFINITIONS, ACRONYMS AND THE ABBREVIATIONS

- Market Data provider: One who analyze the product and distribute the news.
- Customer: One who takes order of product
- Sales person: One who maintains the stock detail

1.4 REFERENCES:

IEEE Software Requirement Specification format.

1.5 TECHNOLOGIES TO BE USED:

- Visual Studio
- VB Script

1.6 TOOLS TO BE USED:

- Eclipse IDE (Integrated Development Environment)
- Rational Rose tool (for developing UML Patterns)

1.7 OVERVIEW:

SRS includes two sections overall description and specific requirements Overall Description will describe major role of the system components and interConnections Specific Requirements will describe roles & functions of the actors.

2.0 OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE:

The Stock maintenance acts as an interface between the 'customer' and the 'sales person'. This system tries to make the interface as simple as possible and at the same time not risking the work of data stored in

2.2 SYSTEM FUNCTIONS:

- Secure order of information by the customer
- Schedule the customer an appointment for manual delivery of the product.

2.3 USER CHARACTERISTICS:

- Customer: The person who orders for the item.
- Validate customer: The items ordered by the customer are validated.
- Sales Detail: Maintains the stock details after delivering the items to the customer.

2.4 CONSTRAINTS:

- The customer should wait until the trade contractor and other to analyse the product.
- After the distribution of the news about the product. The customer can take order and request of sales person to fill it.
- Finally, the sales person delivers the order.

2.5 SYSTEM REQUIREMENTS:

1) Hardware Requirements:

Processor: Intel Core i5 or equivalent.

RAM: 8GB or higher.

Storage: Minimum 100GB SSD.

Network Interface Card (NIC): Ethernet or Wi-Fi for network connectivity.

Display: Minimum 1366x768 resolution.

2) Software Requirements:

Operating System: Windows 10, macOS, or Linux.

Web Browser: Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge.

Database Management System: MySQL, PostgreSQL, or Oracle.

Development Framework: Node.js, Django, Ruby on Rails, or .NET Core.

Web Server: Apache HTTP Server, Nginx, or Microsoft IIS.

Version Control: Git or Subversion.

2.6 FUNCTIONAL REQUIREMENTS:

Stock Inventory Management:

- The system should allow users to add new stock items with details such as name, description, quantity, and location.
- Users should be able to update existing stock item information, including quantity adjustments and location changes.

Order Management:

- The system should facilitate the creation of new orders, enabling users to specify the items, quantities, and delivery details.
- Users should be able to track the status of orders from processing to shipment and delivery.

Reporting and Analytics:

- The system should generate reports on various aspects of stock management, including stock levels, sales performance, order history, and inventory turnover.
- Users should be able to customize report parameters such as date range, product categories, and sales channels.

User Management:

- The system should support multiple user roles with different levels of access and permissions, such as administrators, stock managers, and order processors.
- Administrators should be able to create and manage user accounts, assign roles, and reset passwords as needed.

2.7 NON-FUNCTIONAL REQUIREMENTS:

Performance:

- The system should respond to user actions promptly, with minimal latency for loading pages and processing requests.
- It should be capable of handling a large volume of concurrent user sessions without significant degradation in performance.

Security:

- The system should implement robust security measures to protect sensitive data, including stock information, customer details, and financial transactions.
- It should utilize encryption techniques to secure data transmission over the network and protect against unauthorized access.

Reliability:

- The system should operate reliably under normal conditions, with high availability and minimal downtime for maintenance or updates.
- It should have mechanisms in place to handle errors gracefully, providing informative error messages and ensuring data integrity.

Usability:

- The system should have an intuitive and user-friendly interface, with clear navigation paths and informative prompts to guide users through various functionalities.
- It should be accessible across different devices and screen sizes, supporting responsive design principles for optimal user experience.

Scalability:

- The system should be scalable to accommodate growth in data volume, user base, and transactional load over time.
- It should be able to scale horizontally by adding more server resources or vertically by upgrading hardware components as needed.

EX NO:3	TO IDENTIFY THE USE CASES AND DEVELOP THE USE CASE MODEL
DATE:	

AIM:

To identify the use cases and develop use case model

INTRODUCTION OF USE CASE DIAGRAM:

A use case diagram is a type of behavioural diagram in the Unified Modelling Language (UML) that represents the interactions between a system and its actors (users or external systems) in terms of the system's behaviour. It provides a graphical overview of the functionalities provided by a system and the actors involved in those functionalities. Use case diagrams are widely used in software engineering to capture the requirements of a system and to visualize the high-level system structure.

PURPOSE OF USE CASE DIAGRAM:

The purpose of a use case diagram is to visually represent the functional requirements of a system from the perspective of its users or external systems. It helps in understanding the system's behaviour by illustrating how users interact with the system to accomplish specific tasks or goals. Use case diagrams facilitate communication between stakeholders, including developers, designers, and clients, by providing a clear and concise representation of the system's functionality.

COMPONENTS OF USE CASE DIAGRAM:

USE CASES:

Use cases represent the functionalities or services provided by the system. Each use case describes a specific interaction between an actor and the system to achieve a particular goal. In the Stock maintenance system, examples of use cases include "product details “,"purchase details", "sales details", "stock details", "purchase the product", and “supply the product”.

RELATIONSHIPS:

Relationships, such as associations and dependencies, depict the connections between actors and use cases. Associations represent the interaction between actors and use cases, while dependencies represent the reliance of one-use case on another.

ACTORS:

The actors used in this system are

1. **Customer:** The person who orders for the item.

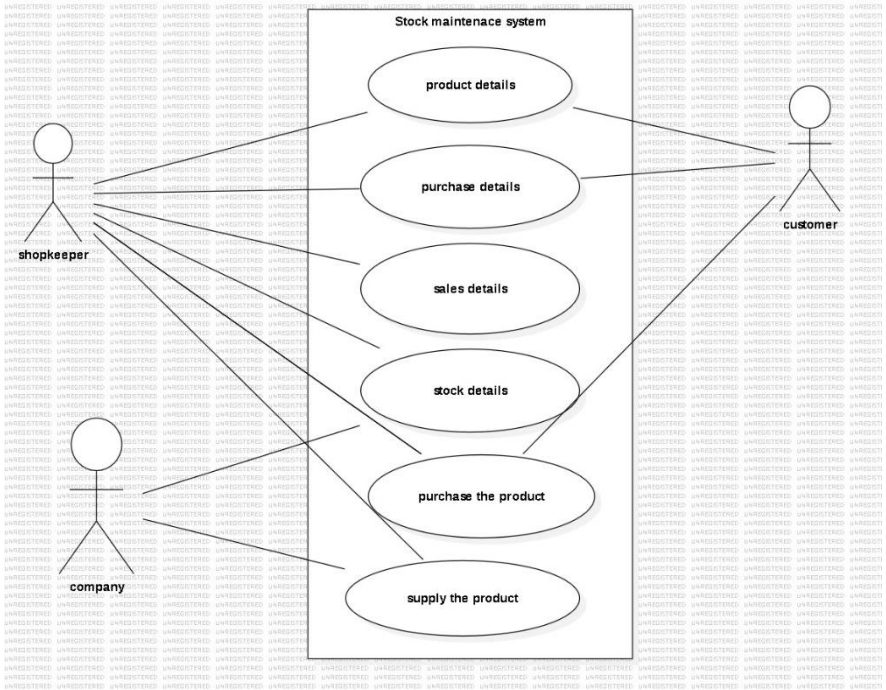
2. **Shopkeeper:** The items ordered by the customer are validated.

3. **Company:** Maintains the stock details after delivering the items to the customer.

BENEFITS OF USE CASE DIAGRAM:

- **Requirements Elicitation**: Use case diagrams help in eliciting and defining the functional requirements of a system by identifying the interactions between users and the system.
- **Visualization**: Use case diagrams provide a visual representation of the system's functionality, making it easier for stakeholders to understand and analyze the system's behaviour.
- **Communication**: Use case diagrams serve as a communication tool between stakeholders, including developers, designers, and clients, by providing a common understanding of the system's requirements and behaviour.
- **Requirements Verification**: Use case diagrams can be used to verify whether all the necessary functionalities and interactions have been captured and addressed in the system design.

USE CASES OF STOCK MAINTENANCE SYSTEM:



FUNCTIONALITY OF USE CASE DIAGRAM:

The functionality of a system can be described in a number of different use-cases, each of which represents a specific flow of events in a system. It is a graph of actors, a set of use-cases enclosed in a boundary, communication, associations between the actors and the use-cases, and generalization among the use-cases.

The use cases used in this system are

Product details:

- The system should provide comprehensive details about each stock item, including its name, description, category, quantity available, location in the warehouse, unit price, and any additional attributes relevant to the product.
- Users should be able to easily access this information through the system's user interface, allowing for informed decision-making regarding stock management and order processing.

Purchase details:

- Users can view the purchase history to track previous purchases made for stock items.
- Users can add a new entry to record purchases made for restocking inventory.
- Users can track the status of purchase orders placed with vendors to replenish stock inventory.

Sales details:

- Users can view a comprehensive history of sales transactions to track past sales activities and analyze sales performance over time.
- Users can generate detailed reports on sales activities to gain insights into sales trends, patterns, and performance metrics.
- Users can track real-time sales performance metrics to monitor sales progress, identify sales trends, and make informed business decisions.

Stock details:

- Users can view comprehensive details of each stock item stored in the inventory.
- Authorized users can update the details of stock items as needed to reflect changes in inventory status or product information.
- Users can track the movement and history of stock items within the inventory over time.

Purchase the product:

- After finalizing the product selection and reviewing the order details, the user proceeds to select a preferred payment method to complete the purchase transaction.
- After selecting the payment method and providing necessary details, the user confirms the purchase to initiate the payment process and finalize the transaction.
- Upon successful completion of the purchase transaction, the user receives an order confirmation notification or email containing details of the purchased product(s) and transaction summary.

Supply the product:

- Before supplying the product, the system checks the availability of the requested product in the inventory.
- After confirming the availability of the product, the system updates the stock levels to reflect the supplied quantity.
- The system generates documentation to record details of the supplied product, including supplier information, quantity supplied, and delivery date.

RESULT:

Thus, the use case diagram for Stock maintenance system is implemented and executed successfully.

EX NO:4	IDENTIFY THE CONCEPTUAL CLASSES AND DEVELOP A DOMAIN MODEL AND DERIVE CLASS DIAGRAM FOR STOCK MAINTENANCE SYSTEM.
DATE:	

AIM:

To identify the conceptual classes and develop a domain model and derive class diagram for Stock maintenance system.

CONCEPTUAL CLASSES:

Product:

It represents the individual products managed in the stock maintenance system and the attributes are Product ID, Name, Description, Quantity, Price, Category.

Supplier:

It represents the suppliers from whom the organization procures stock items and the attributes are Supplier ID, Name, Contact Information, Address.

Stock Item:

It represents the specific instances of products available in the stock inventory and the attributes are Stock Item ID, Product ID, Supplier ID, Quantity, Location.

Order:

It represents the orders placed by the organization to replenish stock inventory and the attributes are Order ID, Date, Supplier ID, Status, Total Amount.

Customer:

It represents the customers who purchase products from the organization and the attributes are Customer ID, Name, Contact Information, Address.

SalesTransaction:

It represents the transactions generated for sales made to customers and the attributes are Transaction ID, Customer ID, Date, Total Amount, Payment Status.

InventoryLocation:

It represents the physical locations within the organization's premises where stock items are stored and the attributes are Location ID, Name, Description.

Category:

It represents the categories or classifications of products for organization and management purposes and the attributes are Category ID, Name, Description.

Shipment:

It represents the shipments received from suppliers containing ordered stock items and the attributes are Shipment ID, Order ID, Supplier ID, Date, Delivery Status.

ReturnTransaction:

It represents the transactions initiated for returning defective or unwanted stock items by customers and the attributes are Return ID, Order ID, Customer ID, Date, Reason.

DOMAIN MODEL FOR STOCK MAINTENACE SYSTEM:

ProductDetails:

Attributes:

- prodCode: integer
- prodName: string
- prodQty: integer
- prodPrice: float

Operations:

- prodAdd()
- prodDelete()
- prodUpdate()
- prodDetails()

purchaseDetails:

Attributes:

- purchCode: integer
- purcDate: date
- subid: integer
- subname: string
- purcQty: integer
- purcPrice: float

Operations:

- save()
- delete()
- purchaseedit()
- purchaseDetails()

salesDetails:

Attributes:

- salid: integer
- salDate: date

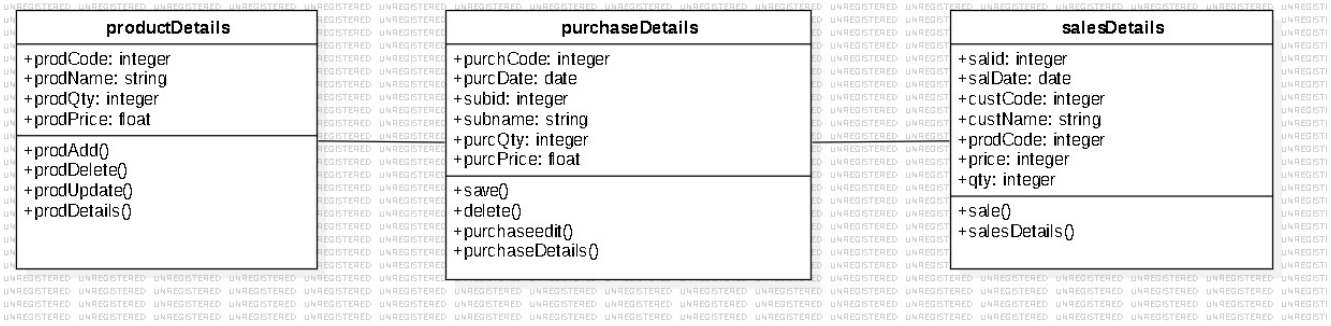
- custCode: integer
- custName: string
- prodCode: integer
- price: integer
- qty: integer

Operations:

- sale()
- salesDetails()

This domain model provides a high-level overview of the main entities and their relationships within the Stock Maintenance System

CLASS DIAGRAM FOR STOCK MAINTENANCE SYSTEM:



RESULT:

Thus, the class diagram for Stock maintenance system is implemented and executed Successfully.

EX NO:5	USING THE IDENTIFIED SCENARIOS, FIND THE INTERACTION BETWEEN OBJECTS AND REPRESENT THEM USING UML SEQUENCE AND COLLABORATION DIAGRAM
DATE:	

AIM:

To find the interaction between objects and represent them using UML Sequence and Collaboration diagram.

SEQUENCE DIAGRAM:

An object is shows as a box and the top of dash vertical line. This vertical line is called object life line. The life line represents objects life during interaction this was given by "Jackupson". Each message is represented by an arrow between the lifeline of Two object.

The order of message is occurred from top to bottom of a page. Message contain Message's name, argument and some control information.

Self-call is a message that an object sends to itself by sending messages arrow back to the same lifeline.

A sequence diagram illustrates a kind of format in which each object interacts via message. It is generalized between two or more specialized diagram.

This interactive behaviour is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

The purposes of interaction diagrams are to visualize the interactive behaviour of the system. Now visualizing interaction is a difficult task. So, the solution is to use different types of models to capture the different aspects of the interaction, that is why sequence and collaboration diagrams are used to capture dynamic nature but from a different angle.

The purposes of interaction diagram can be describing as:

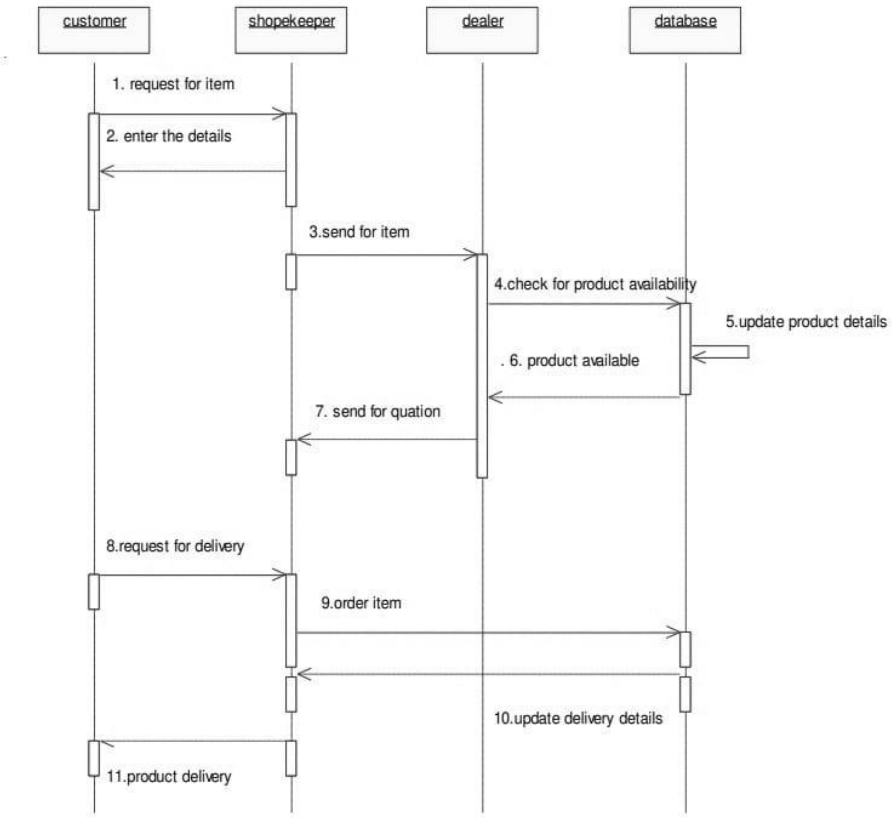
i) To capture dynamic behaviour of a system.

ii) To describe the message flow in the system.

iii) To describe structural organization of the objects.

iv) To describe interaction among objects.

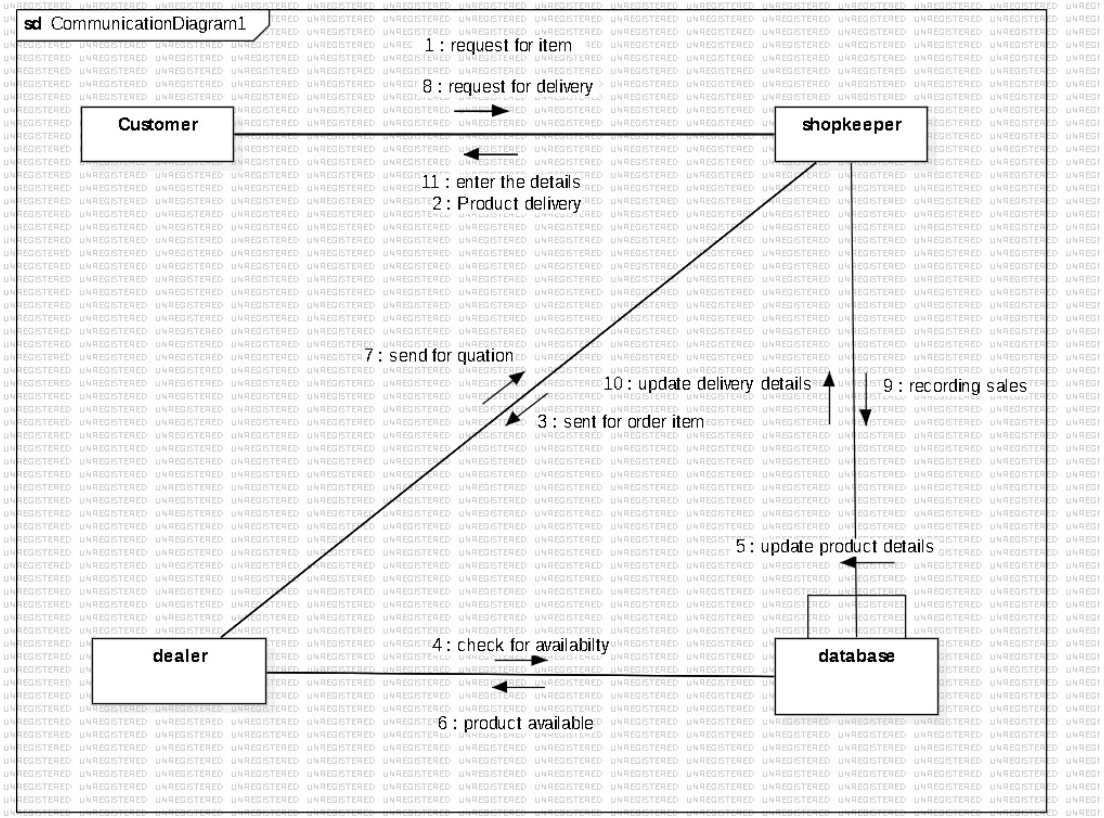
SEQUENCE DIAGRAM FOR STOCK MAINTENANCE SYSTEM:



COLLABORATION DIAGRAM:

- Communication diagram illustrate that object interact on a graph or network format in which object can be placed where on the diagram.
- In collaboration diagram the object can be placed in anywhere on the diagram.
- The collaboration comes from sequence diagram. The collaboration diagram represents the collaboration which is a set of objects related to achieve and decide outcome.
- In collaboration the sequence is indicated by numbering the messages several numbering schemes are available.

COLLABORATION DIAGRAM FOR STOCK MAINTENANCE SYSTEM:



RESULT:

Thus, the UML Sequence diagram and Collaboration diagram for Stock Maintenance System was developed successfully.

EX NO:6	DRAW RELEVANT STATE CHART AND ACTIVITY DIAGRAM FOR THE SAME SYSTEM FOR STOCK MAINTENANCE SYSTEM
DATE:	

AIM:

To draw State chart diagram and Activity diagram for Stock Maintenance System.

STATE CHART DIAGRAM:

Graphical Representation: State chart diagrams are graphical representations used to model the behaviour of objects in a system over time. They depict states, transitions between states, and events that trigger transitions.

States and Transitions: States represent conditions or situations during the lifecycle of an object, while transitions indicate the change from one state to another in response to events.

Event-Driven Behaviour: State chart diagrams capture the event-driven behaviour of objects, showing how they respond to external events or internal conditions by transitioning between states.

State Representation: States are depicted as rounded rectangles, each labeled with a name representing a specific condition or mode of the object.

Transition Lines: Transitions are represented by arrows connecting states, indicating the flow of control from one state to another in response to events.

Event Triggers: Events trigger transitions between states. These events can be external stimuli, such as user input or system signals, or internal conditions, such as the completion of a task or the passage of time.

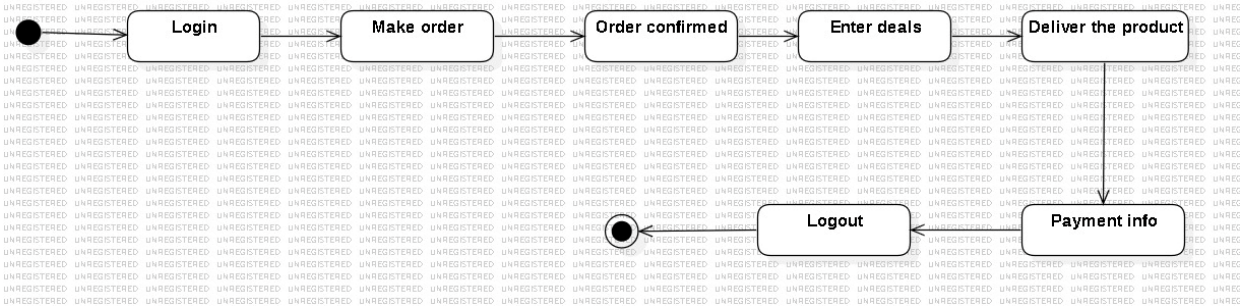
Concurrent States: State chart diagrams can represent concurrent states, where an object can be in multiple states simultaneously, showing parallel behaviour or concurrent activities.

Hierarchy and Nesting: States can be hierarchical, allowing for the modeling of complex behaviours with nested states and sub-states.

Dynamic Behaviour: State chart diagrams capture the dynamic behaviour of objects, showing how they transition between different states based on events and conditions.

Executable Models: State chart diagrams can be used to generate executable models, enabling simulation and verification of system behaviour before implementation.

STATE CHART DIAGRAM FOR STOCK MAINTENANCE SYSTEM:



ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system.

An activity diagram shows the overall flow of control. An activity is shown as a rounded box containing the name of the operation. This activity diagram describes the behaviour of the system.

An activity diagram is variation or special case or a state machine, in which the states are activities representing a performance of operation and the transition or triggered by the completion of operation.

Activity diagram is similar to state chart diagram where the token represented as an operation. An activity shows as a round box containing name of operation. The concurrent control is indicated by multiple arrows, leaving a synchronization bar represented by short or thick bar with incoming and outgoing arrows. Activity diagram is another important diagram in UML to describe dynamic aspects of the system.

This diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So, the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent.

Activity diagrams deals with all type of flow control by using different elements like fork, join etc. The basic purposes of activity diagrams are similar to other four diagrams. It captures the dynamic behaviour of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

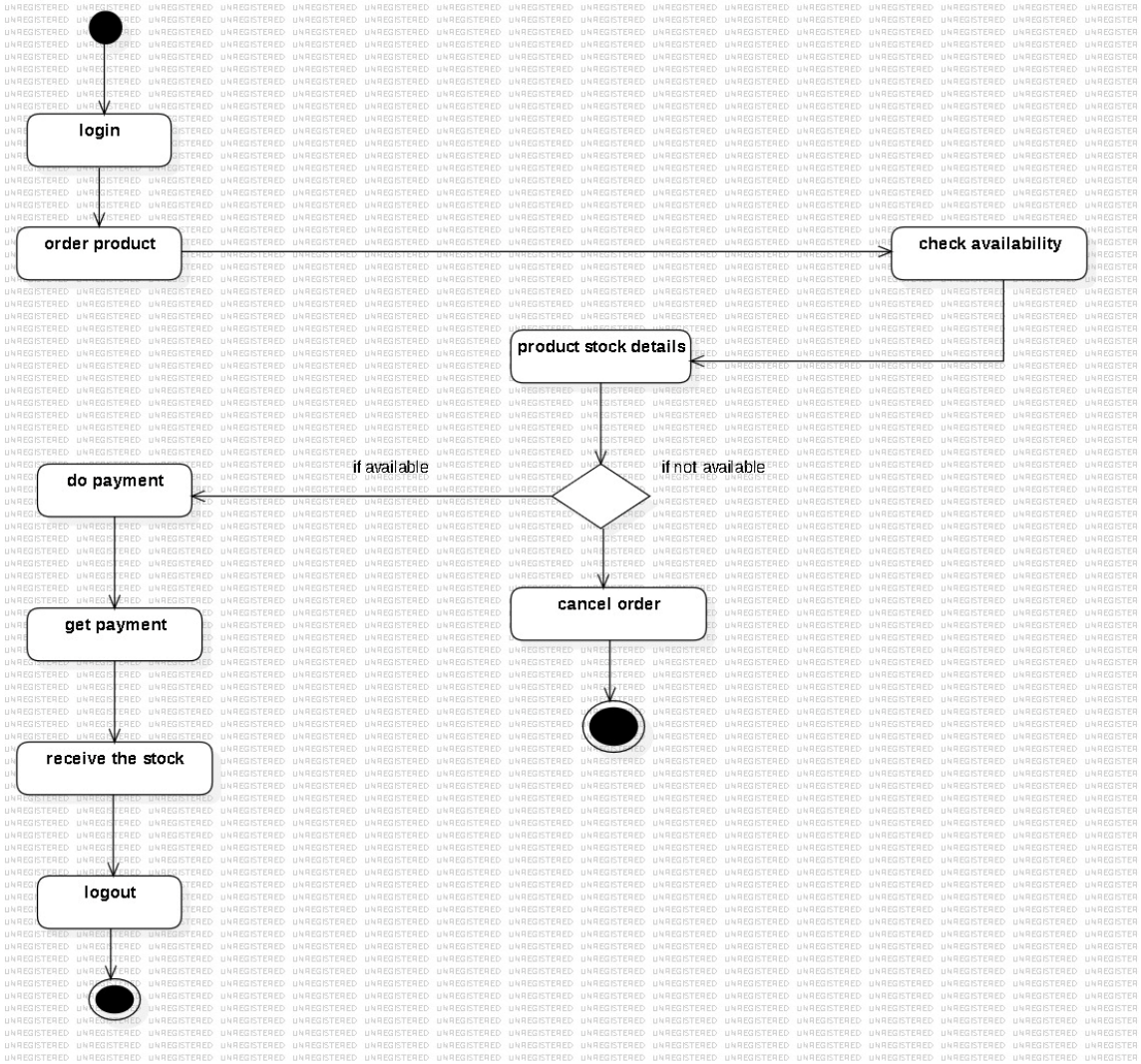
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another.

Activity diagram is some time considered as the flow chart. Although the diagrams look like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

The purposes can be described as:

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

ACTIVITY DIAGRAM FOR STOCK MAINTENANCE SYSTEM:



RESULT:

Thus, the state chart diagram and activity diagram for Stock maintenance system was developed successfully.

EX NO:7	IMPLEMENT THE SYSTEM AS PER THE DETAILED DESIGN
DATE:	

AIM:

To implement the system as per the detailed design.

INTRODUCTION:

In today's dynamic business environment, efficient management of inventory and stock plays a crucial role in ensuring the smooth operation of organizations across various industries. A Stock Maintenance System serves as a central hub for managing and controlling the inflow, outflow, and storage of products or materials within an organization's inventory.

PROBLEM STATEMENTS:

Inefficient Inventory Management: Many organizations struggle with inefficient inventory management processes, leading to stockouts, overstock situations, and poor utilization of resources.

Manual Tracking and Record-Keeping: Traditional methods of tracking stock inventory using spreadsheets or manual record-keeping are prone to errors, inconsistencies, and lack real-time visibility into stock levels and movements.

Lack of Stock Visibility: Organizations often face challenges in obtaining real-time visibility into their stock inventory across multiple locations or warehouses, making it difficult to monitor stock levels, track movements, and make informed decisions.

Ineffective Order Fulfillment: Inaccurate inventory data and inefficient order management processes can result in delays, errors, and customer dissatisfaction in fulfilling orders promptly and accurately.

Poor Forecasting and Demand Planning: Without robust forecasting and demand planning capabilities, organizations struggle to anticipate future stock requirements, leading to stockouts, excess inventory costs, and missed sales opportunities.

Inadequate Supplier Management: Ineffective supplier management processes, including delayed deliveries, poor communication, and lack of supplier performance tracking, can impact stock availability and overall operational efficiency.

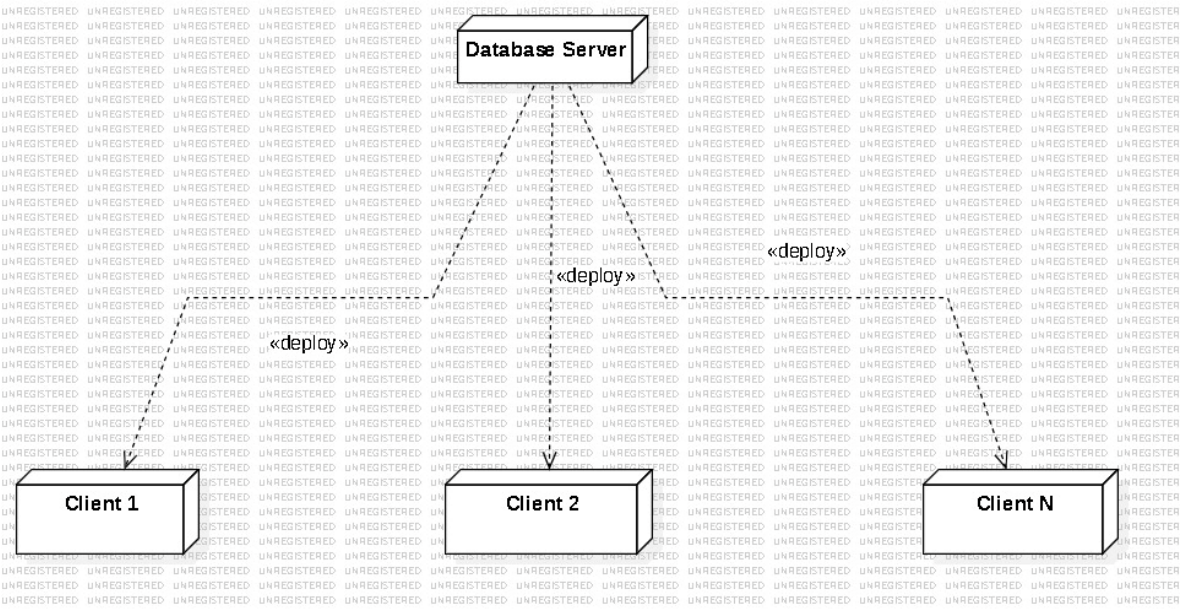
Limited Stock Valuation Methods: Organizations may lack sophisticated methods for valuing their stock inventory, resulting in inaccurate financial reporting, misrepresentation of asset values, and compliance issues.

Integration Challenges: Integration challenges between stock maintenance systems and other business systems such as accounting, sales, and procurement can hinder data flow, automation, and end-to-end business processes.

Security and Compliance Concerns: Security vulnerabilities and compliance risks associated with stock inventory data, including unauthorized access, data breaches, and regulatory non-compliance, pose significant challenges to organizations.

Scalability and Flexibility: As organizations grow and expand their operations, they may encounter scalability and flexibility issues with their existing stock maintenance systems, hindering their ability to adapt to changing business requirements and market dynamics.

DEPLOYMENT DIAGRAM:



IMPLEMENTATION CODE:

Interface1.java:

```
import java.util.*;

/**
 *
 */
public interface Interface1 {

}
```

ProductDetails.java:

```
import java.util.*;

/**
 *
 */
```

```
public class productDetails {

    /**
     * Default constructor
     */
    public productDetails() {
    }

    /**
     *
     */
    public integer prodCode;

    /**
     *
     */
    public string prodName;

    /**
     *
     */
    public integer prodQty;

    /**
     *
     */
    public float prodPrice;

    /**
     *
     */
    public void prodAdd() {
        // TODO implement here
    }

    /**
     *
     */
    public void prodDelete() {
        // TODO implement here
    }

    /**
     *
     */
    public void prodUpdate() {
        // TODO implement here
    }

    /**
     *
```

```
     */
    public void prodDetails() {
        // TODO implement here
    }

}
```

purchaseDetails.java:

```
import java.util.*;

/**
 *
 */
public class purchaseDetails {

    /**
     * Default constructor
     */
    public purchaseDetails() {
    }

    /**
     *
     */
    public integer purchCode;

    /**
     *
     */
    public date purcDate;

    /**
     *
     */
    public integer subid;

    /**
     *
     */
    public string subname;

    /**
     *
     */
    public integer purcQty;

    /**
     *
     */
    public float purcPrice;
```

```
    /**
     *
     */
    public void save() {
        // TODO implement here
    }

    /**
     *
     */
    public void delete() {
        // TODO implement here
    }

    /**
     *
     */
    public void purchaseedit() {
        // TODO implement here
    }

    /**
     *
     */
    public void purchaseDetails() {
        // TODO implement here
    }

}
```

salesDetails.java:

```
import java.util.*;


/**
 *
 */
public class salesDetails {


    /**
     * Default constructor
     */
    public salesDetails() {
```

```
}

/**
 *
 */
public integer salid;

/**
 *
 */
public date salDate;

/**
 *
 */
public integer custCode;

/**
 *
 */
public string custName;

/**
 *
 */
public integer prodCode;

/**
 *
 */
public integer price;

/**
```

```
     *
     */
    public integer qty;

    /**
     *
     */
    public void sale() {
        // TODO implement here
    }

    /**
     *
     */
    public void salesDetails() {
        // TODO implement here
    }

}
```

RESULT:

Thus, the implementation of the Stock maintenance system as per the detailed design was executed successfully.

EX NO:08	TEST THE SOFTWARE SYSTEM FOR ALL THE SCENARIOS IDENTIFIED AS PER THE USE CASE DIAGRAM
DATE:	

AIM:

To test the Foreign Trading software system for all the scenarios identified as per the use case diagram.

STOCK MAINTENANCE SYSTEM OVERVIEW:

The Stock Maintenance System is designed to manage stock inventory, including tracking stock levels, updating product information, processing orders, and generating reports. In the dynamic landscape of business operations, effective management of stocks and inventory is crucial for ensuring smooth operations and meeting customer demands. The Stock Maintenance System is a comprehensive software solution designed to streamline the management of stocks and inventory for businesses of all sizes.

Review Use Case Diagram:

- Gain a comprehensive understanding of the depicted interactions and functionalities portrayed in the use case diagram.
- Identify primary actors and their specific goals within the system to ensure accurate representation and alignment with business requirements.

Identify Test Scenarios:

- Explore various perspectives, including those of different user roles, to ensure comprehensive coverage of test scenarios.
- Include a diverse range of functionalities, encompassing both basic operations and edge cases, to validate system behavior under various conditions and scenarios.

Create Test Cases:

- Translate identified test scenarios into detailed steps, inputs, expected outputs, and test conditions.
- Include positive (expected behavior) and negative (error or exception handling) test cases, boundary tests, and error-handling scenarios to thoroughly validate the system's functionality and robustness.

Prepare Test Environment:

- Establish a dedicated setup that isolates testing activities from the production system, minimizing risks of interference or disruption.

- Ensure that the test environment replicates the production environment's configuration and data accurately to validate system behavior and performance realistically.

Execute Test Cases:

- Adhere to predefined steps diligently and document both actions performed and observed outcomes during test case execution.
- Utilize diverse test data to cover a range of scenarios, including boundary values and invalid inputs, comprehensively validating the system's functionality and behavior.

Report Bugs:

- Document any discrepancies or defects encountered during testing in a dedicated bug tracking system.
- Include detailed information such as steps to reproduce, system configurations, and screenshots to assist developers in diagnosing and resolving issues efficiently.

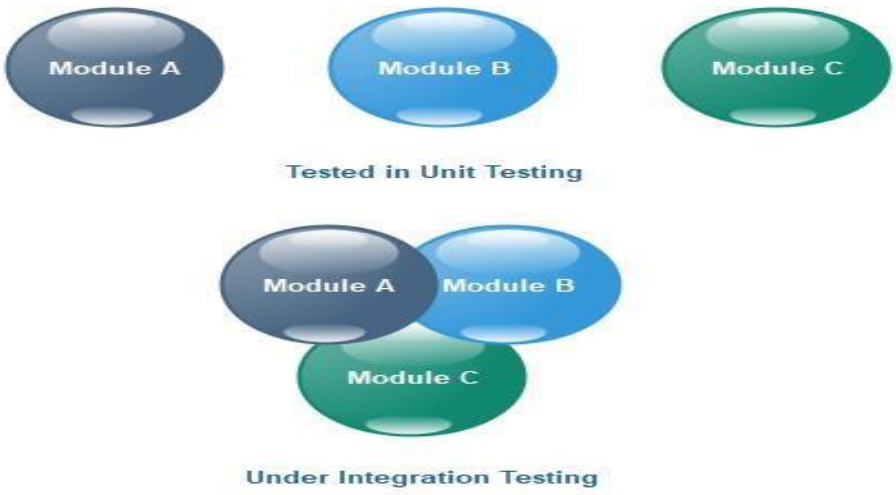
Integration Testing:

- Conduct comprehensive integration testing to verify seamless interaction and functionality of different modules and components within the Stock Maintenance System.
- Verify data integrity and consistency across integrated modules, ensuring accurate information flow and processing.
- Test APIs and interfaces between modules to validate data exchange, communication protocols, and system interoperability.

Example:

Components to Integrate: Stock tracking module, Inventory management module

Test Case: Add a new stock item and verify that it reflects accurately in the inventory, ensuring proper synchronization and data consistency across modules.



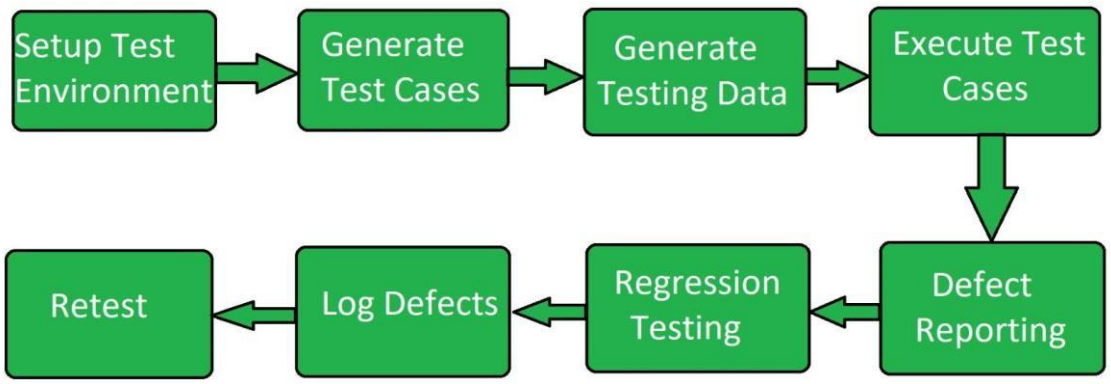
System Testing Overview:

- System testing evaluates the entire stock maintenance system to validate its behavior and performance against defined requirements and standards.
- It encompasses testing end-to-end scenarios, including stock management, inventory tracking, reporting functionalities, and user interactions, to ensure seamless operation under various conditions.

Example:

End-to-End Workflow: User interaction with the Stock Maintenance System

Test Case: Simulate a stock management session where a user logs in, adds new stock items, updates quantities, checks inventory status, and generates a stock report. Confirm that each step of the workflow functions correctly and maintains data integrity.



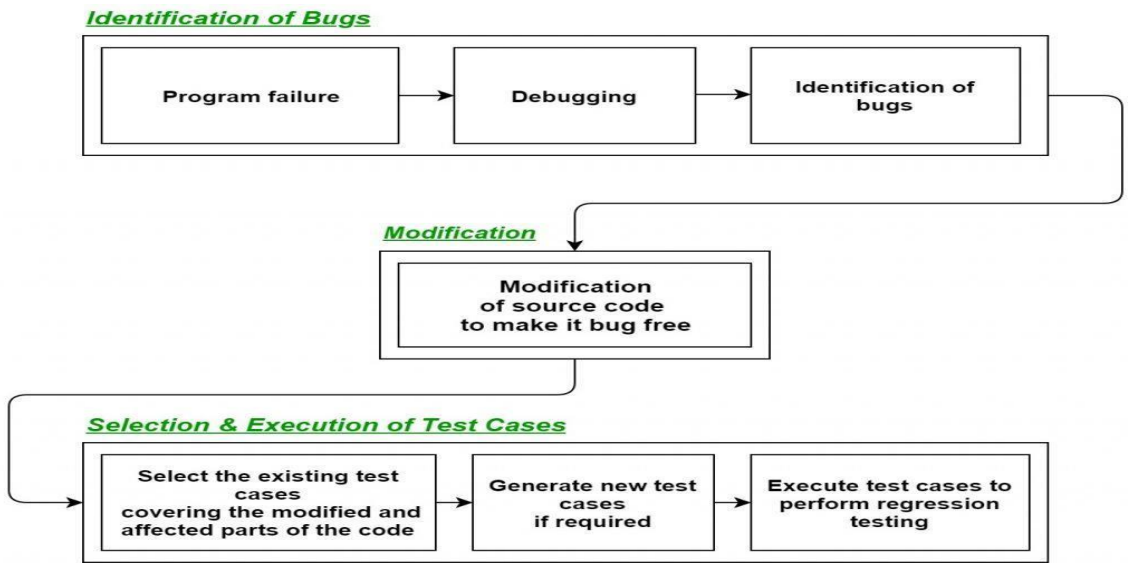
Regression Testing Overview:

- Regression testing is crucial for a Stock Maintenance System to detect and prevent the introduction of new defects or issues when modifications or enhancements are implemented.
- It involves re-executing previously conducted tests to ensure that existing functionalities still operate correctly after changes are made, thereby maintaining system stability and reliability.

Example:

Scenario Change: Enhance the stock tracking module to support new stock categories

Test Case: After enhancing the stock tracking module, re-run previous test cases (e.g., stock addition, inventory update) to verify that existing functionalities remain intact and function as expected without any adverse impacts.



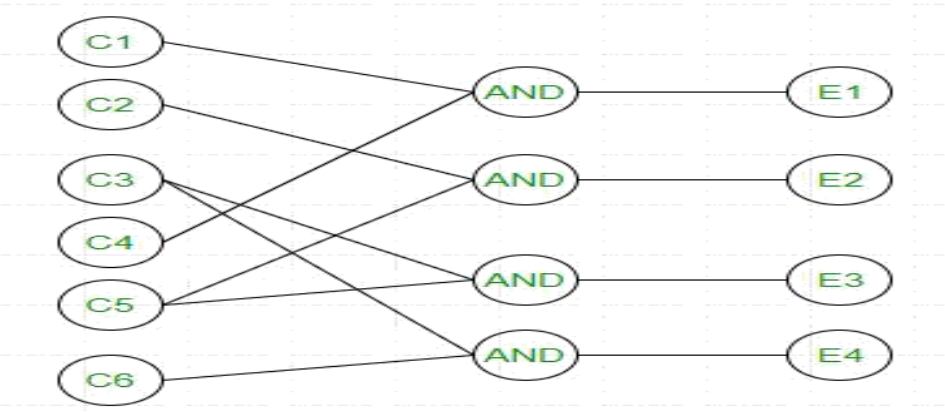
Black Box Testing Overview:

- Black box testing focuses on evaluating the functionalities of the Stock Maintenance System without examining its internal structure or implementation specifics.
- It validates user-facing features such as stock addition, inventory tracking, and reporting functionalities to ensure they meet defined requirements and expectations.

Example:

Functionality to Test: Stock Addition

Test Case: Input valid stock details (e.g., name, quantity, category) and verify that the system correctly adds the stock item and updates relevant information in the inventory logs and stock reports.



		1	2	3	4
CAUSES	C1 C2 C3 C4 C5 C6	1 0 0 1 0 0	0 1 0 0 1 0	0 0 1 0 1 0	0 0 1 0 0 1
EFFECTS	E1 E2 E3 E4	x - - -	- x - -	- - x -	- - - x

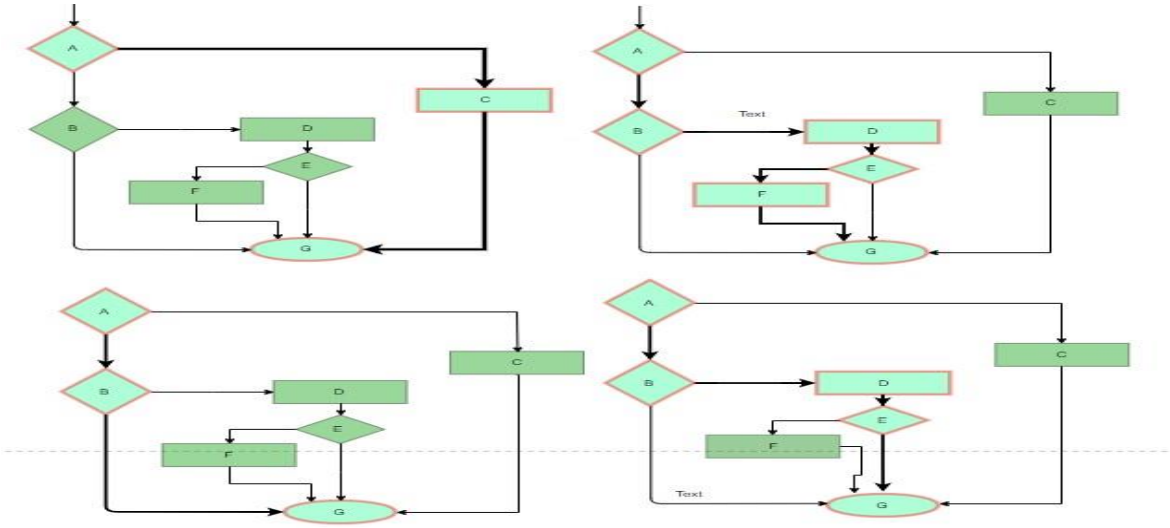
White Box Testing Overview:

- White box testing involves analyzing the internal structure and logic of the Stock Maintenance System, including code paths, decision branches, and data flows.
- It verifies the accuracy of algorithms, error handling procedures, and data validations integrated into the system.

Example:

Component: Stock Addition Logic

Test Case: Validate that the system correctly adds new stock items according to predefined rules (e.g., uniqueness constraints, category assignments), handles exceptions such as duplicate entries, and updates inventory records accurately.



Unit Testing Overview:

Unit testing is highly suitable for verifying the functionality of individual components or units within the Stock Maintenance System (e.g., modules, classes, functions) in isolation.

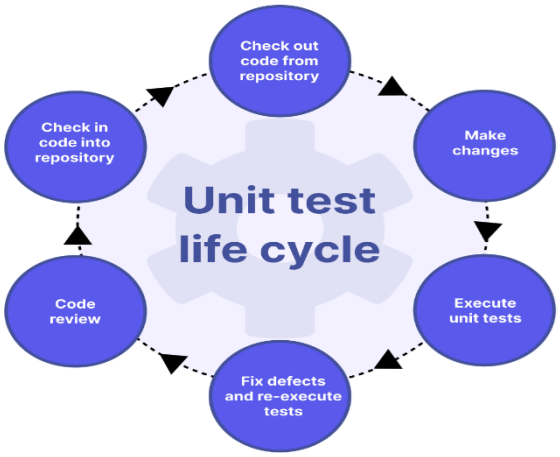
It ensures that critical functionalities such as stock addition, inventory update mechanisms, and reporting functions operate correctly at a granular level.

Example:

Component: Stock Addition Module

Functionality to Test: addStockItem() function

Test Case: Confirm that the addStockItem() function within the Stock Addition Module correctly adds a new stock item with valid details (e.g., name, quantity, category) to the inventory and returns the expected result indicating success or failure.



RESULT:

Thus, the software system for all the scenarios identified as per the use case diagram for Stock Maintenance System and was tested successfully.

EX NO:09	IMPROVE THE REUSABILITY AND MAINTAINABILITY OF THE SOFTWARE SYSTEM BY APPLYING APPROPRIATE DESIGN PATTERNS
DATE:	

AIM:

To improve the reusability and maintainability of the software system by applying appropriate design patterns.

STOCK MAINTENANCE SYSTEM:

A Stock Maintenance System is a software application designed to manage and track stocks, inventory, and related operations within an organization. It includes functionalities such as stock addition, removal, tracking, inventory management, reporting, and analysis.

To enhance the reusability and maintainability of the Stock Maintenance System, we can incorporate several design patterns. Here are some design patterns that can be applied:

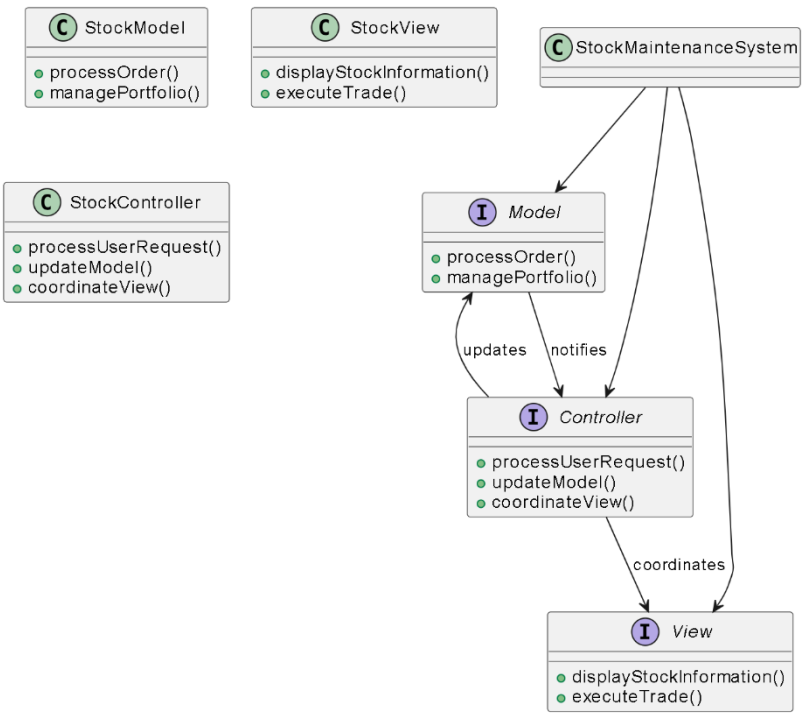
Model-View-Controller Pattern:

- The Model encompasses the stock data and operational algorithms, handling tasks such as inventory management and transaction processing.
- The View showcases stock information to users, furnishing interfaces for stock analysis and monitoring.
- The Controller interprets user commands, modifies the model based on trading activities, and synchronizes with the view to exhibit real-time stock updates and statuses.

Context: MVC segregates stock data administration, UI representation, and user engagement.

Problem: Absence of MVC escalates code intricacy, thereby complicating maintenance efforts.

Solution: Employing MVC partitions the system into Model, View, and Controller components, streamlining development processes and enhancing code structuring.



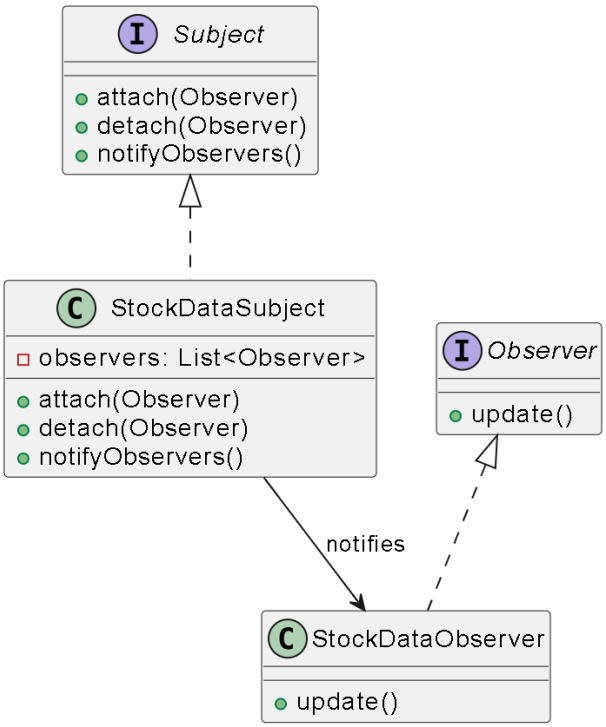
Observer Pattern:

• The Observer Pattern facilitates notifications to components regarding system alterations, thereby boosting flexibility and scalability. It simplifies the integration of new notifications, thereby enhancing system maintainability.

Context: Employed when objects necessitate notification of changes in another object's state.

Problem: Establishes a one-to-many relationship devoid of tightly coupling objects, ensuring automatic updates.

Solution: Define a subject interface comprising methods for attaching, detaching, and notifying observers. Observers register with subjects to receive updates on state alterations.



Publish-Subscribe Pattern:

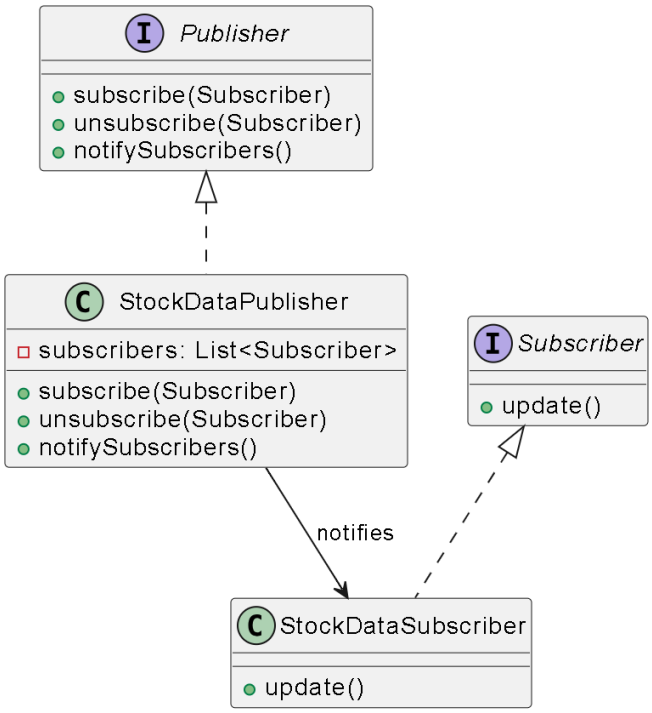
• The Publish-Subscribe Pattern facilitates real-time updates and notifications across various components within trading systems.

• This pattern augments scalability, flexibility, and responsiveness within the architecture of the trading system.

Context: Real-time updates play a vital role in trading operations.

Problem: Disseminating real-time market data lacking the Publish-Subscribe Pattern results in data latency and inefficiencies.

Solution: By utilizing the pattern, efficient data distribution is enabled, allowing providers to publish to specific topics. This ensures that pertinent updates reach subscribing modules, thereby enhancing the system's scalability and responsiveness.



Strategy Pattern:

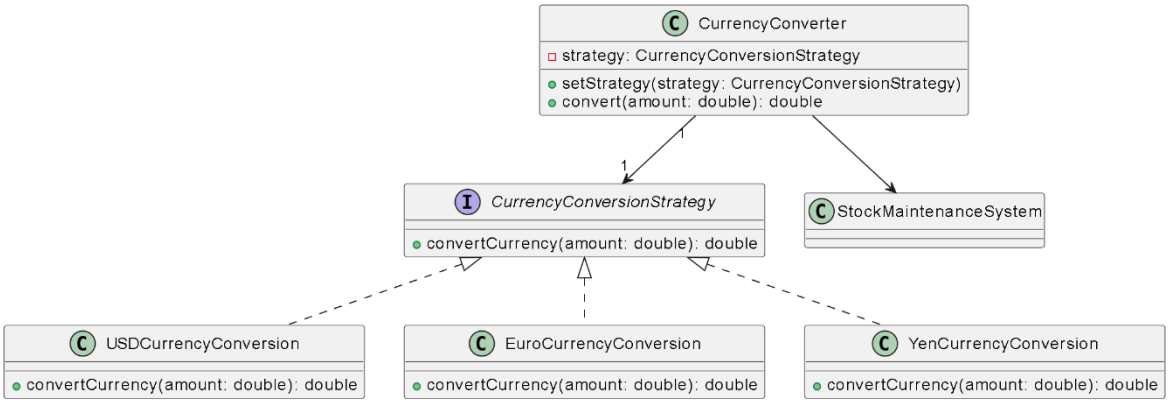
• The Strategy Pattern is applicable for managing various algorithms related to international currency conversion.

• Through the implementation of the Strategy Pattern, the system facilitates the interchangeability of algorithms, offering flexibility for users to choose the most appropriate currency conversion strategy.

Context: Utilized to manage a collection of algorithms interchangeably.

Problem: Supports dynamic selection and switching of algorithms, fostering code flexibility and reusability.

Solution: Define a strategy interface comprising algorithmic methods. Implement concrete strategy classes for each algorithm and enable runtime switching between them.



Adapter Pattern:

• The Adapter Pattern is employed to seamlessly integrate various trading data sources.

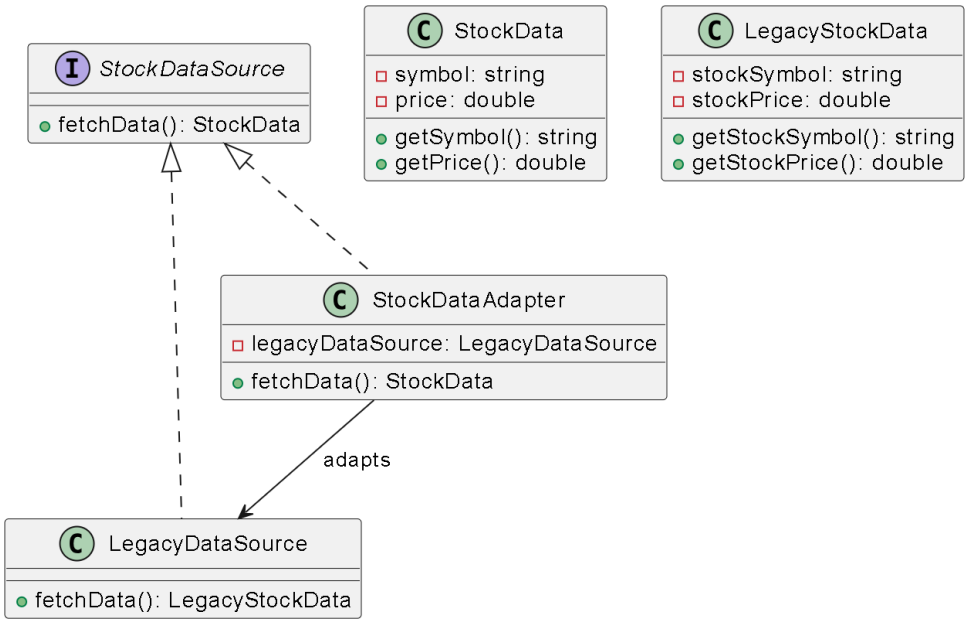
• It addresses the challenge of incompatible interfaces between data sources and the trading system by offering a wrapper that translates data formats and protocols.

• This pattern ensures smooth communication and data exchange, thereby enhancing interoperability and flexibility within the trading ecosystem.

Context: Integrating diverse data sources with varying formats and protocols is a common requirement.

Problem: Incompatibility between these data sources and the trading system results in communication issues and data parsing errors.

Solution: The Adapter Pattern provides a wrapper that translates data formats and protocols, facilitating seamless integration without compromising data integrity or communication.



Command Pattern:

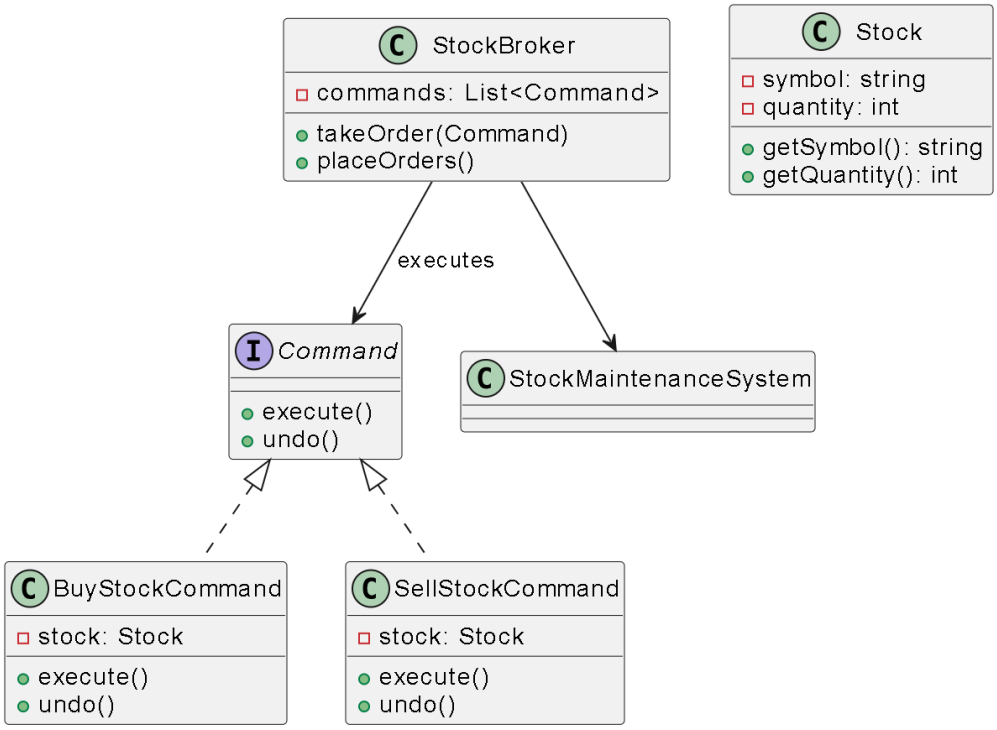
• The Command Pattern encapsulates requests as objects, facilitating parameterization of clients with different requests, queuing, logging, and supporting undoable operations.

• This pattern enables the system to support undoable operations, transactional behavior, and logging, thereby enhancing maintainability, scalability, and reliability within the system.

Context: Utilized to encapsulate requests as objects, enabling parameterized and queued requests.

Problem: Decouples sender and receiver, supports undoable operations, and provides structured request handling.

Solution: Define command objects that encapsulate requests and parameters. Clients create and queue commands, and invokers execute them when needed.



Proxy Pattern:

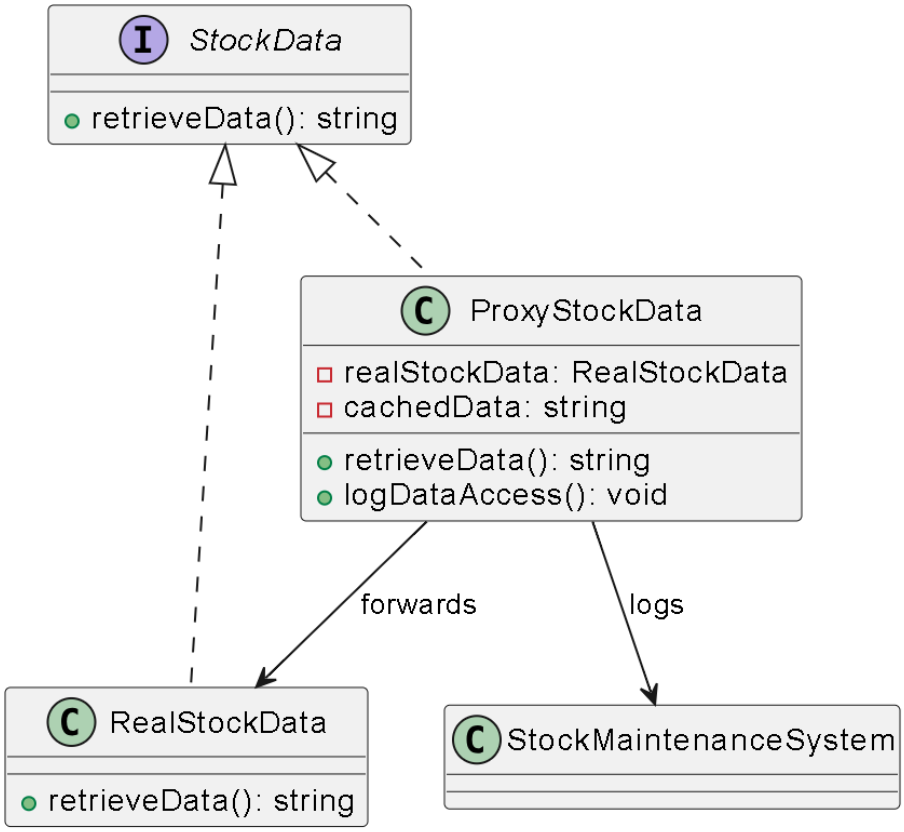
• The Proxy Pattern is utilized to regulate access to sensitive trading data or manage expensive operations.

• It addresses the challenge of direct access to such resources by providing a surrogate object that serves as a representative, managing access, and incorporating functionalities like caching or logging.

Context: There is a necessity to control access to sensitive trading data or manage expensive operations.

Problem: Direct access to such resources can pose security risks or lead to performance issues.

Solution: The Proxy Pattern offers a surrogate object that manages access, adds functionalities like caching or logging, thereby enhancing security, performance, and resource management within the trading system.



Facade Pattern:

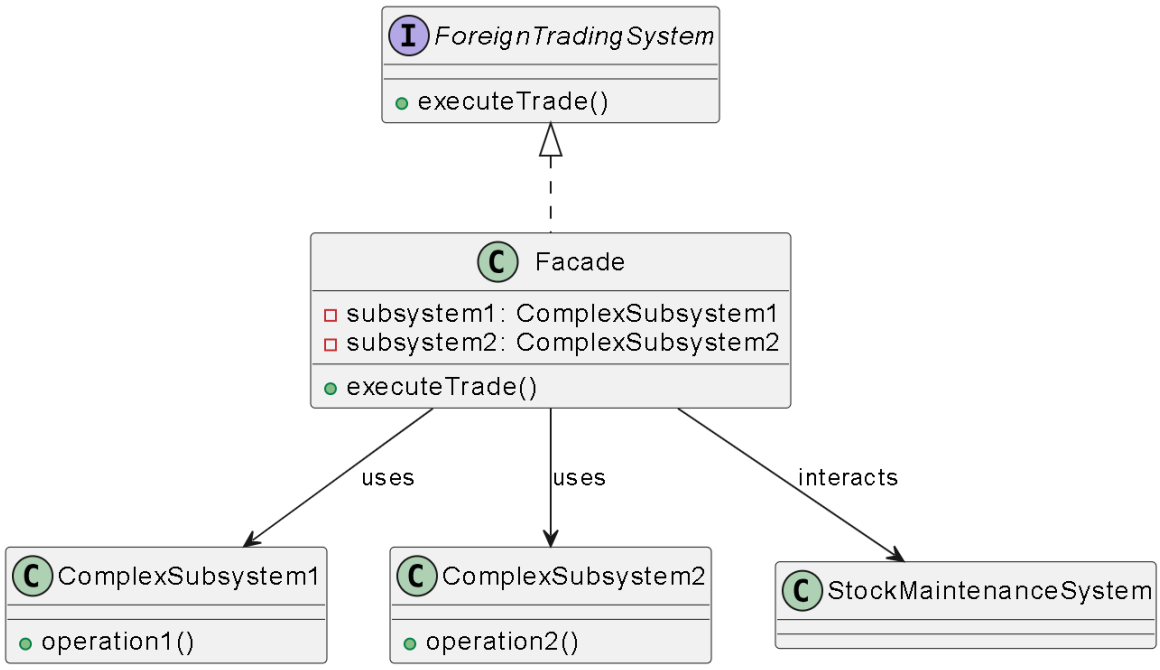
• The Facade Pattern simplifies interactions with complex subsystems within a Foreign Trading System by providing a unified interface.

• It addresses the challenge of managing multiple subsystems with varying complexities by abstracting their functionality behind a single facade class.

Context: Direct interactions with subsystems cause code complexity and increased dependencies.

Problem: Managing multiple subsystem interactions leads to complex code and dependencies.

Solution: Facade Pattern simplifies interactions, hiding subsystem complexities for easier system management and maintenance.



By integrating these design patterns into the Stock Maintenance System, we can enhance reusability and maintainability by encouraging code reuse, encapsulating complexity, and decoupling components. This results in a more modular, flexible, and maintainable architecture, simplifying extension, modification, and maintenance of the system over time.

RESULT:

Thus, the reusability and maintainability of the software system by applying appropriate design pattern was improved successfully.