UNIVERSITY COLLEGE OF ENGINEERING NAGERCOIL

(ANNA UNIVERSITY CONSTITUENT COLLEGE)

KONAM, NAGERCOIL – 629 004

RECORD NOTE BOOK

CCS356-OBJECT ORIENTED SOFTWARE ENGINEERING

REGISTER NO :

NAME :

YEAR/SEMESTER :

DEPARTMENT :

UNIVERSITY COLLEGE OF ENGINEERINGNAGERCOIL

(ANNA UNIVERSITY CONSTITUENT COLLEGE)

KONAM, NAGERCOIL – 629 004

Register No:

Certified that, this is the bonafide record of work done by ***Mr./Ms.****…… of VI Semester in Computer Science and Engineering of this college, in the* ***CCS356 – OBJECT ORIENTED SOFTWARE ENGINEERING*** *during academic year 2023-2024 in partial fulfillment of the requirements of the B.E Degree course of the Anna University Chennai.*

Staff-in-charge **Head of the Department**

This record is submitted for the University Practical Examination held on .....

Internal Examiner **External Examiner**

INDEX

Exp No	Date	Title	Page	Sign
1.		Identify a software system that needs to be developed		
2.		Document the Software Requirements Specification (SRS) for the Foreign Trading System		
3.		Identify the use cases and the develop the Use Case model		
4.		Identify the conceptual classes and develop a Domain Model and also derive a Class diagram for the Foreign Trading System		
5.		Using the identified scenarios, find the interaction between objects and represent them using UML Sequence and Collaboration diagram		
6.		Draw relevant State Chart and Activity Diagram for the Foreign Trading System		
7.		Implement the system as per the detailed design		
8.		Test the software system for all the scenarios identified as per the Use Case diagram		
9.		Improve the reusability and maintainability of the software system by applying appropriate design patterns		

Ex.No:01	Identify a software system that needs to be developed Foreign Trading System

AIM:

To identify a software system that needs to be developed – Foreign Trading System.

INTRODUCTION:

This project centers around the Foreign Trade System (FTS), serving as a pivotal interface connecting account holders with the market. The primary objective of the FTS is to streamline and enhance foreign trade operations. In its initial phase, the system meticulously gathers comprehensive details about various currencies, focusing on assessing the profit and loss associated with currency holdings. By doing so, the FTS aims to provide account holders with a robust platform for informed trading decisions and improved market engagement.

FEATURES OF FOREIGN TRADING SYSTEM:

1. User Authentication:

- Swift and secure verification of user credentials for seamless access.

2. Commodity Information:

- Efficient retrieval of detailed commodity information for traders' informed decisions.

3. Order Placement:

- Seamless functionality for precise and quick placement of trader orders.

4. Transaction Processing:

- Secure and prompt processing of transactions, including payment verification.

5. Security Infrastructure:

- Implementation of robust security measures for data protection and cyber threat prevention.

6. Real-time Market Data:

- Timely provision of accurate market updates, empowering traders with the latest information.

ARCHITECTURE OF FOREIGN TRADING SYSTEM:

1. **Client Side:**
   - Users interact with the trading system through a user interface, submitting buy or sell orders and receiving real-time market data.
   - The client's interface displays market information, order status, and portfolio management tools.

2. **Server Side:**
   - The server processes client orders, matches buy and sell orders, and executes trades based on market conditions.
   - It manages market data, risk controls, and communication with exchanges and clearinghouses.
3. **Communication:**
   - Communication between clients and the server is facilitated over a network, ensuring timely order processing and data exchange.
   - The server also communicates with external entities such as exchanges, regulatory bodies, and data providers to maintain a seamless trading environment.

Testing:

Functional Testing:

Ensures that individual components, modules, and the entire system function correctly according to specified requirements.

Non-functional testing:

Evaluates aspects of a system beyond its basic functionality, such as performance, security, usability, and reliability.

Regression Testing:

Verifies that recent changes or updates have not adversely affected existing functionalities.

User Acceptance Testing (UAT):

Involves end-users to validate the system's functionality, usability, and suitability for their business needs.

RESULT:

Thus, a software system that needs to be developed the foreign trading system was identified successfully.

Ex.No:02	Document the Software Requirements Specification (SRS) for Foreign Trading System

AIM:

To document the Software Requirements Specification (SRS) for the Foreign Trading System.

PROBLEM STATEMENT:

The steps involved in Foreign Trading System are:

- The forex system begins its process by getting the username and password from the trader.
- After the required commodities are chosen, the trader places the order.
- The administrator checks for the availability for the required commodities and updates it in the database.
- After the commodities are ready for the trade, the trader pays the amount to the administrator.
- The administrator in turn provides the bill by receiving the amount and updates it in the database.
- The trader logouts after the confirmation message have been received.

1. INTRODUCTION:

1.1. Purpose:

This project centers around the Foreign Trade System (FTS), serving as a pivotal interface connecting account holders with the market. The primary objective of the FTS is to streamline and enhance foreign trade operations. In its initial phase, the system meticulously gathers comprehensive details about various currencies, focusing on assessing the profit and loss associated with currency holdings. By doing so, the FTS aims to provide account holders with a robust platform for informed trading decisions and improved market engagement.

1.2. Document Conversion:

The document follows the IEEE format standard (IEEE Std. 830 – 1998).

	Font	Style	Size
Heading	Times New Roman	Bold	14
Sub-Heading	Times New Roman	Bold	13
Other's	Times New Roman	Regular	13

1.3. Intended Audience and Reading Suggestions:

1. **Developers/Programmers:** Individuals responsible for designing, developing, and maintaining the Foreign Trading System. This could include software engineers, programmers, and system architects.
2. **System Administrators:** Professionals who are responsible for configuring, deploying, and maintaining the Foreign Trading System within an organizational setting.
3. **Traders/End Users:** Individuals who will be interacting with the Foreign Trading System for performing foreign trades. This could include traders, investors, and other users interested in international commodities trading.
4. **Project Managers:** Those overseeing the development and implementation of the Foreign Trading System, responsible for ensuring the project's success within the given scope and timeline.
5. **Stakeholders:** Any individuals or entities with an interest in the success of the Foreign Trading System, including business owners, investors, or regulatory bodies.

1.4. Project Scope:

The proposed online Foreign Trading System provides users with a secure platform for 24-hour access to currency investment updates. This system ensures seamless user authentication, enabling traders to retrieve comprehensive commodity details, place orders, and conduct secure transactions. With a focus on eliminating exchange fees and commissions, the system facilitates margin and leverage trading, allowing investors to potentially increase earnings by up to 100 times. Furthermore, the system incorporates a Currency Converter feature for accurate and instant foreign currency conversions. It caters to various investment preferences, offering options for minimum currency trading investments and providing a mini account option with a minimal initial

investment, allowing users to engage in leveraged foreign currency trading with the potential for high rewards and minimal risk.

2.OVERALL DESCRIPTION:

2.1. Product Perspective:

The description of an online Foreign Trading System, taking live feeds of international currency status, aligns with the problem statement's emphasis on creating a system that enables traders to engage in foreign currency trading. The online nature of the system and the integration of live currency feeds reflect the digital and dynamic aspects expected from the proposed solution.

2.2. Product function:

1. Live Currency Feed:
   - Implement a mechanism to take live feeds of international currency status.
2. User Authentication and Authorization:
   - Develop a secure system to authenticate users based on their username and password.
3. Commodity Information Retrieval:
   - Enable traders to access comprehensive details about various commodities.
4. Order Placement:
   - Allow traders to place orders for selected commodities.
5. Database Management:

- Implement a database to store and update real-time information on commodity availability.

6. Transaction Processing:
   - Facilitate secure payment transactions from traders to administrators.
7. Billing and Confirmation:
   - Generate and provide bills to traders upon receiving payments.
   - Communicate confirmation messages to traders.

2.3. User Characteristics:

There are various kinds of users for this product,

- **Investors**-Focuses on long-term growth, typically holding assets for an extended period, and often seeks dividends or interest income.
- **Traders**-Actively buys and sells financial instruments in the short term to capitalize on market fluctuations.
- **Brokers**-Facilitates buying and selling of financial instruments on behalf of clients, earning commissions or fees for their services.
- **Banks**-Works within financial institutions, managing client accounts, providing financial advice, and facilitating various banking services.

2.4. Design and Implementation Constraints:

- **LOGIN:** The Login module contains the form which contain membership name and member password. It includes Username and Password.
- **TRADING ACCOUNT DETAILS:** This form contains the information about account holder, market status, currency held, trading histories, etc.
- **BUY:** After the user logged in they can buy stocks online the user can buy stock only it is available for buying.
- **SELL:** After the user logged in they can sell stocks online, the user can sell his own stocks only
- **BANK ACCOUNT DATABASE:** After the trading is finished user has to select the type of transaction whether credit card Demat account.

3. FUNCTIONAL REQUIREMENTS:

3.1. External Interface Requirements:

The Foreign Trading System employs a Graphical User Interface (GUI) to facilitate user interaction, ensuring a seamless and user-friendly experience. The GUI plays a crucial role in managing user authentication and authorization, securing the login process for traders. Through the GUI, traders can access forms for sourcing commodity details, placing orders, and viewing comprehensive trading histories. This interface design enhances the overall accessibility and usability of the system

3.2. User Interfaces:

There are various kinds of users for this product,

- **Investors**-Focuses on long-term growth, typically holding assets for an extended period, and often seeks dividends or interest income.
- **Traders**-Actively buys and sells financial instruments in the short term to capitalize on market fluctuations.
- **Brokers**-Facilitates buying and selling of financial instruments on behalf of clients, earning commissions or fees for their services.
- **Banks-**Works within financial institutions, managing client accounts, providing financial advice, and facilitating various banking services.

3.3. Hardware Interfaces:

- Needed: Computers
- Hard Disk: 100-150 GB
- RAM: 512-1 GB required
- Internet Connection required.
- Cables, wires, Network adapters are required.

3.4. Communication Interfaces:

The Foreign Trading System relies on a stable internet connection for seamless trader-server communication. Through this connectivity, real-time data exchange ensures prompt updates and efficient trading. Email notifications play a pivotal role, providing traders with instant transaction confirmations and timely system updates. Additionally, the system supports secure file transfer services, enhancing the overall reliability and communication capabilities. The key concept of E-Shopping is integrated, emphasizing the system's commitment to facilitating secure and efficient online transactions.

4. SYSTEM FUNCTION:

1. **User Authentication:**
    - Swift and secure verification of user credentials for seamless access.

2. **Commodity Information:**
    - Efficient retrieval of detailed commodity information for traders' informed decisions.
3. **Order Placement:**
    - Seamless functionality for precise and quick placement of trader orders.
4. **Transaction Processing:**
    - Secure and prompt processing of transactions, including payment verification.
5. **Security Infrastructure:**
    - Implementation of robust security measures for data protection and cyber threat prevention.
6. **Real-time Market Data:**
    - Timely provision of accurate market updates, empowering traders with the latest information.

5. OTHER NON-FUNCTIONAL REQUIREMENTS:

5.1. Performance Requirements:

The system must be able to perform in adverse weather conditions like heavy rain, hot climate etc. Connection can be interrupted but service must be uninterrupted. Transportation of stocks must be efficient and punctual. Database should support largeamount of data and also must preserve integrity, consistency of data and should prevent data loss.

5.2. Software system attributes:

System attributes or quality factors will have the following requirements.

- The software is more reliable as it causes no damage to the system in which it works. If it leads to any malfunction, it has the capability to recover from that damage.
- The system is available at all the places where the users fell it comfortable to work with this.
- The software is as secure as it is user-friendlier and the user can use the system efficiently. It won’t cause any system attacks
- The software is easy to maintain and the updating over the system are verified daily, in order to have more secure data

- The software can be installed in any system that satisfies the hardware and software requirement of this system. It is more portable.
- The system provides easiness for the customer to use it and of course it is more user-friendly.

Appendix A: Glossary

Definitions, Acronyms and Abbreviations

- ✓ **SRS:** Software Requirement Specification
- ✓ **Client/User:** The entity who will be using the passport registration.
- ✓ **Server:** A system that runs in Linux that monitors the Foreign Trading System.
- ✓ **RAM:** Random Access Memory
- ✓ **PHP:** Hypertext Preprocessor
- ✓ **MySQL:** A relational database management system
- ✓ **HTTP:** Hyper Text Transfer Protocol
- ✓ **PDF:** Portable Document Format
- ✓ **Username:** Unique name given to each account of digital library
- ✓ **Password:** Unique word given to each user as a secret code.
- ✓ **HTML**
  - JSP
  - JavaScript
  - Java
  - XML
  - AJAX

RESULT:

Thus, the Software Requirements Specification (SRS) for the Foreign Trading System was documented successfully.

Ex. No:03	Identify the use cases and develop the Use Case model

AIM:

To identify use cases and develop the use case model for foreign trading system.

INTRODUCTION OF USE CASE DIAGRAM:

Use case diagrams are graphical representations that depict the interactions between users and system to achieve specific goals or tasks. They are an essential tool in software development for capturing and communicating system requirements in a clear and visual manner.

Purpose:

- **Requirement Analysis:** Use case diagrams help stakeholders, including developers and clients, understand the system's functionality and behavior from a user's perspective. They provide a high-level overview of the system's features and interactions.
- **Communications:** Use case diagrams facilitate communication between stakeholders by providing a common visual language to discuss system requirements and functionalities. They help ensure that everyone involved in the project has a shared understanding of the system's scope and objectives.
- **Design Validation:** Use case diagrams assist in validating the system design by identifying potential gaps, inconsistencies, or missing functionalities. They allow stakeholders to review and refine the system requirements before proceeding with the implementation phase.

Components:

- **Actors:** Actors represent external entities that interact with the system to accomplish specific goals. Actors are depicted as stick figures and are connected to use cases by association relationships.
- **Use Cases:** Use cases represent the specific functionalities or tasks that the system performs to fulfil user goals. Each use case describes a sequence of interactions between the system and actors to achieve a particular outcome. Use cases are depicted as ovals within the diagram.

- **Relationships:** Relationships between actors and use cases are depicted using association lines. These relationships indicate the interactions between actors and the system to accomplish various tasks. There are different types of relationships, including include, extend, and generalization.

Benefits:

- **Clarity:** Use case diagrams provide a visual representation of system requirements, making it easier for stakeholders to understand and visualize the system's behavior and functionality.
- **Requirements Prioritization:** Use case diagrams allow stakeholders to prioritize system requirements based on user goals and objectives. By identifying critical use cases, stakeholders can focus on implementing essential functionalities first.
- **Change Management:** Use case diagrams facilitate change management by providing a structured framework for evaluating and incorporating new requirements or modifications to existing functionalities. They help assess the impact of changes on the overall system.

Use cases diagram for Foreign Trading System:

In a Foreign Trading System, the use case diagram could include actors like "Investor", "Market Authority", "Bank Authority" and "Broker". Some potential use cases might be "Account details", "Database", "Trading Account", "Buy", "Sell" and "Market Status".

Actors:

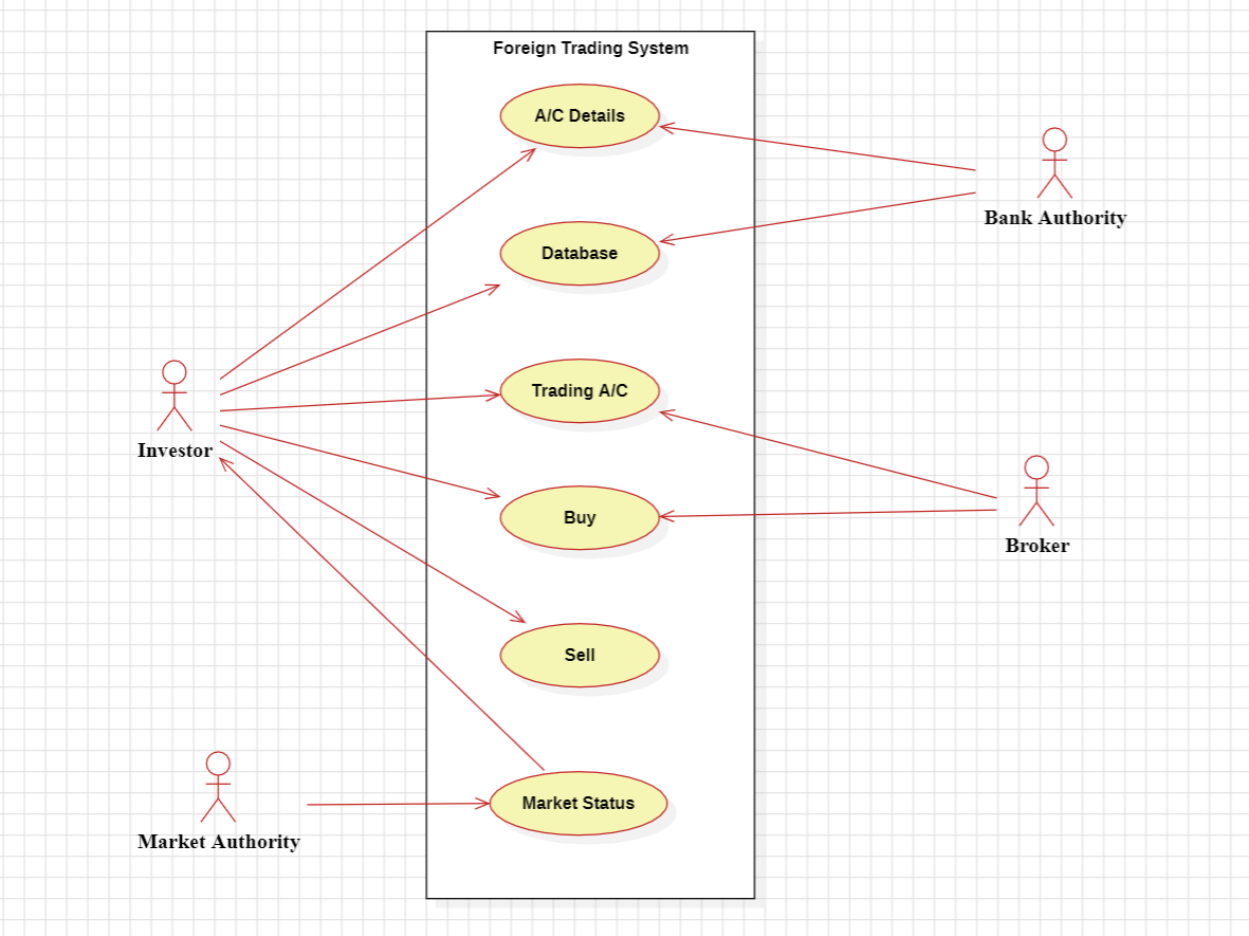
- Investor
- Market Authority
- Bank Authority
- Broker

Use Cases:

- Account Details
- Database
- Trading Account
- Buy
- Sell

- Market Status

Use Case Diagram:



Investor:

Investors in a foreign trading system are individuals or entities who participate in buying and selling financial instruments such as stocks, bonds, currencies, and commodities across international markets with the goal of earning profits or hedging against risks.

Market Authority:

Market Authority refers to regulatory bodies or organizations responsible for overseeing and regulating the operations of financial markets, ensuring compliance with laws, and maintaining fair and transparent trading practices.

Bank Authority:

Bank Authority typically refers to central banks or regulatory bodies that oversee and regulate financial institutions such as banks involved in foreign exchange transactions. They may set policies, regulations, and guidelines for banks participating in foreign trading activities to ensure stability and compliance with financial regulations.

Broker:

A broker is a financial intermediary or agent who facilitates buying and selling transactions of securities, commodities, or other financial assets on behalf of clients in exchange for a commission or fee.

Account Details:

Information including but not limited to user identification, financial balances, transaction history, and personal information associated with a financial or online account.

Database:

A structured collection of data organized for efficient retrieval, storage, and manipulation, often used to store account details, transaction records, and other relevant information in financial systems.

Trading account:

An account specifically designated for buying and selling financial instruments such as stocks, bonds, currencies, and commodities, typically linked to a brokerage or financial institution.

Buy:

The action of acquiring a financial instrument or asset with the expectation of its value increasing over time, often associated with a long position in trading.

Market status:

The current condition or state of a financial market, indicating factors such as price movements, trading volume, liquidity, volatility, and overall market sentiment.

RESULT:

Thus, the Use Case Model was identified and developed successfully.

Ex. No:04	Identify the Conceptual classes and develop a domain model and also derive a Class Diagram

AIM:

To identify the conceptual classes and develop a Domain Model and also derive a Class diagram from the Foreign Trading System.

CONCEPTUAL CLASSES:

In the Foreign Trading System, several conceptual classes can be identified to represent the various entities and functionalities of the system. Here are some key conceptual classes for the Foreign Trading System:

i. Investor:

Represents individuals or entities participating in the foreign trading system, submitting orders to buy or sell financial instruments.

ii. Broker:

Represents brokerage firms or individuals that act as intermediaries, executing trades on behalf of investors.

iii. Market Authority:

Represents regulatory bodies or entities overseeing and maintaining the integrity of the trading market.

iv. Bank Authority:

Represents financial institutions involved in the foreign trading system, facilitating fund transfers, settlements, and providing banking services.

DOMAIN MODEL FOR FOREIGN TRADING SYSTEM:

Investor:

- **Attributes:**
  - AC number
  - Login Password
- **Operations:**
  - submit()
  - reset()

- cancel()

Broker:

❖ **Attributes:**

- Buy
- Sell

❖ **Operations:**

- renew()
- trading()
- retrieve()

MarketAuthority:

❖ **Attributes:**

- Date
- Current Price

❖ **Operations:**

- SendStatus()

BankAuthority:

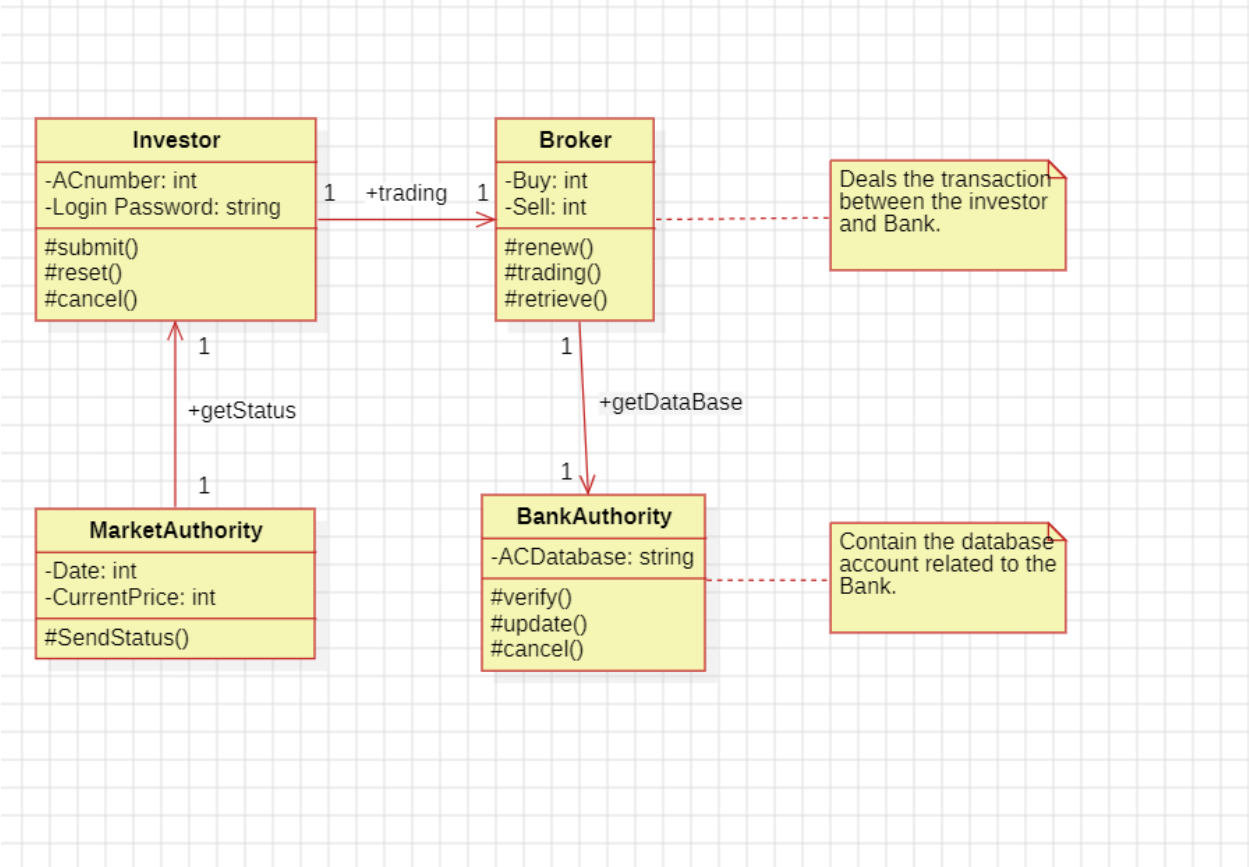
❖ **Attributes:**

- AC Database

❖ **Operations:**

- verify()
- update()
- cancel()

CLASS DIAGRAM FOR FOREIGN TRADING SYSTEM:



RESULT:

Thus, the conceptual classes, Domain Model and Class Diagram for foreign trading system was identified and developed successfully.

Ex.No:05	Using the identified scenarios, find the interaction between objects and represent them using UML Sequence and Collaboration Diagram

AIM:

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

INTRODUCTION OF INTERACTION DIAGRAM:

An interaction diagram is a visual representation that illustrates how objects in a system interact and communicate with each other over time, typically used in software design and modeling to depict the flow of messages or method calls between objects during the execution of a use case or scenario.

There are two main types of interaction diagrams commonly used in software development:

- Sequence Diagram
- Communication (or) Collaboration Diagram

SEQUENCE DIAGRAM :

A sequence diagram is a type of interaction diagram in UML (Unified Modeling Language) that shows how objects interact in a particular scenario or use case over time. It illustrates the sequence of messages exchanged between objects, along with their lifelines, to depict the flow of control and communication in a system or software application.

A sequence diagram is a type of interaction diagram that illustrates the flow of messages or method calls between objects in a system over time. Its main components include:

Objects:

Represented by rectangles with the object name at the top, indicating the entities or instances participating in the interaction.

Lifelines:

Vertical dashed lines extending from each object's rectangle, representing the object's existence and lifespan during the interaction.

Messages:

Arrows indicating communication or interaction between objects, with labels specifying the type of message. There are different types of messages,

- Synchronous Message
- Asynchronous Message
- Return Message

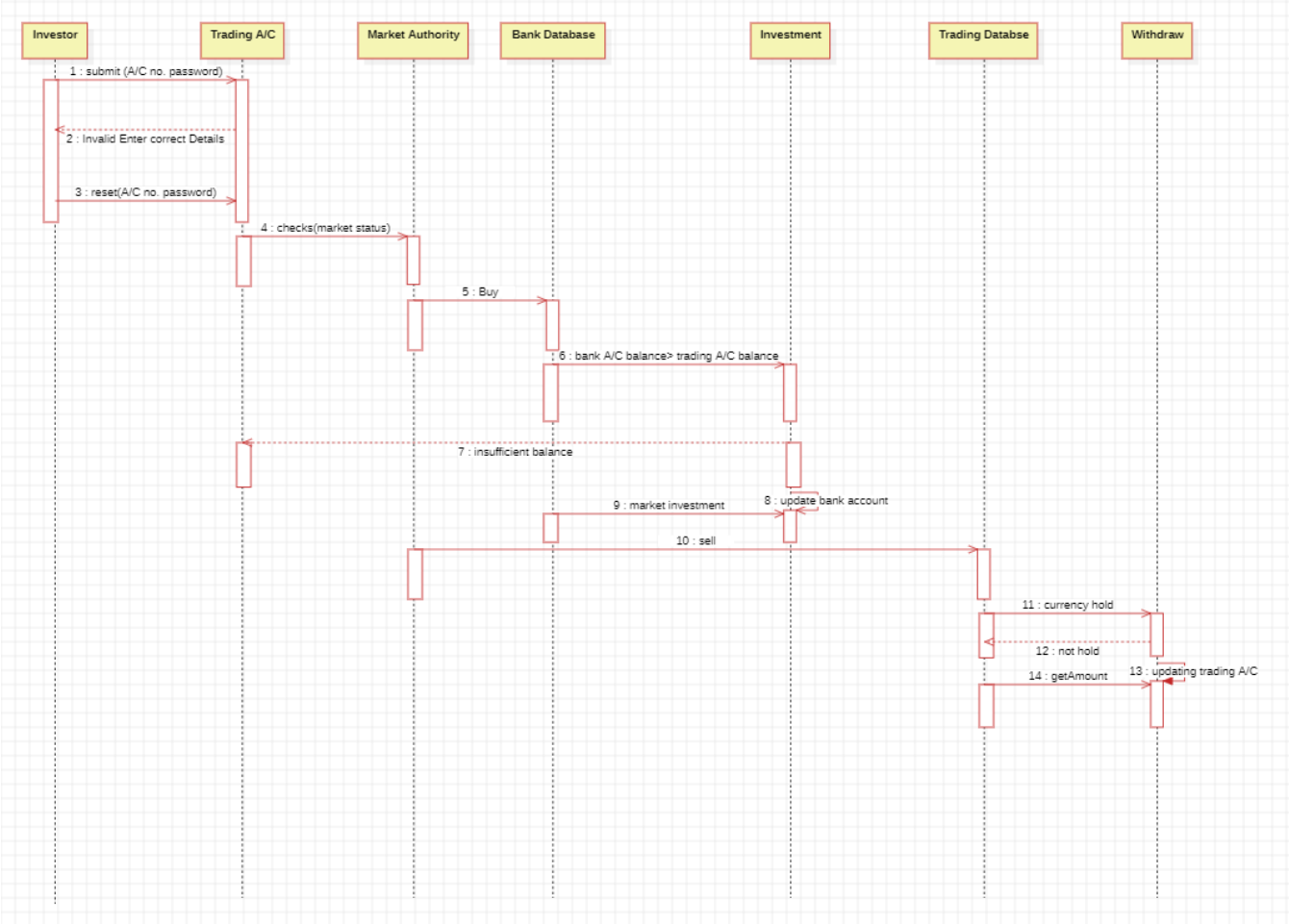
Activation Boxes:

Horizontal bars on lifelines representing the period when an object is actively processing or executing a message.

Focus of Control:

Indicates the flow of control between objects during the interaction, showing which object is actively performing actions or responding to messages at any given time.

Sequence Diagram for Foreign Trading System:



COLLABORATION DIAGRAM:

A collaboration diagram, also known as a communication diagram in UML (Unified Modeling Language), is a type of interaction diagram that visualizes the interactions and relationships between objects or actors in a system or software application. It emphasizes the communication flow and messages exchanged between objects to achieve specific functionalities or scenarios within the system.

Collaboration diagrams are used to visualize and model the interactions, collaborations, and dependencies between objects in a system, helping to understand the dynamic behavior and communication patterns within the software or system architecture.

A collaboration diagram is an interaction diagram in UML (Unified Modeling Language) that illustrates the interactions and relationships between objects in a system or software application. Its main components include:

Objects:

Represented by rectangles with the object's name, showing the entities or instances participating in the collaboration.

Links:

Lines connecting objects to represent associations or relationships between them, indicating how objects interact or communicate.

Messages:

Arrows indicating the flow of communication or interactions between objects, with labels specifying the type of message (e.g., synchronous, asynchronous).

Roles:

Descriptions or labels attached to links or arrows indicating the role or purpose of the interaction between objects.

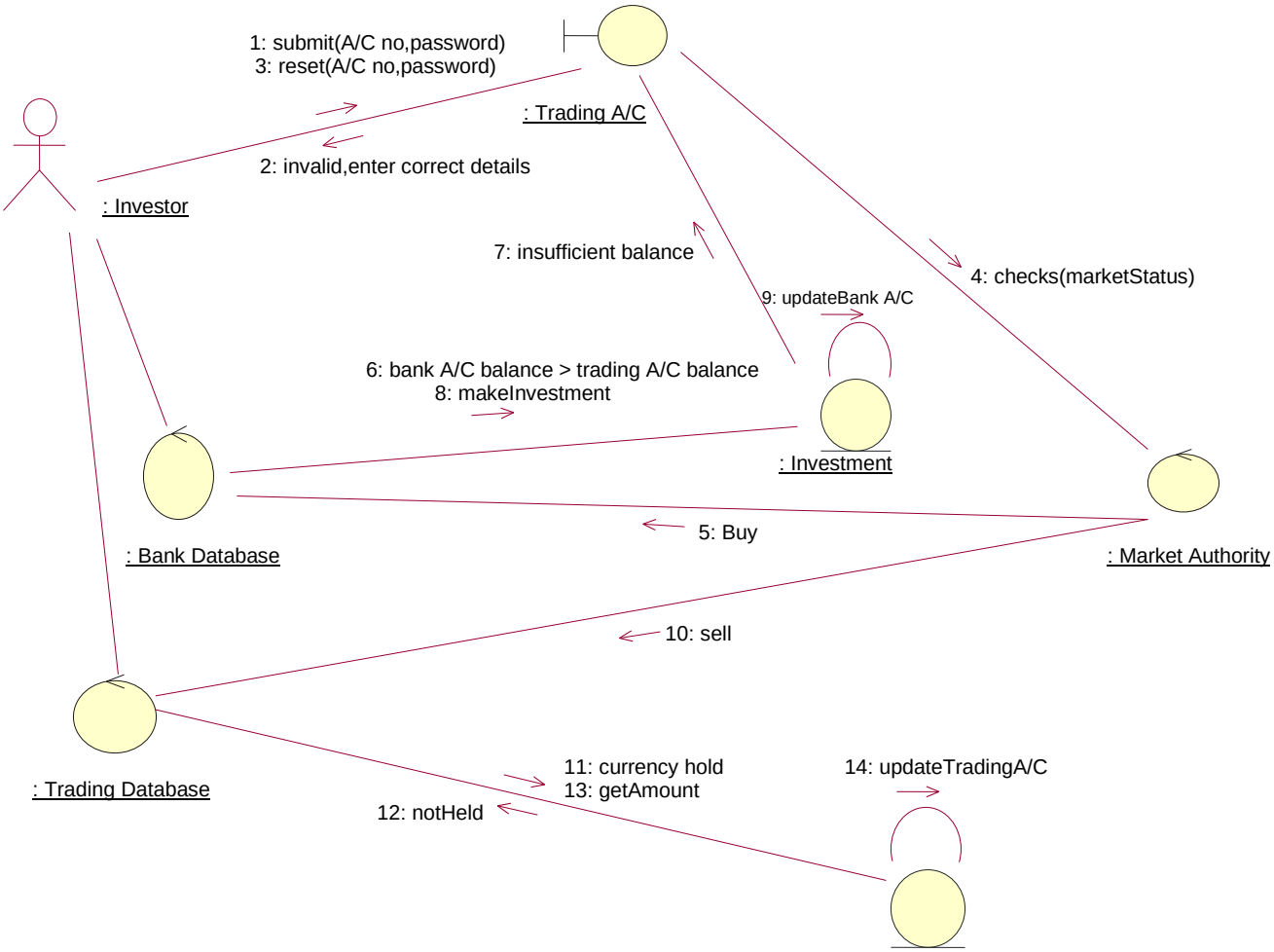
Multiplicity:

Optional notation on links or arrows indicating the cardinality or multiplicity of the relationship between objects (e.g., one-to-one, one-to-many).

Notes:

Additional information or comments placed in rectangular boxes attached to relevant parts of the diagram, providing explanations or details about the collaboration.

Collaboration Diagram for Foreign Trading System:



RESULT:

Thus, the interaction between objects and represent them using UML Sequence and Collaboration Diagrams were identified successfully.

Ex. No:06	Draw relevant State Chart and Activity Diagrams for the Foreign Trading System

AIM:

To draw the relevant State chart diagram and Activity diagram for the Foreign Trading system.

INTRODUCTION TO STATE CHART DIAGRAM:

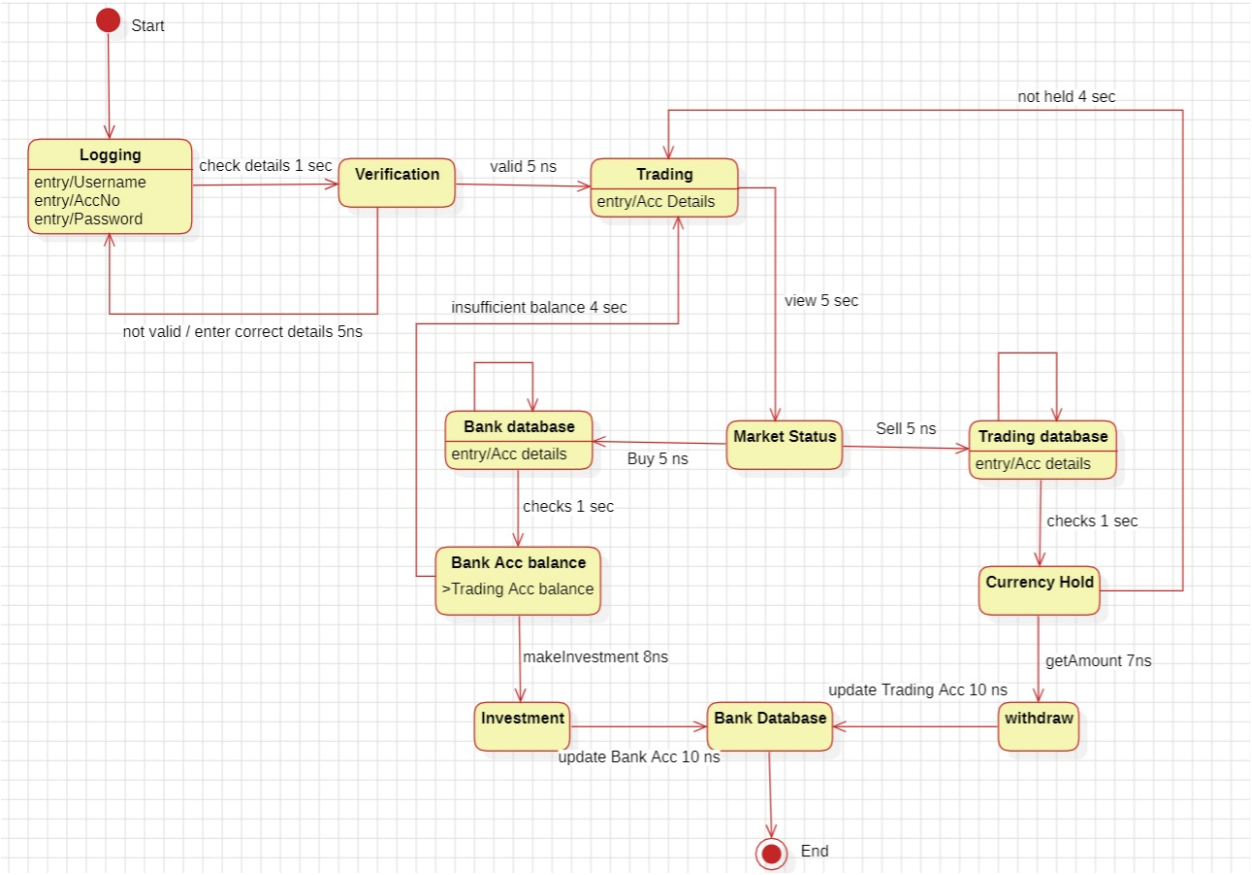
A state chart, also known as a state machine diagram, is a visual representation used in software engineering to model the behavior of a system or object over time. It depicts the various states that an object can be in and the transitions between these states based on events or conditions. State charts are particularly useful for modeling complex systems with multiple states and defining the sequence of actions or operations that occur as the system transitions between states.

COMPONENTS:

- **States:** States represent the different conditions or situations that an object or system can be in at any given time. Each state is typically depicted as a rounded rectangle with a name inside, describing the state (e.g., "Idle," "Active," "Stopped").
- **Transitions:** Transitions indicate the movement or change of state that occurs in response to events or conditions. They are represented by arrows between states, with labels describing the triggering event or condition that causes the transition (e.g., "Start," "Stop," "Timeout").
- **Events:** Events are external stimuli or triggers that initiate state transitions. They can be user actions, system inputs, or timed events that cause the system to change from one state to another (e.g., "Button Clicked," "Data Received," "Timer Expired").
- **Actions:** Actions represent the operations or behaviors that occur when a state transition takes place. They are often associated with transitions and can include tasks, computations, or changes in system variables (e.g., "Initialize System," "Update Data," "Send Notification").
- **Initial State:** The initial state is the starting point of the state chart diagram, indicating the initial condition or state of the system when it first begins execution. It is typically denoted by a filled circle connected to the initial state with a transition arrow.
- **Final State:** The final state represents the end or termination point of the state chart, indicating that the system has completed its process or reached a final state where no further transitions occur. It is usually denoted by a filled circle with a dot inside or a rounded rectangle with the word "End."

- **Composite State:** Composite states are hierarchical states that contain sub-states within them. They allow for the organization and nesting of states to represent complex behavior more effectively. Composite states are depicted as a larger state box containing nested states.

STATE CHART DIAGRAM:



INTRODUCTION TO ACTIVITY DIAGRAM:

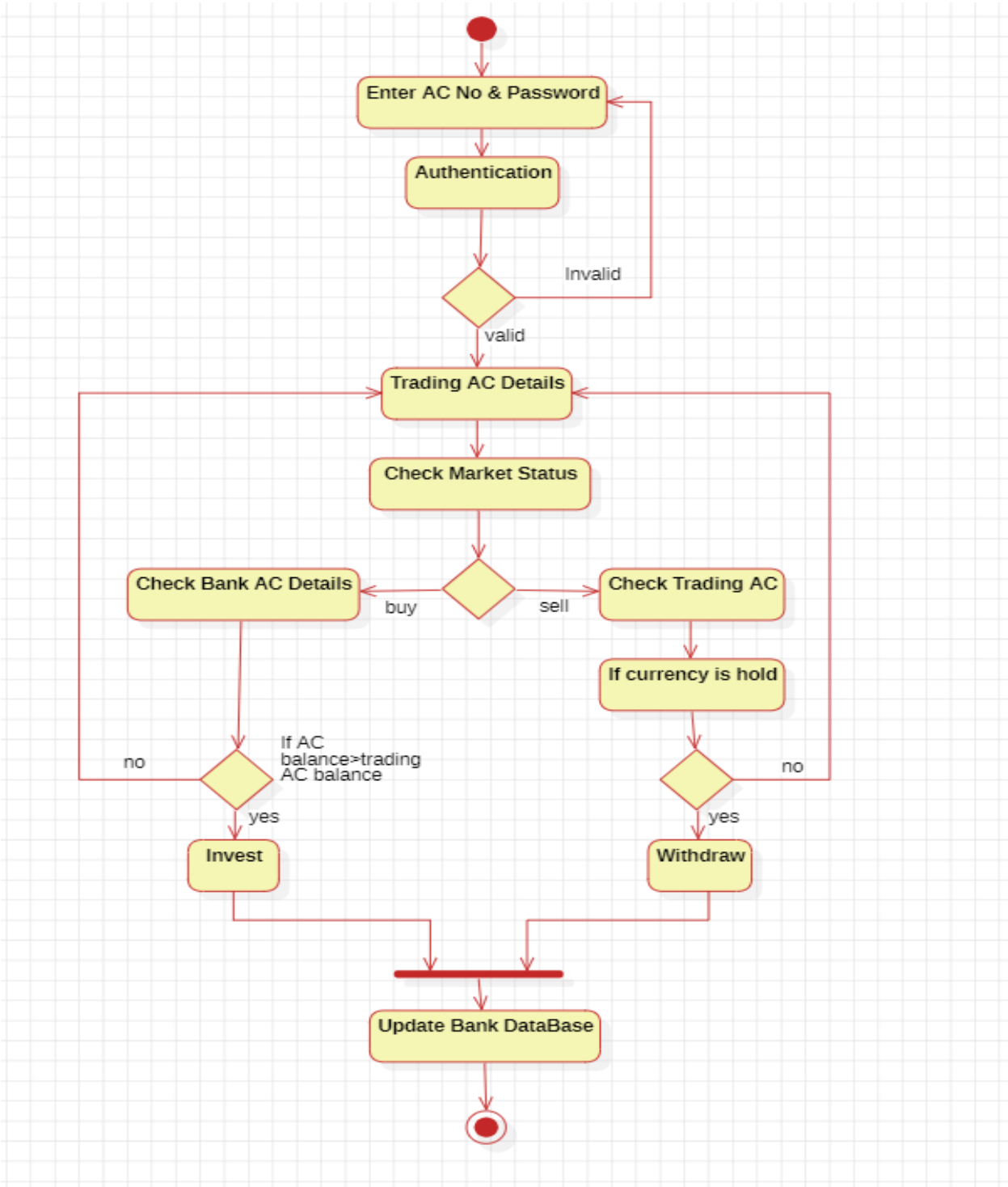
Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We can depict both sequential processing and concurrent processing of activities using an activity diagram i.e an activity diagram focuses on the condition of flow and the sequence in which it happens.

COMPONENTS:

- **Initial State:** The initial state is the starting point of the state chart diagram, indicating the initial condition or state of the system when it first begins execution. It is typically denoted by a solid circle or a filled circle.

- **Action or Activity State:** An activity represents execution of an action on objects or by objects. It is represented by a rectangular shape with rounded corners.
- **Action Flow or Control flows**: Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another activity state. They are represented by solid lines with arrows indicating the direction of flow.
- **Decision node and Branching:** Also known as a decision point or branch point, this node represents a decision to be made within the process flow. It typically involves evaluating a condition and branching the flow based on the result. It is represented by a diamond shape.
- **Fork:** This node is used to split the flow of control into multiple parallel paths, allowing activities to be performed concurrently. It is represented by a solid line with a single incoming flow and multiple outgoing flows.
- **Join:** The join node is used to synchronize multiple parallel paths back into a single path. It is represented by a solid line with multiple incoming flows and a single outgoing flow.
- **Final State:** This represents the end point of the activity diagram. It is typically depicted as a solid circle with a border or an unfilled circle.
- **Object Flow:** Object flows represent the flow of objects or data between activities or nodes in the activity diagram. They are represented by dashed lines with arrows indicating the direction of flow.

ACTIVITY DIAGRAM:



RESULT:

Thus the relevant State chart diagram and Activity diagram for the Foreign Trading system was drawn successfully.

Ex. No:07	Implement the Foreign Trading system as per the detailed design

AIM:

To implement the system as per the detailed design.

INTRODUCTION:

The foreign trading system is a crucial platform that facilitates global trade by enabling the exchange of goods and services between countries. It plays a pivotal role in promoting economic growth and fostering international relationships. Through advanced technologies and financial instruments, the foreign trading system streamlines transactions, reduces barriers to trade, and enhances market efficiency. Its impact extends across industries, influencing everything from import-export businesses to multinational corporations.

PROBLEM STATEMENTS:

1. **Currency Exchange Fluctuations:**

Rapid changes in currency exchange rates can lead to financial uncertainties and impact the profitability of international trade transactions.

2. **Trade Compliance and Regulatory Challenges:**

Adhering to diverse trade regulations, customs procedures, and sanctions imposed by different countries poses complexity and compliance risks for businesses.

3. **Supply Chain Disruptions:**

Delays or disruptions in the global supply chain, such as transportation bottlenecks or geopolitical tensions, can affect the timely delivery of goods and services, leading to customer dissatisfaction.

4. **Market Volatility:**

Uncertainties in global markets, including fluctuations in commodity prices and geopolitical events, can significantly influence trade dynamics and investment decisions.

5. **Payment and Settlement Issues:**

Issues related to payment methods, currency conversions, and cross-border transaction costs can create challenges in completing trade transactions efficiently and securely.

6. **Data Security and Cyber Threats:**

Protecting sensitive trade data and mitigating cyber threats, such as data breaches or hacking attempts, is crucial to maintaining trust and confidentiality in the foreign trading system.

SOFTWARE REQUIREMENTS:

1. Operating System:

Choose a stable and widely supported operating system such as Linux (e.g., Ubuntu, CentOS) for server environments to ensure reliability and security.

2. Database Management System (DBMS):

Utilize a scalable and efficient DBMS like PostgreSQL, MySQL, or Oracle for storing and managing trade data, ensuring data integrity and performance.

3. Web Server:

Use Apache HTTP Server or Nginx as the web server to host the trading platform's web interface, ensuring fast and reliable access for users.

4. Programming and Frameworks:

Develop the trading platform using robust programming languages and frameworks such as Java (Spring Boot), Python (Django), or Node.js (Express) for scalability and maintainability.

5. Authentication and Authorization:

Implement secure authentication and authorization mechanisms using protocols like OAuth 2.0 or JWT (JSON Web Tokens) to control access to trading features based on user roles and permissions.

6. Frontend Technologies:

Use modern frontend technologies like React, Angular, or Vue.js to create responsive and interactive user interfaces for traders and administrators.

7. Security Tools:

Employ security tools such as SSL/TLS certificates for encrypted communication, intrusion detection systems (IDS), and firewalls to protect against cyber threats and ensure data privacy and confidentiality.

HARDWARE REQUIREMENTS:

1. Server Infrastructure:

High-performance servers with multi-core processors (e.g., Intel Xeon) and ample RAM (at least 16GB or more) to handle concurrent user requests and data processing tasks efficiently.

2. Storage:

Fast and reliable storage solutions such as Solid State Drives (SSDs) or high-speed hard disk drives (HDDs) with RAID configurations for data redundancy and improved read/write speeds.

3. Network Equipment:

Gigabit Ethernet switches and routers with sufficient bandwidth capacity to support high-speed data transfers and seamless connectivity between trading platforms, databases, and external networks.

4. Backup Systems:

Implement automated backup systems with redundant storage arrays or cloud backup services to ensure data availability and disaster recovery in case of hardware failures or data loss incidents.

5. Monitoring Tools:

Install monitoring tools and systems (e.g., Network Monitoring Software, Server Monitoring Tools) to track hardware performance metrics, detect anomalies, and proactively address issues to maintain system availability and reliability.

6. Security Hardware:

Deploy hardware-based security appliances such as Unified Threat Management (UTM) devices, Intrusion Prevention Systems (IPS), and Next-Generation Firewalls (NGFW) to protect against cyber threats and unauthorized access attempts.

7. Client Devices:

Modern desktop computers, laptops, or mobile devices with sufficient processing power, memory, and graphics capabilities to run the trading platform's frontend interface smoothly.

IMPLEMENTATION:

BankAuthority:

```
import java.util.*;
public class BankAuthority {
   public BankAuthority() {
   }
   private void AC Database;
   protected void verify() {   }
   protected void update() {    }
   protected void cancel() {    }
}
```

Broker:

```
import java.util.*;
public class Broker {
   public Broker() {
   }
   private void Buy;
   private void Sell;
   protected void renew() {    }
   protected void trading() {    }
   protected void retrieve() {    }
}
```

Investor:

```
import java.util.*;
public class Investor {
   public Investor() {
   }
   private void AC no;
   private void Login Password;
   protected void submit() {    }
   protected void reset() {    }
   protected void cancel() {    }
}
```

MarketAuthority:

```
import java.util.*;
public class MarketAuthority {
   public MarketAuthority() {
   }
   private void Date;
   private void CurrentPrice;
   protected void SendStatus() {    }
}
```

RESULT:

Thus, the detailed design of Foreign Trading System was implemented successfully.

Ex. No:08	Test the Software System for all the scenarios identified as per the use case diagram

AIM:

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

Foreign Trading System:

In today's generation, the foreign trading system has evolved into a highly digitized and interconnected network, leveraging advanced technologies such as artificial intelligence, big data analytics, and blockchain. Traders and investors can access global markets in real-time, execute transactions with speed and precision, and analyze vast amounts of market data for informed decision-making.

Automation and algorithmic trading algorithms play a significant role, optimizing trade execution, reducing latency, and enhancing market liquidity. Additionally, regulatory compliance, cybersecurity measures, and risk management strategies are paramount to ensure trust, security, and stability in the modern foreign trading ecosystem.

I can provide you with a general approach to testing the software system for the scenarios identified in the use case diagram for Foreign Trading System.

Review Use Case Diagram:

- Review the use case diagram by gaining a deep understanding of the depicted interactions and functionalities.
- Identify primary actors and their specific goals within the system to ensure accurate representation and alignment with business requirements.

Identify Test Scenarios:

- Identify test scenarios by exploring various perspectives, including those of different user roles, to ensure comprehensive coverage.
- 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:

- Create test cases by translating identified test scenarios into detailed steps, inputs, expected outputs, and test conditions.
- Include positive (where the system behaves as expected) and negative (where errors or exceptions are expected) test cases, boundary tests, and error-handling scenarios to thoroughly validate the system's functionality and robustness.

Prepare Test Environment:

- Prepare the test environment by establishing a dedicated setup that isolates testing activities from the production system, minimizing risks of interference or disruption.
- Validate that the test environment replicates the production environment's configuration and data, ensuring accurate testing of system behavior and performance under realistic conditions.

Execute Test Cases:

- Execute test cases diligently by adhering to the predefined steps and documenting both the actions performed and the observed outcomes.
- Utilize diverse test data to cover a range of scenarios, including boundary values and invalid inputs, to comprehensively validate the system's functionality and behavior.

Report Bugs:

- Report bugs by documenting 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:

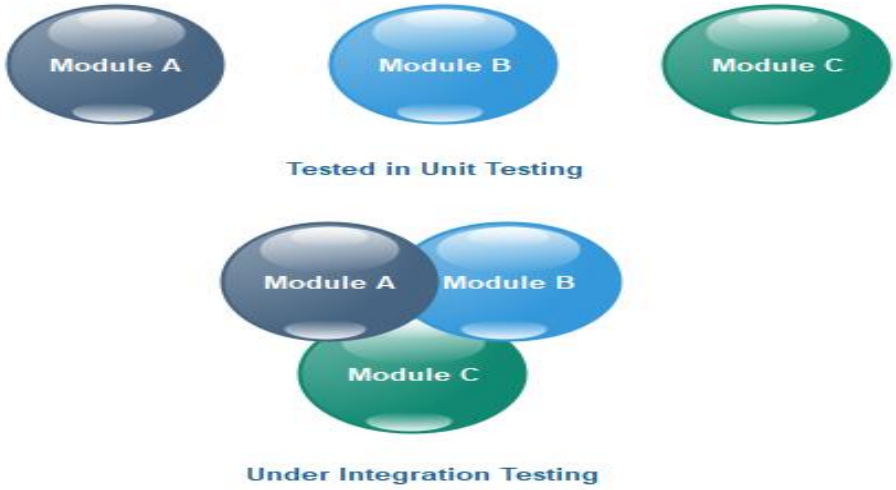
- Conduct comprehensive integration testing to verify the seamless interaction and functionality of different modules and components within the Foreign Trading System.
- Verify data integrity and consistency across integrated modules during integration testing, ensuring accurate information flow and processing.

- Test APIs and interfaces between modules in the FTS to validate data exchange, communication protocols, and system interoperability.

Example:

Components to Integrate: Trade execution module, Risk management module

Test Case: Execute a trade and verify that risk management protocols are applied correctly, ensuring compliance with regulatory requirements and seamless transaction processing.



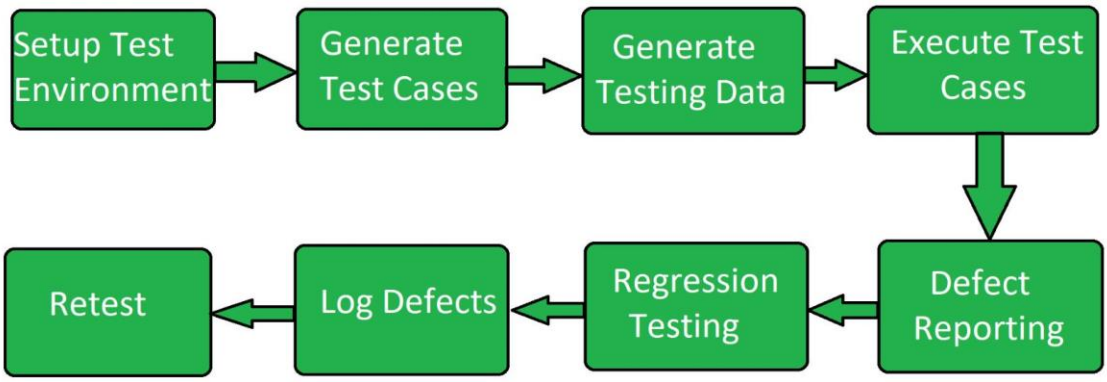
System Testing:

- System testing assesses the entire foreign trading system to validate its behavior and performance against defined requirements and standards.
- It includes testing end-to-end scenarios such as trade execution, order processing, risk assessment, and reporting functionalities to ensure the system operates seamlessly under various conditions.

Example:

End-to-End Workflow: User interaction with the Foreign Trading System

Test Case: Simulate a trading session where a user logs in, places an order, verifies trade execution, checks risk assessment, and generates a trade report. Confirm that each step of the workflow functions correctly and meets regulatory compliance.



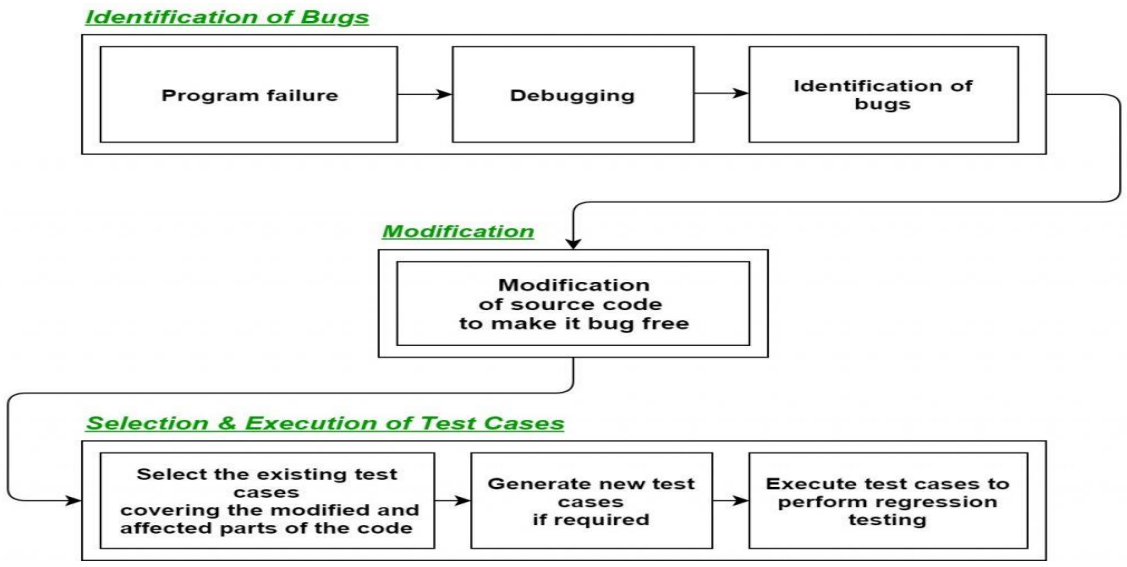
Regression Testing:

- Regression testing is critical for a Foreign Trading System to identify and prevent the introduction of new defects or issues when modifications or enhancements are applied.
- It involves re-executing previously conducted tests to ensure that existing functionalities still operate correctly after changes are made, thereby preserving system stability and reliability.

Example:

Scenario Change: Enhance the trade execution module to support new order types

Test Case: After enhancing the trade execution module, re-run previous test cases (e.g., order placement, risk assessment) to verify that existing functionalities remain intact and function as expected without any adverse impacts.



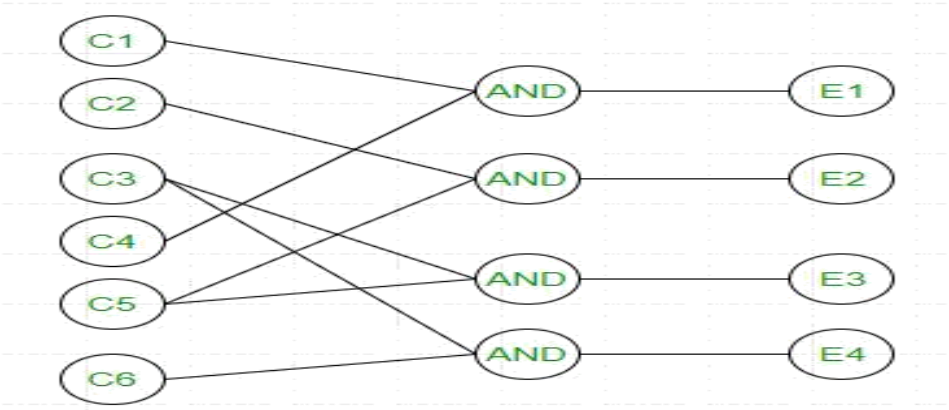
Black Box Testing:

- Black box testing concentrates on assessing the functionalities of the Foreign Trading System without delving into its internal structure or implementation specifics.
- It validates user-facing features like order placement, trade execution, and reporting functionalities to ensure they align with defined requirements and expectations.

Example:

Functionality to Test: Order Placement

Test Case: Input valid order details (e.g., quantity, price, instrument) and verify that the system correctly processes the order and updates relevant information in the trade logs and risk assessment 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

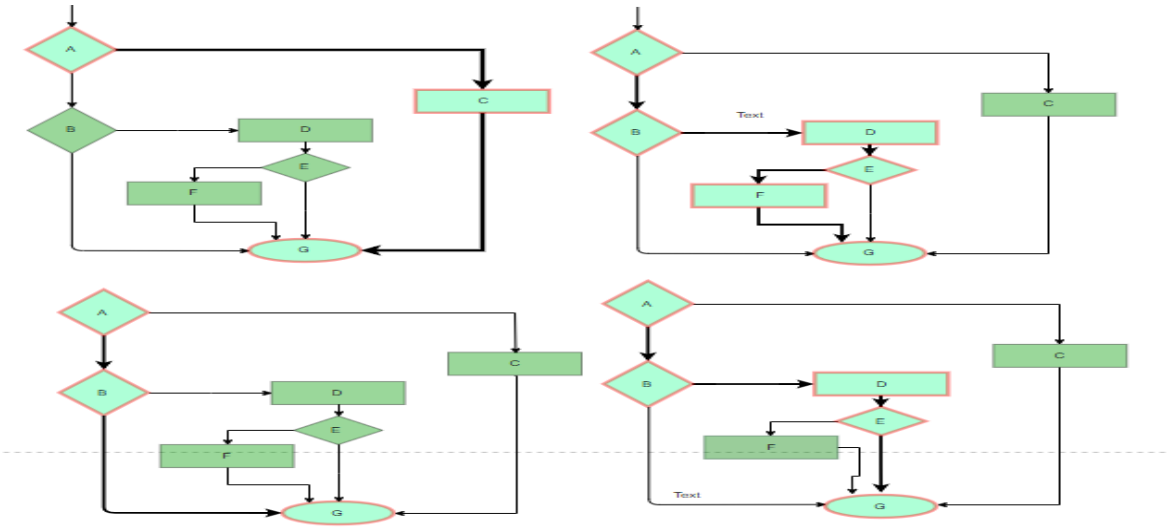
White Box Testing:

- White box testing entails analyzing the internal structure and logic of the Foreign Trading 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: Trade Execution Algorithm

Test Case: Validate that the system correctly executes trades according to predefined rules (e.g., order priority, market conditions), handles exceptions such as invalid orders, and updates trade status and risk metrics accordingly.



Unit Testing:

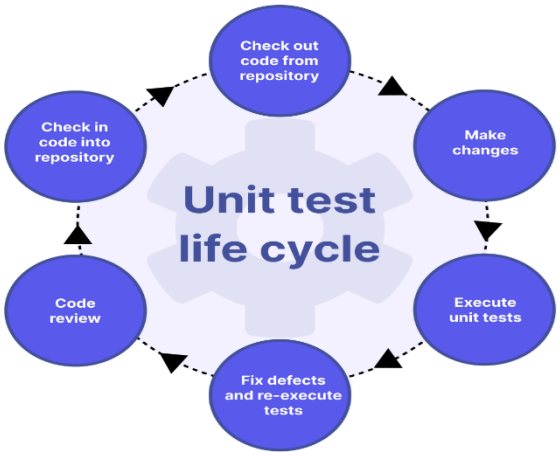
- Unit testing is highly suitable for verifying the functionality of individual components or units within the Foreign Trading System (e.g., modules, classes, functions) in isolation.
- It ensures that critical functionalities such as order processing, risk assessment algorithms, and trade execution mechanisms operate correctly at a granular level.

Example:

Component: Order Processing Module

Functionality to Test: validateOrder() function

Test Case: Confirm that the validateOrder() function within the Order Processing Module correctly validates order details (e.g., quantity, price, instrument) against predefined rules and returns the expected validation result.



RESULT:

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

Ex. No:09	Improve the Reusability and Maintainability of the Software System by Applying Appropriate Design Patterns

Aim:

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

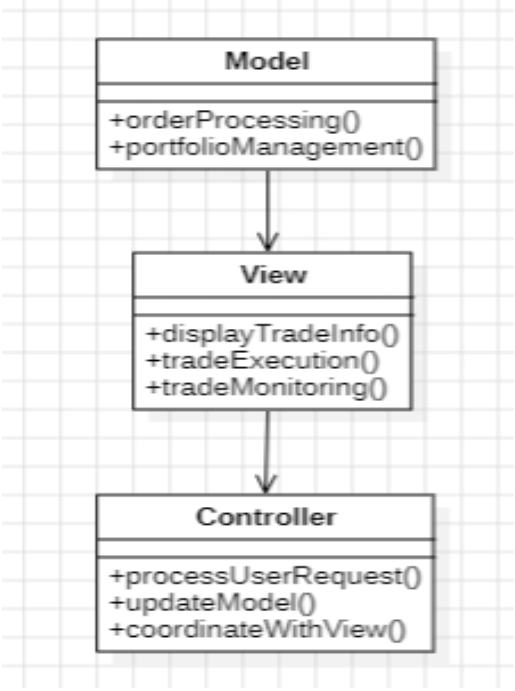
Foreign Trading System:

A Foreign Trading System is a software application that facilitates trading operations between countries or entities across different nations. It includes user registration, login, product listing, search, order placement, payment processing, currency conversion, inventory management, order tracking, feedback/rating systems, security, trade regulation compliance, and customer support.

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

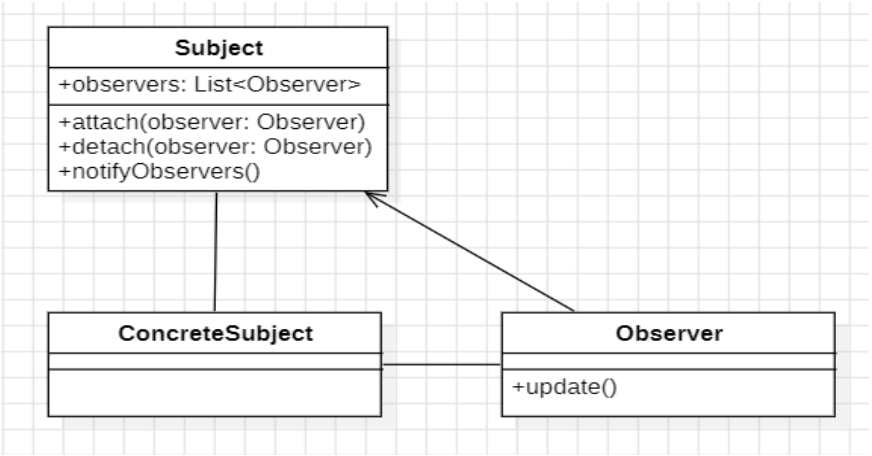
1. Model-View-Controller Pattern:

- The Model represents trade data and business logic, managing operations like order processing and portfolio management.
- The View displays trade information to users, providing interfaces for trade execution and monitoring.
- The Controller processes user requests, updates the model based on trading actions, and coordinates with the view to present real-time trade updates and status.
  - ❖ Context: MVC separates trade data management, UI presentation, and user interaction.
  - ❖ Problem: Without MVC, code complexity increases, making maintenance difficult.
  - ❖ Solution: MVC divides the system into Model, View, and Controller, simplifying development and improving code organization.



2. Observer Pattern:

- The Observer Pattern notifies components about system changes, enhancing flexibility and scalability. It enables easy addition of new notifications, improving system maintainability.
  - ❖ Context: Used when objects need to be notified of changes in another object's state.
  - ❖ Problem: Establishes a one-to-many relationship without tightly coupling objects, ensuring automatic updates.
  - ❖ Solution: Define a subject interface with methods for attaching, detaching, and notifying observers. Observers register with subjects to receive updates on state changes.



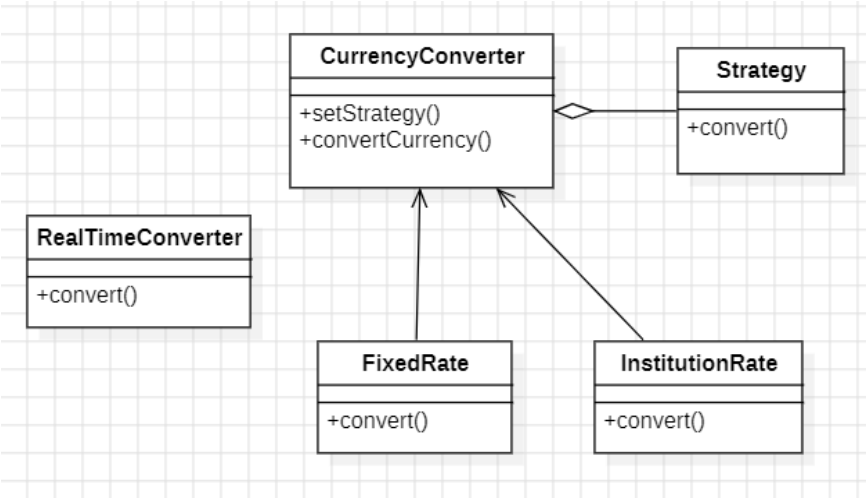
3. Publish-Subscribe Pattern:

- The Publish-Subscribe Pattern enables real-time updates and notifications across multiple components in trading systems.
- This pattern enhances scalability, flexibility, and responsiveness within the trading system architecture.
  - ❖ Context: Real-time updates are crucial in trading operations.
  - ❖ Problem: Distributing real-time market data without the Publish-Subscribe Pattern leads to data latency and inefficiencies.
  - ❖ Solution: The pattern enables efficient data distribution, letting providers publish to specific topics, ensuring relevant updates reach subscribing modules, enhancing system scalability and responsiveness.

4. Strategy Pattern:

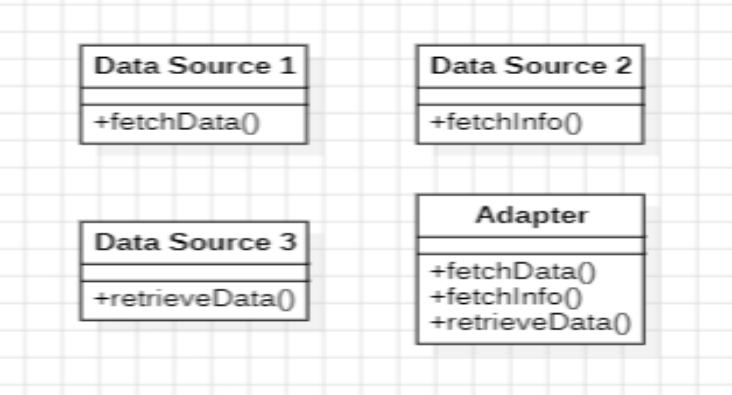
- The Strategy Pattern can be employed to handle different algorithms for international currency conversion.
- By implementing the Strategy Pattern, the system allows for interchangeable algorithms, providing flexibility and enabling users to select the most suitable currency conversion strategy.
  - ❖ Context: Used to manage a family of algorithms interchangeably.

- ❖ Problem: Facilitates dynamic algorithm selection and switching, promoting code flexibility and reusability.
- ❖ Solution: Define a strategy interface with algorithm methods. Implement concrete strategy classes for each algorithm and allow runtime switching between them.



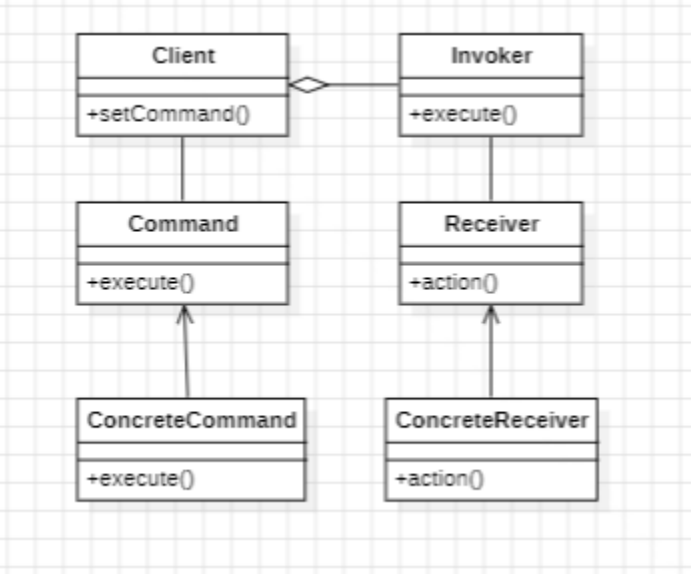
5.Adapter Pattern:

- The Adapter Pattern is used to integrate different trading data sources seamlessly.
- It solves the problem of incompatible interfaces between data sources and the trading system by providing a wrapper that translates data formats and protocols.
- This pattern ensures smooth communication and data exchange, enhancing interoperability and flexibility in the trading ecosystem.
  - ❖ Context: Integrating diverse data sources with different formats and protocols is common.
  - ❖ Problem: Incompatibility between these data sources and the trading system leads to communication issues and data parsing errors.
  - ❖ Solution: Adapter Pattern provides a wrapper translating data formats and protocols for seamless integration without compromising data integrity or communication.



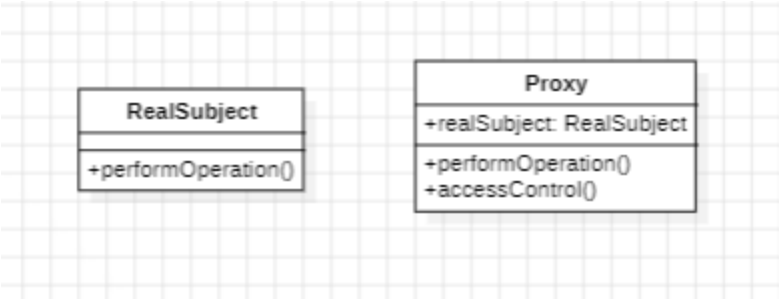
6. Command Pattern:

- The Command Pattern encapsulates requests as objects, allowing 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, enhancing maintainability, scalability, and reliability within the system.
  - ❖ Context: Used 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.



7. Proxy pattern:

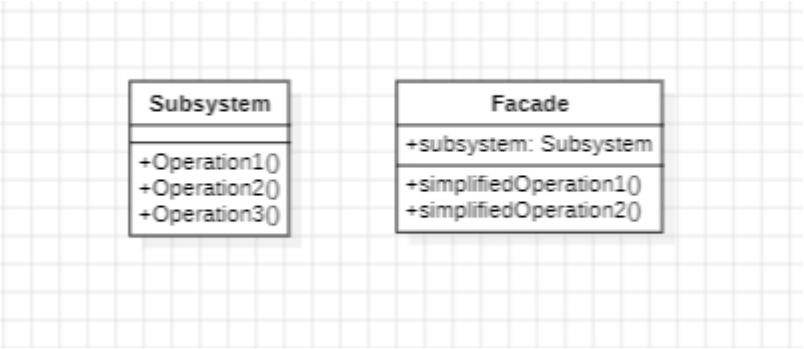
- The Proxy Pattern is used to control access to sensitive trading data or expensive operations.
- It solves the problem of directly accessing such resources by providing a surrogate object that acts as a representative, managing access and adding functionalities like caching or logging.
  - ❖ Context: Need to control access to sensitive trading data or manage expensive operations.
  - ❖ Problem: Direct access to such resources can lead to security risks or performance issues.
  - ❖ Solution: Proxy Pattern provides a surrogate object managing access, adding functionalities like caching or logging, enhancing security, performance, and resource management in the trading system.



8. Facade pattern:

- In a Foreign Trading System, the Facade Pattern simplifies interactions with complex subsystems by providing a unified interface.
- It solves the problem 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 incorporating these design patterns into the Foreign Trading System, we can improve the reusability and maintainability by promoting code reuse, encapsulating complexity, and decoupling components. This leads to a more modular, flexible, and maintainable architecture, making it easier to extend, modify, and maintain the system over time.

Result:

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