



RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

Laboratory Record Note Book

NAME

BRANCH

UNIVERSITY REGISTER No.

COLLEGE ROLL No.

SEMESTER

ACADEMIC YEAR



RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

BONAFIDE CERTIFICATE

NAME

ACADEMIC YEAR SEMESTERBRANCH.....

UNIVERSITY REGISTER No.

Certified that this is the bonafide record of work done by the above student in the

..... Laboratory during the year 20 - 20

Signature of Faculty - in - Charge

Submitted for the Practical Examination held on

Internal Examiner

External Examiner

INDEX

S.No.	Name of the Experiment	Expt. Date	Faculty Sign
1.	Preparing Problem Statement		
2.	Software Requirement Specification (SRS)		
3.	Entity-Relational Diagram		
4.	Data Flow Diagram		
5.	Use Case Diagram		
6.	Activity Diagram		
7.	State Chart Diagram		
8.	Sequence Diagram		
9.	Collaboration Diagram		
10.	Class Diagram		

EXPT NO : 1

DATE :

Writing the Complete Problem Statement

Aim :

To design and develop a Blood Bank Management System that efficiently manages and automates the process of blood donation, storage, and distribution, ensuring quick and easy access to blood and related information.

Algorithm :

1. Initialize the system with basic data and structure.
2. Accept Donor Registration:
 - Capture details like donor name, age, blood type, contact info, etc.
 - Verify donor eligibility (age, health conditions).
3. Store Donor and Blood Information:
 - Store blood type, quantity, and donor information in a database.
4. Accept Requests for Blood:
 - Capture details of the blood recipient (blood group, volume needed, urgency).
 - Check availability in the inventory.
5. Process Blood Requests:
 - If blood is available:
 - Deduct the quantity from inventory.
 - Notify the requester.
 - If unavailable:
 - Add the request to a waiting list.
6. Generate Reports:
 - Produce reports on blood inventory, donation history, and requests.
7. Close System and Update Database.

Objectives

1. To streamline the process of blood donation and distribution.
2. To maintain an updated database of donors, recipients, and blood inventory.
3. To efficiently manage blood requests and donations.
4. To provide quick accessibility to blood stocks in times of emergency.

5. To facilitate easy communication with donors and recipients.

Requirements :

Functional Requirements :

- User registration and login for donors and hospital staff.
- Blood inventory management.
- Blood donation scheduling.
- Real-time availability tracking.
- Notifications for blood requests and donation drives.
- Reporting system for generating various data insights.

Non-Functional Requirements :

- Reliability: System must have high accuracy in inventory tracking and user information.
- Scalability: Should support a large number of users and database expansion.
- Usability: Interface should be user-friendly, especially for non-technical users.
- Security: Ensure data privacy for donors and recipients.

Problem Statement :

The current blood bank management process is often manual, leading to inefficiencies, errors, and delays in handling emergency blood requests. Many blood banks struggle to maintain accurate records of available blood types, donor eligibility, and recipient needs, which can hinder timely assistance during emergencies. Additionally, there is a need for a system that can track inventory, manage donations and requests, and facilitate communication between the blood bank and the hospital in a secure and scalable manner.

The objective of this project is to design an automated Blood Bank Management System that provides a streamlined platform for donors, recipients, and blood bank administrators to interact and manage blood donations and inventory. The system will help manage blood donation events, store and update information regarding blood stocks in real time, and process blood requests efficiently.

Solutions :

1. Centralized Database: A well-organized database to store details of donors, recipients, blood types, and inventory.
2. Real-Time Updates: Inventory will be updated in real-time, ensuring immediate reflection of new donations or issued blood units.

3. Donor and Recipient Matching: An algorithm that matches recipients with the available blood of the needed blood type and notifies the parties involved.
4. Notification System: Alerts for low stock levels, upcoming blood donation camps, and notifications for registered donors when blood is needed.
5. Report Generation: Automated report generation for analysis and inventory management.

Sample Input :

1. Donor Registration:
 - Name: John Doe
 - Age: 30
 - Blood Type: O+
 - Contact: john.doe@example.com
2. Blood Request:
 - Blood Type Required: A+
 - Quantity: 2 units
 - Urgency: High

Sample Output :

1. After Donor Registration:
 - Confirmation message: "Donor John Doe successfully registered with blood type O+."
 - Inventory update: "Blood type O+ count increased by 1 unit."
2. After Blood Request Processing:
 - Success message: "2 units of blood type A+ available. Request successfully processed."
 - Inventory update: "Blood type A+ count decreased by 2 units."
 - Notification to recipient: "Blood is available and ready for collection."

Result:

The problem statement was written successfully by following the steps described above.

EXPT.NO : 2

DATE :

Writing the Software Requirement Specification Document

1. Introduction :

Purpose:

The purpose of this document is to define the software requirements for the Blood Bank Management System (BBMS). This system will facilitate the management of blood donations, inventory tracking, blood requests, and processing. It aims to streamline blood bank operations, enhance the availability of blood to those in need, and provide an efficient workflow for staff and administrators.

Scope

The BBMS will enable blood banks to:

- Register and manage donor and recipient information.
- Track blood inventory by type, quantity, and expiration.
- Process and track blood donations.
- Fulfill or manage blood requests.
- Generate reports on inventory, donation history, and pending requests.

The system will serve administrators, staff, donors, and recipients, ensuring accurate data handling and enhancing the availability of blood units.

Definitions, Acronyms, and Abbreviations

- **BBMS:** Blood Bank Management System
- **Donor:** A person who donates blood.
- **Recipient:** A person who requests or receives blood.
- **Inventory:** The stock of blood units available.

References

- Relevant regulatory documents on healthcare data management.
- User interface standards for healthcare applications.

2. Overall Description

Product Perspective

The BBMS is a standalone system that will integrate with the blood bank's existing databases and user workflows. It will be accessible through a web-based interface, allowing secure login for different user roles.

Product Functions

- **Donor Management:** Register, update, and view donor information.
- **Recipient Management:** Register, update, and view recipient information.
- **Blood Inventory Management:** Track blood types, quantities, and expiration dates.
- **Blood Request Management:** Process requests from recipients and update inventory.
- **Donation Processing:** Record and manage blood donations.
- **Reporting:** Generate reports on blood stock, donations, and pending requests.

User Classes and Characteristics

- **Admin:** Manages the system, updates inventory, processes donations, and oversees blood requests.
- **Staff:** Assists in managing donor and recipient data, processes blood requests.
- **Donor:** Registers and schedules blood donations.
- **Recipient:** Requests blood and checks availability.

Operating Environment

- Web-based application accessible via common browsers (Chrome, Firefox).
- Server with a relational database (e.g., MySQL, PostgreSQL).
- Internet or intranet connectivity for access within the blood bank premises.

Design and Implementation Constraints

- Compliance with healthcare data privacy laws (e.g., HIPAA, GDPR).
- Secure authentication and role-based access control.

Assumptions and Dependencies

- Assumes consistent internet or intranet access.
- Depends on timely entry and update of data for accuracy.

3. Specific Requirements

Functional Requirements

Donor Management

- **Register Donor:** System allows new donor registration with details like name, contact, blood type, and health status.
- **Update Donor Information:** Admin/Staff can update donor details as needed.
- **View Donor History:** Admin/Staff can view the donation history of a donor.

Recipient Management

- **Register Recipient:** System allows recipient registration with details like name, blood type, contact information.
- **Update Recipient Information:** Admin/Staff can update recipient details.
- **View Recipient Details:** Admin/Staff can access recipient details to verify requests.

Blood Inventory Management

- **Add Blood Units:** System allows adding blood units to the inventory after a donation.
- **Update Inventory:** System automatically updates inventory after fulfilling requests.
- **View Blood Levels:** Admin/Staff can view available blood types and quantities.

Blood Request Management

- **Request Blood:** Recipients or Staff can submit requests specifying blood type and quantity.
- **Fulfill Request:** System checks availability, updates inventory if fulfilled, and notifies status.
- **View Request Status:** Recipients or Staff can view the status of pending or completed requests.

Donation Processing

- **Record Donation:** System records donation details, updates donor record, and updates blood inventory.
- **Schedule Donation:** System allows donors to schedule donation appointments.

Reporting

- **Generate Reports:** System generates reports on current inventory, donation history, and pending requests.

Non-Functional Requirements

Performance Requirements

- The system should respond to user actions (e.g., data entry, search) within 2 seconds.
- Able to support simultaneous access by multiple users.

Security Requirements

- **Authentication:** Only authenticated users can access the system.

- **Role-Based Access Control:** Access is restricted based on user roles (Admin, Staff, Donor, Recipient).
- **Data Encryption:** Sensitive information (e.g., contact info) is encrypted.

Usability Requirements

- **User Interface:** Simple and intuitive for staff with limited technical experience.
- **Accessibility:** Complies with accessibility standards for visually impaired users.

Reliability Requirements

- System uptime of at least 99% during blood bank operating hours.
- Data backup every 24 hours.

4. External Interface Requirements

User Interfaces

- **Login Page:** For users to securely log in to the system.
- **Dashboard:** Displays quick stats on inventory, requests, and donations.
- **Forms:** For donor and recipient registration, blood request submission, and donation recording.

Hardware Interfaces

- Compatible with standard PCs, tablets, and mobile devices.

Software Interfaces

- Relational database management system (MySQL or PostgreSQL).
- Web server (Apache or Nginx) to host the application.

Communication Interfaces

- HTTPS for secure data transmission.

5. Other Requirements

- **Data Privacy:** Compliance with privacy standards to protect donor and recipient data.
- **Audit Log:** All actions performed by users should be recorded for audit purposes.

Result:

The SRS was made successfully by following the steps described above.

EXPT.NO: 3

DATE :

Entitiy Relationship Diagram

Aim :

To draw the Entity Relationship diagram for Blood Bank Management System.

Algorithm:

Step 1: Mapping of regular Entity types

Step 2: Mapping of Weak Entity Types

Step3: Mapping of Binary 1:1 Relation types

Step4: Mapping of Binary 1:N Relationship Types

Step5: Mapping of Binary M:N Relationship Types.

Step6: Mapping of Multivalued attributes.

Input:

Entities

ER matrix

Primary Keys

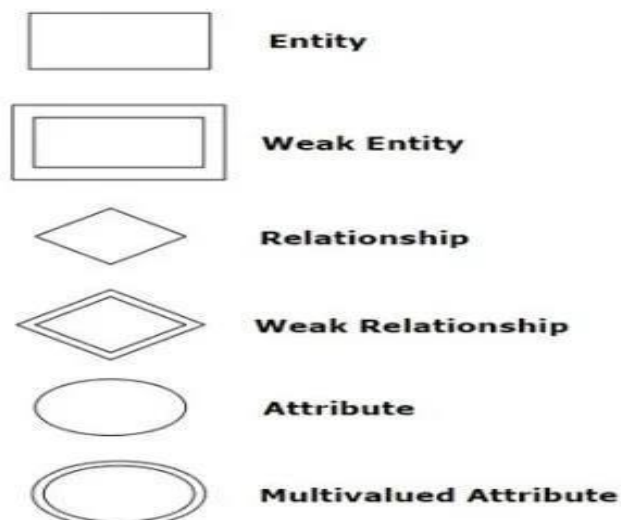
Attributes

Mapping of Attributes with Entities

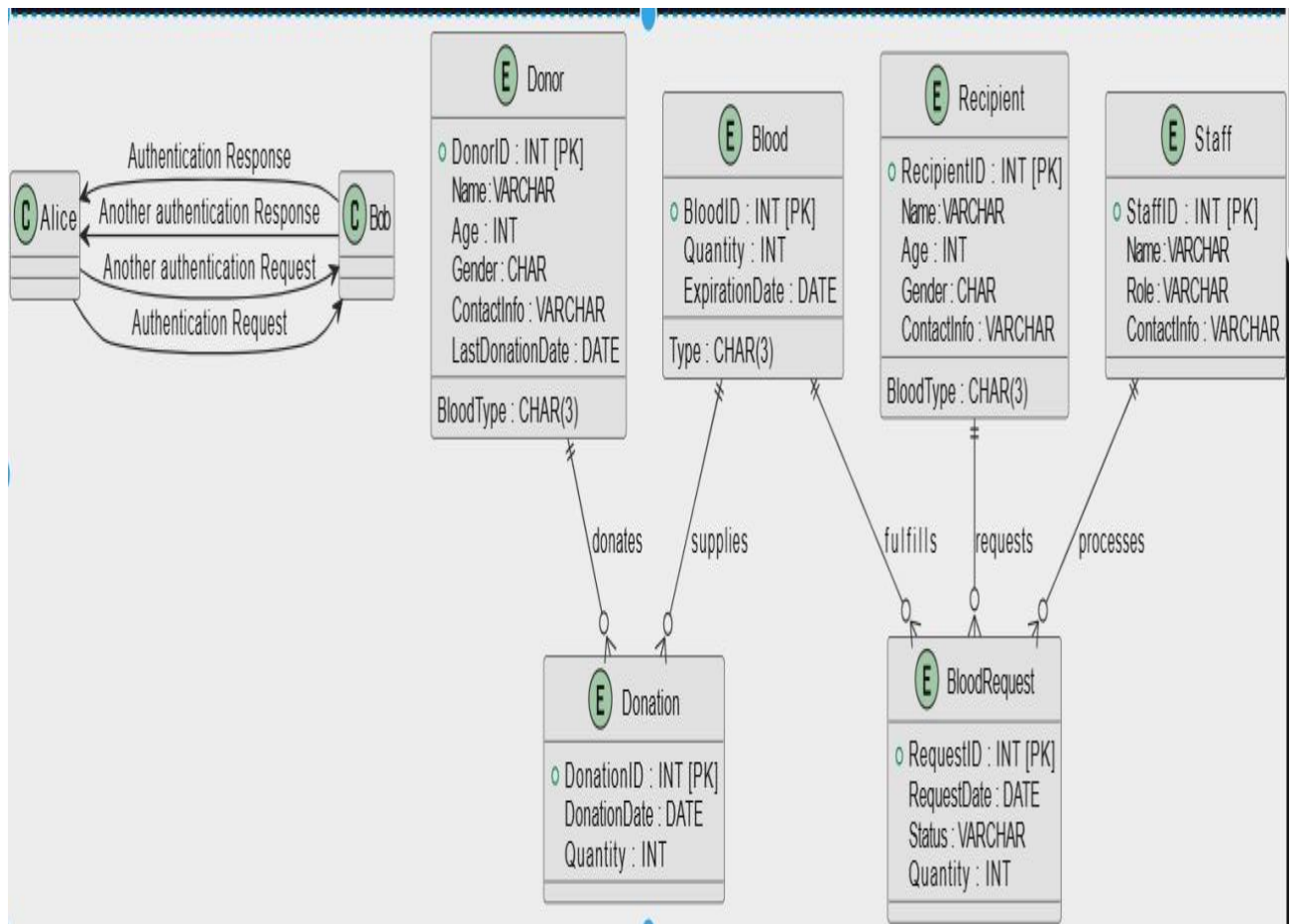
Sample Output:

The fully attributed ERD

SYMBOLS:



ER DIAGRAM :



Result :

The ER Diagram was made successfully by following the steps described above.

EXPT.NO : 4

DATE :

Data Flow Diagrams at Level 0 and Level 1

Aim:

To draw the Data Flow Diagram for Blood bank system management project and list the modules in the application.

Algorithm :

1. Open the visual paradigm to draw DFD(Ex.Lucidchart)
2. Select a data flow diagram template
3. Name the data flow diagram
4. Add an external entity that starts the process
5. Add a Process to the DFD
6. Add a data store to the diagram
7. Continue to add items to the DFD
8. Add data flow to the DFD
9. Name the data flow
10. Customize the DFD with colours and fonts
11. Add a title and share your data flow diagram






Input :

Processes

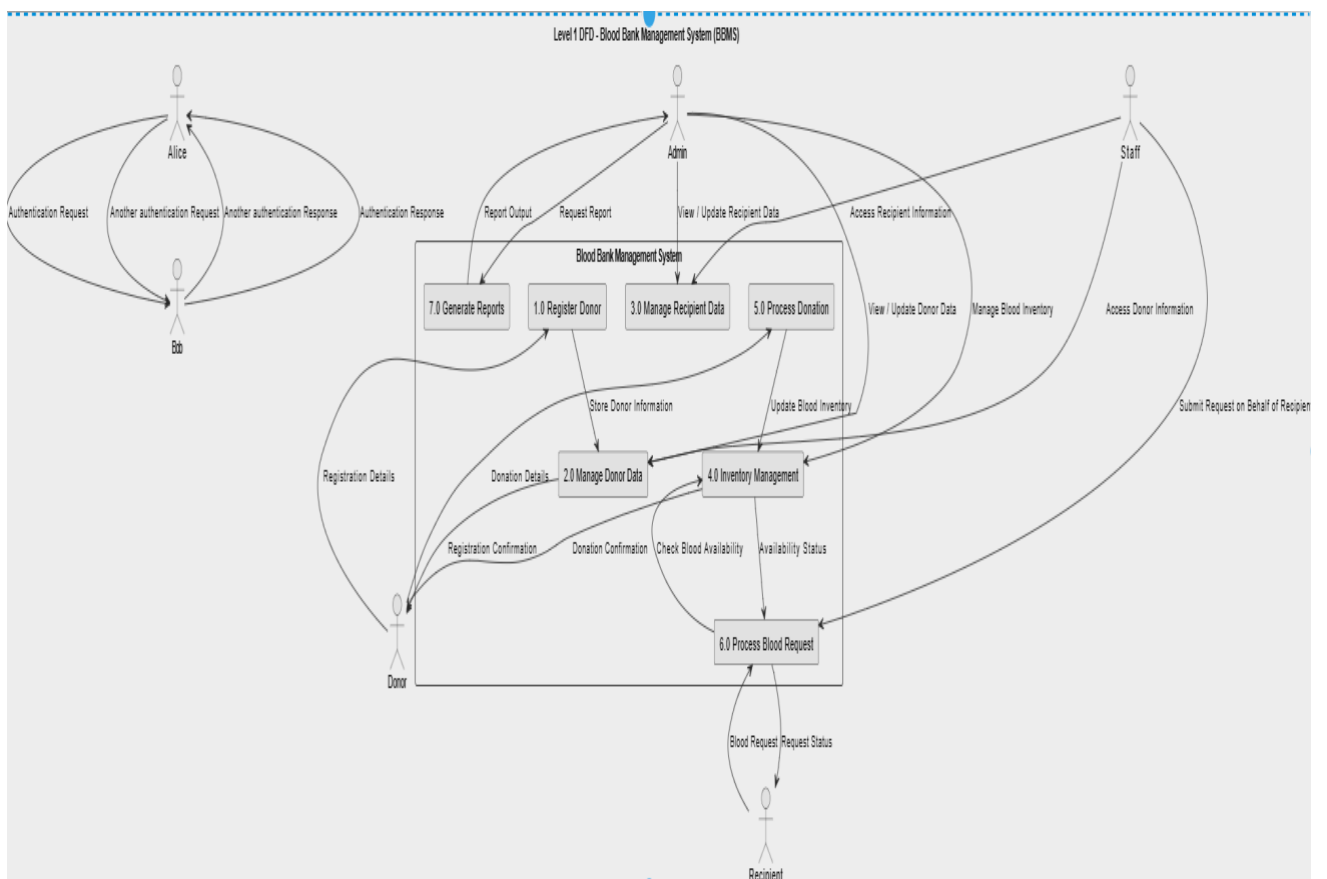
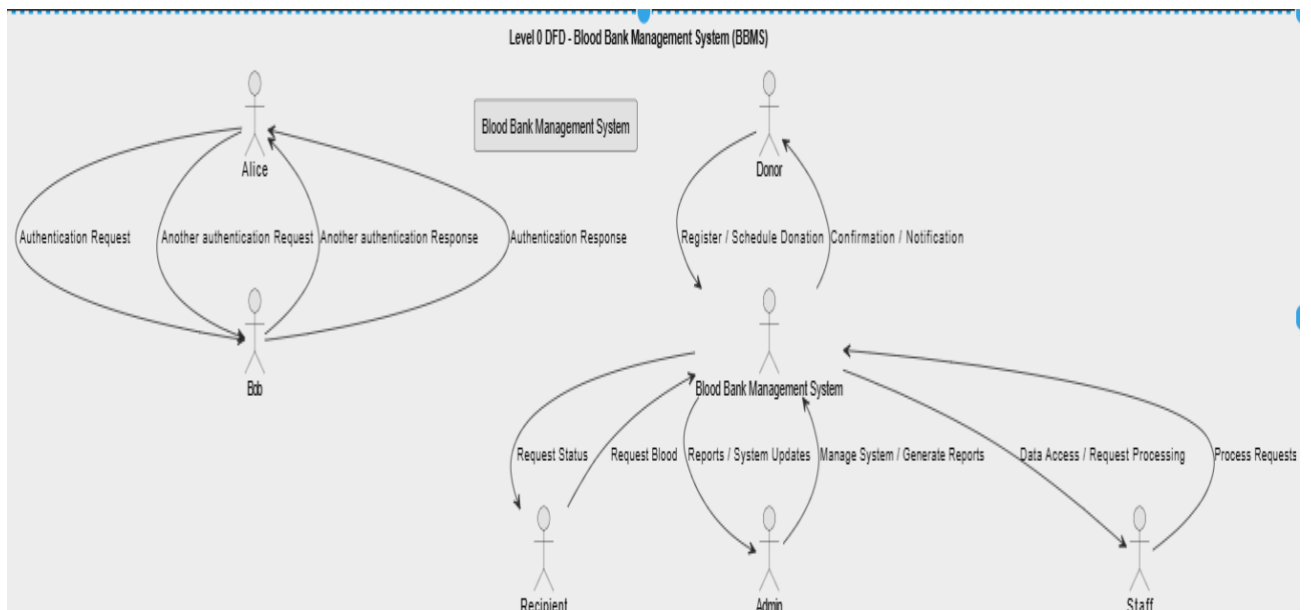
Datastores

External Entities

Symbols :

Notation	Yourdon & De Marco	Gene & Sarson	SSADM	Unified
External Entity				
Process				
Data Store				
Data Flow				

Sample Output:



Result:

The Data Flow diagram was made successfully by following the steps described above.

EXPT.NO : 5

DATE :

Use Case Diagram

Aim :

To Draw the Use Case Diagram for Blood Bank Managem.

Algorithm :

Step 1: Identify Actors

Step 2: Identify Use Cases

Step 3: Connect Actors and Use Cases

Step 4: Add System Boundary

Step 5: Define Relationships

Step 6: Review and Refine

Step 7: Validate

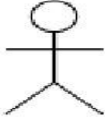
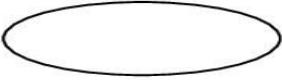


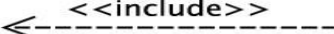

Input :

Actors

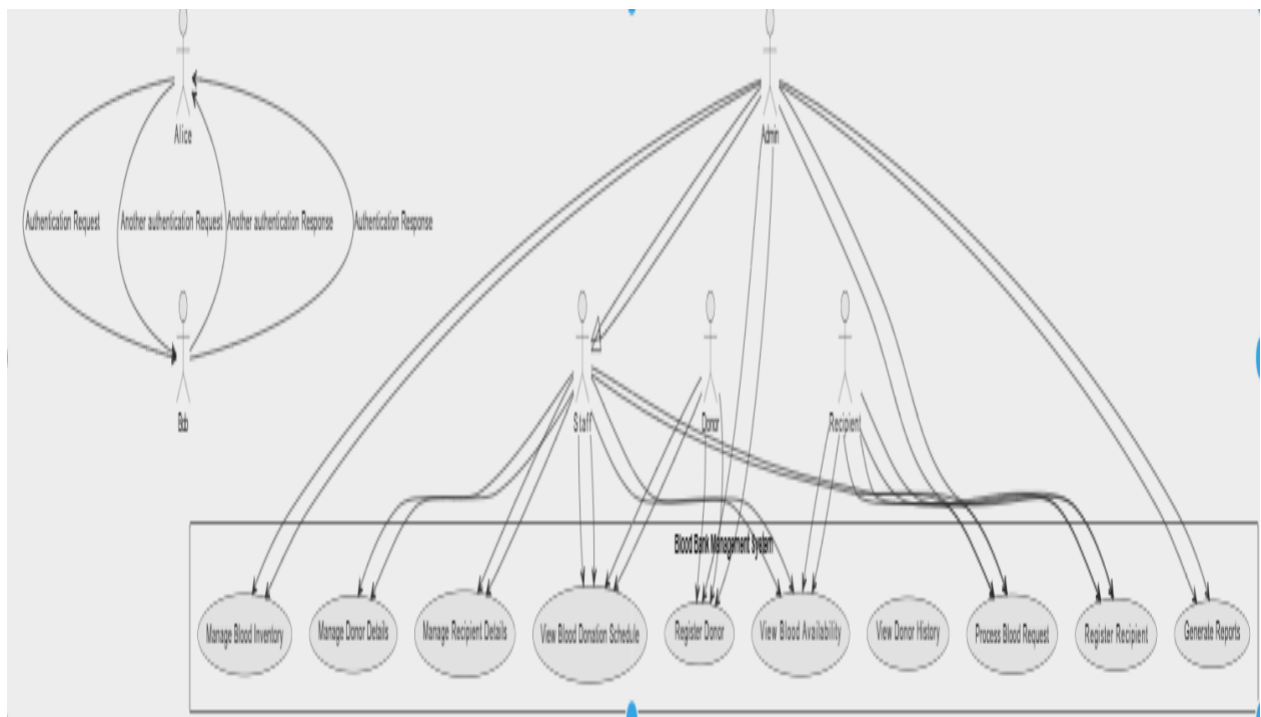
Use Cases

Relations

Symbols :

Symbol	Reference Name
	Actor
	Use case
   	Relationship

Sample Output :



Result :

The use case diagram has been created successfully by following the steps given.

EXPT NO : 6

DATE :

Activity Diagram of All Use Cases

Aim:

To Draw the activity Diagram for Blood Bank Management System.

Algorithm :

Step 1: Identify the Initial State and Final States

Step 2: Identify the Intermediate Activities Needed

Step 3: Identify the Conditions or Constraints

Step 4: Draw the Diagram with Appropriate Notations

Inputs :

Activities

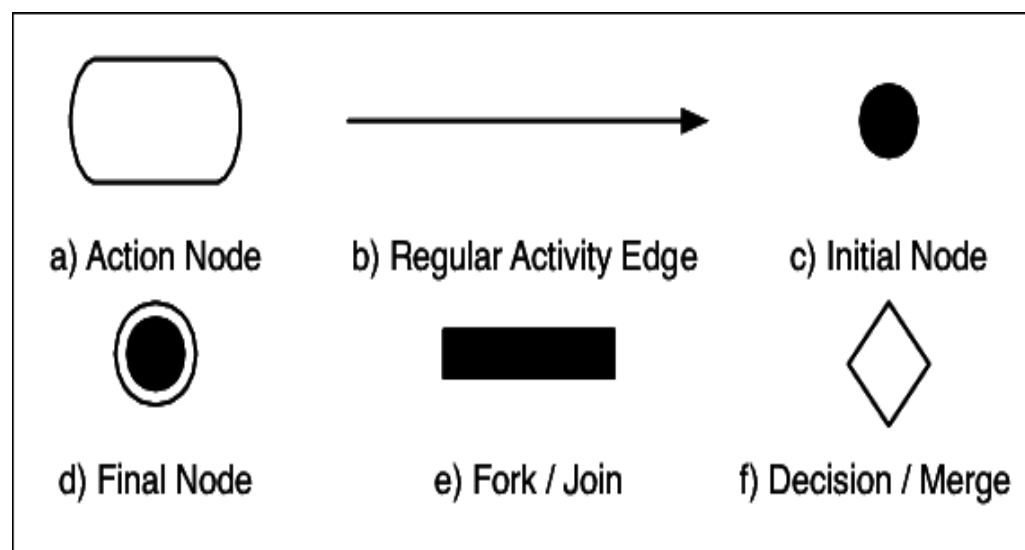
Decision Points

Guards

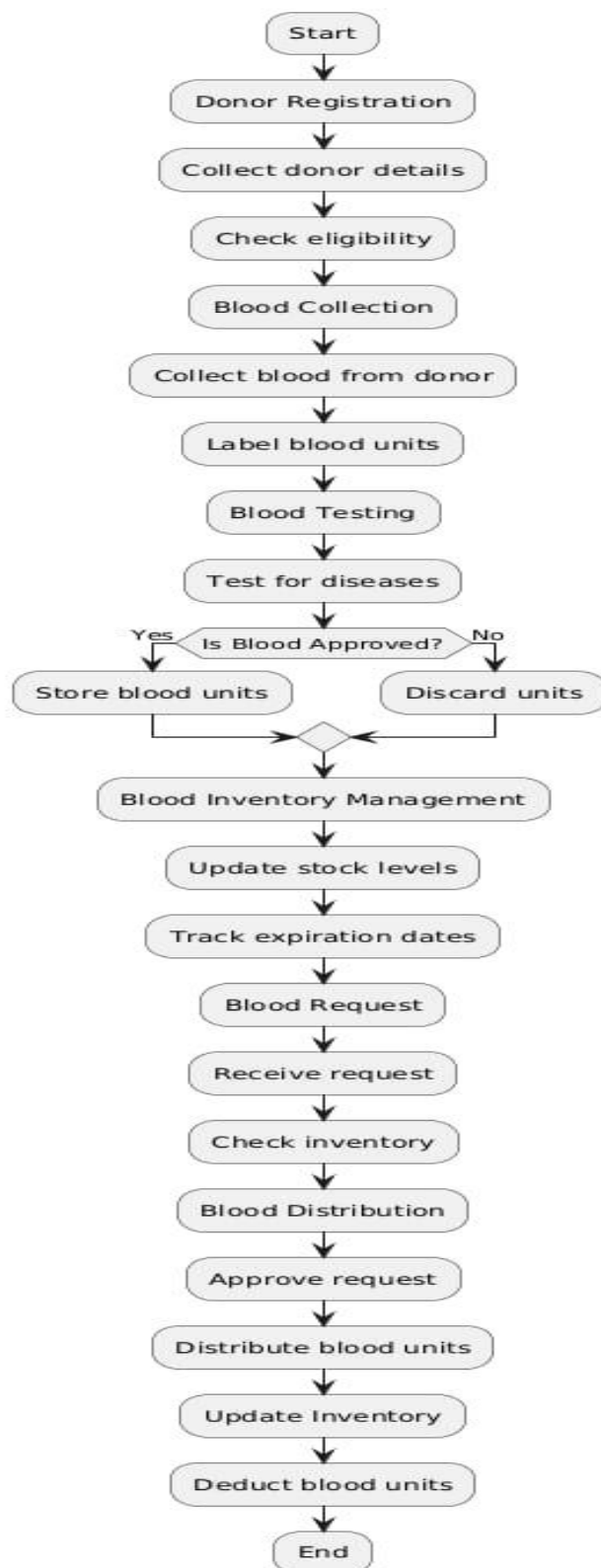
Parallel Activities

Conditions

Symbols :



Sample Output :



Result : The Activity diagram has been created successfully by following the steps given.

EXPT NO : 7

DATE :

State Chart Diagram of all use cases

Aim :

To Draw the State Chart Diagram for Blood Bank Management System.

Algorithm :

STEP-1: Identify the important objects to be analysed.

STEP-2: Identify the states.

STEP-3: Identify the events.

State chart diagram is used to describe the states of different objects in its life cycle.

Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyse and implement them accurately. State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

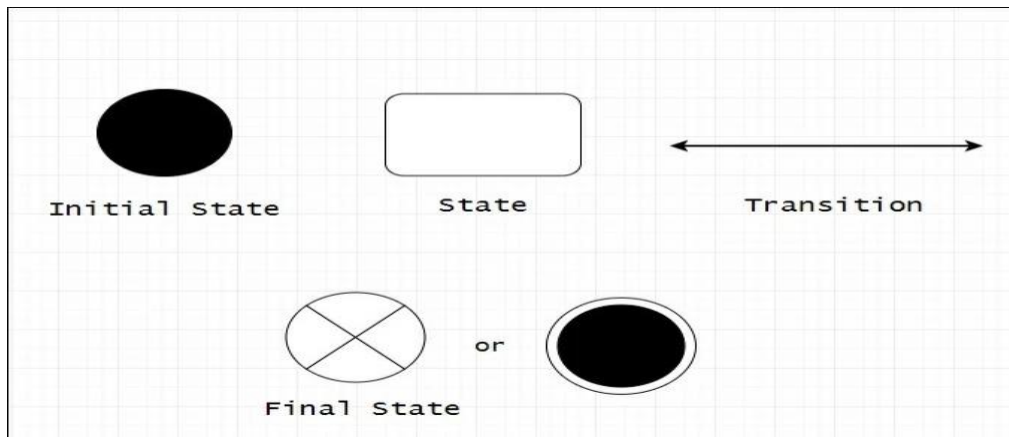
Inputs :

Objects

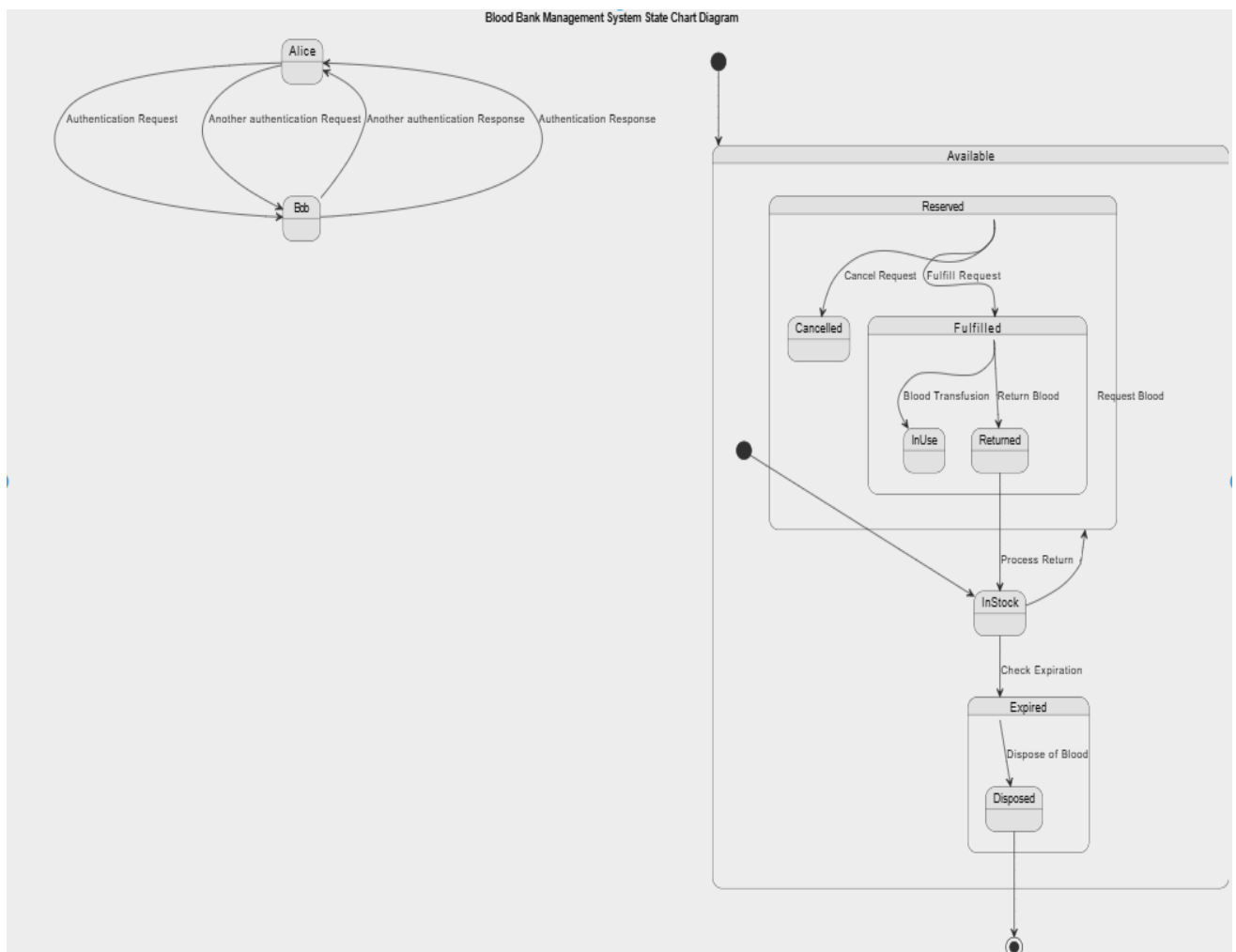
States

Events

Symbols :



Sample Output :



Result :

The State Chart diagram has been created successfully by following the steps given.

EXPT NO : 8

DATE :

Sequence diagram of all use cases

Aim :

To Draw the Sequence Diagram for Blood Bank Management Diagram.

Algorithm :

1. Identify the Scenario
2. List the Participants
3. Define Lifelines
4. Arrange Lifelines
5. Add Activation Bars
6. Draw Messages
7. Include Return Messages
8. Indicate Timing and Order
9. Include Conditions and Loops
10. Consider Parallel Execution
11. Review and Refine
12. Add Annotations and Comments
13. Document Assumptions and Constraints
14. Use a Tool to create a neat sequence diagram

Inputs :

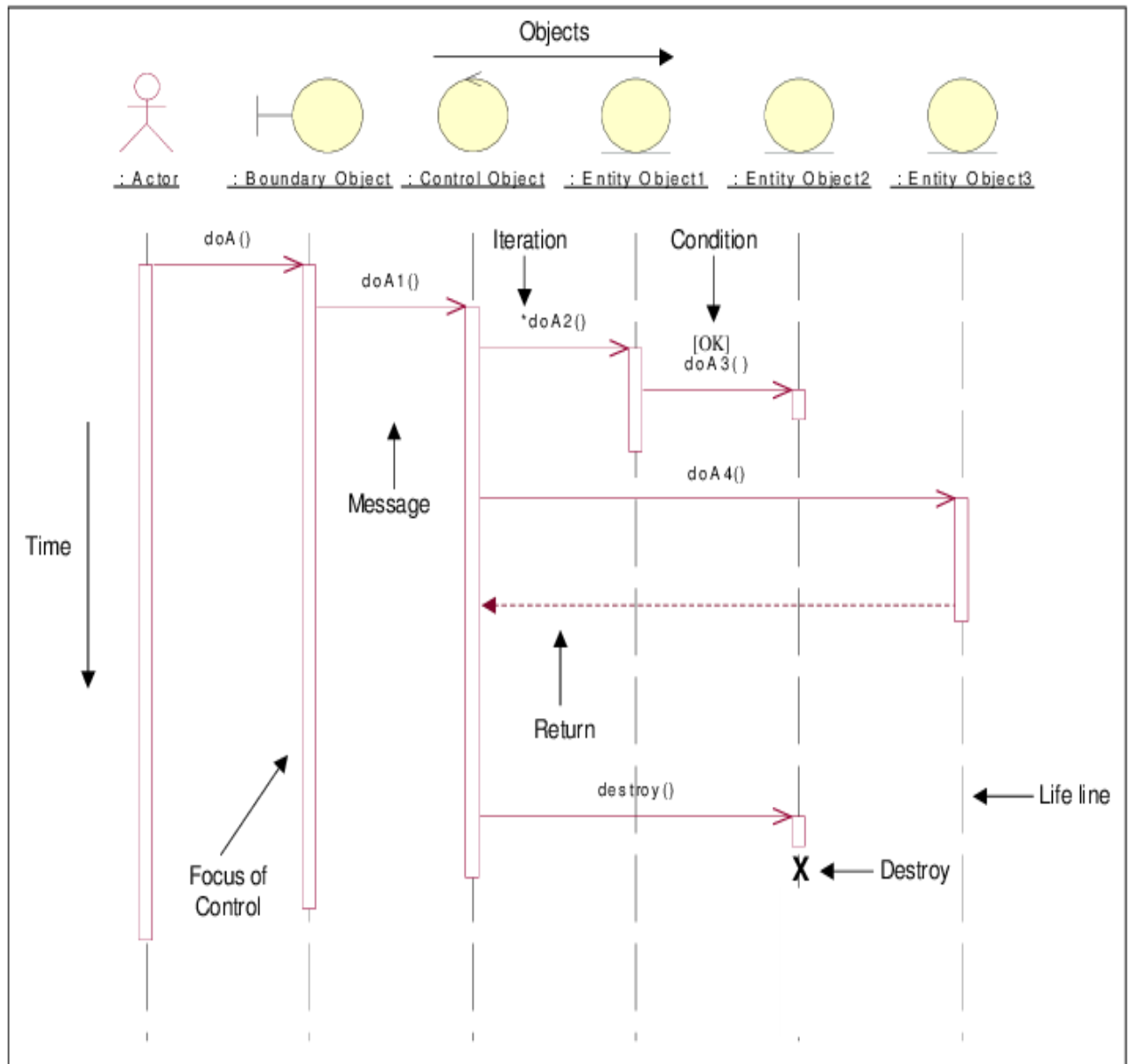
Objects taking part in the interaction.

Message flows among the objects.

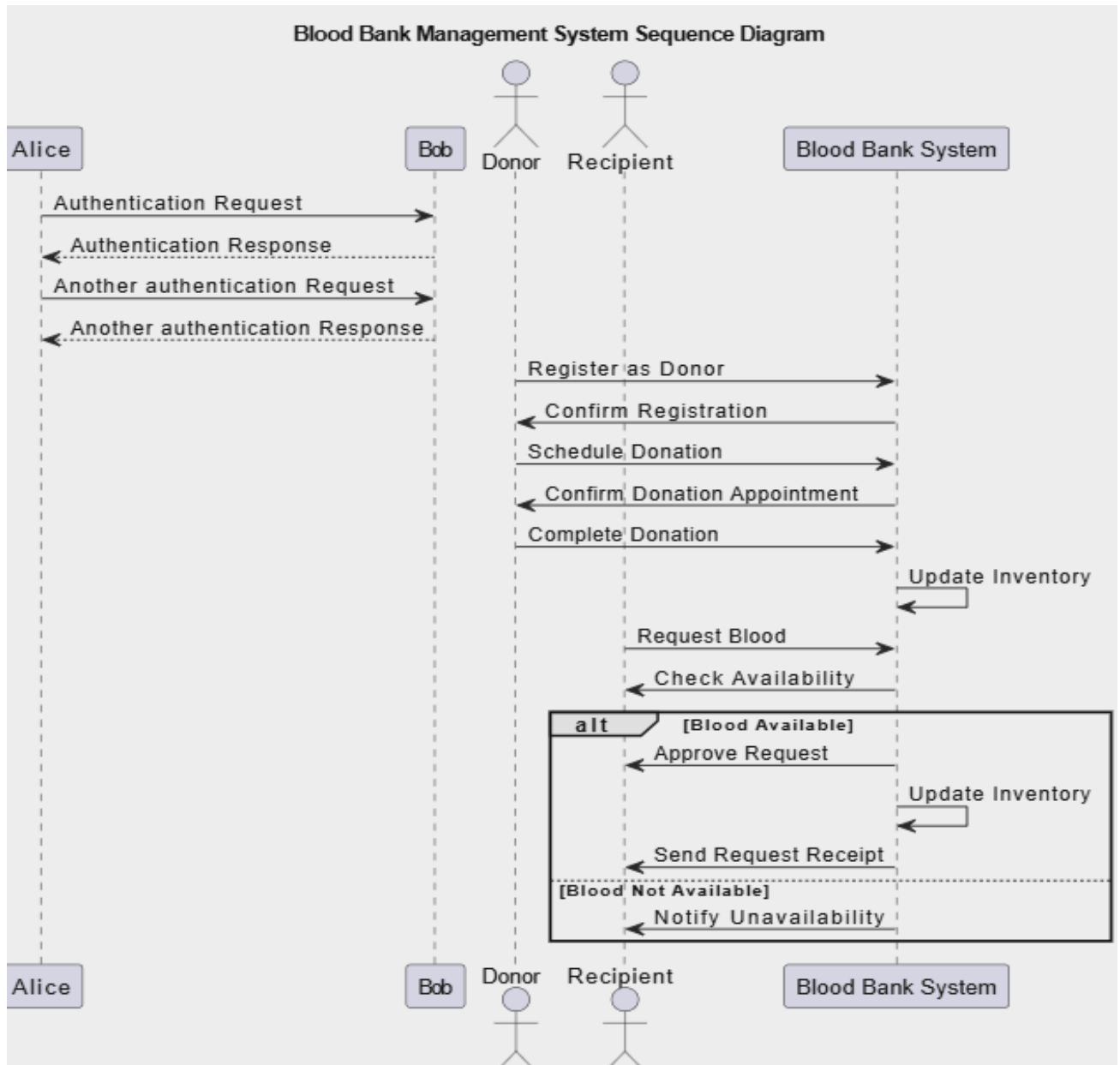
The sequence in which the messages are flowing.

Object organization.

Symbols :



Sample Output



Result :

The Sequence diagram has been created successfully by following the steps given.

EXPT NO : 9

DATE :

Collaboration diagram of all use cases

Aim :

To Draw the Collaboration Diagram for Student Result Analysis Project.

Algorithm :

Step 1: Identify Objects/Participants

Step 2: Define Interactions

Step 3: Add Messages

Step 4: Consider Relationships

Step 5: Document the collaboration diagram along with any relevant explanations or annotations.

Inputs :

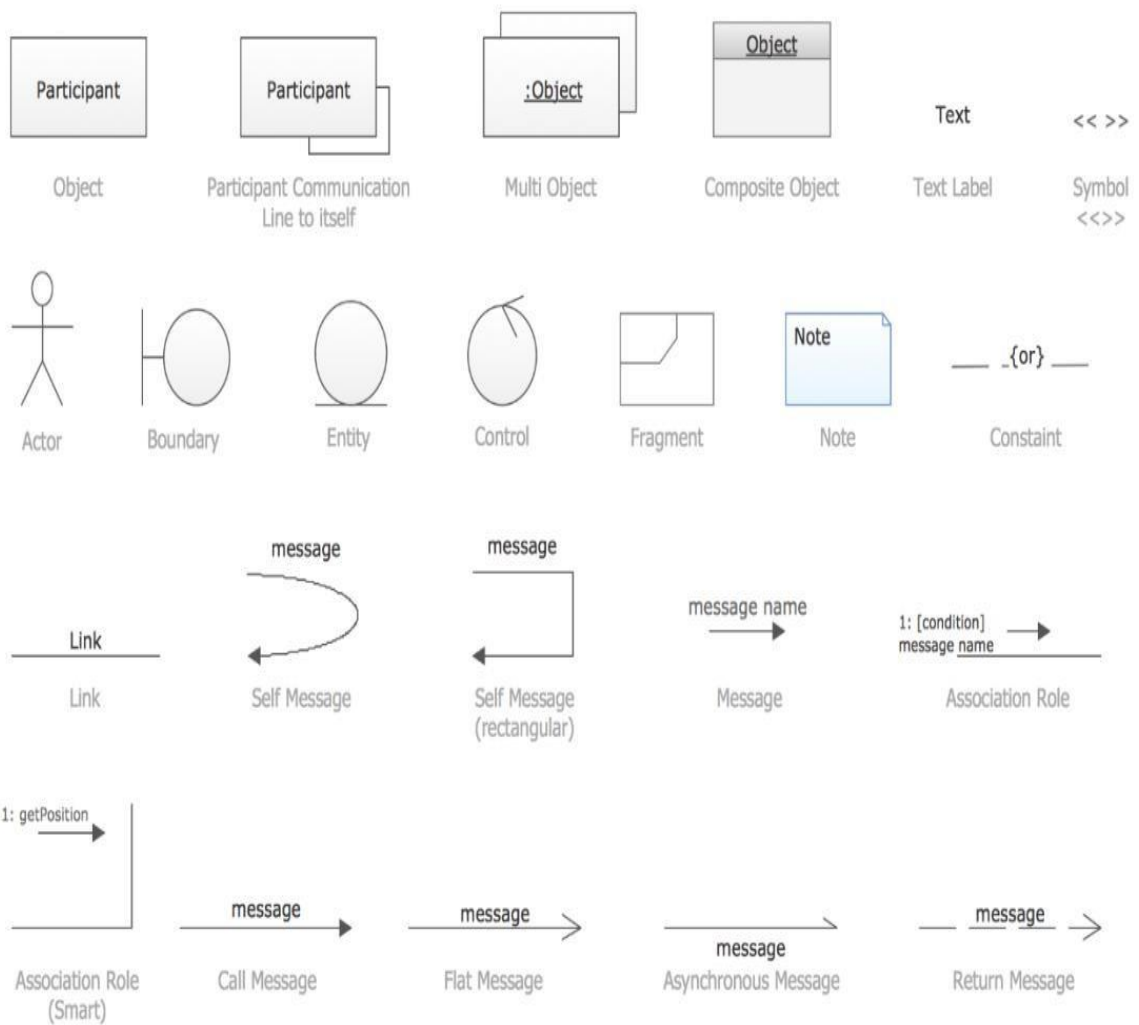
Objects taking part in the interaction.

Message flows among the objects.

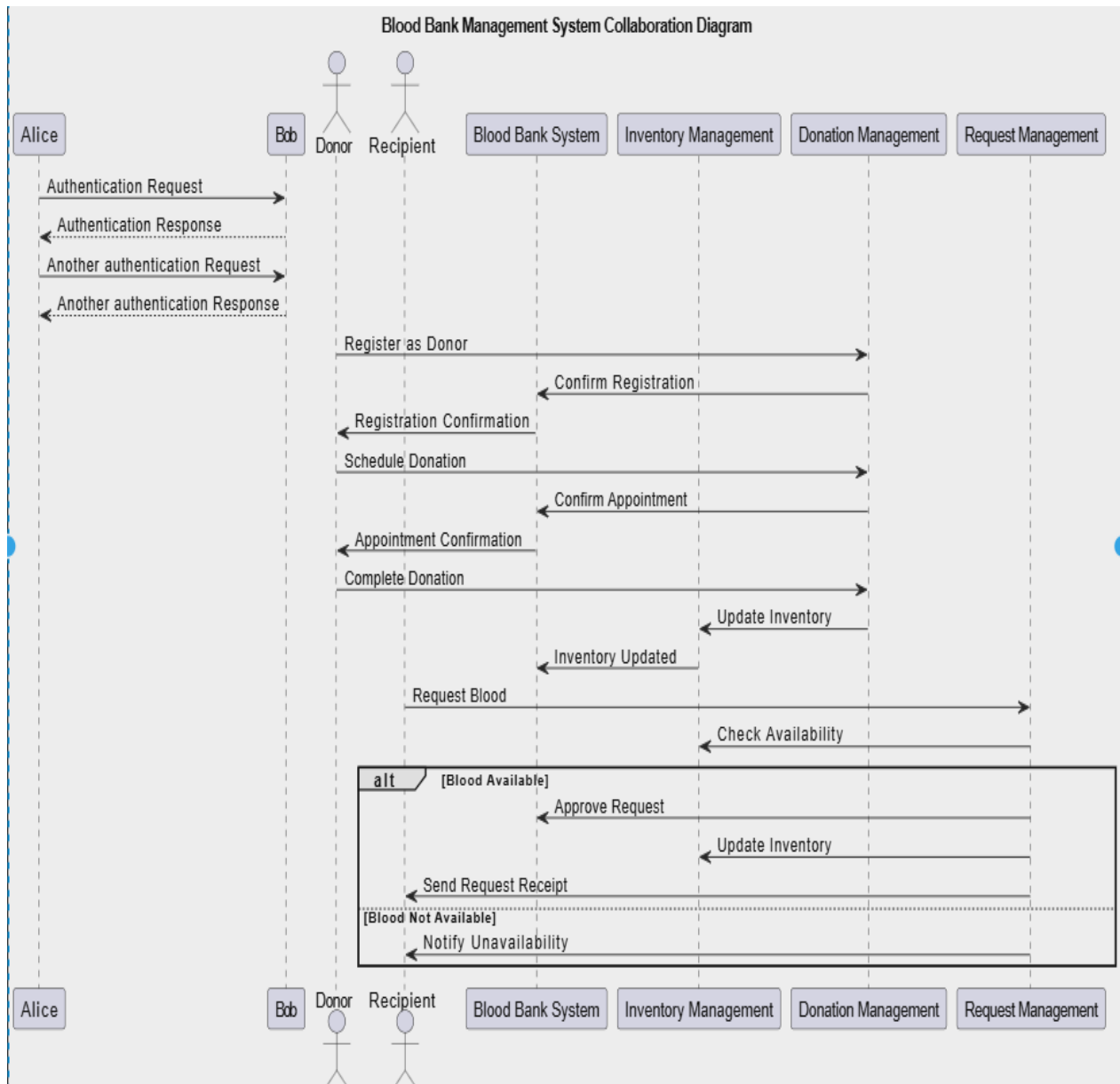
The sequence in which the messages are flowing.

Object organization.

Symbols :



Sample Output :



Result :

The Collaboration diagram has been created successfully by following the steps given

EXPT NO : 10

DATE :

CLASS DIAGRAM

Aim:

To Draw the Class Diagram for Blood Bank Management System.

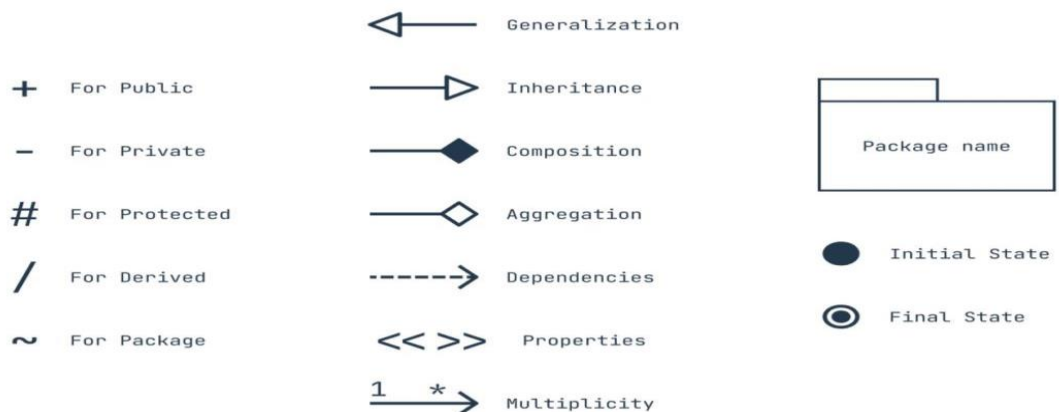
Algorithm :

1. Identify Classes
2. List Attributes and Methods
3. Identify Relationships
4. Create Class Boxes
5. Add Attributes and Methods
6. Draw Relationships
7. Label Relationships
8. Review and Refine
9. Use Tools for Digital Drawing

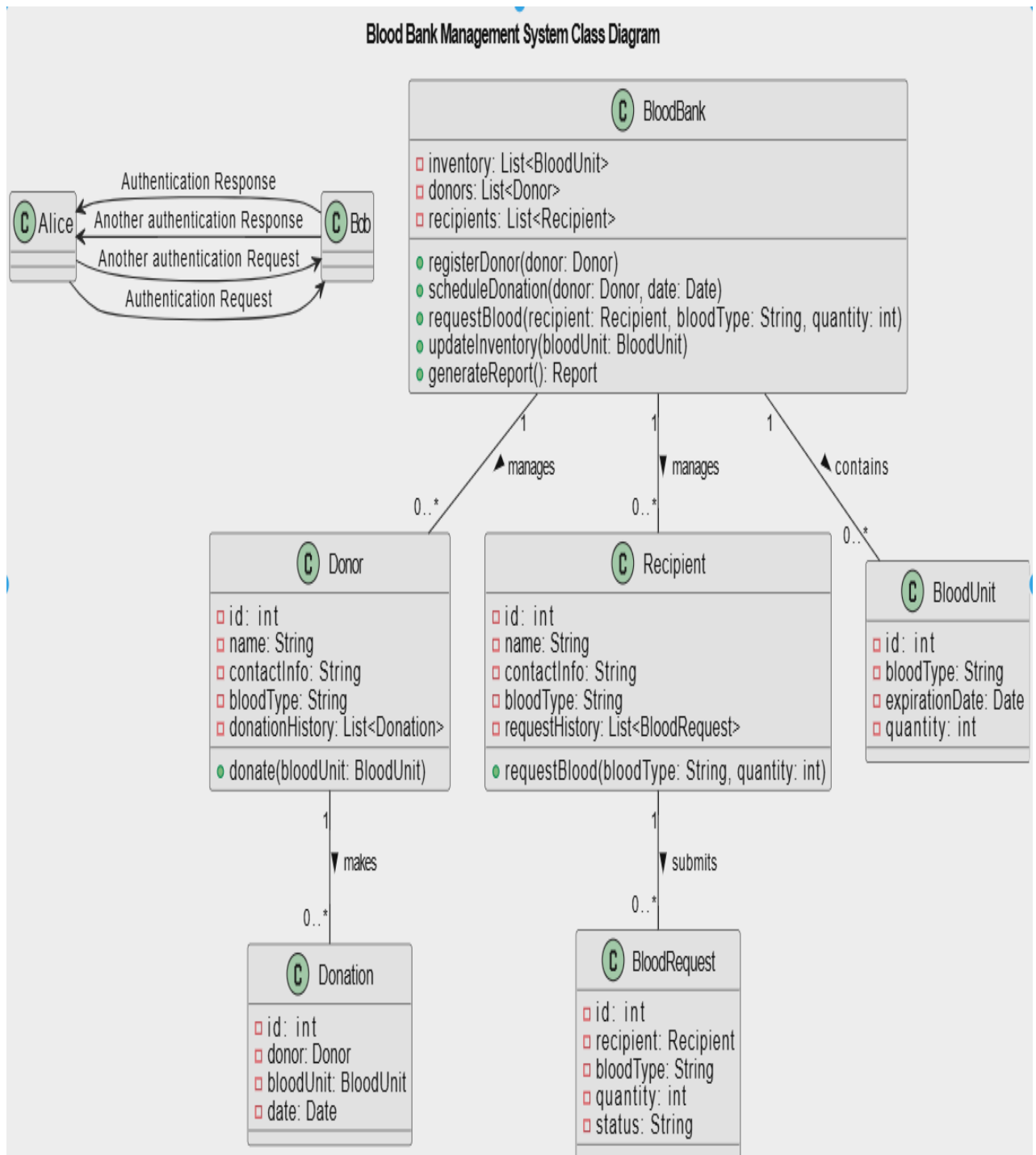
Inputs :

1. Class Name
2. Attributes
3. Methods
4. Visibility Notation

Symbols :



Sample Output :



Result:

The Class diagram has been created successfully by following the steps given.

BLOOD BANK MANAGEMENT SYSTEM CODE

```
import sqlite3

from datetime import datetime

def connect_db():
    return sqlite3.connect("blood_bank.db")

def add_donor(name, blood_type, contact, address, last_donation_date):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO donors (name, blood_type, contact, address,
last_donation_date)
        VALUES (?, ?, ?, ?, ?)", (name, blood_type, contact, address,
last_donation_date))
    conn.commit()
    conn.close()
    print("Donor added successfully.")

def view_donors():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM donors")
    donors = cursor.fetchall()
    conn.close()
    for donor in donors:
        print(f"ID: {donor[0]}, Name: {donor[1]}, Blood Type: {donor[2]}, Contact: {donor[3]},
"
            f"Address: {donor[4]}, Last Donation Date: {donor[5]}")

def update_blood_stock(blood_type, quantity):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT quantity FROM blood_stock WHERE blood_type = ?",
(blood_type,))
    result = cursor.fetchone()
```

```

    if result:

        cursor.execute("UPDATE blood_stock SET quantity = quantity + ? WHERE
blood_type = ?", (quantity, blood_type))

    else:

        cursor.execute("INSERT INTO blood_stock (blood_type, quantity) VALUES (?, ?)",
(blood_type, quantity))

    conn.commit()

    conn.close()

    print("Blood stock updated successfully.")
def check_blood_stock(blood_type):

    conn = connect_db()

    cursor = conn.cursor()

    cursor.execute("SELECT quantity FROM blood_stock WHERE blood_type = ?",
(blood_type,))

    result = cursor.fetchone()

    conn.close()

    if result:

        print(f"Available stock for {blood_type}: {result[0]} units")

    else:

        print(f"No stock available for blood type {blood_type}.")
def main():

    while True:

        print("\nBlood Bank Management System")

        print("1. Add Donor")

        print("2. View All Donors")

        print("3. Update Blood Stock")

        print("4. Check Blood Stock")

        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':

```

```
name = input("Enter Name: ")
blood_type = input("Enter Blood Type: ")
contact = input("Enter Contact: ")
address = input("Enter Address: ")
last_donation_date = input("Enter Last Donation Date (YYYY-MM-DD): ")

try:
    datetime.strptime(last_donation_date, "%Y-%m-%d")
    add_donor(name, blood_type, contact, address, last_donation_date)
except ValueError:
    print("Invalid date format. Please enter in YYYY-MM-DD format.")

elif choice == '2':
    view_donors()

elif choice == '3':
    blood_type = input("Enter Blood Type: ")

    try:
        quantity = int(input("Enter Quantity to Add: "))
        update_blood_stock(blood_type, quantity)
    except ValueError:
        print("Please enter a valid integer for quantity.")

elif choice == '4':
    blood_type = input("Enter Blood Type: ")
    check_blood_stock(blood_type)

elif choice == '5':
    print("Exiting...")
    break

else:
    print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()
```

OUTPUT :

Blood Bank Management Syste

1. Add Donor
2. View All Donors
3. Update Blood Stock
4. Check Blood Stock
5. Exit

Enter your choice: 1

Enter Name: John

Enter Blood Type: O+

Enter Contact: 9893045678

Enter Address: 123 Elm Street

Enter Last Donation Date (YYYY-MM-DD): 2024-11-01

Donor added successfully.

Blood Bank Management System

1. Add Donor
2. View All Donors
3. Update Blood Stock
4. Check Blood Stock
5. Exit

Enter your choice: 2

ID: 1, Name: John, Blood Type: O+, Contact: 9893045678, Address: 123 Elm Street, Last Donation Date: 2024-11-01

Blood Bank Management System

1. Add Donor
2. View All Donors
3. Update Blood Stock
4. Check Blood Stock

5. Exit

Enter your choice: 3

Enter Blood Type: O+

Enter Quantity to Add: 5

Blood stock updated successfully.

Blood Bank Management System

1. Add Donor

2. View All Donors

3. Update Blood Stock

4. Check Blood Stock

5. Exit

Enter your choice: 3

Enter Blood Type: A-

Enter Quantity to Add: 10

Blood stock updated successfully.

Blood Bank Management System

1. Add Donor

2. View All Donors

3. Update Blood Stock

4. Check Blood Stock

5. Exit

Enter your choice: 4

Enter Blood Type: O+

Available stock for O+: 5 units

Blood Bank Management System

1. Add Donor

2. View All Donors

3. Update Blood Stock

4. Check Blood Stock

5. Exit

Enter your choice: 4

Enter Blood Type: AB-

No stock available for blood type AB-.

Blood Bank Management System

1. Add Donor

2. View All Donors

3. Update Blood Stock

4. Check Blood Stock

5. Exit

Enter your choice: 5

Exiting...

