|  |  |  |  |
| --- | --- | --- | --- |
|  | **Assignment No. 01 (Graded) Semester: Fall 2024**  **CS304- Object Oriented Programming**  **Registration No. BC230423882**  **Name: Muhammad Yasir**  **BS (Software Engineering)** | | **Total Marks: 20**  **Submission Date:**  **03 November 2024** |
| **Assignment No. 1** | |  | |
| **Scenario:**  A blood bank plays a vital role in saving people’s lives. A blood bank management system is a system which manages the donation and storage of blood, and inventory operations etc. It manages the collection, storage and distribution of blood to hospitals. Donors are individuals who donate blood. They can register, update their profiles, and view their donation records. On the other hand, patients in need of blood can register request and track status of their request. They have specific blood type requirements and medical history.  The Inventory tracks available blood types and quantities, including inventory ID and expiration dates, with methods for checking availability and updating records. Similarly, hospitals can view donor details, and select required donor on biases of donor’s information. The admin must handle donor registrations, manage blood inventory, and facilitate blood requests from hospitals. It should also monitor blood expiry, check donor eligibility and ensures that blood is available for critical patients. Admin also send reminders to donors for future donations.  You have to perform the following tasks in this assignment:   1. Extract the main objects (entities) of above system. 2. Find the necessary attributes and functions that need to be associated with each object. 3. Identify the relationships among identified objects. 4. Construct a final comprehensive Class diagram showing all objects and their relationships along with their attributes and functions. | | | |
| **Solution** | |  | |
| **Step 1: Identify the Main Objects (Classes)**  The primary objects (entities) for the blood bank system are:  1. Hospital  2. Admin  3. Patient  4. Donor  5. Blood Inventory  **Step 2: Define Attributes and Functions for Each Object**  Following OOP principles, let’s assign attributes and functions to each entity.  1. Hospital  **Attributes**:  hospitalID  name  location  contactInfo  **Functions**:  viewDonorDetails()  selectDonor()  requestBlood()  2. Admin  **Attributes**:  adminID  name  contactInfo  **Functions**:  manageDonorRegistration()  manageBloodInventory()  facilitateBloodRequest() // Process blood requests from hospitals.  monitorBloodExpiry()  sendDonationReminders()  checkDonorEligibility()  3. Patient  **Attributes**:  patientID  name  age  bloodTypeRequired  medicalHistory  contactInfo  requestStatus  **Functions**:  register()  requestBlood()  trackRequestStatus()  4. Donor  **Attributes**:  donorID  name  age  bloodType  contactInfo  lastDonationDate  medicalHistory  **Functions**:  register()  updateProfile()  viewDonationRecords()  checkEligibility() // Checks if donor meets criteria for donation.  5. Blood Inventory  **Attributes**:  inventoryID  bloodType  quantity  expirationDate  **Functions**:  checkAvailability() // Checks blood availability by type.  updateInventory()  monitorExpiry()  **Step 3: Define Relationships among Objects**  1. **Donor - Blood Inventory**: Donors donate blood, which is then stored in the blood inventory.  2. **Patient - Blood Inventory**: Patients request blood from the inventory.  3. **Hospital - Donor**: Hospitals can view and select donor details.  4. **Admin - Donor**: Admin manages donor registration and can send reminders.  5. **Admin - Blood Inventory**: Admin manages the blood inventory, including monitoring expirations and facilitating hospital requests.  6. **Admin - Patient**: Admin handles patient requests and ensures blood availability.  7. **Hospital - Patient**: Hospitals handle patient requests for blood.  **Step 4: Construct a Comprehensive Class Diagram**  Incorporating principles of **Composition** and **Association**:  **Composition**: Admin has a composition relationship with Inventory since the admin directly manages blood inventory.  **Aggregation**: Donors and Patients are associated with Inventory through aggregation since their relationship isn’t tightly bound (they exist independently).  Here’s an outline of the diagram:    C:\Users\Yasir\Downloads\Flowcharts (2).png  **Relationships:**  **Donor → Blood Inventory**: Donates blood.  **Patient → Blood Inventory**: Requests blood.  **Hospital → Donor**: Views and selects donor.  **Admin → Donor**: Manages registrations and eligibility.  **Admin → Blood Inventory**: Manages and monitors blood.  **Admin → Patient**: Facilitates patient requests.  **Hospital → Patient**: Processes patient blood requests.  This structure aligns with OOP principles such as **inheritance**, **composition**, **encapsulation**, and **polymorphism**. | | | |