C.W

Task 1:  
You have been asked to design a digital wallet application that users can top up, spend from, and check their balances. Each user can have only one wallet. The system needs to prevent unauthorized or invalid operations (for instance, negative balances or exceeding certain transaction limits).

**Tasks:**

1. Create a Wallet class that:
   * Holds the user’s balance as a private member.
   * Allows only deposit (top-up) and withdrawal (spend) operations through public methods.
   * Enforces maximum daily transaction limits (for deposit and withdrawal).
2. Create a User class that:
   * Stores minimal user details (like userID, possibly a name, etc.) as private.
   * Has a single Wallet instance hidden from direct manipulation.
3. Expose only the methods that the rest of the code needs for normal usage. For example, the outside world should not directly manipulate the Wallet’s internal details; it should just say user.deposit(...) or user.withdraw(...).
4. Ensure the system refuses invalid operations (like overdrawing or exceeding a daily limit).
5. Demonstrate usage in main() with multiple user objects and different deposit/withdraw scenarios.

**Key Points to Consider:**

* How do you design the Wallet class so that internal states (balance, daily limit usage) are hidden?
* How do you structure User to ensure the wallet cannot be manipulated outside the intended public interface?

Task 2:

large software system requires a centralized logging service. Various parts of the application produce logs of different severities (INFO, WARN, ERROR). The logging service needs to:

* Collect and store log messages internally.
* Expose only a few “log this message” methods to the outside world.
* Enforce internal rules, such as limiting the maximum stored logs or automatically categorizing messages.

**Tasks:**

1. Implement a Logger class that:
   * Keeps an internal data structure (like a std::vector<std::string>) for log messages in private scope.
   * Has public methods for logging information (logInfo, logWarning, logError, or a single method with severity as a parameter).
   * Possibly has a private method to handle the real formatting and storing logic so that outside callers don’t see how logs are internally processed.
2. Create a function or class that retrieves logs for auditing, but only if the auditor is authenticated. Expose the data in a safe, read-only manner (e.g., returning a copy or providing an iterator interface) to preserve encapsulation.
3. Demonstrate usage in main() with multiple logging calls from different parts of your hypothetical system (e.g., “Network module,” “Database module,” etc.).
4. Enforce a hard maximum number of messages (for example, 1000). Once that limit is reached, either refuse new logs or remove the oldest logs.

**Key Points to Consider:**

* How to hide the internal representation (e.g., vector, list, or file-based) from external users of Logger.
* Ensuring that log retrieval is tightly controlled (abstraction of who can or can’t see the data).
* Handling boundary conditions (limit on messages) in a well-encapsulated manner.

Task 3:  
You are designing a hospital patient management system. You need classes to store and retrieve sensitive patient information (medical history, diagnoses, billing). The system must ensure that:

* Only authorized roles (doctor, billing staff) can view or update certain parts of the data.
* Others can only see minimal, necessary information.

**Tasks:**

1. Create a PatientRecord class that:
   * Maintains patient data (name, ID, date of birth) and sensitive medical data (history, current treatments, etc.) in private members.
   * Exposes methods getPublicData() and getMedicalData() that return data in a controlled format.
2. Create Doctor and Billing classes (or roles) that:
   * Interact with PatientRecord through dedicated interfaces. For example, a Doctor can call record.updateMedicalHistory(...), but a Billing class might only call record.addBillingDetails(...).
3. Demonstrate usage in main() or in a test function, showing attempts by unauthorized roles (maybe a Receptionist or some unprivileged system user) to access private details, which should be prevented or limited.
4. Show how you enforce that certain methods remain private or protected, ensuring only authorized classes can call them. This might involve friend classes, specialized interfaces, or an inheritance-based approach that limits accessibility.

**Key Points to Consider:**

* Abstracting medical information access such that each role gets only the data it needs.
* Encapsulating sensitive fields so no direct read/write is allowed from unauthorized classes.
* Considering advanced ways to handle role-based access—possibly using friend classes, or distinct interface classes inherited by each role.

Task 4:

You are designing an airline booking system with flight routes, seats, and passengers. The system must:

* Keep track of flight details (departure, arrival, seat capacity).
* Allow seat booking, seat cancellation, or seat upgrades.
* Restrict direct changes to seat counts or passenger lists outside the official booking flows.

**Tasks:**

1. Create a Flight class that:
   * Stores flightNumber, departure, arrival, capacity, etc. in private.
   * Has public methods bookSeat(), cancelSeat(), upgradeSeat(), etc.
2. Create a Passenger class that:
   * Has private personal data (like ID, name).
   * Interacts with the Flight only through an abstracted interface. For instance, Passenger::requestBooking(Flight&).
3. Enforce rules:
   * Cannot book more seats than available.
   * Cannot cancel or upgrade seats that are not already booked.
4. Demonstrate usage in main() with multiple flights and passengers, ensuring that direct manipulation (e.g., capacity--) is disallowed from outside the Flight class.

**Key Points to Consider:**

* Separation of concerns: Flight manages seat availability, Passenger only requests seat changes through methods.
* Encapsulation so that flight capacity and seat booking lists remain private data.

H.W:  
**Task 1:** Define abstract class “Shape” that provides interface (through virtual functions) to the

two derived classes “Rectangle” and “Triangle” to implement the function called.

“getArea()”. The program will output the area of rectangle and area of triangle.

**Task 2:** Create an abstract class Car (model, price).  Include get and set methods from these

fields. The setPrice Method must be abstract. Create two subclasses Color() and

Company() from Car and include appropriate setPrice method in these classes. Finally

write a code that uses Car class and sub classes to display information about instances.

**Task 3**: Write a program to calculate bonus of the employees. The class master derives the

information from both admin and account classes which intern derives information from

class person. Create base and all derived classes having same member functions called

getdata, display data and bonus. Create a base class pointer that capable of accessing

data of any class and calculates bonus of the specified employee. (Hint: Use virtual

functions)

**Task 4:**

Implement the Student class in C++ as an abstract class with a pure virtual function getTuition().

In addition, implement the GraduateStudent class that inherits from Student and overrides the getTuition() function.

The GraduateStudent class should include a member variable for the student's research topic and a method to set and get the value of the research topic.

The getTuition() function in the Student class should take two arguments: the student's status (undergraduate, graduate, or doctoral) and the number of credit hours. The function should calculate and return the tuition amount based on the student's status and the number of credit hours.

The tuition rates are as follows:

**Undergraduate students pay $200 per credit hour**

**Graduate students pay $300 per credit hour**

**Doctoral students pay $400 per credit hour**

Your implementation should include:

1. Definition of the Student abstract class with a pure virtual function getTuition()
2. Definition of the GraduateStudent class that inherits from Student and overrides the getTuition() function
3. The GraduateStudent class should include a member variable for the student's research topic and a method to set and get the value of the research topic