

ATM System Implementation: -

**Classes:**

1)User Class

**Attributes:**

* **user\_id:** Unique identifier for the user.
* **password:** User's password.
* **balance:** Current balance in the user's account.
* **login\_attempts:** Number of failed login attempts.
* **is\_locked:** Flag indicating if the account is locked.
* **transaction\_history:** List of user's transaction history.

**Methods:**

* **deposit(amount, denominations**): Adds money to the user's account.
* **withdraw(amount, denominations):** Deducts money from the user's account.
* **check\_balance():** Retrieves the user's current balance.
* **change\_credentials(current\_password, new\_password):** Changes the user's password.
* **add\_tag\_to\_last\_transaction(tag):** Adds a tag to the last transaction.
* **\_current\_time():** Returns the current date and time.
* \_**update\_transaction\_history(transaction):** Updates the user's transaction history.

2)Admin Class (Inherits from User)

**Attributes:**

* Inherits attributes from the User class.
* **MAX\_BALANCE:** Maximum balance allowed for the admin.
* **MAX\_DEPOSIT:** Maximum deposit limit for the admin.

**Methods:**

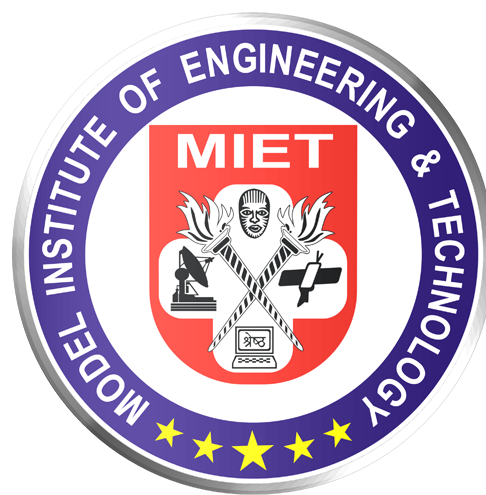
**total\_balance(user\_balances):** Calculates the total balance of all users.

**total\_deposit(deposits):** Calculates the total deposit across all users.

* **cash\_deposit(amount, denominations):** Allows the admin to deposit money.
* **notify\_low\_balance():** Notifies the admin if their balance is below a certain threshold.

3)ATM Class

**Methods:**

* **\_\_init\_\_():** Initializes the ATM system, loading existing user data.
* **load\_data():** Loads user data from a file.
* **save\_data():** Saves user data to a file.
* **create\_account(user\_id, password):** Creates a new user account.
* **change\_credentials(user\_id, current\_password, new\_password):** Changes user credentials.
* **check\_timeout(last\_interaction\_time):** Checks for user and admin inactivity.
* **login(user\_id, password):** Handles user and admin login.
* **display\_transaction\_history(user):** Displays the transaction history for a user.
* **admin\_menu():** Provides options for admin tasks.
* **user\_menu(user):** Provides options for user transactions.
* **main():** Main menu for the ATM system.
* **\_get\_denominations():** Takes input for denominations during transactions.

**Operation Flow:**

**1. Account Creation (Option 1)**

* Users can create a new account by providing a unique user ID and a password.

**2. Login (Option 2 and 3)**

* Users and admins can log in by entering their credentials.

**3. User Menu**

* Users can perform various operations:
* Check balance
* Deposit money
* Withdraw money
* Change password
* View transaction history
* Add a tag to the last transaction

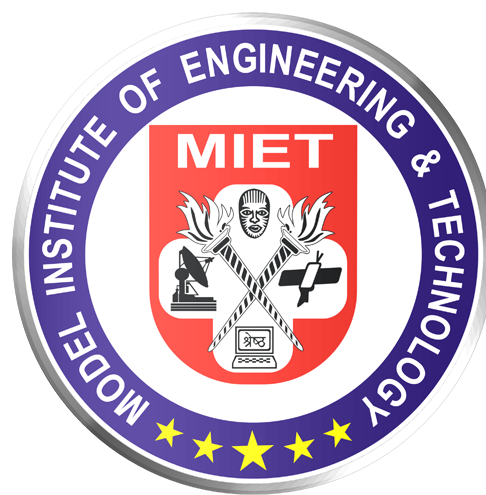
**4. Admin Menu**

* Admins can perform various operations:
* Check total balance of all users
* Check total deposit across all users
* Deposit money into the admin's account
* Receive notifications for low admin balance

**5. Exit (Option 4)**

* Users and admins can exit the system, saving any changes made during the session.

Deposit Operation (deposit method in the User class):

**1. User Initiation:**

* The user chooses to deposit money, selecting the deposit option from the user menu.

**2. Input:**

* The user provides the amount they want to deposit and the denominations of the currency they are depositing (e.g., {"100": 2, "200": 5}).

**3. Denomination Conversion:**

* The provided denominations are converted into a dictionary for easier calculation.

denominations = {int(denomination): int(count) for denomination, count in denominations.items()}

**4. Total Deposit Calculation:**

* The total amount to be deposited is calculated by summing the product of each denomination and its count.

total\_deposit = sum([denomination \* count for denomination, count in denominations.items()])

**6. Deposit Limit Check:**

* The system checks if the total deposit is within the maximum limit (e.g., $100,000).

if total\_deposit <= 100000:

**7. Balance Update:**

* If the deposit is within the limit, the user's balance is updated.

self.balance += total\_deposit

**8. Transaction History Update:**

* The deposit transaction is recorded in the user's transaction history with a timestamp.

self.\_update\_transaction\_history(f"Deposited ${total\_deposit} on {self.\_current\_time()}")

**9. Return Status:**

* The method returns True to indicate a successful deposit.

Withdraw Operation (withdraw method in the User class):

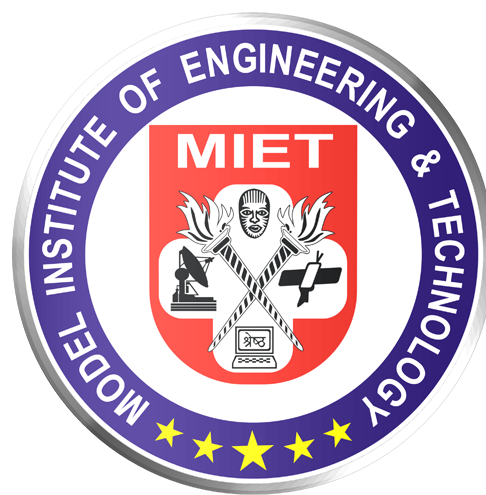
**1. User Initiation:**

* The user chooses to withdraw money, selecting the withdrawal option from the user menu.

**2. Input:**

* The user provides the amount they want to withdraw and the denominations of the currency they are withdrawing (e.g., {"100": 2, "200": 5}).

**3. Denomination Conversion:**

* The provided denominations are converted into a dictionary for easier calculation.

denominations = {int(denomination): int(count) for denomination, count in denominations.items()}

**4. Total Withdrawal Calculation:**

* The total amount to be withdrawn is calculated by summing the product of each denomination and its count.

total\_withdrawal = sum([denomination \* count for denomination, count in denominations.items()])

**5. Withdrawal Limit Check:**

* The system checks if the total withdrawal is within the withdrawal limit and if the user has sufficient funds.

if total\_withdrawal <= 50000 and self.balance - total\_withdrawal >= self.MIN\_BALANCE:

**6. Balance Update:**

* If the withdrawal is within the limit and the user has sufficient funds, the user's balance is updated.

self.balance -= total\_withdrawal

**7. Transaction History Update:**

* The withdrawal transaction is recorded in the user's transaction history with a timestamp.

self.\_update\_transaction\_history(f"Withdrew ${total\_withdrawal} on {self.\_current\_time()}")

**8. Return Status:**

* The method returns True to indicate a successful withdrawal.

OUTPUTS:

