# Banking System Project (OOP in Python)

## Objectives:

* - To design a banking system using OOP concepts.  
  - Implement account creation, deposit, withdrawal, and transaction history functionalities.  
  - Ensure Abstraction, encapsulation, inheritance, and polymorphism.

1. Exception Handling in required place

## Project Plan:

### Step 1: Requirements Gathering and Planning

1. 1. Core Functionalities:  
    - Account creation (for individual or business)  
    - Deposit and withdrawal transactions  
    - Balance inquiry  
    - Transaction history for each account  
    - Interest calculation (for savings accounts)  
    - Overdraft limit (for business accounts)  
    - Simple admin and customer access

### Step 2: Class Structure

1. 1. Class Diagram Outline:  
    - Base classes: Person, Account  
    - Derived classes: Customer, Admin, SavingsAccount, BusinessAccount  
    - Transaction, Bank

### Step 3: Define Classes and Methods

#### Person

##### Attributes:

* + name (str): Name of the person
  + address (str): Address of the person
  + phone (str): Contact number

##### Methods:

* + \_\_init\_\_(): Constructor to initialize the person’s details
  + get\_details(): Returns the person's details

#### Account (Abstract)

##### Attributes:

* + account\_number (str): Unique identifier for the account
  + balance (float): Current balance of the account
  + owner (Customer object): Owner of the account

##### Methods:

* + deposit(amount): Adds funds to the account
  + withdraw(amount): Deducts funds, ensuring no overdrafts occur in general
  + get\_balance(): Returns the current balance
  + show\_account\_details(): Displays account details

#### Customer (Inherits from Person)

##### Attributes:

* + accounts (list of Account objects): All accounts linked to the customer

##### Methods:

* + add\_account(account): Adds a new account to the customer's account list
  + remove\_account(account\_number): Removes an account from the customer’s list
  + get\_accounts\_summary(): Displays a summary of all accounts

#### Admin (Inherits from Person)

##### Attributes:

* + employee\_id (str): Unique identifier for the admin

##### Methods:

* + approve\_account(customer, account): Approves and activates a new account
  + view\_all\_customers(): Provides a summary of all bank customers

#### SavingsAccount (Inherits from Account)

##### Attributes:

* + interest\_rate (float): Interest rate for the savings account

##### Methods:

* + apply\_interest(): Calculates and applies interest to the account balance periodically

#### BusinessAccount (Inherits from Account)

##### Attributes:

* + overdraft\_limit (float): Limit for overdrafts

##### Methods:

* + Override withdraw(amount): Allows withdrawal within the overdraft limit

#### Transaction

##### Attributes:

* + transaction\_id (str): Unique identifier for the transaction
  + transaction\_type (str): Type of transaction ("Deposit" or "Withdrawal")
  + amount (float): Amount transacted
  + date (datetime): Date of the transaction

##### Methods:

* + get\_transaction\_details(): Returns the transaction details

#### Bank

##### Attributes:

* + customers (list of Customer objects): All bank customers
  + admins (list of Admin objects): All bank admins

##### Methods:

* + create\_customer\_account(): Initiates a new account creation for a customer
  + delete\_customer\_account(): Deletes a customer’s account
  + view\_transaction\_history(account\_number): Returns all transactions for a given account
  + find\_customer\_by\_account(account\_number): Finds and returns the customer associated with an account

### Step 4: Define Functionalities and Logic

1. 1. Account Creation:  
    - Admin uses create\_customer\_account() method to create a new account (savings or business).  
    - The add\_account() method in Customer links the new account to the customer.  
     
   2. Deposit:  
    - Customer initiates a deposit with the deposit() method in Account.  
    - A Transaction record is created and saved for each deposit.  
     
   3. Withdraw:  
    - Customer initiates a withdrawal using withdraw() in Account.  
    - For BusinessAccount, overdraft limit is considered.  
     
   4. Interest Calculation:  
    - apply\_interest() in SavingsAccount applies interest to the account balance based on interest\_rate.  
     
   5. Transaction History:  
    - Transaction details stored in Transaction class.  
    - Use view\_transaction\_history() in Bank to view the history for any account.

### Step 5: Code Implementation and Testing

1. 1. Code Implementation:  
    - Implement each class and method as per design.  
    - Use OOP principles to ensure encapsulation (by making attributes private and using getters/setters).  
    - Use inheritance and polymorphism where necessary.  
     
   2. Testing:  
    - Create unit tests for each method.  
    - Perform testing of different scenarios such as:  
    - Creating accounts, depositing, withdrawing (both within and outside overdraft limits).  
    - Checking interest calculation.  
    - Testing transaction history retrieval.

### Step 6: Documentation and Final Presentation

1. 1. Document Code:  
    - Add docstrings for each class and method.  
    - Comment code for readability.  
     
   2. Presentation:  
    - Summarize the functionality of each component.  
    - Highlight OOP principles used in the project.