# COMS-280 Final Project

## Class Hierarchy Design Document

The banking system is composed of a base class (“BankAccount”) and multiple derived classes called SavingAccount and CheckingAccount. This design enables polymorphism and code reusability while ensuring a scalable and maintainable system

Relationship:

1. BankAccount is the parent class, defining core banking operations
2. SavingAccount and CheckingAccount are child classes, implementing specific features.
3. The system uses polymorphism, allowing objects of SavingsAccount and CheckingAccount to be manaed using BankAccount pointers.

Class Hierarchy:

1. BankAccount (abstract base class)
   1. Attributes:
      1. String owner – name of the account holder
      2. Double balance – current account balance
   2. Methods:
      1. Virtual void deposit(double amount) = 0 - abstract method for deposits.
      2. Virtual void withdraw(double amount) = 0 – abstract method for withdrawals.
      3. Virtual void display() const – return the current balance
   3. Purpose:
      1. Servers as the base class for all account types.
      2. Ensures that all derived classes implement core banking operations
2. SavingsAccount (derived from BankAccount)
   1. Attributes:
      1. Double interestRate – interest rate applied to the savings account.
   2. Methods:
      1. Void deposit(double amount) override – adds amount to balance
      2. Void withdraw(double amount) override – prevents withdrawal if balance is insufficient
      3. Void display() const override – displays account details including interest rate.
   3. Purpose:
      1. Represents a savings account with an interest rate.
      2. Enforces withdrawal constraints to avoid overdrafts
3. CheckingAccount (Derived from BankAccount)
   1. Attributes:
      1. Double overdraftlimit – maximum overdraft allowed for the account.
   2. Methods:
      1. Void deposit(double amount) override – adds amount to balance.
      2. Void withdraw(double amount) override – allows overdraft within a predefined limit.
      3. Void display() const override – displays account details including overdraft limit.
   3. Purpose:
      1. Represents a checking account with overdraft protection
      2. Allows users to withdraw beyond their balance up to the overdraft limit.

## Implementation Plan

1. Transaction History:
   1. Transactions will be stored using linked list (std::list<Transaction>).
   2. The TransactionHistory class will manage transactions, allowing new ones to be added and retrieved
   3. Each transaction will have a type ( examples: deposit, withdrawal) and an amount.
   4. A function displayHistory() will allow users to view past transactions
2. Account management:
   1. Accounts will be created through an AccountManager template class.
   2. Users can open SavingsAccount of CheckingAccount.
   3. Each account type will support deposits, withdrawals, and balance checks.
   4. Accounts will be stored using std::vector<unique\_ptr<BankAccount>> to enable dynamic memory allocation
3. User authentication:
   1. Basic authentication can be added using a user class that stores usernames and passwords.
   2. Authentication will require users to provide correct credentials before accessing their accounts.
   3. Future implementations could integrate encryption for security.
4. Operator overloading:
   1. The += operator will be overloaded for deposits, allowing account += amount syntax
   2. The -= operator will be overloaded for withdrawals, allowing account -= amount syntax.
   3. The << operator will be overloaded for displaying account details in an easy to read format.
5. Function: main
   1. Entry point of the banking system.
   2. Creates instances of SavingsAccount and CheckingAccount.
   3. Adds accounts to AccountManager and performs banking operations.
6. Int main() {

AccountManager<BankAccount> manager;

manager.addAccount(make\_unique<SavingsAccount>("Laurie", 5000, 2.5));

manager.addAccount(make\_unique<CheckingAccount>("Larry", 1000, 500));

manager.addAccount(make\_unique<SavingsAccount>("David", 10000, 2.5));

manager.addAccount(make\_unique<CheckingAccount>("Luis", 2000, 500));performBankingOperations(manger);

return 0;

}