Int customerID  
string customerFirstName  
string customerLastName  
long customerContactNum  
string custType  
IFeesStrategy myFeesStrategy

int customerNumber  
string accountType  
int accountID  
double interestRate  
int overdraftLimit  
int accountFee  
double accountBalance  
double previousBalance  
double interestTotal

Customers

Accounts

CalcFine()

CalcFine()

Other

BankStaff

Investment (int customerNumber, int accountID)

base.accountType = “Investment”  
base.overdraft

Everyday (int customerNumber, int accountID)

base.accountType = “Everyday”  
base.overdraft

Omni (int customerNumber, int accountID)

base.accountType = “Omni”  
base.overdraft

Everyday

Investment

Omni

Accounts(int customerIDNumber, int newAccountID, string newAccountType)  
Accounts(int customerIDNumber, int newAccountID, string newAccountType, double balance)  
Accounts()  
CustomerNumber {get; set}  
AccountType {get; set}  
AccountID {get; set}  
Balance {get; set}  
Interest {get; set}  
Overdraft {get; set}  
Fee {get; set}

Customer()  
SetFeesStrategy(IFeesStrategy newStrategy)  
Customer(int customerIDNum, string customerType, string firstName, string lastName, long contactNum)  
ID {get; set}  
firstName {get; set}  
lastName {get; set}  
contactNum {get; set}  
customerType {get; set}

1

0..\*

The controller class is responsible for carrying out methods that would ordinarily require communication between different classes. Instead the controller class acts as a sort of mediator.

For the controller class to add accounts and facilitate transactions on them and between them, the following classes have been added:

WriteAccountsBinaryData()  
ReadAccountsData()  
setAccountId(CustomerNum, accType)  
GetCustomerAccounts(customerNumber)  
getInterest(customerNumber, accountNumber, accountType)  
UpdateAccountBalance(customerNumber, accountNumber, accountType, newBalance)  
AddAccount(customerNumber, accountType)  
GetAccountBalance(customerNumber, accountNumber, accountType)  
CompleteDeposit(customerNumber, accountNumber, depositAmount, accountType)  
CompleteWithdrawal(customerNumber, accountNumber, withdrawalAmount, accountType)  
WithdrawalSuccessBalance(previousBal, withdrawalAmount)

Along with these new methods, new variables (nextAccountID and accountID) were added and so were the lists:  
public List<Accounts> customerAccounts = new List<Accounts>();

List<Accounts> CorrespondingAccounts = new List<Accounts>();

WriteAccountsBinaryData()  
WriteCustomersBinaryData()  
ReadAccountsData()  
ReadCustomerData()  
setAccountID(customerNum, accType)  
setCustomerID()  
CreateCustomer(custType, firstName, lastName, contactNumber)  
DeleteCustomer(customerNumber)  
GetCustomerType(customerNumber)  
EditCustomer(customerNumber)  
UpdateCustomerDetails(customerNumber, custType, firstName, lastName, contactNumber)  
FindCustomer(customerNumber)  
GetFirstName(customerNumber)  
GetLastName(customerNumber)  
GetContactNumber(customerNumber)  
GetCustomerAccounts(customerNumber)  
getInterest(customerNumber, accountNumber, accountType)  
UpdateAccountBalance(customerNumber, accountNumber, accountType, newBalance)  
AddAccount(customerNumber, accountType)  
GetAccountBalance(customerNumber, accountNumber, accountType)  
CompleteDeposit(customerNumber, accountNumber, depositAmount, accountType)  
CompleteWithdrawal(customerNumber, accountNumber, withdrawalAmount, accountType)  
WithdrawalSuccessBalance(previousBal, withdrawalAmount)  
CustomerInfoDisplay(customerNumber)

Controller