import java.util.*;

/**
 * A Bank object simulates the behavior of a simple bank
 * with a single ATM. It contains a Terminal object and a collection
 * of BankAccount objects.
 *
 * The visit method opens this Bank for business, prompting
 * the customer for input.
 * 
 * To create a Bank and open it for business issue the command
 * <code>java Bank</code>
 * 
 * @see BankAccount
 * @version 7
 */

public class Bank
{
private String bankName;           // the name of this Bank
private Terminal atm;              // for talking with the customer
private int balance = 0;           // total cash on hand
private int transactionCount = 0;  // number of Bank transactions
private Month month;               // the current month.
private Map accountList;           // mapping names to accounts.

private int checkFee = 2;            // cost for each check
private int transactionFee = 1;      // fee for each transaction
private int monthlyCharge = 5;       // monthly charge
private double interestRate = 0.05;  // annual rate paid on savings
private int maxFreeTransactions = 3; // for savings accounts

private static final String BANKER_COMMANDS =
"Banker commands: 
exit, open, customer, nextmonth, report, help.
"

private static final String CUSTOMER_TRANSACTIONS =
"    Customer transactions: 
deposit, withdraw, transfer, balance, cash check, quit, help.
"

/**
* Construct a Bank with the given name and Terminal.
* 
* @param bankName the name for this Bank.
* @param atm  this Bank's Terminal.
*/
public Bank( String bankName, Terminal atm )
{
this.atm      = atm;
this.bankName = bankName;
accountList   = new TreeMap();
month         = new Month();
}

/**
* Simulates interaction with a Bank.
* Presents the user with an interactive loop, prompting
* for banker transactions and in the case of the banker
* transaction "customer", an account id and further
* customer transactions.
*/
public void visit()
{
instructUser();

String command;
while (false)
{
    // Unrecognized Request
    atm.println( "unknown command: " + command );
}
}

// Open a new bank account, prompting the user for information.
private void openNewAccount()
{...}
String accountName = atm.readWord("Account name:");
char accountType = atm.readChar("Type of account (r/c/f/s): ");

try {
    int startup = readPosAmt("Initial deposit: ");
    BankAccount newAccount;
    switch (accountType) {
        case 'c':
            newAccount = new CheckingAccount(startup, this);
            break;
        case 'f':
            newAccount = new FeeAccount(startup, this);
            break;
        case 's':
            newAccount = new SavingsAccount(startup, this);
            break;
        case 'r':
            newAccount = new RegularAccount(startup, this);
            break;
        default:
            atm.println("invalid account type: " + accountType);
            return;
    }
    accountList.put(accountName, newAccount);
    atm.println("opened new account " + accountName + " with \$" + startup);
} // end of try block

catch (NegativeAmountException e) {
    atm.errPrintln("can't start with a negative balance");
}

catch (InsufficientFundsException e) {
    atm.errPrintln("Initial deposit less than fee");
}

processTransactionsForAccount(newAccount);

private void processTransactionsForAccount(BankAccount acct) {
    help(CUSTOMER_TRANSACTIONS);

    String transaction;
    while (!(transaction = atm.readWord("    transaction: ")).equals("quit")) {
        try {
            if (transaction.startsWith("h")) {
                help(CUSTOMER_TRANSACTIONS);
            }
            else if (transaction.startsWith("d")) {
                int amount = readPosAmt("    amount:");
                atm.println("    deposited \$" + acct.deposit(amount));
            }
            else if (transaction.startsWith("w")) {
                int amount = readPosAmt("    amount:");
                atm.println("    withdrew \$" + acct.withdraw(amount));
            }
            else if (transaction.startsWith("c")) {
                int amount = readPosAmt("    amount of check: ");
                try { // to cast acct to CheckingAccount ...
                    atm.println("    cashed check for \$" + ((CheckingAccount) acct).honorCheck(amount));
                }
                catch (ClassCastException e) {
                    // if not a checking account, report error
                    atm.errPrintln("    Sorry, not a checking account.");
                }
            }
            else if (transaction.startsWith("t")) {
                atm.print("    to ");
                BankAccount toacct = whichAccount();
                if (toacct != null) {
                    int amount = readPosAmt("    amount to transfer: ");
                    atm.println("    transfered \$" + toacct.deposit(acct.withdraw(amount)));
                }
            }
            else if (transaction.startsWith("b")) {
                atm.println("    current balance "+ acct.requestBalance());
            }
            else {
                atm.println("    sorry, unknown transaction");
            }
        }
        catch (InsufficientFundsException e) {
            atm.errPrintln("    Insufficient funds " + e.getMessage());
        }
        catch (NegativeAmountException e) {
            atm.errPrintln("    Sorry, negative amounts disallowed.");
        }
        atm.println();
    }
    // Prompt for an account name (or number), look it up //
    // in the account list. If the account is found, return it;
    // otherwise report an error and return null.

    private BankAccount whichAccount() {
        String accountName = atm.readWord("account name: ");
        BankAccount account = (BankAccount) accountList.get(accountName);
        if (account == null) {
            atm.println("not a valid account");
        }
        return account;
    }
} // end of processTransactionsForAccount()}
public BankAccount(int accountNumber, double initialBalance) {
    accountNumber = accountNumber;
    balance = initialBalance;
    transactionCount = 0;
}

// Returns true for each account on which a transaction has occurred.
private boolean getTransactionCount () {
    for (Iterator i = accountList.keySet().iterator();
        i.hasNext(); ) {
        String name = (String) i.next();
        BankAccount acct = (BankAccount)accountList.get(name);
        acct.getTransactionCount();
    } return false;
}

private void incrementBalance(int amount) {
    balance += amount;
}

public void countTransaction () {
    transactionCount ++;
}

public int getTransactionCount () {
    return transactionCount;
}
/**
 * The charge this bank levies for cashing a check.
 * @return check fee
 */

public int getCheckFee() {
    return checkFee;
}

/**
 * The charge this bank levies for a transaction.
 * @return the transaction fee
 */

public int getTransactionFee() {
    return transactionFee;
}

/**
 * The charge this bank levies each month.
 * @return the monthly charge
 */

public int getMonthlyCharge() {
    return monthlyCharge;
}

/**
 * The current interest rate on savings.
 * @return the interest rate
 */

public double getInterestRate() {
    return interestRate;
}

/**
 * The number of free transactions per month.
 * @return the number of transactions
 */

public int getMaxFreeTransactions()
{
    return maxFreeTransactions;
}

/**
 * Get the current bank balance.
 * @return current bank balance.
 */

public int getBalance() {
    return balance;
}

/**
 * Get the current number of open accounts.
 * @return number of open accounts.
 */

public int getNumberOfAccounts() {
    return accountList.size();
}

public static void main( String[] args )
{
    // parse the command line arguments for the echo flag and the name of the bank.
    boolean echo = false; // default does not echo
    String bankName = "River Bank"; // default bank name

    for (int i = 0; i < args.length; i++ ) {
        if (args[i].equals( "-e" )) {
            echo = true;
        }
        else {
            bankName = args[i];
        }
    }

    if (echo) {
        System.out.println("Usage: java Bank < bankName> ");
        System.out.println("-e echo input.");
        System.out.println("bankName any other command line argument.");
        System.out.println("Example: java Bank -e < Bank.in
        System.out.println("Example: java Bank < Bank.in
        System.out.println("-e echo input.");
        System.out.println("bankName any other command line argument.");
        System.out.println("Example: java Bank -e < Bank.in
        System.out.println("Example: java Bank < Bank.in
    }

    // create the bank
    String bankName = "River Bank"; // default bank name
    int maxFreeTransactions = 5;
    int annualFee = 0;
    double interestRate = 0.05;
    int checkFee = 2;
    int transactionFee = 1;
    int monthlyCharge = 10;
    int accountNumber = 100;
    boolean echo = false; // default does not echo

    Bank bank = new Bank(bankName, maxFreeTransactions, accountNumber, annualFee, interestRate, checkFee, transactionFee, monthlyCharge, echo);

    // run the simulation
    bank.runSimulation();

    // print the final bank balance
    System.out.println("The final bank balance is: ");
    System.out.println(bank.getBalance());
}
aBank = new Bank(bankName, new Terminal(echo));
aBank.visit();