import java.util.\*;

// Abstract Account class

public abstract class Account {

protected String accountNo;

protected double balance;

protected String accountType;

protected List<Transaction> transactions = new ArrayList<>();

public Account(String accountNo, double balance, String accountType) {

this.accountNo = accountNo;

this.balance = balance;

this.accountType = accountType;

}

public void deposit(double amount) {

balance += amount;

transactions.add(new Transaction(UUID.randomUUID().toString(), amount, new Date(), "deposit", this));

}

public boolean withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

transactions.add(new Transaction(UUID.randomUUID().toString(), amount, new Date(), "withdraw", this));

return true;

}

return false;

}

public boolean transfer(Account target, double amount) {

if (withdraw(amount)) {

target.deposit(amount);

return true;

}

return false;

}

public double getBalance() {

return balance;

}

public String getAccountNo() {

return accountNo;

}

public String getAccountType() {

return accountType;

}

public List<Transaction> getTransactions() {

return transactions;

}

public String getAccountInfo() {

return "AccountNo: " + accountNo + ", Type: " + accountType + ", Balance: " + balance;

}

}

// SavingsAccount subclass

class SavingsAccount extends Account {

private double interestRate;

public SavingsAccount(String accountNo, double balance, double interestRate) {

super(accountNo, balance, "Savings");

this.interestRate = interestRate;

}

public void calculateInterest() {

double interest = balance \* interestRate / 100;

deposit(interest);

}

}

// CurrentAccount subclass

class CurrentAccount extends Account {

private double overdraftLimit;

public CurrentAccount(String accountNo, double balance, double overdraftLimit) {

super(accountNo, balance, "Current");

this.overdraftLimit = overdraftLimit;

}

@Override

public boolean withdraw(double amount) {

if (balance + overdraftLimit >= amount) {

balance -= amount;

transactions.add(new Transaction(UUID.randomUUID().toString(), amount, new Date(), "withdraw", this));

return true;

}

return false;

}

public boolean checkOverdraft() {

return balance < 0;

}

}

// Transaction class

class Transaction {

private String transactionId;

private double amount;

private Date date;

private String type;

private Account account;

public Transaction(String transactionId, double amount, Date date, String type, Account account) {

this.transactionId = transactionId;

this.amount = amount;

this.date = date;

this.type = type;

this.account = account;

}

public String toString() {

return "TransactionID: " + transactionId + ", Type: " + type + ", Amount: " + amount + ", Date: " + date;

}

}

// Customer class

class Customer {

private String customerId;

private String name;

private String address;

private List<Account> accounts = new ArrayList<>();

public Customer(String customerId, String name, String address) {

this.customerId = customerId;

this.name = name;

this.address = address;

}

public void openAccount(Account account) {

accounts.add(account);

}

public List<Account> getAccounts() {

return accounts;

}

public String getCustomerId() {

return customerId;

}

public String getName() {

return name;

}

public List<Transaction> viewTransactionHistory(Account account) {

if (accounts.contains(account)) {

return account.getTransactions();

}

return Collections.emptyList();

}

// Accessor method for account details

public String getAccountDetails(Account account) {

if (accounts.contains(account)) {

return account.getAccountInfo();

}

return "Account not found for this customer.";

}

}

// Branch class

class Branch {

private String branchName;

private String location;

private List<Account> accounts = new ArrayList<>();

private List<Customer> customers = new ArrayList<>();

public Branch(String branchName, String location) {

this.branchName = branchName;

this.location = location;

}

public String getBranchName() {

return branchName;

}

public void addAccount(Account account) {

accounts.add(account);

}

public void addCustomer(Customer customer) {

customers.add(customer);

}

public List<Account> getAccounts() {

return accounts;

}

public List<Customer> getCustomers() {

return customers;

}

// Get all transactions in the branch from all accounts

public List<Transaction> getAllTransactions() {

List<Transaction> allTransactions = new ArrayList<>();

for (Account acc : accounts) {

allTransactions.addAll(acc.getTransactions());

}

return allTransactions;

}

}

// Bank class

class Bank {

private String bankName;

private String bankAddress;

private List<Branch> branches = new ArrayList<>();

public Bank(String bankName, String bankAddress) {

this.bankName = bankName;

this.bankAddress = bankAddress;

}

public void addBranch(Branch branch) {

branches.add(branch);

}

public List<Branch> getBranches() {

return branches;

}

public Branch selectBranch(String branchName) {

for (Branch branch : branches) {

if (branch.getBranchName().equalsIgnoreCase(branchName)) {

return branch;

}

}

return null;

}

public Customer selectCustomer(Branch branch, String customerId) {

for (Customer customer : branch.getCustomers()) {

if (customer.getCustomerId().equals(customerId)) {

return customer;

}

}

return null;

}

// Get all transactions in a branch

public List<Transaction> viewBranchTransactions(Branch branch) {

return branch.getAllTransactions();

}

}

// Clerk class

class Clerk {

private String clerkId;

private String name;

public Clerk(String clerkId, String name) {

this.clerkId = clerkId;

this.name = name;

}

public Customer createCustomer(String customerId, String customerName, String address, Branch branch) {

Customer newCustomer = new Customer(customerId, customerName, address);

branch.addCustomer(newCustomer);

return newCustomer;

}

}

// Manager class

class Manager {

private String managerId;

private String name;

public Manager(String managerId, String name) {

this.managerId = managerId;

this.name = name;

}

// View all accounts in the bank

public void viewAllAccounts(Bank bank) {

System.out.println("Manager View: All Accounts in the Bank");

for (Branch branch : bank.getBranches()) {

System.out.println("Branch: " + branch.getBranchName());

for (Account account : branch.getAccounts()) {

System.out.println("\t" + account.getAccountInfo());

}

}

}

}