**Bank System**

System Description

Requirements:

The bank system holds a client list and allows them to deposit and withdraw cash money and also manage accounts

Bank system provides the following:

* client management – which includes
  + add/remove/update client
* account management
  + each client may own several accounts
  + clients can add/remove accounts
  + daily interest calculations – done automatically by the bank system
* Bank cash management
  + bank fortune is the total amount taken from client balances and accounts
  + operations that effect bank cash are:
    - commission for each withdraw or deposit made by the clients
    - closing / opening accounts
    - interest calculations on each account
* Activity log
  + Logs are records that describes the events that occurs along time
  + each Log object describes a single event
  + all the following events, that might effects bank cash balance, are logged:
    - add / remove client
    - update client balance
    - add / remove account
    - client deposit & withdraw
    - daily account balance update of accounts balance according to each interest
  + the system allows to view a logged information

Here is a description of the general structure and functionality of the Bank system:

**Bank**

* data members:
  + client list
  + log service
  + accounts commission updater
* functionality:
  + view balance - this operation must calculate the total bank balance each time the operation is called. The balance is calculated by summing the total client balance and the total accounts balance
  + add / remove client - (effects bank total balance & should be logged)
  + get client list
  + view logs – allows to view activities
  + start account updater daily process

**Client**

* details:
  + id
  + rank (regular, gold, platinum)
  + name
  + balance (instant money that can be withdrawn or deposited)
  + list of accounts
  + commission rate
  + interest rate
* functionality:
  + get methods for: id, name, balance, accounts
  + set methods for: name, balance
  + get fortune – returns the sum of client balance + total accounts balance
  + add account
  + remove account – money is transferred to the clients balance – no change in the bank total balance
  + deposit & withdraw – adds & removes money to clients balance – each action adds a commission to the bank total according to the following:
    - regular clients pays a commission rate of 3%
    - gold clients pays a commission rate of 2%
    - platinum clients pays a commission rate of 1%
  + update accounts balance (daily auto process)
    - regular clients gets a daily interest rate of 0.1%
    - gold clients gets a daily interest rate of 0.3%
    - platinum clients gets a daily interest rate of 0.5%

**Account**

* details:
  + id
  + balance
* functionality:
  + get methods for: id, balance
  + set methods for: balance

**Log – each Log instance represents a single action (event) in the Bank system**

* details:
  + time stamp
  + client-id
  + amount – the cash amount (+/-) of the operation
  + description – describes event details and will be one of the following:
    - "client added"
    - "client removed"
    - "client balance updated – deposit"
    - "client balance updated – withdraw"
    - "account update – closed"
    - "account update – opened"
    - "bank auto account interest update"
* functionality:
  + print details

**Logger – is used to store and view Log messages**

* details:
  + driver name (which will hold a DB driver for future use)
* functionality:
  + log – accepts a Log object and stores it in the Log DB
  + get logs – loads and returns all logs

**Bank System**

Phase 1 - System Design



* 1. Module 6- encapsulation & basic flow control

Phase 2 – Building Classes

In this phase you'll create the building blocks of the bank system.

Later on, other bank features will be added to those classes.

In this phase the following classes will be defined:

Log

Logger

Account

Client

Bank

Classes details :

Log class

* Attributes:
  + timestamp – long – describes the time of action occurrence
  + client-id – int – the client-Id owning this action
  + description – String – short description of the action (messages are detailed in previous section)
  + amount – float – the amount of money of this action
* Methods:
  + Constructor (timestamp, client-id, description, amount)
  + getData() : String – append all action information into a string.

For example:

"2387781632, client-123, account update – opened, 120,000"

Logger class

* Attributes:
  + driverName – String – will be empty string by now
* Methods:
  + Constructor (driverName)
  + log( Log) : void - implement to print Log.getData() on screen.

NOTE: every time you see in this document "Log this operation" it means – create a new Log instance and use Logger.log(Log) to print its details.

* + *getLogs() : Log[]* - leave empty for now

Account class

* Attributes:
  + id - int
  + balance - float
* Methods:
  + Constructor (id, balance)
  + get methods for: id, balance
  + set methods for: balance - log this operation

Client class

* Attributes:
  + id - int
  + name - String
  + balance - float
  + accounts – Account [5]
  + commission rate – float (value=0 for now)
  + interest rate – float (value=0 for now)
  + logger - Logger
* Methods:
  + Constructor (id, name, balance) – constructs a new client with zero accounts. Constructor also instantiates the Logger.
  + get methods for: id, name, balance, accounts
  + set methods for: name, balance
  + addAccount (Account) : void - add the account to the array and log the operation.

You should seek the array and place the Account where the first null value is found.

Clients shouldn't be able to add Accounts with already existing IDs.

* getAccount(int id) : Account – returns the account with the specified id or null if does not exist
  + remove account (int id) : void - remove the account with the same id from the array (by assigning a 'null' value to the array[position]) & transfers the money to the clients balance. Log the operation via creating Log object with appropriate data and sending it to the Logger.log(..) method.
  + deposit(float) & withdraw(float) : void - implement to add / remove the amount from clients balance according to the commission (which is now zero). Use the commission data member in your calculation). Example: if a client deposits 1000, the actual amount added to his account will be = 1000-commission. If a client withdraws 1000, the actual amount taken from his account will be = 1000+commission.
  + autoUpdateAccounts() : void – this method loops over the client accounts, calculates the amount to add according to the client interest (meanwhile it is zero) and adds it to each account balance. Use the interest data member in your calculation. Log this operation.

Note: this method will be invoked once a day by a background thread – we will add it in future phases.

* + getFortune() : float – returns the sum of client balance + total accounts balance.

Bank class

* Attributes:
  + clients – Client [100]
  + logService - Logger
  + account updater – leave this one for now, we'll add it later
* Methods:
  + An empty constructor that instantiates the clients array
  + getBalance() : float - this operation returns the bank balance. The balance is calculated by summing the total clients balance and the total accounts balance – you should use Client.getFortune() method of each client. The function sums all clients balances and return the total as the bank current balance.
  + addClient(Client) : void - adds the client to the client array and log the operation.

You should seek the array and place the Client where the first null value is found.

* + removeClient(int id) : void - remove the client with the same id from the array (by assigning a 'null' value to the array[position]). Log the operation
  + getClients() : Client []
  + view logs – prints all logs that are stored in the logger – leave empty for now
  + *startAccountUpdater() : void* - leave empty for now

Note :

* + Currently the timestamp value sent to the Log constructor should be zero.
  + Currently, "log this operation" means creating a Log object, filling it with the action details and print it via Logger.log(Log) which invokes and prints the Log.getData();
  1. Module 6-7 Inheritance & polymorphism

Phase 3 – System Core

In this phase you'll create the different types of Clients and implement more functionality in the bank system.

In this phase the following classes will be defined all extends Client.class:

* RegularClient
* GoldClient
* PlatinumClient

Bank class:

* Turn this class into a Singleton
* Add a private float static field totalCommission to sum all commissions due to client withdraw & deposit operations.
* Add public static method updateTotalCommission(float) that allows to update the totalCommission field. Clients will use this static method in order to add withdraw/deposit commissions. We will update Client.class to do it in the next step.
* Update setBalance() to add the commissionSum to the bank total balance
* Add a new method – printClientList() : void that prints the client details using the new toString() implementation you'll create in the next step.
* Update removeClient(id) to take a Client: Bank.removeClient(Client) and perform the check using Client.equals(Object) method

Client class:

* Add the following attributes
  + commissionRate – final, float, protected
  + interestRate – final, float, protected
* Update Client to be an abstract class
* Change both interest & commission access modifiers to be

protected instead of private

* Update withdraw() method to calculate commission according to
* the amount & commission rate and call Bank.updateTotalCommission(float) in
* Bank class to add the commission to the bank totalCommission.
* Override the equals() method to perform the check according to

client id value.

Sub-Classes details :

RegularClient, GoldClient, PlatinumClient

* all extends Client
* all have default constructor which populates commissionRate & interestRate according to the following:
  + commission rate – float with a fixed values:
    - regular clients pays a commission rate of 3%
    - gold clients pays a commission rate of 2%
    - platinum clients pays a commission rate of 1%
    - relevant for withdraw operation only
  + interest rate – float with a fixed values:
    - regular clients gets a daily interest rate of 0.1%
    - gold clients gets a daily interest rate of 0.3%
    - platinum clients gets a daily interest rate of 0.5%
    - later, a system thread will use it
  + Note: all calculations should work fine since now the relevant interest & commission values are used.
* Override toString() method to return the client type (as a String – "Platinum", "Gold", "Regular") and ID

Account class:

* Override the equals() method to perform the check according to
* the id value.
* Update the Client.removeAccount(id) to take an Account
* [Client.removeAccount(Account)] and perform the check
* using Account.equals(Object) method

Logger class:

* Update log(Log) method to be static
* Update all log creators to use Logger.log(...) method. Replace your code with Logger.log(...) wherever you instantiate a Logger object. Logger instances should never be created.

Log class

* Override toString() method to print log details (client ID, message & timestamp) – you may simply change the 'getData()' method name