BANKING APPLICATION Approach Document

We will be building a Banking Application incrementally

1. As a first step let us identify a name for our bank, say BankOfPratianAppln. Let us create a new Java Project called BankOfPratianAppln.
2. Our bank has two types of accounts : Savings and Current.
3. What should an Account provide?
   Class Account should provide the following

```
String accNo;
String name;
String pin;
boolean active;
Date dateOfOpening;
double balance;
PrivilegeType privilegeType;
```

   where PrivilegeType is an enum having the following constants *REGULAR*, *GOLD*, *PREMIUM.*

   - ? What methods should the **Class Account** provide?
     - ✓ Get, Set Methods for instance variables
     - ✓ open()
       - Open an account
       - Generate an AccNo
       - Set active to true
     - ✓ close()
       - Close an account
       - Set active to false
       - Set balance to 0
4. Savings and Current will inherit from Account
   - ? Do they need any instance variables?
     - ✓ No, since they are inheriting from Account, they can use the instance variables of the base class
   - ? So what should be the access specifier of the instance variables of the Class Account
     - ✓ Protected
   - ? What methods will Class Savings and Class Current have?
     - ✓ Necessary parameterized constructors
     - ✓ boolean open()
     - ✓ boolean close()
     - ✓ String getAccType() which will return “Savings” or “Current appropriately
   - ? Every Account has an accNo. How do we give/generate values for it?
     - ✓ We can have create an utility class IDGenerator to generate Account nos.

```
Class IDGenerator
{
   public static int ID=1000;
   public static int generateID()
    {        return ID++;
```

```
    }
  }
```

- ✓ We can assign in the generated ID in the constructor

```
public class Current extends Account {
    public Current(){
        accNo= "CUR"+IDGenerator.generateID();
                }
     ……}
```

- ? Should a user be allowed to instantiate Account class?
  - ✓ No. Hence it has to be made abstract
- ? To bring in conformity to specification or enforce a contract what needs to be done?
  - ✓ Create an interface IAccount containing the public methods from Account including the following methods
    - String getAccType()
    - boolean open()
    - boolean close()
- ? What implementation should we provide for the methods open() and close() in Class Account ?
  - ✓ Since these methods can be given implementations only in the subclasses, we can make as abstract methods in the Class Account

5. How does a customer like you and me open an account?
   - ✓ We go to a bank and approach the AccountManager who will ask us what type of account we want (Savings or Current) and then get necessary details from us and create an account for us and probably give us a passbook.
   - ✓ So to simulate the real world, let us create a Class AccountManager
6. What functionality does the AccountManager provide?
   1. IAccount createAccount(String name, String pin, **double** balance, PrivilegeType privilegeType, AccountType accType)
7. In the createAccount method the AccountManager has to create either a Savings Account or Current Account depending on the AccountType requested by the user. If in future our Bank decides to introduce Savings-Corporate, Savings-NRI, Current-NRI accounts then the Account manager should constantly keep modifying the code of the createAccount Method.
   1. Is there an alternate way to avoid continuous if-else ladder of checking the AccountType and creating the account?
      ✓AccountFactory is a solution to the above problem.
8. Class AccountFactory has a method createAccount() which accepts AccountType as argument and returns a new object of type savings or current.
   - AccountType can be an enum with *SAVINGS*, *CURRENT as constants*
   - InvalidAccountTypeException is thrown
9. Let us add more functionality to AccountManager. AccountManager should be able to deposit to an account and withdraw from an account. Before a withdrawal, checks needs to be performed and appropriate exceptions needs to be thrown.
   - boolean withdraw(IAccount fromAccount, double amount, int pin)
     - ✓ InvalidPinException
     - ✓ InsufficientbalanceException
     - ✓ InactiveAccountException

✓ AccountDoesNotExistsException

- boolean deposit(IAccount toAccount, double amount)

10. An AccountManager should be able to **transfer funds** from one account to another
    1. What will be the signature of the method TransferFunds be?
       ✓ boolean transferFunds(fromAcc, toAcc, amt , pin)
    2. The above signature looks fine but what is the RBI or any regulatory body expects us to verify the pan card no also.
       ✓boolean transferFunds(fromAcc, toAcc, amt , pin, pan)
    3. If the Aadhar project is implemented then what if we need to include that no. also?
    4. Is there an alternate way to standardize on the signature and yet accommodate changes
       ✓Yes . Create a Class Transfer which has fromAcc, toAcc, amt , pin as instance variables and pass the transfer object as an argument as
       - boolean transferFunds(Transfer transfer)
    5. Is both fromAccount and toAccount necessary in Class Transfer
       ✓ Yes the Account Manager needs to know the said info.
    6. Is pin necessary ?
       ✓Yes, the pin given in the transfer will be verified with the pin stored in the fromAccount

11. Before transfering funds Account manager should
    1. Check accounts are active else throw inactiveAccountException
    2. Verify the given pin with the pin of fromAccount else throw invalidPinException
    3. Check balance else throw insufficientbalanceException
    4 Check daily limit - of the person transferring amount
12. What is daily limit?
    1. Every PrivilegeType will have a limit as to how much money can the account holder transfer in a day
    2. The following are the guidelines
       - REGULAR=100000.0
       - GOLD=200000.0
       - PREMIUM =300000.0
13. Since the AccountManager has enough work on hand, let us introduce an AccountPrivilegeManager who maintains the daily limit for each PrivilegeType.
14. Class AccountPrivilegeManager will read from configuration file dailyLimits.properties and get the limit for the privilege type and pass it to the AccountManager.
    1. Why a configuration file?
       ✓ Configuration file helps us to add a new privilege say PREMIUM_ELITE and more, dynamically rather than changing the code.
       ✓Even if couple of new privileges are introduced, the change is only done to the properties file which is a text file and the code in not touched.
    - AccountPrivilegeManager should have static map<PrivilegeType, double> which fills when class is loaded
    - AccountPrivilegeManager should also have a method
      - **double** getDailyLimit(PrivilegeType privilegeType)
      - If privilegeType is invalid, it should throw InvalidPrivilegeTypeException
15. Having got the dailyLimit from AccountPrivilegeManager, how will the AccountManager check whether the dailyLimit is exceeded?

- To check the daily limit we need to know all the transactions done by the account in the day like deposits, withdraw, transferFunds
- We need the common details from the above transactions like fromAcc, amount , dateTime
- To encapsulate the common details let us create a Class Transaction

16. Class Transaction has the following data
    - int transID; - Can generate using IDGenerator
    - IAccount fromAccount;
    - Date tranDate;
    - double amount;
17. How can we store all the Transactions?
    1. What transactions do we need to keep track of?
       ✓We need to keep a track of all the deposits, withdrawals and transferOfFunds
       ✓ That is, we need to know for a particular fromAcc what are the transactions done
       ✓Transactions are of 3 categories : deposits, withdrawals, transferOfFunds
       ✓So we have to store the list of deposits, list of withdrawals and list of transferOfFunds
       ✓We can have Map<TransactionTypes ,List<Transaction>>. Here TransactionTypes is a enum with constants as *TRANSFER* ,*WITHDRAW* , *DEPOSIT* .
       ✓So for one account holder, we can have this Map
    2. How do we store the list of transaction according to transactiontypes for all the account holders?
       ✓ Solution: Map<FromAccountNo, Map<TransactionTypes ,List<Transaction>>>
    3. Where do we store this Map of a Map?
       ✓Create a Class Transaction Log which has
       Map<FromAccountNo, Map<TransactionTypes ,List<Transaction>>> and the following **static** methods.
       - Method to get all transactions ie the transactionLog
         - **public static** Map<String, Map<TransactionTypes, List<Transaction>>> getTransactions()
       - Method to get all the transactions for an account
         - **public static** Map<TransactionTypes, List<Transaction>> getTransactions(String accNo)
       - Method to get all the transactions for an account and type of transaction
         - **public static** List<Transaction> getTransactions(String accNo, TransactionTypes type)
       - Method to log a transaction
         - **public static void** logTransaction(String accNo, TransactionTypes type, Transaction transaction)
    4. Are there any exceptions to be handled?
       ✓ TransactionNotFoundException – is thrown if the entire Map of a Map is null
       ✓InvalidTransactionTypeException – is thrown if the TransactionType is not one of DEPOSIT, WITHDRAW, TRANSFER
18. After reading the story in a story, we should not lose track of the start point of the original story , which in our case is TransferOfFunds (Step 11)
    All set to transferFunds (please refer Step 11)
    * Check 1,2,3 done
    * Check daily limit and if exceeded then throw daily limit exceeded exception
    * Withdraw, Deposit and Transfer should have unique transID generated. (Use IDGenerator)

* When transaction type is a transfer then Withdraw, deposit and transfer should have the same transaction ID

19. Let us spice it up. There it goes….
    1. Every Account has a Policy.
    2. Policy has minimumBalance to be maintained and the rateOfInterest applicable.
    3. Every Policy class has a contract to adhere to – **IPolicy** Interface which has public methods
       ✓**double** getMinBalance();
       ✓**double** getRateOfInterest();
    4. Since any policy is bound to change let us have a **PolicyFactory** which is a **Singleton** and a **Factory Pattern.**
    5. PolicyFactory reads from the configuration file – Policies.properties
       SAVINGS-REGULAR:5000.0,4.0
       SAVINGS-GOLD:25000.0,4.25
       SAVINGS-PREMIUM:100000.0,4.75
       CURRENT-REGULAR:25000.0,2.0
       CURRENT-GOLD:100000.0,2.25
       CURRENT-PREMIUM:300000.0,2.75
    6. PolicyFactory has a method
       ✓ IPolicy createPolicy(String accType, String privilege)
    7. An InvalidPolicyTypeException is thrown if the Policy Type is not amongst the one listed.
20. What changes to the existing code needs to be done because of Policy Inclusion?
    1. IAccount should have
       - **public** IPolicy getPolicy();
    2. Class Account should have
       - **protected** IPolicy policy;
       - Get set methods for policy
    3. AccountManager should modify the **createAccount** method
       - Use PolicyFactory to create a policy object and set it to the account object
       - Check if the balance remitted at the time of account opening satisfies the minBalance criteria of the Account Holder's , if not throw MinBalanceNeedsToBeMaintainedException.
       - If all the criteria are satisfied then Open the account and return the object else throw UnableToOpenAccountException
    4. AccountManager should modify the **withdraw, transferFunds** method
       - Check minBalance() before withdraw, transferFunds else throw MinBalanceNeedsToBeMaintainedException.
21. A new twist and turn …in other words An ENHANCEMENT. Till now we were able to transferFunds within our BankOfPratian. Let us introduce ExternalTransfer which will help us transferFunds to another bank.
    1. Create Class ExternalTransfer **IS A** Transaction having
       - toExternalA/c
       - fromAcPin
    2. Create Class ExternalAccount
       - String AccNo
       - String BankCode - ICICI, CITI
       - String BankName

3. Modify enum TransactionType
   - Add a TransactionType EXTERNALTRANSFER in the enum
4. Create Enum TransactionStatus with constants {OPEN, CLOSE};
5. Modify **Class Transaction**
   - Add **status** of type enum TransactionStatus in Transaction Class
6. If a transaction is an EXTERNALTRANSFER then the status is set to OPEN.
   - ? Can the status for TRANSFER be set to OPEN?
     - ✓ It can be set to OPEN and CLOSE it once the transaction is complete, meaning amount from fromAcc is deducted and added to ToAcc.
   - ? When will the status for EXTERNALTRANSFER be set to CLOSE?
     - ✓ The status will be set to CLOSE when the amount from the FromAcc is successfully transferred to the ToAcc in the External Bank like ICICI or CITI Bank.

22. Let us now also include Database Integration. Let us create a new database. The following are the tables in the database
    1. **TABLE : ACCOUNT**
       1. accNo– Varchar(15) (PK)
       2. name– Varchar(30)
       3. pin– Varchar(4)
       4. active– boolean
       5. dtOfOpening–Date
       6. balance– Double
       7. privilegeType– Varchar(15) – use Check Constraint to check if privilegeType IN (*REGULAR*, *GOLD*, *PREMIUM)*
       8. accType– Varchar(15) – use Check Constraint to check if AccountType IN (*SAVINGS, CURRENT)*
    2. **TABLE: TRANSACTION**
       1. TransID (pk) – INT
       2. TransactionType – use Check Constraint to check if TransactionType in (Withdraw, Deposit, Transfer, ExternalTransfer)
       3. accNo– Varchar(15) (FK)
       4. TransDate–Date
       5. amount– Double
    3. **TABLE : HCLBANK**
       1. AccId – Varchar(15)
       2. amt - double
    4. **TABLE :CITIBANK**
       1. AccId – Varchar(15)
       2. amt - double
    5. **TABLE :ICICIBANK**
       1. AccId – Varchar(15)
       2. amt - double
23. When an EXTERNALTRANSFER request comes in, we have to invoke a method of the Class provided by External Bank say CITI or ICICI or SBI and deposit the money in the toAccount of that respective bank.
    If each of these banks expose their own classes with different method names , we first need to create objects of these classes and call the methods.

In order to have an external bank transfer, our bank has to liaise with CITI, ICICI to start with and many other banks in due course. If we had liaised with around 25 banks, then we would have to check which external bank our transfer is for and create an object of that class and call its respective deposit method. This scenario becomes so tedious. With respect to coding , the external transfer method would have an unending if else if ladder.
A better solution for us would be to expose a Contract to these Banks.
In order to bring in conformity amongst the Banks that we liaise with for external transfer, we provide them with an Interface

```
public interface IExternalBankService
 {
  boolean deposit(String accId ,double amt) throws AccountDoesNotExistException;
}
```

The Banks (CITI, ICICI) would create their own **class** CITIBankService and **class** ICICIBankService which implements the interface and provide implementation for the deposit method , which on invocation would update the respective table (CITIBANK, ICICIBANK) in the database.

24. In our BankOfPratianAppln we are doing a role play. We play both the roles, i.e of BankOfPratian and of CITIBank and ICICI Bank…
    So we create class

```
public class ICICIBankService implements IExternalBankService
{
    public boolean deposit(String accNo, double amt) throws AccountDoesNotExistException
          { // Write ICICIBank Specific code to deposit
          }
          }
```

25. By exposing our IExternalBankService Interface, we have ensured the method name is the same for all the External Banks. But we still have to create objects of these individual classes and call the deposit().
    A solution would be to create an ExternalBankServiceFactory should accept bankCode and create object of respective class
    Let ExternalBankServiceFactory follow Singleton + Factory pattern,
    have a Map<String BankCode, Interface ExternalBankService> serviceBankPool;
    The factory uses the configuration file named serviceBanks.properties which has
    BankCode:ClassName
    ICICI :ICICIBankService
    CITI :CITIBankService
    As and when the configuration file is read using the properties, the BankCode is the key and the object of the ClassName is created and populated in the Map.

```
String bankCode= key.toString();
//Code Snippet
for(Object key: properties.keySet())
 {
  String bankCode= key.toString();
  String className= properties.getProperty(bankCode);
 IExternalBankService bankObj=
(IExternalBankService)Class.forName(className).newInstance();
serviceBankPool.put(bankCode, bankObj);
```

}

26. AccountManager will have a method transferFunds(ExternalTransfer).
    He would have to check the following
    1. Check if FromAcc is active
    2. Check if given Pin is valid
    3. Check Sufficient Balance is available
    4. Check if min balance is maintained after transfer
    5. Check if the transaction is within the permissible daily limits
    6. After all checks, the transaction has to be added to the transactionLog and the status of the transaction should be set as OPEN.
27. ExternalTransfer to other banks take varying no. of days. We need a periodical check to see if we have successfully transferred the amount to the ExternalBank successfully and to debit the fromAcc of our Bank and set the status as CLOSED.
    To do a periodical check if any external transfers done have the status as OPEN and to do a transfer by calling the deposit method of the respective Bank Class, we create
    class ExternalTransferService extends Thread{ } and override the run().
    In the run(), you should iterate infinitely and
    * Get the List<EXTERNALTRANSFER> TransactionLog
    * Iterate thro the transactionLog
    * If a transaction is found with status OPEN
    * then use ExternalBankServiceFactory to create object of the Bank Class say
    * CITIBankService class and call the deposit method which will update
    * the table and set the status to close
    * Also update the tables of our Bank (Account, Transaction)
28. Create a Class ResultGenerator – a utility class and have the following static methods
    1. void PrintAllLogTransactions()
    2. void PrintAllLogTransactions(AccountId)
    3. void PrintAllLogTransactions(TransactionType)
    4. int getTotalNoOfAccounts() - To get the total number of accounts in the bank and display the same
    5. void displayNoOfAccTypeWise() -To get the total number of accounts for each type in the bank and display the same
       To display the report like

Account Type	No Of Accounts
Savings	15
Current	12

    6. void DispTotalWorthOfBank() - Total amount available in the bank
       To display – Total balance available : Rs 12,23,23,344.00
    7. void dispPolicyInfo() - To get the minimum balance and rate of interest for each type and privilege of accounts available in the bank and display the same
    a. To display the report like

Policy Type	Minimum Balance	RateOfInterest
SAVINGS-REGULAR	5000.00	5.0

       .

.
.

8. To get the list of all transfers done, with their account information and display the same
    a. To display the report like

From To Date Amount

9. To get the list of all withdrawals done, with their account information and display the same
    a. To display the report like

From To Date Amount

10. To get the list of all deposits done, with their account information and display the same
    a. To display the report like

From To Date Amount

11. To get all the transactions done for the day from the log
    a. To display the report like

From To Date Amount

29. One more enhancement that can be done is to create an AccountDAO class and log all the data when the Account is created and all the Transactions done to the database.
30. No project can be signed off as complete and any project is open to more enhancements. So till we await new ideas we put a pause on this Banking Application Approach Document, only to be continued later….