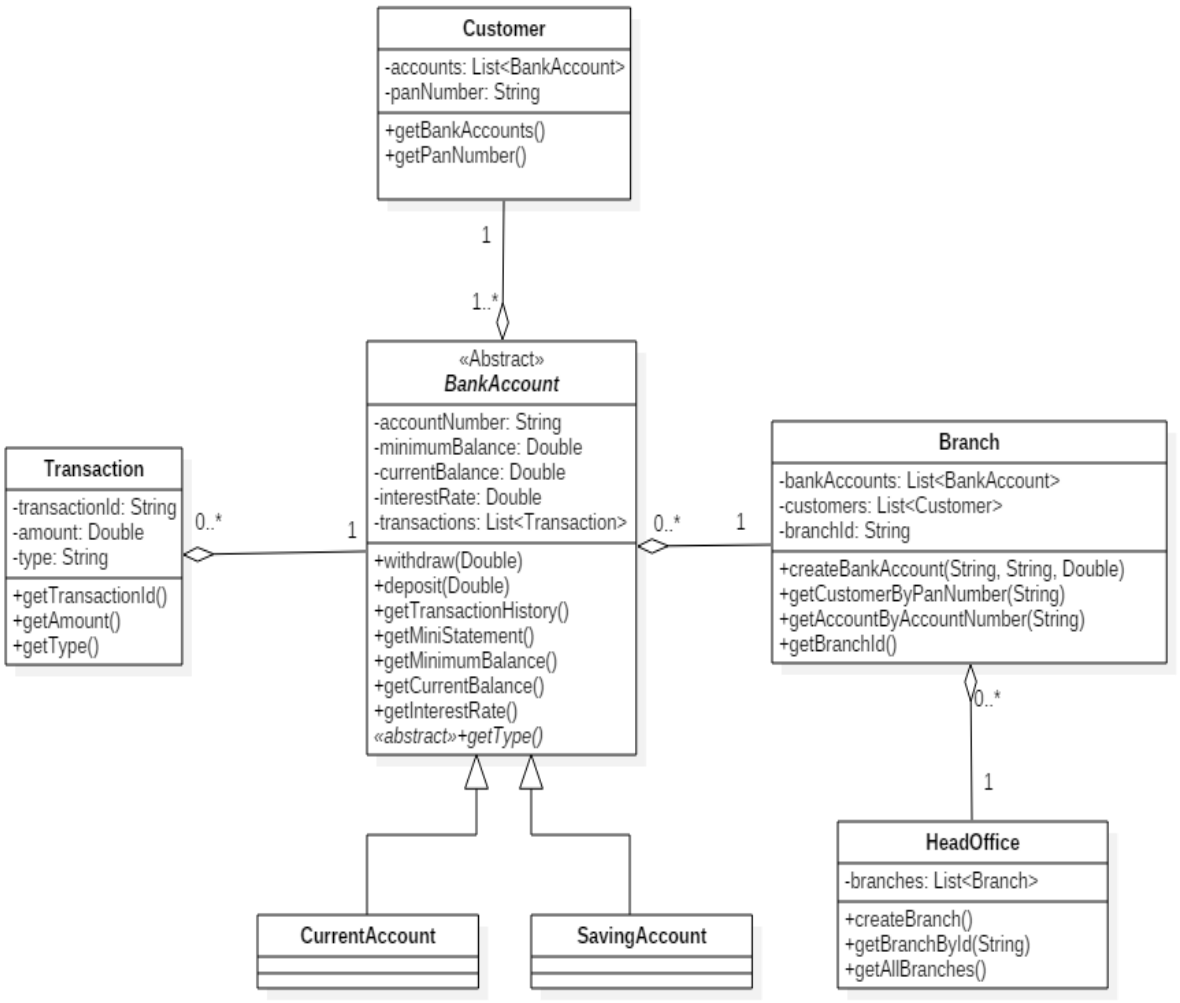
Bank Application

You are required to create a bank application by considering the following specifications:



Transaction (Transaction.java): It represents all the transactions that are made against a bank account.

It contains the following attributes:

1. **transactionId**: It is a unique ID to identify the transactions that are made against a bank account. These IDs are assigned by the bank account.
2. **amount**: It represents the transaction amount.
3. **type**: It contains the value withdraw or deposit depending on the type of transaction.

BankAccount (BankAccount.java): It is an abstract class that defines a bank account.

It contains the following attributes:

1. **accountNumber**: It is a unique number that is used to identify the bank account in a branch. It is generated by the branch in which the account has been opened.
2. **minimumBalance**: It is the minimum amount that must always be available in the bank account. An exception must be raised if this constraint is violated.
3. **currentBalance**: It is the current amount that is in the bank account.

4. **interestRate**: It is the percentage of interest.
5. **transactions**: It is a list that contains all transactions that are associated with the bank account. The latest transaction is always added at the end of the list.

It allows the following operations:

1. **withdraw(Double amount): It** withdraws the **amount** from the bank account. If the value of the **currentBalance** attribute becomes less than the value of the **minimumBalance** attribute, then the transaction is unsuccessful and an exception is raised. In case of a successful transaction, the value of the **currentBalance** attribute is updated and the transaction is added to the **transactions** list with its type set as withdraw. The amount of the transaction is set to what is provided and the **transactionId** attribute is set to one greater than the ID of the last transaction. The ID of the first transaction is 1 and of the second transaction is 2 and so on.
2. **deposit(Double amount)**: It deposits the **amount** into the bank account. If the **amount** is negative, then the transaction is unsuccessful and an exception is raised. In case of a successful transaction, the **currentBalance** attribute is updated and the transaction is added to the **transactions** list with its type set as deposit. The amount of this transaction is updated to the provided amount and the transactionId attribute is set to as one greater than the ID of the last transaction. The ID of the first transaction is 1 and of the second transaction is 2 and so on.
3. **getTransactionHistory()**: It returns the list of **transactions**.
4. **getMiniStatement()**: It returns the last 10 transactions with the latest transaction at the end of the list.

The **CurrentAccount (CurrentAccount.java)** class is a specialization of the **BankAccount** attribute with the following features:

1. **getType()**: Returns a string that is denoted as Current
2. **minimumBalance** is **20000.0**
3. **interestRate** is 0%

Similarly, the **SavingAccount (SavingAccount.java)** class is a specialization of the **BankAccount** attribute with the following features:

1. **getType()**: Returns a string that is denoted as Saving
2. **minimumBalance** is **10000.0**
3. **interestRate** is 4.5%

Customer (Customer.java): It represents the owner of one or more bank accounts. It contains the following fields:

1. **panNumber:** It is used to identify customers uniquely
2. **accounts**: It is a list of the bank accounts (**BankAccount**) that are owned by customers.
The **Branch (Branch.java)** class represents a branch of a bank. It keeps track of all the customers and bank accounts that are created in that specific bank account. It is uniquely identified by its **branchId** that is generated by the **HeadOffice.**

It contains the following attributes:

1. **bankAccounts**: List of all the bank accounts that are created in a branch

2. **customers**: List of all the customers that own a bank account in a branch

It allows the following operations:

1. **createBankAccount(String panNumber, String type, Double amount)**: It creates new **BankAccount** for a customer whose PAN number is equal to the **panNumber** attribute and the **currentBalance** in the account is set to the **amount**. The **SavingAccount** is created if the **type** is equal to Saving and a **CurrentAccount** is created if the **type** is equal to Current. If a customer with the given **panNumber** already exists in this branch, then this account is added to the customer's list of bank accounts. If a customer with given **panNumber** does not exist in the list of customers of this branch, then a new customer is created and added to the list of customers of this branch. The account number of the recently-added customer is equal to one greater than the account number of the last account created by this branch. The account number of the first account is 1 and of the second account is 2 and so on.
2. **getCustomerByPan(String panNumber)**: It returns a **Customer** whose PAN number equal to the **panNumber** attribute. If no such customer exists, then an exception is raised.
3. **getAccountByAccountNumber(String accountNumber)**: It returns a **BankAccount** whose account number is equal to **accountNumber** attribute. If no such account exists, then an exception is raised.

HeadOffice (HeadOffice.java): It represents the head office of the bank. It is responsible for creating and keeping track of all the branches.
It contains the following fields:

1. **branches**: List of all the elements of the **Branch** class. It contains all the branches that are created by this head office.
It allows the following operations:

1. **createBranch():** It creates a new **Branch** and then adds it to the list of **branches**.
The **branchId** is equal to one greater than the **branchId** of the last branch that is created by this head office. The branchID of the first branch is 1 and of the second branch is 2 and so on.
2. **getBranchById(String branchId)**: It returns the branch with the given **branchId**.
3. **getAllBranches()**: It returns the list of all the branches.