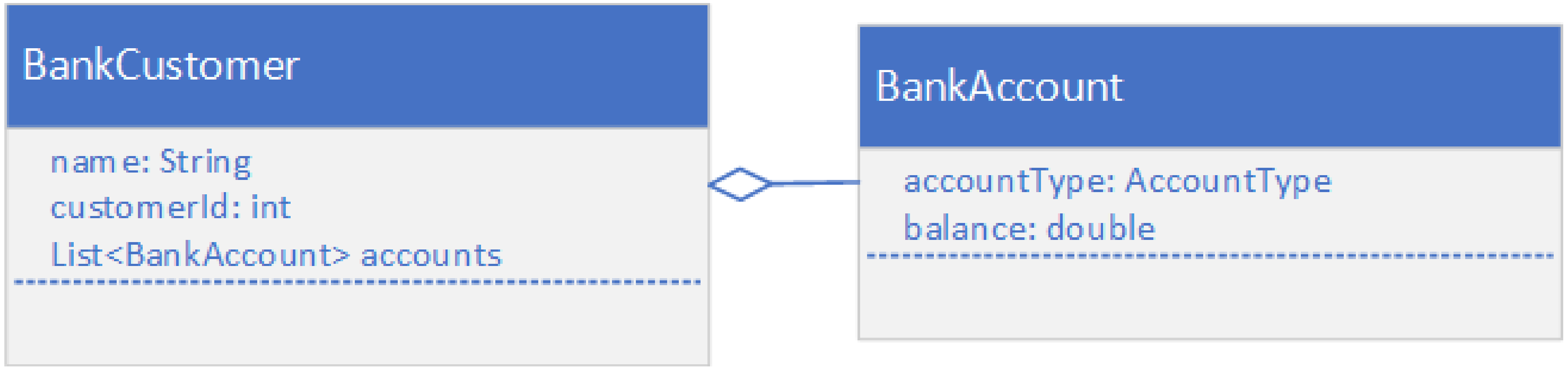
Java Advanced Course

Challenge 28. Unmodifiable

A bank

These are the classes we'll be starting with in this challenge.

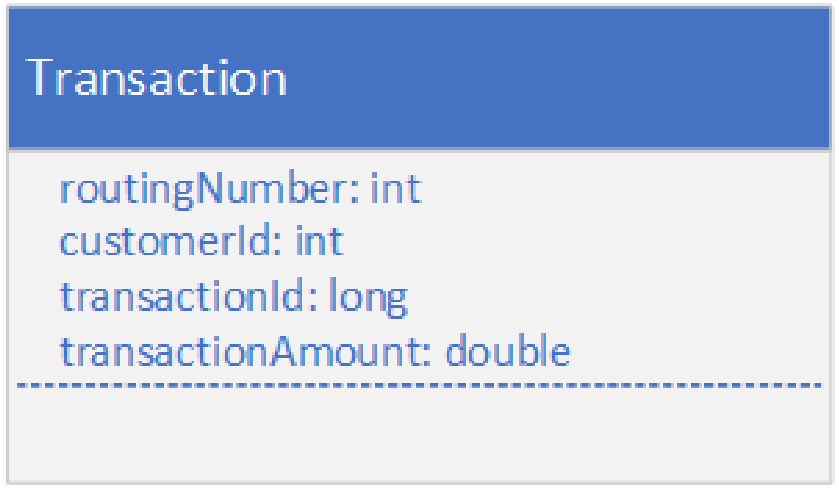


Create a Transaction class

Create a **Transaction** class in a separate package that will mirror a data table.

This class should have the fields shown here, and you need to **include getters and setters** for all fields.

Also include a constructor that takes all fields, for ease of use.



Modify the BankAccount class

For this challenge, you’ll need to modify your BankAccount class.

First, you'll want to change the balance so that it's mutable.

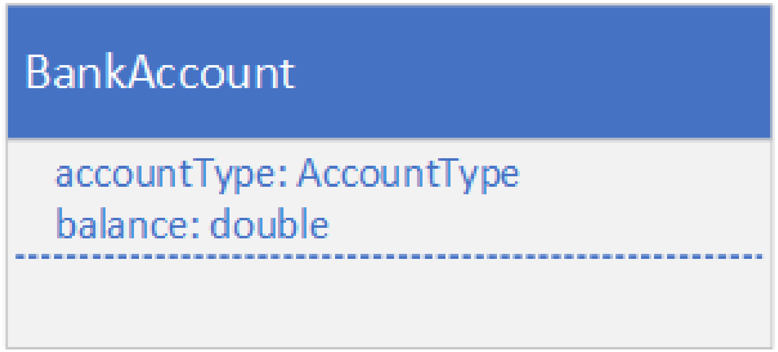
Include a Transaction Collection.

Provide a getter, or accessor method, for the transaction data.

Provide a method to adjust the balance and add the transaction data to the transaction collection.

Modify the BankAccount class

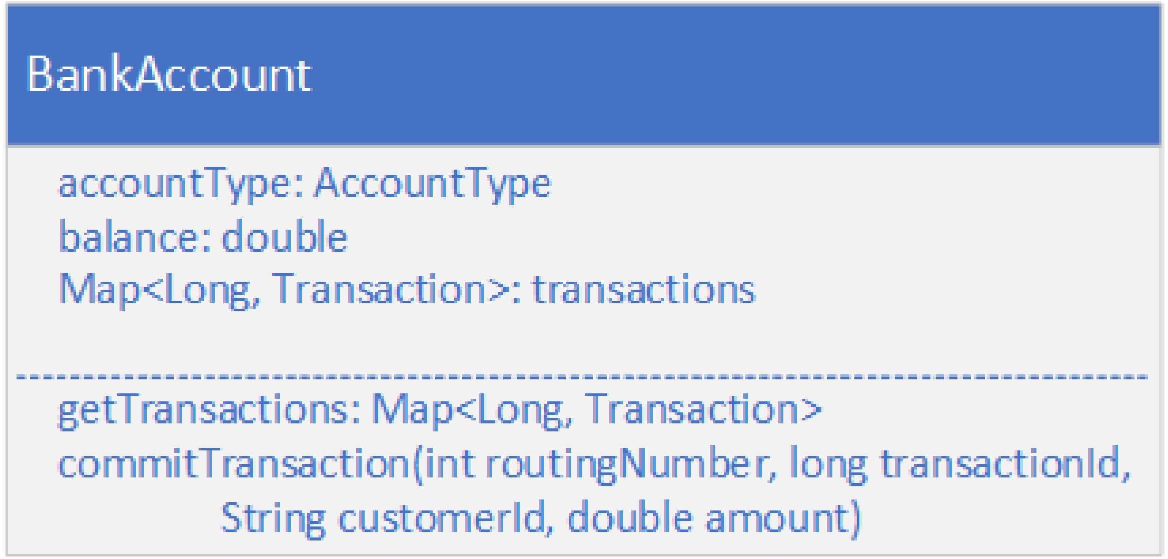
Current



BankAccount

accountType: AccountType
balance: double

After modifications (an example)



BankAccount

accountType: AccountType
balance: double
Map<Long, Transaction>: transactions

getTransactions: Map<Long, Transaction>
commitTransaction(int routingNumber, long transactionId, String customerId, double amount)

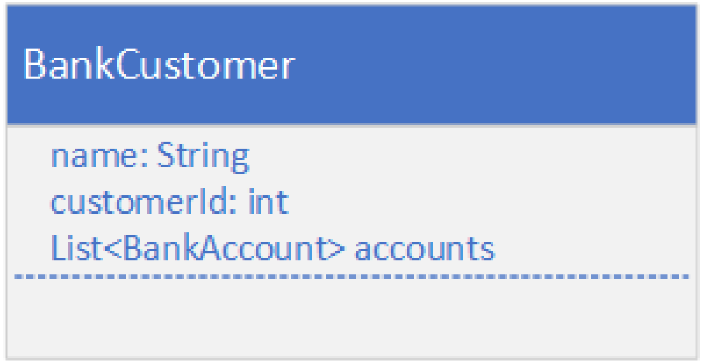
Modify the BankCustomer class

Modify your BankCustomer class.

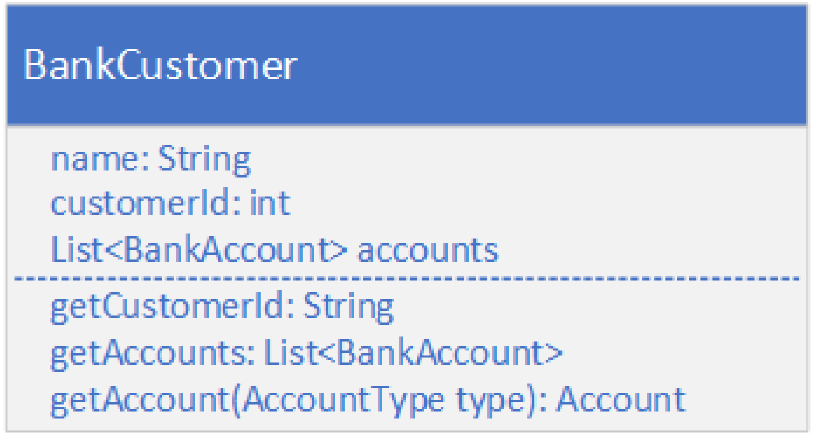
- Return the customer id as a 15-digit string, with leading zeros.
- Design this class so that code in other packages can't instantiate a new Bank Customer.
- Return a defensive copy of the accounts, from the getAccounts method.
- Include a getAccount method to return just one account, based on account type, either savings or checking.
- Assume a customer will have one checking account and one savings account.

Modify the BankCustomer class

Current



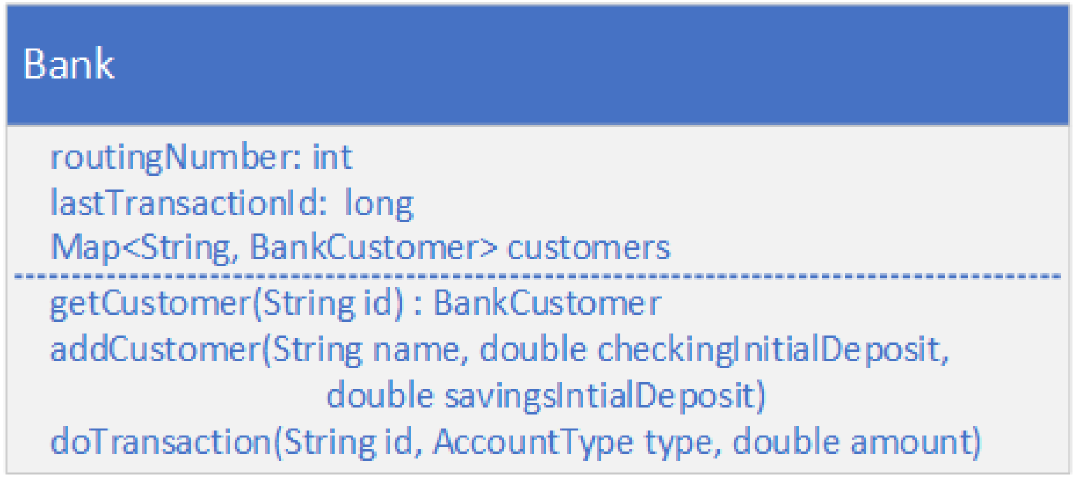
After modifications (an example)


BankCustomer
name: String
customerId: int
List<BankAccount> accounts
getCustomerId: String
getAccounts: List<BankAccount>
getAccount(AccountType type): Account

Implement a Bank

Next, you want to create a **Bank** class that has a routing number, and a collection of customers, as well as an integer that holds the next transaction id to be assigned.

- You should be able to look up a customer by a customer id, a 15-character String.
- Transaction ids should be assigned by using the lastTransactionId field on this instance of the bank.
- A negative amount is a withdrawal, and a positive amount is a deposit.
- Don't let the customer's account balance go below zero.



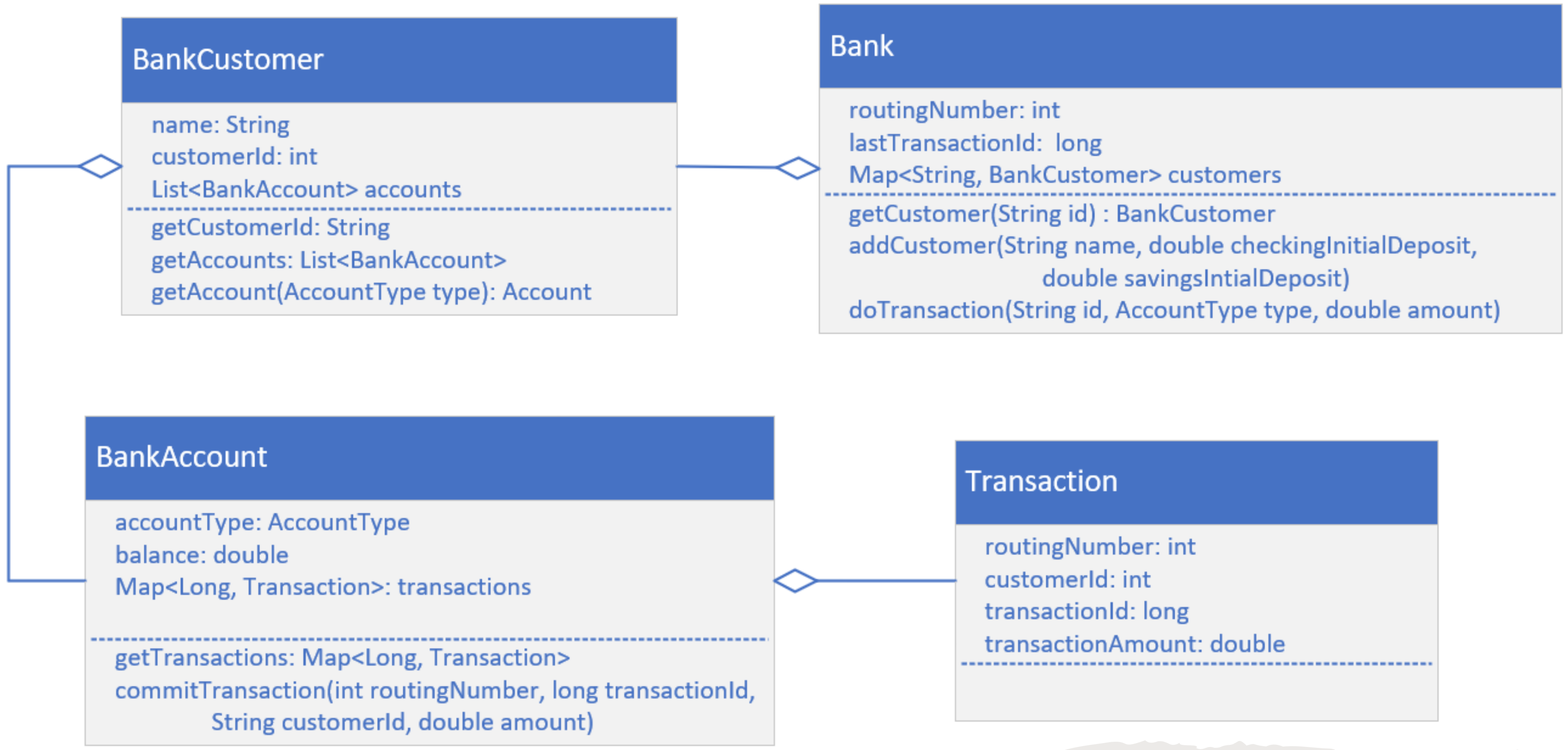
```
Bank

routingNumber: int
lastTransactionId:  long
Map<String, BankCustomer> customers
------------------------------------------------------------
getCustomer(String id) : BankCustomer
addCustomer(String name, double checkingInitialDeposit,
                         double savingsIntialDeposit)
doTransaction(String id, AccountType type, double amount)
```

Implement a Bank

In the Main class's main method:

- Create a bank instance and add a customer.
- Let a client obtain a BankCustomer instance by a customer id and review transactions from a single selected account. These transactions should not be modifiable, or susceptible to side effects.
- You should only be able to perform a withdrawal or deposit of funds through the Bank Instance, passing the customer id as a String, the type of account this transaction will be processed on, and the amount. In other words, the main method should not have access to the commit transaction code on the BankAccount itself.

The class diagram



BankCustomer
name: String
customerId: int
List<BankAccount> accounts
getCustomerId: String
getAccounts: List<BankAccount>
getAccount(AccountType type): Account
Bank
routingNumber: int
lastTransactionId: long
Map<String, BankCustomer> customers
getCustomer(String id) : BankCustomer
addCustomer(String name, double checkingInitialDeposit, double savingsIntialDeposit)
doTransaction(String id, AccountType type, double amount)
BankAccount
accountType: AccountType
balance: double
Map<Long, Transaction>: transactions
getTransactions: Map<Long, Transaction>
commitTransaction(int routingNumber, long transactionId, String customerId, double amount)
Transaction
routingNumber: int
customerId: int
transactionId: long
transactionAmount: double