Lab 3 in TND002

Mark Eric Dieckmann

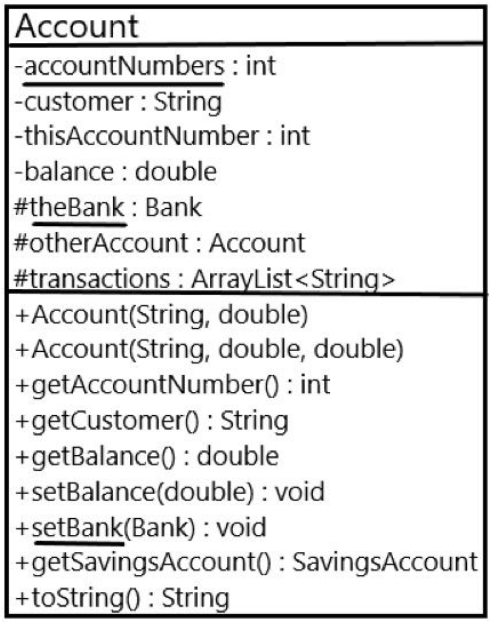
February 7, 2023

1 Summary

You implement a code framework that allows a bank manage accounts. The bank offers various types of accounts and provides methods that allow customers to transfer money between them. The code framework is developed in two labs (labs 3 and 4). In lab 3, you implement two types of accounts that share a superclass. One is a current account, with which you run the day-to-day business. The other is a savings account. Both inherit shared variables and methods from a joint superclass. All accounts are handled by a bank, which is an instance of a dedicated class. In this lab, you implement a method that transfers money from the current account to the savings account and vice versa.

2 Task: The superclass Account

This superclass contains variables and methods that are shared by all subclasses. This class is not used as a standalone class (the constructor is not called directly).



accountNumbers is the lowest free number that can be used as the account number of a new account. Initialize it to 1.

customer is the name of the person owning the account.

thisAccountNumber is the account number of this instance of **Account**.

balance is the money on this instance of **Account**.

theBank gets you to the instance of the bank, which handles this account.

otherAccount is the account connected to this one (both have the same customer). If this account is a current account, *otherAccount* is a savings account and vice versa.

transactions lists all money transfers to and from this instance of **Account**.

Account(arg1, arg2) is used when we want to create one account. The value of *arg1* initializes *customer* and *arg2* initializes *balance*. *thisAccountNumber* is initialized with *accountNumbers* and the latter is increased by 1.

Account(arg1, arg2, arg3) is called when you want to create an account together with the second account *otherAccount*. This constructor is only called when you create a current account. You do the same as in the first constructor but you also create a new savings account for the customer with the name *arg1* and you set the value of its balance to *arg3*. You connect this savings account to *otherAccount* and you connect the current account to *otherAccount* of the savings account.

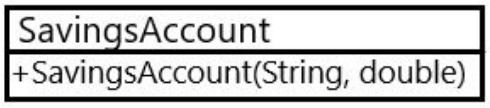
getAccountNumber(), *getCustomer()*, and *getBalance()* do what their names say. *setBalance(double)* and *setBank(Bank)* set the values of *balance* and *theBank*.

getSavingsAccount() returns *otherAccount* if that is a savings account. It returns *null* if *otherAccount* is a current account.

toString() returns information about this account. The returned string depends on the type of account. The first line should either be "Current Account: " or "Savings Account: " (we will add a third type in lab 4), followed by " with account number " followed by *thisAccountNumber* followed by the value of *balance* and a line break. It should append to this line all elements of *transactions*. Each element of *transactions* is written on its own line.

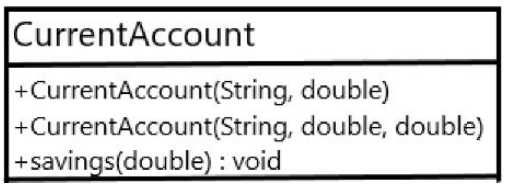
3 Task: The subclasses CurrentAccount and SavingsAccount

Both classes are subclasses of **Account**. The simple one is *SavingsAccount*. It inherits all variables and methods it needs from **Account**. It has only one constructor.



A savings account should not create an instance of *otherAccount*. It can thus only call the constructor of the superclass with two arguments.

All money transfer is done through the current account. For this purpose, it gets several variables and methods that are not part of **Account** (more in lab 4).



CurrentAccount(arg1, arg2) creates a current account. It initializes *customer* with *arg1* and *balance* with *arg2* and sets *otherAccount* to *null*.

CurrentAccount(arg1, arg2, arg3) creates a current account and a savings account through a call to the appropriate superclass constructor. The savings account is created in the constructor of **Account**. Both accounts get the value *arg1* for the customer. *arg2* sets *balance* of the current account and *arg3* sets *balance* of the savings account.

savings(arg) transfers money between the current account and the savings account. First, it checks if a savings account is attached to the current account. If there is no attached savings account, it should do nothing. If the current account has a savings account attached, it should distinguish between the cases $arg > 0$ and $arg \leq 0$. It should also make sure that the balances of both accounts do not become negative.

If $arg > 0$, the money goes from the current account to the savings account. It should check if there is enough money in the current account for the transfer. If there is, it should reduce *balance* of the current account by *arg* and increase that in the savings account by *arg*. If there is not enough on the current account, it should transfer the value of *balance*. After that transfer, all the money went from the current account to the savings account.

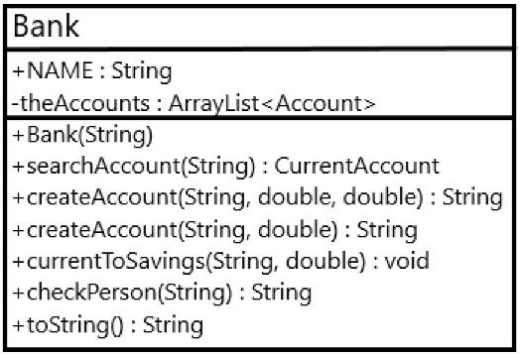
Add a string to *transactions* of the current account that starts with "To savings account: " followed by the transferred money. Add a string to *transactions* of the savings account starting with "From current account: " followed by the received money.

If $arg \leq 0$ the check is reversed. You transfer *arg* from the savings account to the current account if there is enough money in the savings account. Otherwise, you transfer all the money from the savings account to the current account.

Add a string to *transactions* of the current account that starts with "From savings account: " followed by the transferred money. Add a string to *transactions* of the savings account starting with "To current account: " followed by the received money.

4 Task: The class Bank

The bank manages the accounts and transfers money between accounts.



NAME is a constant that holds the name of this instance of the bank.

theAccounts is a dynamic array that holds all current- and savings accounts. You can initialize it when you declare it.

Bank(arg) initializes *NAME* with *arg*. It also copies its address to *theBank* in **Account**. We assume for now that there will only be one bank.

searchAccount(arg) searches *theAccounts* for the current account of the customer with name *arg*. It returns it if it finds it. Otherwise, it returns *null*.

createAccount(arg1, arg2, arg3) creates a current account with the balance *arg2* and a savings account with the balance *arg3* for the customer with the name *arg1*. This is provided that the customer *arg1* does not yet have a current account. If *arg1* already has an account (check with *searchAccount(arg)*), no new accounts should be created. Both accounts should be added to *theAccounts*. The method should return "Current

and savings accounts created for " followed by *arg1*. If no account was created, it should return "Account(s) already exist for " followed by *arg1*.

createAccount(arg1, arg2) does the same as the method above but it does not create a savings account. If an account was created, it should return the string "Current account created for " followed by *arg1*. If no account was created, it should return "Account(s) already exist for " followed by *arg1*.

currentToSavings(arg1, arg2) should find the current account of *arg1* and call its *savings(arg)* method with *arg2*.

checkPerson(arg) should return the account information for the customer with the name *arg*. Its first line should state the name of the customer followed by a line break. It should then search for its current account. If it finds one, it should append the return value of the current account's *toString()* method. If the customer also has a savings account, then you also append the return value of the savings account's *toString()* method. If there is no customer with the name *arg*, the method should return the string "Person does not exist".

toString() returns information from the bank. It starts with the string "Bank: " followed by the value of *NAME* followed by a line break. On the next line, it gives "Accounts: " followed by the number of accounts handled by the bank followed by a line break. It then adds up the money stored in the current accounts in one variable and the money in the savings accounts in a second variable. It then appends to the string "Money in current / savings accounts: " followed by the total money in current accounts and in savings accounts.

Running **Lab3** should give you the console output below and on the next page. Try to get a similar output format to make it easier for lab assistants to check it.

```
Testing createAccount
Current and savings accounts created for Peter
Account(s) already exist for Peter
Current and savings accounts created for Sofia
Current and savings accounts created for Olga
Current account created for Alex
Bank: Great Northern Bank
Accounts: 7
Money in current / savings accounts: 6000.0 / 21000.0

Testing the search function
Peter

Current Account with account number 1: 1000.0

Savings Account with account number 2: 2000.0

Sofia

Current Account with account number 3: 3000.0

Savings Account with account number 4: 18000.0

Olga

Current Account with account number 5: 1000.0

Savings Account with account number 6: 1000.0

Alex

Current Account with account number 7: 1000.0
```

```
Testing the money transfer from current to savings account
Peter

Current Account with account number 1: 2000.0
From savings account: 1000.0

Savings Account with account number 2: 1000.0
To current account: 1000.0

Sofia

Current Account with account number 3: 500.0
To savings account: 2500.0

Savings Account with account number 4: 20500.0
From current account: 2500.0

Olga

Current Account with account number 5: 0.0
To savings account: 1000.0

Savings Account with account number 6: 2000.0
From current account: 1000.0

Alex

Current Account with account number 7: 1000.0
```

```
Peter

Current Account with account number 1: 2000.0
From savings account: 1000.0

Savings Account with account number 2: 1000.0
To current account: 1000.0

Peter

Current Account with account number 1: 0.0
From savings account: 1000.0
To savings account: 2000.0

Savings Account with account number 2: 3000.0
To current account: 1000.0
From current account: 2000.0
```