Bruce Dong

Oct.2.2020

CSDS 132

Professor. Connamacher

Project 1 Test Report

In class BankAccount, there are three different kinds of method, so I choose to test them differently.

Variables in this class that needs to be set manually.

Expected result: successfully set up the variable

Result: success

* interest rate: 0.20
* minimum balance: 1000
* overdraft fee: 10
* ATM fee: 20
* bounced check fee: 30
* withdraw fee: 40
* withdraw limit: 2

test code of setting up the basic constructor:

> BankAccount test = new BankAccount (0.20, 1000, 10, 20, 30)

> test

BankAccount@1deb870

1. Getter and setter method:

This includes all the getter and setter method: (getter method only change set into get)

Expected result: the input of the setter method should equal to the output of the getter method.

Result: Success

* setInterestRate (double)
* setMinimumBalance (int)
* setATMFee (double)
* setOverdraftFee (double)
* setBouncedCheckFee (double)
* setWithdrawFee (double)
* setWithdrawLimit (int)

The actually test code of all the getter and setter methods:

> BankAccount test = new BankAccount (0.20, 1000, 10, 20, 30)

> test.setInterestRate(0.20)

> test.getInterestRate()

0.2

> test.setMinimumBalance (1000)

> test.getMinimumBalance ()

1000.0

> test.setATMFee(20)

> test.getATMFee()

20.0

> test.setOverdraftFee (10)

> test.getOverdraftFee ()

10.0

> test.setBouncedCheckFee(30)

> test.getBouncedCheckFee()

30.0

> test.setWithdrawFee(40)

> test.getWithdrawFee()

40.0

> test.setWithdrawLimit(2)

> test.getWithdrawLimit()

2

1. Method of deposit and withdraw

This includes the following methods, and because each method has it’s own function, I will include separate explanation on each method:

* Deposit and getBalance (both double inputs/outputs)

Expected result: The sum of all deposit value should equal to the return value of getBalance

Result: Success:

Test code:

> test.deposit(2000.5)

> test.deposit(2999.5)

> test.getBalance()

5000.0

* withdraw (double)

In this method, there are three possible situations (balance value is continuing in withdraw test process, and the withdraw limit is 2, the withdraw time is 0, withdraw fee: 40):

1. Withdraw amount is smaller than the balance and not over the withdraw limit

Expected result: balance reduced by the withdraw amount

Result: Success

Test code:

> test.deposit(5000)

> test.withdraw(1000)

true

> test.getBalance()

4000.0

> test.withdraw(1000)

true

> test.getBalance()

3000.0

> test.withdraw(5000)

false

1. Withdraw amount is smaller than the balance and over the withdraw limit

Expected result: balance reduced by the withdraw amount and the withdraw fee

Result: Success

Test code:

> test.withdraw(1000)

true

> test.getBalance()

1960.0

1. Withdraw amount is larger than the balance

Expected result: nothing happens

Result: Success

Test code:

> test.withdraw(5000)

false

> test.getBalance()

1960.0

* withdrawDraft (double)

In this method, there are three possible situations (balance value is continuing in the withdrawDraft test process, and the withdraw limit is 2, the withdraw time is 0, withdraw fee: 40, bounced check fee: 30):

1. Withdraw amount is smaller than the balance and not over the withdraw limit

Expected result: balance reduced by the withdraw amount

Result: Success

Test code:

> test.getBalance()

5000.0

> test.withdrawDraft(1000)

true

> test.getBalance()

4000.0

> test.withdrawDraft(1000)

true

> test.getBalance()

3000.0

1. Withdraw amount is smaller than the balance and over the withdraw limit

Expected result: balance reduced by the withdraw amount and the withdraw fee

Result: Success

Test code:

> test.withdrawDraft(1000)

true

> test.getBalance()

1960.0

1. Withdraw amount is larger than the balance

Expected result: balance is reduced by the bounced check fee

Result: Success

Test code:

> test.withdrawDraft(5000)

false

> test.getBalance()

1930.0

* withdrawATM (double)

In this method, there are three possible situations (balance value is continuing in the withdrawATM test process, and the withdraw limit is 2, the withdraw time is 0, withdraw fee: 40, ATM fee: 20):

1. Withdraw amount is smaller than the balance and not over the withdraw limit

Expected result: balance reduced by the withdraw amount and the ATM fee

Result: Success

Test code:

> test.getBalance()

5000.0

> test.withdrawATM(1000)

true

> test.getBalance()

3980.0

> test.withdrawATM(1000)

true

> test.getBalance()

2960.0

1. Withdraw amount is smaller than the balance and over the withdraw limit

Expected result: balance reduced by the withdraw amount and the withdraw fee and the ATM fee

Result: Success

Test code:

> test.withdrawATM(1000)

true

> test.getBalance()

1900.0

1. Withdraw amount is larger than the balance

Expected result: nothing happens

Result: Success

Test code:

> test.withdrawATM(5000)

false

> test.getBalance()

1900.0

* For all three withdraw method, there is a possible situation that the withdraw limits sets to 0, which means that the withdraw limits is infinite and no withdraw fee is applied anyway and here is the test code(since the methods’ judging part(withdraw, withdrawDraft and withdrawATM) of the withdraw limit are identical, I choose to use withdrawATM as an example.)

Expected result: each time or withdrawing, only the ATM fee and withdraw amount is reduced

Result: Success

Test code:

> BankAccount test = new BankAccount (0.20, 1000, 10, 20, 30)

> test.deposit(5000)

> test.setWithdrawLimit(0)

> test.setWithdrawFee(40)

> test.withdrawATM(1000)

true

> test.getBalance()

3980.0

> test.withdrawATM(1000)

true

> test.getBalance()

2960.0

> test.withdrawATM(1000)

true

> test.getBalance()

1940.0

> test.withdrawATM(1000)

true

> test.getBalance()

920.0

3. Increment Methods

Note: for test purpose, I include the getter and setter method of boolean overdraftFlag and double interestEarned, and in the submitted code, they are deleted. The start interest earned so far is 1000 for test purpose

// test purpose method

// This method allows user to set the account's overdraft flag

public void setOverdraftFlag( boolean overdraftFlag){

this.overdraftFlag = overdraftFlag;

}

// This method allows user to check the account's overdraft flag

public boolean setOverdraftFlag(){

return overdraftFlag;

}

// This method allows user to set the account's interest earned

public void setInterestEarned( double interestEarned){

this.interestEarned = interestEarned;

}

// This method allows user to check the account's interest earned

public double getInterestEarned(){

return interestEarned;

}

incrementDay (void)

In this method, there should be three situations

1. Balance is smaller than the minimum balance and overdraft flag it false

Expected result: balance is deducted by the overdraft fee

Result: Success

Test code:

> BankAccount test = new BankAccount (0.20, 1000, 10, 20, 30)

> test.deposit(500)

> test.setInterestEarned(1000)

> test.incrementDay()

> test.getBalance()

490.0

> test.getOverdraftFlag()

true

> test.getInterestEarned()

1000.0

1. Balance is smaller than the minimum balance and overdraft flag it true

Expected result: Nothing happens

Result: Success

Test code:

> test.incrementDay()

> test.getBalance()

490.0

> test.getOverdraftFlag()

true

> test.getInterestEarned()

1000.0

1. Balance is greater than the minimum balance

Expected result: interest is accumulated

Result: Success

Test code:

> BankAccount test = new BankAccount (0.20, 1000, 10, 20, 30)

> test.deposit(5000)

> test.setInterestEarned(1000)

> test.incrementDay()

> test.getInterestEarned()

1003.2876712328767

> test.getBalance()

5000.0

> test.incrementDay()

> test.getInterestEarned()

1006.5771439294427

> test.getBalance()

5000.0

* incrementMonth (void)

Note: This test code is continuing the test of branch 3 of the test of incrementDay, so currently the interest earned is 1006.5771439294427 and for test purpose, the overdraft flag is true

Expected Result:

* + - * 1. balance is the current balance plus the interested earned so far
        2. interested earned rest to 0
        3. overdraft flag reset to true

Test result: Success

Test Code:

> test.setOverdraftFlag(true)

> test.incrementMonth()

> test.getInterestEarned()

0.0

> test.getBalance()

6006.577143929443

> test.getOverdraftFlag()

false

Until now, each method of class BankAccount is tested and all of them runs correctly

This is the test of the class CreditCardAccount:

Variables in this class that needs to be set manually.

Expected result: successfully set up the variable

Result: success

* credit limit: 5000
* interest rate: 0.20
* minimum payment for each month: 3000
* late payment penalty: 100

test code of setting up the basic constructor:

> CreditCardAccount test = new CreditCardAccount(5000, 0.20, 3000, 100)

> test

CreditCardAccount@c1d61

There are three types of method in this class and they are:

1. getter and setter methods

Here are the methods of this branch, and their expectation are identical. ( each method has both get and set mode)

Expectation: each input of the setter method should be identical to the output of the getter method.

Test Result: Success

* setCreditLimit

Test Code:

> test.setCreditLimit(5000)

> test.getCreditLimit()

5000

* setInterestRate

Test Code:

> test.setInterestRate(0.2)

> test.getInterestRate()

0.2

* setMinMonthlyPayment

Test Code:

> test.setMinMonthlyPayment(3000)

> test.getMinMonthlyPayment()

3000

* setLatePaymentPenalty

Test Code:

> test.setLatePaymentPenalty(100)

> test.getLatePaymentPenalty()

100

1. charge and payment method

There are four method in this branch, and they are: charge, payment, getMonthlyPayment and gatBalance, and their relationship are:

The sum of all charge equals to the monthly payment, and balance equals to monthly payment minus all the payment this month. When the amount that user wants to charge plus the amount of balance is more than the credit limit, the charge request will be denied. Thus, the test is working according to this logic.

Test result: Success

Test Code:

> CreditCardAccount test = new CreditCardAccount(5000, 0.20, 3000, 500)

> test.charge(1000)

true

> test.charge(1000)

true

> test.charge(1000)

true

> test.charge(1000)

true

> test.charge(1000)

true

> test.charge(1000)

false

> test.getMonthlyPayment()

5000.0

> test.payment(1000)

> test.getBalance()

4000.0

> test.charge(1000)

true

> test.getBalance()

5000.0

1. increment method

Note: for test purpose, I include the getter and setter method of boolean overdraftFlag and double interestEarned, and in the submitted code, they are deleted. The start interest earned so far is 1000 for test purpose

// test purpose method

// This method allows user to set the account's plug in full flag

public void setPaidInFullFlag ( boolean paidInFullFlag){

this. paidInFullFlag= paidInFullFlag;

}

// This method allows user to check the account's paid in full flag

public boolean getPaidInFullFlag (){

return paidInFullFlag;

}

// This method allows user to set the account's total interest charged

public void setInterestCharged ( double interestCharged){

this. interestCharged = interestCharged;

}

// This method allows user to check the account's total interest charged

public double getInterestCharged (){

return interestCharged;

}

// This method allows user to check the account's total payment

public double getTotalPayed (){

return totalPayed;

}

There are two methods in this branch, and they are:

* incrementDay

In this method, there are two possible situations:

1. paid-in-full flag is true

Expectation: Nothing happens

Test result: Success

Test Code (The code is continuing for this test of the method incrementDay):

> CreditCardAccount test = new CreditCardAccount(5000, 0.20, 3000, 500)

> test.setPaidInFullFlag(true)

> test.getPaidInFullFlag()

true

> test.setInterestCharged(1000)

> test.charge(1000)

true

> test.getBalance()

1000.0

> test.incrementDay()

> test.getInterestCharged()

1000.0

1. paid-in-full flag is false

Expectation: interested charged is accumulated with balance together

Test result: Success

Test Code:

> test.setPaidInFullFlag(false)

> test.incrementDay()

> test.getInterestCharged()

1001.0958904109589

> test.incrementDay()

> test.getInterestCharged()

1002.1923813098142

> test.getPaidInFullFlag()

false

* incrementMonth

There are two if statement in this method, and they are irrelevant.

1. The first one is to determine whether the user has payed all his debt to the bank.

Expectation: if totally payment is greater than or equal to the monthly payment, paidInFullFlag should be true, in the other way, it is false.

Test result: Success

Test Code:

> CreditCardAccount test = new CreditCardAccount(5000, 0.20, 3000, 500)

> test.charge(1000)

true

> test.payment(1000)

> test.incrementMonth()

> test.getPaidInFullFlag()

true

> test.payment(1001)

> test.incrementMonth()

> test.getPaidInFullFlag()

true

> test.charge(1000)

true

> test.charge(1000)

true

> test.incrementMonth()

> test.getPaidInFullFlag()

false

1. The second if statement is to determine whether the user paid his monthly min payment when his monthly payment exceeds the min monthly payment. If he did reach the limit, there will be a late payed penalty add to his account, and this test is operated according to this logic.

Test result: Success

Test code:

// When the person first spend less than the min monthly payment and payment is 0, and then spend over min monthly payment and total payment is less than the min monthly payment

> CreditCardAccount test = new CreditCardAccount(5000, 0.20, 3000, 500)

> test.setInterestCharged(0)

> test.charge(1000)

true

> test.incrementMonth()

> test.getBalance()

1000.0

> test.charge(3000)

true

> test.incrementMonth()

> test.getBalance()

4500.0

// When the person paid over the line of min monthly payment

> CreditCardAccount test = new CreditCardAccount(5000, 0.20, 3000, 500)

> test.setInterestCharged(0)

> test.charge(1000)

true

> test.charge(3000)

true

> test.payment(3000)

> test.incrementMonth()

> test.getBalance()

1000.0

1. The rest of the method will do the regular account adjustments

Expectation: interested charged so far will accumulate to the balance; interested charged and total payment will reset to zero, and the balance will equal to the monthly payment

Test result: Success

Test code:

> CreditCardAccount test = new CreditCardAccount(5000, 0.20, 3000, 500)

> test.payment(1000)

> test.setInterestCharged(1000)

> test.getBalance()

-1000.0

> test.incrementMonth()

> test.getInterestCharged()

0.0

> test.getBalance()

0.0

> test.getTotalPayed ()

0.0

At this point, all methods in class CreditCardAccount has been tested and each of them runs successfully.

This is the test of class Date since some part of the class Account use the class Date.

Variables that need to set manual:

* day (int)
* month (int)

Expectation: successfully set up the constructor of this class

Test result: Success

Test code:

> Date test = new Date(11,1)

> test

Date: 11 Month: 1

There are three branches of method in this class:

* Getter methods

Expectation: the getter method of both day and month should equal to the input of the constructor

Test result: Success

Test Code:

> test.getDay()

"11"

> test.getMonth()

"1"

* incrementDay method

Note: for test purpose, I include the setter method of day and month, and in the submitted code, they are deleted.

// test purpose method

// This method allows user to set the day

public void setDay ( int day){

this. day = day;

}

// This method allows user to set the month

public void setMonth ( int month){

this. month = month;

}

Expectation: In the increment method, it will increment a day and work correctly when a month is also needed to increment

Test result: Success

Test code:

> Date test = new Date (31,12)

> test.incrementDay()

> test

Day: 1 Month: 1

> Date test = new Date(11,1)

> test.incrementDay()

> test.getMonth()

"1"

> test.getDay()

"12"

> test.setDay(31)

> test.setMonth(1)

> test.incrementDay()

> test.getDay()

"1"

> test.getMonth()

"2"

> test.setDay(27)

> test.setMonth(2)

> test.incrementDay()

> test.getMonth()

"3"

> test.getDay()

"1"

> test.setMonth(4)

> test.setDay(30)

> test.incrementDay()

> test.getDay()

"1"

> test.getMonth()

"5"

* override methods

In this branch, there are two methods

1. equals

Expectation: This method will judge whether the two dates equal or not

Test result: Success

Test code:

> Date test1 = new Date(11,1)

> Date test2 = new Date(11,1)

> Date test3 = new Date(11,2)

> test1.equals(test2)

true

> test1.equals(test3)

false

> test3.equals(test1)

false

> test3.equals(test2)

false

> test3.equals(test3)

true

1. toString

Expectation: This method will turn the date variable into a common people readable string form.

Test result: Success

Test code:

> Date test = new Date(11,1)

> test.toString()

"Day: 11 Month: 1"

> test.getDay()

"11"

> test.getMonth()

"1"

Until now, every method in class Date have been tested and each one runs successfully.

This is the test report of class Account

In this class, there are 9 possible variable and the constructor needs five.

Here is the code of successfully building the constructor:

Succeeded test code:

> Date date = new Date(11,1)

> Account main = new Account ("Bruce", "Dong", "739 Academy Lane", "23603", date)

> main

Bruce Dong 739 Academy Lane 23603

There are three branches of the class Account and they are

1. getter and setter methods

There are two different kinds of method in this branch.

1. Methods that get and set String and date

Expectation: the input of setter method should equal the output of getter method.

Test result: Success

Test code:

> main. setLastName("a")

> main. getLastName()

"a"

> main. setFirstName("b")

> main. getFirstName()

"b"

> main. setStreetAddress("aa")

> main. getStreetAddress()

"aa"

> main. setZipCode("11")

> main. getZipCode()

"11"

> Date date2 = new Date(11,29)

> main.getDate()

Day: 11 Month: 1

> main.setDate(date2)

> main.getDate()

Day: 11 Month: error

1. Methods that get and set CreditCardAccount and BankAccount class

Expectation: the input of setter method should equal the output of getter method.

Test result: Success

Test code:

> CreditCardAccount credit = new CreditCardAccount(5000, 0.20, 3000, 500)

> BankAccount bank1 = new BankAccount (0.20, 1000, 10, 20, 30)

> BankAccount bank2 = new BankAccount (0.20, 10000, 100, 20, 30)

> BankAccount bank3 = new BankAccount (0.20, 100, 10, 20, 30)

> Account main = new Account ("Bruce", "Dong", "739 Academy Lane", "23603", date1)

> main.setSavingsAccount(bank1)

> main.getSavingsAccount()

BankAccount@1ea9e2

> bank1

BankAccount@1ea9e2

> main.setCheckingAccount(bank2)

> main.getCheckingAccount()

BankAccount@2d080c

> bank2

BankAccount@2d080c

> main.setMoneyMarketAccount(bank3)

> main.getMoneyMarketAccount()

BankAccount@17c1335

> bank3

BankAccount@17c1335

> main.setCreditCardAccount(credit)

> main.getCreditCardAccount()

CreditCardAccount@165903c

> credit

CreditCardAccount@165903c

1. increment method

This method should make change to every bank account and also change the date. Expectation: When the month changes, every separate account should also execute the increnmentMonth method.

Note: since the method that operates the BankAccount class are identical except for the variable name, it will be meaningless to test the same thing again and again. Thus, I only test the saving account and credit card account variable.

Test result: Success

Test code:

// Expected result in of CreditCardAccount runs in its won increnmentday and month result

> CreditCardAccount credit = new CreditCardAccount(5000, 0.20, 3000, 500)

> credit.charge(1000)

true

> credit.setInterestCharged(1000)

> credit.incrementDay()

> credit.getInterestCharged()

1001.0958904109589

> credit.incrementMonth()

> credit.getInterestCharged()

0.0

> credit.getBalance()

2001.0958904109589

// Expected result in of BankAccount runs in its won increnmentday and month result

> BankAccount bank1 = new BankAccount (0.20, 1000, 10, 20, 30)

> bank1.deposit(5000)

> bank1.setInterestEarned(1000)

> bank1.incrementDay()

> bank1.getBalance()

5000.0

> bank1.getInterestEarned()

1003.2876712328767

> bank1.incrementMonth()

> bank1.getBalance()

6003.287671232877

> bank1.getInterestEarned()

0.0

// This is the test of how Account variable as a whole that interact with incrementDay()

> Date date = new Date(31,1)

> Account main = new Account ("Bruce", "Dong", "739 Academy Lane", "23603", date)

> CreditCardAccount credit = new CreditCardAccount(5000, 0.20, 3000, 500)

> credit.charge(1000)

true

> credit.setInterestCharged(1000)

> BankAccount bank1 = new BankAccount (0.20, 1000, 10, 20, 30)

> bank1.deposit(5000)

> bank1.setInterestEarned(1000)

> main.setSavingsAccount(bank1)

> main.setCreditCardAccount(credit)

> main.incrementDay()

> date

Day: 1 Month: 2

> bank1.getBalance()

6003.287671232877

> credit.getBalance()

2001.0958904109589

1. override methods

In this branch, there are two different methods.

1. Equals method

Expectation: This method will determine whether two variables of Account are same or not according to whole name, address and zipcode.

Test result: success

Test code:

> Date date = new Date(11,1)

> Account main = new Account ("a", "b", "c", "d", date)

> Account main2 = new Account ("a", "b", "c", "d", date)

> main.equals(main2)

true

> main2.equals(main)

true

> Account main2 = new Account ("a", "b", "c", "e", date)

> main.equals(main2)

false

> Account main2 = new Account ("a", "b", "ce", "d", date)

> main.equals(main2)

false

> Account main2 = new Account ("a", "be", "c", "d", date)

> main.equals(main2)

false

> Account main2 = new Account ("ae", "b", "c", "d", date)

> main.equals(main2)

false

> main2.equals(main)

false

1. toString method

This method should turn the variable which belongs to Account class to a standard format: “name, address, zip code, saving account, checking account, and other account if they exist.”

Test result: Success

Test code (This code is continuing from the test of incrementDay test) :

> BankAccount bank2 = new BankAccount (0.20, 1000, 10, 20, 30)

> BankAccount bank3 = new BankAccount (0.20, 1000, 10, 20, 30)

> bank2.deposit(300)

> bank3.deposit(400)

> main.setCheckingAccount(bank2)

> main.setMoneyMarketAccount(bank3)

> main.toString()

"Bruce Dong 739 Academy Lane 23603. Savings: 6003.0 Checkings: 300.0 Money Markets: 400.0 Credit Card: 2001.0"

At this point, four classes have all been tested and each method can run correctly with no error.