Alexander Laudino

CSC-162-IN1

Dr. Farrett

Lab Assignment 8 – BankAccount & SavingsAccount Reference Documents

---

Pseudo-code

This program demonstrates the BankAccount and SavingsAccount classes.

DemoAccounts

Begin

```
1.  Open new checking account with $2500 at 1.567% interest
2.  View checking account information
3.  Make a $500 withdrawal from checking account
4.  Simulate 1 monthly process
5.  Make a $2000 deposit into checking account
6.  Simulate 12 months of monthly processes
7.  View checking account information
8.  Make a $5000 withdrawal from checking account
9.  Open new savings account with $20 at 3.56% interest
10. Make a $2 deposit into savings account
11. Make a $5 withdrawal from savings account
12. View savings account information
13. Simulate 1 monthly process
14. View savings account information
15. Make a $10 deposit into savings account
16. View savings account information
17. Simulate 1 monthly process
18. View savings account information
19. Open new savings account with $250 at 4.56% interest
20. View savings account 1 information
21. Make 10 withdrawals from savings account 1
22. Simulate 1 monthly process
23. View savings account information
```

Begin balanceAfter(BankAccount : ba, int : months)

```
1. For int i = 0, i < months, i++
2.   ba.monthlyProcess()
```

End

The variables, constructors, and methods needed to create new BankAccount objects.

BankAccount

```
    1. Create static integer variable for total accounts as totalAccounts
    2. Create String variable for account number as accountNumber
    3. Create int[10] array for account number digits as accountNumDigits
    4. Create double variable for balance as balance
    5. Create integer variable for number of deposits this month as deposits
    6. Create integer variable for number of withdrawals as withdrawals
    7. Create double variable for annual interest rate as apr
    8. Create float variable for monthly service charges as fees
    9. Create boolean variable for account type as isSavings

Begin BankAccount()

    1. Empty constructor

End

Begin BankAccount(double : balance, double : rate)

    1. Call incrementAccounts()
    2. Call setAccountNum()
    3. this.balance = balance
    4. this.apr = rate
    5. Print "New account OPENED." + "\nBalance: $%.2f" + "\nAPR: %.4f%%", getBalance(), (apr *
       100)

End

Begin incrementAccounts()

    1. totalAccounts += 1

End

Begin setAccountNum()

    1. String c = string representation of totalAccounts integer
    2. String t = ""
    3. For int i = 0; i < 10 – length of c; i++
    4.   t = t + "0"
    5. accountNumber = t + c

End

Begin deposit(double : amt)

    1. this.balance += amt
    2. Call incrementDeposits()
    3. Print "\n\nAmount deposited: $%.2f" + "\nBalance: $%.2f", amt, getBalance()

End
```

```
Begin incrementDeposits()

    1. this.deposit += 1

End

Begin getDeposits()

    1. return deposits

Begin withdraw(double : amt)

    1. if amt > this.balance
    2.   Print “Insufficient funds.” + “\nBalance: “ + balance + “\nPlease enter a smaller amount.”
    3. else
    4.   this.balance -= amt
    5.   Call incrementWithdrawals()
    4.   Print “\n\nAmount withdrawn: $%.2f” + “\nBalance: $%.2f”, amt, getBalance()

End

Begin incrementWithdrawals()

    1. this.withdrawals += 1

End

Begin getWithdrawals()

    1. return withdrawals

Begin calcInterest()

    1. double rate = (this.apr / 12)
    2. double interest = balance * rate
    3. this.balance += interest

End

Begin monthlyProcess()

    1. Print “Running monthly process.”
    2. Print “\nBalance: $%.2f” + “\nFees: $%.2f”, getBalance(), getFees()
    3. this.balance -= fees
    4. double afterFees = getBalance()
    5. Call calcInterest()
    6. Double interest = getBalance() – afterFees
    7. Print “\nInterest: $%.2f” + “\nUpdated balance: $%.2f”, interest, getBalance()
    8. this.withdrawals = 0
    9. this.deposits = 0
    10. this.fees = 0
```

```
End
Begin getBalance()
    1. return balance
End
Begin setBalance(double : amt)
    1. this.balance = amt
Begin getAccountNum()
    1. return accountNumber
End
Begin getAPR()
    1. return apr
End
Being setAPR(double : rate)
    1. this.apr = rate
End
Begin getFees()
    1. return fees
End
Begin setFees(float : amt)
    1. this.fees = amt
End
Begin setAsSavings()
    1. this.isSavings = true
End
Begin getAccountType()
    1. if isSavings == false
    2.   return “Checking”
    3. else
    4.   return “Savings”
Begin toString()
```

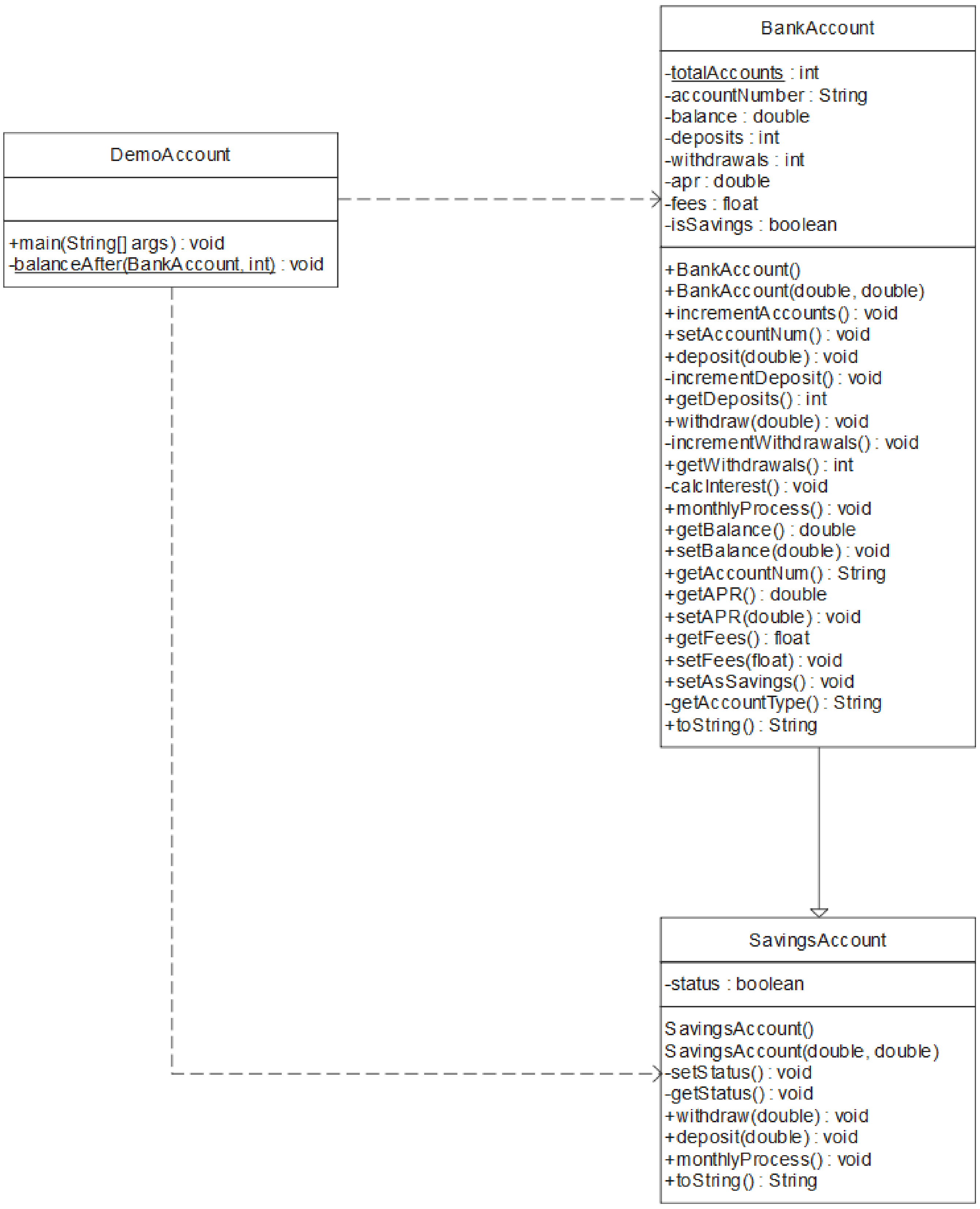
```
    1. return “Account type: %s”, getAccountType() + “Account number: %s”, accountNumber +
       “\nBalance: $%.2f”, balance + “\nInterest rate: %.4f%%”, (apr * 100)
End
```

The variables, constructors, and methods needed to create new SavingsAccount objects.

```
SavingsAccount
    1. Create boolean variable for account status as status

Begin SavingsAccount()
    1. Empty constructor
End

Begin SavingsAccount(double : amt, double : rate)
    1. Call incrementAccounts()
    2. Call setAccountNum()
    3. Call setAsSavings()
    4. Call setBalance(amt)
    5. Call setAPR(rate)
    6. Call setStatus()
    7. Print “New account OPENED.” + “\nBalance: $%.2f” + “\nAPR: %.4f%%” + “\nAccount Status:
       %s\n”, getBalance(), (apr * 100), getStatus()
    8.
End

Begin setStatus()
    1. If getBalance() < 0
    2.   Print “\nMake deposit or account will be closed.”
    3. else if getBalance() > 0 and getBalance() < 25
    4.   Print “Please deposit $%.2f to activate account. \n”, (25 – getBalance())
    5. else if getBalance() > 25
    6.   this.status = true
    7. else
    8.   Print “This should never happen.”
End

Begin getStatus()
    1. if status == false
    2.   return “Inactive”
    3. else
    4.   return “Active”
```

```
End

Begin withdraw(double : amt)

    1. if status == false
    2.   Print "Unable to make a withdrawal." + "\nAccount inactive. $25 minimum required." +
         "\nBalance: $%.2f" + "\n$%.2f Deposit required to activate account.", getBalance(), (25 –
         getBalance())
    3. else
    4.   super.withdraw(amt)

End

Begin deposit(double : amt)

    1. if status == false and amt + getBalance() < 25
    2.   Print "\nUnable to deposit $%.2f" + "\nAccount inactive! Must maintain a minimum balance of
         $25." + "\nBalance: $%.2f" + "\nYou're deposit is $%.2f short." + "\nPlease make a deposit of at
         least $%.2f", amt, getBalance(), (25 – (amt + getBalance())), (25 – getBalance())
    3. else if status == false and (amt + getBalance()) > 25
    4.   this.status = true
    5.   super.deposit(amt)
    6. else
    7.   super.deposit(amt)

End

Begin monthlyProcess()

    1. if getWithdrawals() > 4
    2.   float serviceCharge = getWithdrawals() – 4
    3.   setFees(serviceCharge)
    4.   super.monthlyProcess()
    5.   setStatus()
    6. else
    7.   super.monthlyProcess()

End

Begin toString()

    1. return super.toString() + "\nStatus: %s", getStatus()

End
```

UML


DemoAccount
+main(String[] args) : void
-balanceAfter(BankAccount, int) : void
BankAccount
-totalAccounts : int
-accountNumber : String
-balance : double
-deposits : int
-withdrawals : int
-apr : double
-fees : float
-isSavings : boolean
+BankAccount()
+BankAccount(double, double)
+incrementAccounts() : void
+setAccountNum() : void
+deposit(double) : void
-incrementDeposit() : void
+getDeposits() : int
+withdraw(double) : void
-incrementWithdrawals() : void
+getWithdrawals() : int
-calcInterest() : void
+monthlyProcess() : void
+getBalance() : double
+setBalance(double) : void
+getAccountNum() : String
+getAPR() : double
+setAPR(double) : void
+getFees() : float
+setFees(float) : void
+setAsSavings() : void
-getAccountType() : String
+toString() : String
SavingsAccount
-status : boolean
SavingsAccount()
SavingsAccount(double, double)
-setStatus() : void
-getStatus() : void
+withdraw(double) : void
+deposit(double) : void
+monthlyProcess() : void
+toString() : String