**Account**

This is an abstract class. Other classes will inherit from this class.

This class has been built for you. Do not make any changes to this class.

Fields

* ID is a random four-digit ID number generated at account creation.
* Type is the type of account (Checking or Savings)
* Balance represents the amount of money in the account.

Public abstract methods

deposit

* Argument(s): double amount
* Return type: boolean

Withdraw

* Argument(s): double amount
* Return type: boolean

Reconcile

* Argument(s): none
* Return type: none

Public non-abstract methods

getID

* Argument(s): none
* Return type: integer
* Return the ID associated with this Account

getType

* Argument(s): none
* Return type: String
* Return the type of this Account

getBalance

* Argument(s): none
* Return type: double
* Return the balance of this Account

**Checking**

This class should inherit from the Account class.

Fields

Create a constant field that sets a minimum balance of $100

Constructor

* Accept one double argument and set the balance to this value.
* Throw exception if argument is not a positive value

Methods

Deposit

* Argument(s): double amount
* Return type: boolean
* Add amount to balance
* Return true if transaction is successful; otherwise return false
* Throw exception if amount is not a positive value

Withdraw

* Argument(s): double amount
* Return type: boolean
* Subtract amount from balance
* Return true if transaction is successful; otherwise return false
* Throw exception if amount is not a positive value

Check

* Argument(s): double amount
* Return type: boolean
* Processing a check is similar to a withdrawal, except that you cannot process a check if the account does not have enough money
  + Should display a message or throw an exception
* Return true if transaction is successful; otherwise return false
* Throw exception if amount is not a positive value

Reconcile

* Argument(s): none
* Return type: none
* Apply $8 Minimum Balance Fee if balance is less than minimum balance

toString

* Overrides the toString() method inherited from the Object class
* Argument(s): none
* Return type: String
* Display the account type, ID number, and balance on one line
  + e.g. “Checking 1234 | Balance $5,678.90”

**Savings**

This class should inherit from the Account class.

Fields

interestRate: Create a double field that holds the annual interest rate as a decimal   
(e.g. 5% = 0.05)

Constructors

* Accept one double argument, set the balance to the argument value, and set the annual interest rate to 2%
* Accept two double arguments, set the balance to the first argument value, and set the annual interest rate to the second argument value
* Both arguments should be positive. Throw an exception if either argument is not a positive value.

Methods

deposit

* Argument(s): double amount
* Return type: boolean
* Add amount to balance
* Return true if transaction is successful; otherwise return false
* Throw exception if amount is not a positive value

withdraw

* Argument(s): double amount
* Return type: boolean
* Subtract amount from balance only if the account has enough money
  + i.e. savings account cannot be overdrawn
* Return true if transaction is successful; otherwise return false
* Throw exception if amount is not a positive value

reconcile

* Argument(s): none
* Return type: none
* Add one month’s worth of interest to the balance
  + e.g. if the balance if $5,000 and the interest rate is 6%,   
    the monthly interest is $25 (5000 \* 0.06 / 12)

toString

* Overrides the toString() method inherited from the Object class
* Argument(s): none
* Return type: String
* Display the account type, ID number, and balance on one line
  + e.g. “Savings 2468 | Balance $3,579.01”

Bank Demo

Create an array (or ArrayList) of accounts in this order

* Checking account with $500
* Savings account with $1,000
* Checking account with $250
* Savings account with $5,000 and 4.5% interest rate

Display each account to the console.

Open the text file named transactions.txt.

The first line of the text file are headers and should be discarded.

Then read and process each transaction.

Note: Each transaction has the following pattern: String, int, String, double. Examples:

* 7/8/2024,1,deposit,75.25
* 7/14/2024,4,withdraw,54.11
* 7/29/2024,3,check,84.95

Notes:

* The date should be saved as a String.
* The number after the date refers to the position in the array but is off by one.   
  In other words, 1 refers to the first element in the array, 2 refers to the second element, etc. This is common as non-technical users start counting at 1, not 0.
* The transaction type will either be “deposit”, “withdraw”, or “check”. There should be no “check” transactions on a savings account.
* If a transaction has a negative amount, the program is expected to handle the exception gracefully and continue processing any remaining transactions.

After each transaction, display an appropriate message to the console. Remember that processing the check is the same as a withdrawal. Examples:

* “Deposit $50.00 to checking account 1234 on 7/1/2024”
* “Withdraw $35.00 from savings account 5678”
* “Deposit $152.00 to checking account 1234”
* “Insufficient funds in savings account 2468”

Extra credit: send the transaction messages to an output file and display a single message to the console. Example: “Transaction log saved to banklog.txt”. Do not save any other output to the output file.

After processing all the transactions, reconcile each account then display each account to the console.

Here are the final account balances in the order they should appear in your array. Note that your account numbers will be different.

* Checking 1234 | Balance $3,579.01
* Savings 5678 | Balance $3,579.01
* Checking 2468 | Balance $3,579.01
* Savings 7531 | Balance $3,579.01

Submit the entire package as a .zip file to Blackboard.