**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.
* Balance represents the amount of money in the account.

Public abstract methods

deposit

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

Withdraw

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

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

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: none
* Add amount to balance
* Print message to console that states amount and account ID
  + e.g. “Deposit $50.00 to checking account 1234”
* Throw exception if amount is not a positive value

Withdraw

* Argument(s): double amount
* Return type: none
* Subtract amount from balance
* Print message to console that states amount and account ID
  + e.g. “Withdraw $35.00 from checking account 1234”
* Throw exception if amount is not a positive value

Reconcile

* Argument(s): none
* Return type: none
* Apply ONE of these fees, in order of precedence, if applicable
  + Overdraft fee $35, if balance is less than zero
  + Minimum balance fee $14, 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

Create a double field that holds the 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
* Throw exception if either argument is not a positive value

Methods

deposit

* Argument(s): double amount
* Return type: none
* Add amount to balance
* Print message to console that states amount and account ID
  + e.g. “Deposit $75.00 to savings account 2468”
* Throw exception if amount is not a positive value

withdraw

* Argument(s): double amount
* Return type: none
* Subtract amount from balance only if the account has enough money
  + i.e. savings account cannot be overdrawn
* Print appropriate message to console
  + e.g. “Withdraw $35.00 from savings account 2468”
  + e.g. “Insufficient funds in savings account 2468”
* 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 5% interest rate

Display each account to the console.

There are three text files with transactions. Each transaction has the following pattern:

* Position in your array (0-3), “deposit” or “withdraw”, amount
* e.g. “1, deposit, 75.25”
* e.g. “2, withdraw, 54.11”

Starting with the first text file (july.txt), read and process each transaction. After the final transaction, reconcile and display each account to the console.

Repeat the previous step after processing the transactions in august.txt, and again after september.txt. Hint: if you save the file names to an array, you can use a for loop to read each text file then reconcile and display the accounts.

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

* Checking 2468 | Balance $3,579.01
* Savings 2468 | Balance $3,579.01
* Checking 2468 | Balance $3,579.01
* Savings 2468 | Balance $3,579.01