Making Bank

A Bank has a set of Customers, and a set of Employees. Customers and Employees both have contact information (ContactInfo), which is a name string and a phone number string. Both name and phone strings can contain arbitrary text, but neither can be null or empty. ContactInfo for Customers and Employees is the same content and format, so it should be stored in the same class. In addition, both Customers and Employees have an ID number unique to the customer or employee. The ID number is assigned by the Bank when the Customer or Employee is added.

An Employee has a salary, which is the amount in dollars they are paid each month. To simplify things, an Employee has an attribute totalSalary, which is the total amount the bank has paid her and is updated every time the Employee is paid.

No individual can be both an Employee and a Customer.

A Customer can have a checking account, a savings account, or both, or neither. All accounts have a balance which must be at least 0. All accounts support deposit and withdraw operations. Trying to deposit or withdraw a negative amount is an error (IllegalArgumentException). Trying to withdraw more than the account's balance is an error (InsufficientFundsException). The Bank may close an account – if a Checking Account has insufficient funds to cover its monthly charge, for example. Trying to deposit to or withdraw from a closed account is an error (AccountClosedException).

The Bank itself also has "funds on hand" – a dollar amount is which required to be at least 50% of the total balances of all its accounts. If its funds on hand drop below that level, the bank must throw a BrokenBankError. Notice that since BrokenBankError is an Error rather than an Exception, it does not need to be explicitly caught.

Accounts are of two types: Checking Accounts and Savings Accounts. A Customer may deposit to and withdraw funds from either account, provided the account is open and there are sufficient funds.

A Checking account incurs a monthly fee of $1.00. A Savings account earns 1% interest per month.

Once a month the Bank runs a procedure that

1. Assesses the fee on all checking accounts and adds that amount to the bank's funds on hand. If the checking account balance is less than $1.00, it closes the account. It adds the account fee to its funds on hand.
2. Adds interest to all savings accounts and deducts that amount from the bank's funds on hand
3. Pays its employees: for each Employee, increments the employee's totalSalary by the amount of her salary, and reduces funds on hand by that amount.

Every time a transaction (deposit, withdrawal, monthly fee, interest payment) is made to an account, a TransactionRecord is created. The TransactionRecord is a pair (transactionType, amount) where transactionType is one of (deposit, withdrawal, fee, interest). An account holder may not view these records directly, but they may be displayed in debugging contexts.

Code and Files Provided

- These files are checked in to Github, in the Assignment4 directory
    - BankInterface.java – your Bank class must implement this interface
    - AccountInterface.java – your CheckingAccount and SavingsAccount classes must implement this interface
    - The Person class illustrates the idiom for assigning a new unique ID to each new instance of that class or subclass
    - The Bank class has a line that formats a double as currency (debugging only)
    - Three error/exception classes
    - A Driver function and transcript showing the solution code in action. The solution code Bank implements a report method to display to a debugging console. Your solution does not need to implement the report method, and does not need to have a driver class.

Requirements for Your Implementation

- You must implement a class Bank in a package bank, which implements BankInterface as provided
- Even though not in the interface, Bank must throw BrokenBankError if its funds on hand falls below the 50% limit
- All additional classes that comprise your solution must also be in the package bank
- Checking account and Savings account must both be subclasses of a parent class Account, and all common storage and methods must be implemented in the superclass
- Customer and Employee must both be subclasses of a parent class Person, and all common storage and methods must be implemented in the superclass
- Name and phone information for Customer and Employee must be implemented in a class ContactInfo

---

What to Hand In

- Hand in a single zip file
- The zip file must contain *only* a folder src, containing your source code.
- The src folder must contain *only* a subfolder bank containing your code in the package with the same name

You do not need a README file for this exercise, but you can add one to the top-level directory if you want. Please be sure not to hand in anything not requested above.