Program #2

Due:	**10/16/18 at 10:00 PM**
Late Collections:	**10/17/18 at 10:00 PM**
	10/18/18 at 10:00 PM
Files (4):	**Bank.java, BankAccount.java, CheckingAccount.java, SavingsAccount.java**

For this assignment, you will write a simple program to support a banking institution. You will write four classes: **Bank**, **BankAccount**, **CheckingAccount** and **SavingsAccount**. There is a composition relation between **Bank** and **BankAccount**, and both **CheckingAccount** and **SavingsAccount** are subclasses of **BankAccount**. You will explore concepts of inheritance and polymorphism while strengthening your skills regarding reading from files.

The details of each class are given in UML form below Note, it is essential that the names and types of all data fields and the names, return types and parameter lists of all methods appear exactly as described. Otherwise, you may lose points on the assignment.

BankAccount	
#accountNum:int	The account number
#customerName:String	The customer's full name
#balance:double	The account balance, in dollars and cents
+BankAccount(accountNum:int, customerName:String, balance:double)	Create a new bank account with given account number, customer, and balance
+BankAccount(accountNum:int, customerName:String)	Create a new bank account with the given parameters, initializing the balance to 0.
+getAccountNum():int	Get method for **accountNum**
+getCustomerName():String	Get method for **customerName**
+getBalance():double	Get method for **balance**
+makeDeposit(depositAmt:double):void	Add **depositAmt** to the account's balance
+printAccountInfo():void	Prints information about the account to the screen in the format specified below.

CheckingAccount	*A subclass of BankAccount!*
-monthlyFee:double	The fee that is applied to the account every month (in dollars and cents).
+CheckingAccount(accountNum:int, customerName:String, balance:double, monthlyFee:double)	Create a new checking account with the given account number, customer, balance, and monthly fee.
+getMonthlyFee():double	Get method for **monthlyFee**
+setMonthlyFee(monthlyFee:double):void	Set method for **monthlyFee**
+applyFee():void	Subtracts **monthlyFee** from the balance.
+printAccountInfo():void	Prints information about the account to the screen in the format specified below.

SavingsAccount	*A subclass of BankAccount!*
-interestRate:double	The <u>monthly</u> interest rate of the account in decimal form, e.g., 0.5% is 0.005.
+SavingsAccount(accountNum:int, customerName:String, balance:double, interestRate:double)	Create a new savings account with the given account number, customer, balance, and interest rate (in decimal form as above)
+SavingsAccount(accountNum:int, customerName:String, interestRate:double)	Create a new savings account with the given parameters, initializing the balance to 0
+getInterestRate():double	Get method for **interestRate**
+accrueInterest():void	Adds **interestRate** * **balance** to the **balance**.
+printAccountInfo():void	Prints information about the account to the screen in the format specified below.

Bank	
-name:String	The name of the bank
-accounts:BankAccount[]	The set of accounts held by the bank
-totalAccounts:int	The number of accounts the bank holds
<u>+MAX_ACCOUNTS:int</u>	The maximum number of accounts the bank could hold. Is a constant with value 20.
+Bank(name:String)	Create a new bank with the given name. Initialize its **totalAccounts** to 0, and make **accounts** to be size **MAX_ACCOUNTS**
+getName():String	Get method for bank **name**
+addAccount(newAcct:BankAccount):void	Add the given account to **accounts** and update **totalAccounts** accordingly.
+printBankSummary():void	Print the bank name and information about each account, using the appropriate **printAccountInfo** method, with one account per line.
+accrueInterestToSavingsAccounts():void	Accrue one month's interest to each savings account, using its **interestRate.**
+applyFeesToCheckingAccounts(): void	Subtract the **monthlyFee** from the balance of each CheckingAccount.
+loadAccountsFromFile(acctFile:File):void	Open a Scanner on **acctFile,** and for each line, create a checking or savings account, and add the accounts to the bank. See below for information on the format of the file.

Format for printAccountInfo()

The format for **printAccountInfo** in **BankAccount** should be :
accountNum customerName balance
For a **CheckingAccount**, the format should be:
accountNum customerName balance   Monthy fee: $*fee*
For a **SavingsAccount**, the format should be:
accountNum customerName balance   Interest rate: *rate*%

Account numbers should be in columns of width 5, customer names have width 20 and are left-justified, and the balance should be a floating point with two places after the decimal, and 5 places reserved before the decimal. Put one space between each column. The monthly fee should have two places reserved before the decimal, and two after. The interest rate should be printed as a percentage, have two places reserved before the decimal and one after. See the example output at the end of the assignment.

Format for input file
A sample input file is given below:

```
S      42001   Gordon Gecko        85234.12    0.001
C      44001   Flower Power        12.83   9.95
C      44002   Joe Schmo           392.52  4.50
```

The first letter indicates whether the account is savings ("S") or checking ("C"). This is followed by an integer account number, the name of the account holder and the account balance. The last column of a savings account is the monthly interest rate written in decimal form (i.e., 0.0001 is 0.1%). The last column of checking account is the monthly fee, in dollars and cents. All fields are separated by the **tab** character.

The main method
Bank should also have a **main** method that does the following. First it checks if there is exactly one command-line argument, and exits with a user-friendly message if not. Then it creates a new bank named "Java S&L". It reads the accounts from the file specified by the command-line argument and prints a "bank summary." Then it accrues interest to all of the savings accounts, and subtracts the monthly fee from all of the checking accounts. Finally, it prints the "bank summary" again.

Assuming **acctinfo.txt** looks like the file described above, your output should look like:

```
> java Bank acctinfo.txt
Bank Name: Java S&L
42001 Gordon Gecko         85234.12   Interest Rate:  0.1%
44001 Flower Power            12.83   Monthly fee: $ 9.95
44002 Joe Schmo              392.52   Monthly fee: $ 4.50

Bank Name: Java S&L
42001 Gordon Gecko         85319.35   Interest Rate:  0.1%
44001 Flower Power             2.88   Monthly fee: $ 9.95
44002 Joe Schmo              388.02   Monthly fee: $ 4.50
>
```

Hints:

- Note that the '#' before the fields in BankAccount mean they are protected. This is discussed in Sect. 11.14 (pp. 442-444) of the book, and will be discussed in class soon.
- The **instanceof** operator can be used to test the actual type of a polymorphic variable. See Sect. 11.9 (pp. 429-433) for details.

- You may find it useful to call the **useDelimiter()** method of your Scanner object with the string argument "\\t|[\\n\\r\\f]+" (this specifies that the delimiter between tokens read by the Scanner is the tab character or any sequence of characters that indicates the end of one line and the start of a new one). This affects the behavior of the **next()** and **nextInt()** methods.
- If you use **throws Exception** on a method other than **main.** you will need to add it to every method that invokes it.
- In a **printf** statement, use "%%" to output the percent character (since a single % indicates the start of a format specifier).

Comments
At a minimum provide a Javadoc comment explaining what each method does, and another one that summarizes the class. Include additional comments for lines that are especially complicated. At the top of the program include a comment with the following form:

```
/*
CSE 17
Your name
Your Lehigh e-mail
Tutor: name (e-mail)   [only include if a tutor aided you]
Program #2      DEADLINE: October 16, 2018
Program Description: Simple Bank
*/
```

Submission
Make sure that you have named your classes and files as specified, and that your main method is in the **Bank** class. Once the program compiles, runs, and has been tested to your satisfaction, upload all four**.java** files to Course Site. To do so, click on the name of the assignment in the Course Site page, and then press the "Add submission" button at the bottom of the next page. Drag and drop each file into the area under "File submissions". If necessary, you can update your submission at any time before the deadline passes.