Date, Money, Bill

Purpose:

The purpose of this lab is to write some classes that you will be using in later assignments, to explore the issue of privacy leaks, to consider testing, and to write some larger classes with more methods.

Program Behavior:

There is no specific behavior for this program – except that the classes you create must pass a test case that the instructor will apply which will test the functionality for the classes.

Implementation Summary:

Build three classes (**Money**, **Date** and **Bill**) for use in managing a set of outstanding and paid bills. Submit these **three classes and a test driver (4 files total, 5 if you have both driver and Junit tests)** that exercises all functionality for these classes. Submit only the .java source code files inside a single .zip file. In this assignment, we'll explore class composition, accessors and mutators, privacy leaks, toString(), and equals().

For your **test driver** you must use **at least three tests in Junit** to get experience with the test harness. Overall, you may use the JUnit test model (preferred) –OR- you may simply may write a main method which exercises the classes. We have provided a sample test driver for each style. (BillMoneDateDriver and DateMoneyBillJUnit ***).*** These drivers are not complete and must be extended to test all methods to receive full credit.

Please note that for all of the classes in the project, you MUST create exactly the methods described below. We will be testing this with our own test driver that will depend on the exact method names and signatures. As always, you may make additional private methods. The test driver for grading will be more extensive and check many different cases for input data.

Class Money:

The Money class is used to track a USD amount consisting of two integers to manage dollars and cents. All amounts will be positive or 0. We should never return any cents greater than 99. The data and methods below will be called from the test driver I'll use, so be sure to adhere to

the names of the functions, as well as the number, order, and type of arguments handed to each function, as indicated below.

If the value of cents is ever greater than 99, we will add to the dollar amount the appropriate value. IE if the user calls setMoney(3, 550) this is equivalent to a call of setMoney(8,50);

Money Methods:

- Money (int dol)
  - Constructor which sets the dollar amount, and sets cents to 0
- Money(int dol, int cent)
  - Constructor which initialized dollars and cents.
- Money(Money other)
  - Copy Constructor
- int getDollars()
- int getCents()
- void setMoney(int dollars, int cents)
- double getMoney( )
  - Gets the money amount as a double
  - Returns 5.75 for example
- void add(int dollars);
- void add (int dollars, int cents);
- void add(Money other)
  - Adds the amounts in other to our money object – reducing cents appropriately.
- Boolean equals (Object o)
- String toString()
  - Prints out the amount as a string IE “$3.75” or “$4.00”. Note the number of digits displayed for cents.

Class Date:

The date class represents a date in our world. Do not use any built-in date classes in this assignment,

- All days should be between [1-31]
- Each month should be between [1-12]
- Each month has exactly 31 days. (We’ll use this for simplification.)
- All years should be between [2016 – 2026]
- Entering an invalid date should print an error message to the console, and the program should immediately exit.

The Savitch book talks about an implementation for dates that is similar (but not exactly) like what we are describing here.

Date Methods:

- Date (int month, int day, int year)
  - Constructor
- Date (Date other)
  - Copy Constructor
- int getYear();
- int getMonth();
- int getDay();
- void setYear(int year);
- void setMonth(int month);
- void setDay(int day);
- boolean isAfter(Date compareTo)
  - Return true if the compareTodate is after the date.
  - WARNING: the direction of this comparison may be the opposite of what you're expecting – make sure you get it right.
- boolean equals(Object date)
- String toString()
  - Prints out the date in the form "mm/dd/yyyy". Note that "1/23/2015" is invalid. It should be "01/23/2015"

Bill Class

Represents an outstanding or paid bill. The class should contain the amount of the bill as a Money object, the due date of the bill as a Date object, and a Date object to track the date paid (null if not yet paid). Check to be sure that when paying the bill, the date isn't past the due date.

Think carefully when returning objects or setting objects so that you do not leak information or access.

Instance Variables:

- amount – a Money object
- dueDate – a Date Object
- paidDate – a Date object – null if not yet paid
- originator – a string containing the company or institution that initiated the bill.

Bill Methods:

- Bill (Money amount, Date dueDate, String originator)
    - Constructor. (paidDate would be set to null)
- Bill (Bill toCopy);
- Money getAmount()
- Date getDueDate()
- String getOriginator();
- boolean isPaid(); - returns true if bill is paid. Else false.
- boolean setPaid(Date datePaid) –
    - if datePaid is after the dueDate, the call does not update anything and returns false. Else updates the paidDate and returns true
- boolean setDueDate(Date nextDate)
    - Resets the due date – If the bill is already paid, this call fails and returns false. Else it resets the due date and returns true.
- boolean setAmount(Money amount)
    - Change the amount owed. If already paid, returns false and does not change the amount owed else changes the amount and returns true.
- void setOriginator();
- String toString() –
    - Build a string that reports the amount, when due, to whom, if paid, and if paid the date paid
- boolean equals (object toCompare)
    - return if the two objects are the same.

Driver Class

See the driver classes below as an example of a starting point. JunitTestDriver and MainTestDriver can serve as a starting point for testing.

Note that these are starting points only. At a minimum you should test every method of every class at least once. Most methods will need to be tested more than once as there is more than one case of interest. IE make sure that when you set the Amount on a paid bill, nothing changes and the method returns false. – or that when you set a paidDate, that an external user cannot change it to an invalid date

Hints:

- Start with the smallest class first (Date or Money) and then build the larger classes (Bill).

- First, use the sample driver with your classes — then extend the driver to test each function in Bill, Date and Money.
- Be sure to document your code with comments.
- Again, make sure you comment all methods and the class. If you have a driver that has code someone else has written (e.g., is based on a skeleton), you have to comment the methods someone else wrote also.
- Watch out for privacy leaks in your getters, setters, and constructors.