Project #2 – 150 points

Due Date

Monday, October 16, 11:59pm.

Submission

1. Zip your project folder and submit the zipped file to Canvas. Include the following grading items.
   - **Source folder src,** containing the source files (*.java) [100 points]
     (a) MUST create a Java package to hold the source files; MUST use all lowercase letters for the package name; you will **lose 2 points** if this is not done properly.
     (b) MUST include the **@author** tag in the comment block on top of EVERY Java class and put the name(s) who implemented the Java class, or you will **lose 2 points**.
   - **Class Diagram** [15 points]
   - **JUnit** test classes
     (a) Date class, isValid() method. [20 points]
     (b) AccountDatabase class, close() methods [10 points]
   - **Javadoc** folder **doc**, including all the files generated. [5 points]
2. The submission button on Canvas will disappear after **October 16, 11:59pm**. It is your responsibility to ensure your Internet connection/speed is good for submitting the project on time. **You get 0 points** if you do not have a submission on Canvas. **Projects sent through the emails will not be accepted**.

Project Description

Your team will develop a simple software to process the banking transactions for RU Bank. The transactions will be entered through the command lines on the terminal. The software is an interactive system to produce the output whenever a transaction is entered. The software shall be able to process transactions of opening an account, closing an existing account, depositing money to an existing account, withdrawing money from an existing account, and print the details of all accounts. RU Bank provides four types of banking accounts listed in the table below. Note that, same person can hold different types of accounts, however, cannot hold a College Checking and Checking at the same time. For all account types, must be age of 16 or older to open, for College Checking, must be under the age of 24 to open. The interest rates and monthly fees are different based on the account types and account options.

Account Type	Monthly Fee	Annual Interest Rate	Account Options/Rules
Checking	$12	1.0%	• Monthly fee is waived if account balance is >= $1000.
College Checking			• No monthly fee for College Checking.
Savings	$25	4 %	• Savings with loyal customer status get additional 0.25% for the annual interest rate, i.e., 4.25% • No monthly fee for Savings with balance >= $500
Money Market Savings		4.5%	• Minimum deposit $2000 to open Money Market. • Money Market by default has loyal customer status and get additional 0.25% for the annual interest rate, i.e., 4.75% • No monthly fee for Money Market with balance >= 2000 • Uncheck loyal customer status when account balance dropped below $2000, check the loyal customer status when balance is >= $2000 again. • $10 fee applies if the number of withdrawals > 3 times.

A banking transaction is a command line that always begins with a command, in uppercase letters, and followed by several data tokens delimited by one or more spaces. **Commands are case-sensitive**, which means the commands with lowercase letters are invalid. Your software shall support the following commands.

- **O** command, to **open an account** with the desired account type. There are 4 account types: C – checking, CC – college checking, S – savings, MM – money market savings. Each account holder is identified by his/her profile, which includes first name, last name, and date of birth. There must be an initial deposit to open the account. When an account is opened, the account is added to the account database. Below is a list of sample transactions for opening an account. Note that, it is possible that the user didn't enter enough data tokens for opening an account. You must handle the exceptions.

```
O  C    John Doe 2/19/2000 599.99
O  CC   Jane Doe 10/1/2000 999.99 0
O  S     april march 1/15/1987 1500 1
O  MM  Roy Brooks 10/31/1979 2909.10
```

  The above transactions start with the O command followed by the account type, first name, last name, date of birth, and the amount of the initial deposit. When opening a College Checking, a campus code is required: 0 – New Brunswick, 1 – Newark, 2 – Camden. Other campus codes are invalid. Savings accounts must enter the loyal customer status: 0 – non-loyal customer, 1 – loyal customer. Money Market requires a minimum of $2000 to open, and it is set to loyal customer status by default. The names are not case-sensitive. If the date of birth entered is today or a future date, it is invalid.

- **C** command, to **close an existing account**. When an account is closed, it will be removed from the account database. Below are the sample transactions.

```
C  MM   Jane Doe 10/1/1995
C  C    April March 1/15/1987
C  CC   John Doe 2/19/1989
C  S   April March 1/15/1987
```

- **D** command, to **deposit money** to an existing account. You should reject the transaction if an invalid amount is entered. Below are the sample transactions.

```
D  C    John Doe 2/19/1990 100
D  CC   Kate Lindsey 8/31/2001 100
D  MM   Roy Brooks 10/31/1979 100.99
D  S     April March 1/15/1987 100
```

- **W** command, to **withdraw money** from an existing account. Command line formats are the same with the D command, and the same rules apply. However, you must check if there is enough balance for the withdrawal. For Money Market accounts, the loyal customer status will be unchecked if balance dropped below $2000.
- **P** command to **display** all the accounts in the account database, sorted by the account types. For the same account type, sort by the account holder's profile (last name, first name and dob.)
- **PI** command, to **display** all the accounts in the account database**,** the same order with the P command. In addition, **display** the calculated fees and monthly interests based on current account balances. Fees and interests should be displays with 2 decimal places, see the sample output.
- **UB** command, to **update the account balance** for all accounts by applying the fees and interests earned. This should reset the number of withdrawals of the Money Market accounts to 0.
- **Q** command, to stop the program execution and display `"Transaction Manager is terminated."`

Project Requirement

1. You MUST follow the Coding Standard posted on Canvas under Week #1 in the "Modules". **You will lose points** if you are not following the rules.
2. You are responsible for following the Academic Integrity Policy. See the **Additional Note #14** in the syllabus.
3. Test cases for grading are included in the file **Project2TestCases.txt** and the associated output is in the file **Project2ExpectedOutput.txt**. Your project should be able to read the test cases from console in the same order with the test cases provided in **Project2TestCases.txt**, line by line without getting any exceptions. Your program should be able to ignore the empty lines. The graders will run your project with the test cases in **Project2TestCases.txt** and compare your output with the expected output in **Project2ExpectedOutput.txt**. You will **lose 2 points** for each output not matching the expected output, OR for each exception causing your project to terminate abnormally. You MUST use the **Scanner class** to read the command lines from the standard input (**System.in**), DO NOT read from an external file, or you will **lose 5 points**.
4. Each source file (.java file) can only include one public Java class, and the file name is the same with the Java class name, or you will lose **-2 points**.
5. Your program MUST handle bad commands; **-2 points** for each bad command not handled, with a **maximum of losing 6 points**.
6. You are not allowed to use any Java library classes, EXCEPT the **Scanner**, **StringTokenizer, Calendar** and **DecimalForamt** class. **You will lose 5 points** FOR EACH additional Java library class imported, with a **maximum of losing 10 points**.
7. You are not allowed to use the Java library class **ArrayList** anywhere in the project OR use any Java library classes from the Java Collections, or **you will get 0 points for this project**!
8. When you import Java library classes, be specific and DO NOT import unnecessary classes or import the whole package. For example, `import java.util.*;`, this will import all classes in the `java.util` package. You will **lose 2 points** for using the asterisk "*" to include all the Java classes in the `java.util` package, or other java packages, with a **maximum of losing 4 points.**
9. You **CANNOT** perform read or write **with Scanner or System.out** in all classes, EXCEPT the user interface class TransactionManager.java, and the print methods in the AccountDatabase class. You will lose **2 points** for each violation, with a **maximum of 10 points off.**
10. **Polymorphism** is required, i.e., dynamic binding for the equals(), compareTo(), toString(), and the abstract methods based on the actual types, or you will **lose 10 points.** You can define overloading methods if necessary.
11. You must create the Java classes below with the proper inheritance relationships, or **-5 points** for each class missing, or incorrect inheritance. You must always add the **@Override** tag for overriding methods, **or -2 points** for each violation. All instance variables must be "private". You cannot add additional instance variables, or change the method signatures for the methods listed; **-2 points** for each violation.
    - **Account** class, this is **an abstract class** that is the general type of the other account types; each account has a profile that uniquely identifies the account holder.

```
public abstract class Account implements Comparable<Account> {
     protected Profile holder;
     protected double  balance;
     public abstract double monthlyInterest();
     public abstract double monthlyFee();
}
```

    - **Checking** class, extends the Account class without defining any additional instance variable, but defines the constants for interest rate and fee.
    - **CollegeChecking** class, extends the Checking class; includes one instance variable only, and defines the constants for interest rate and fee. `private Campus campus; //campus code`

- **Savings** class, extends the Account class; includes one instance variable only, and defines the constants for interest rate and fee. `protected boolean isLoyal; //loyal customer status`
- **MoneyMarket** class, extends the Savings class; includes one instance variable only, and defines the constants for interest rate and fee. `private int withdrawal; //number of withdrawals`

12. You MUST include the Java classes below. **-5 points** for each class missing or NOT used.
    (1) **Date class.** Reuse the code from your Project 1.
    (2) **Profile class.** Define the profile of an account holder as follows. You cannot add additional instance variables. **-2 points** for each violation.

```
public class Profile implements Comparable<Profile>{
    private String fname;
    private String lname;
    private Date   dob;
}
```

    (3) **AccountDatabase class.** An instance of this class is a linear data structure using an array to hold a list of accounts with different types. The initial capacity of the array is 4. The capacity will be automatically increased by 4 whenever the array is full. You must implement and use the methods listed and you cannot add additional instances variable or change the signatures of the methods. **-2 points** for each violation.

```
public class AccountDatabase {
    private Account [] accounts; //list of various types of accounts
    private int numAcct; //number of accounts in the array

   private int find(Account account) {} //search for an account in the array
   private void grow() //increase the capacity by 4
   public boolean contains(Account account){} //overload if necessary
   public boolean open(Account account){} //add a new account
   public boolean close(Account account){} //remove the given account
   public boolean withdraw(Account account){} //false if insufficient fund
   public void deposit(Account account){}
   public void printSorted(){} //sort by account type and profile
   public void printFeesAndInterests(){} //calculate interests/fees
   public void printUpdatedBalances(){} //apply the interests/fees
 }
```

    (4) **TransactionManager class.** This is the **user interface class** that processes the transactions entered on the terminal and performs all Input/Ouput. This class **handles all Java exceptions and invalid data** before it calls the methods in AccountDatabase class to complete the transactions. For example, InputMismatchException, NumberFormatException, NoSuchElementException, invalid dates of birth, invalid campus codes, and invalid amounts. Whenever there is an exception or invalid data, display a message on the terminal. See the **Project2ExpectedOutput.txt** for the proper messages to display. **-2 points** for each exception not caught or invalid data not checked in this class or messages not displayed. You must include a run() method to process the command lines. You will **lose 5 points** for not including this method, or the method exceed 40 lines.
    (5) **RunProject2 class** as the driver to run Project 2.

13. **JUnit test classes.** Write code to test the following methods.
    - **isValid()** method in the Date class, 5 invalid cases, 2 valid cases.
    - **close()** method in the AccountDatabase class, with 1 true case and 1 false case.

14. **Class Diagram.** Create a class diagram to document your software structure for Project 2. Hand-drawing the diagram is not acceptable, and you'll **lose 15 points**. You can follow the links below to create a class diagram online and save it to your device as a picture, which can be included in your submission. You can also use other UML tools if you like. For each rectangle (class), include the instance variables and public methods. NO need to include the constants and private methods. DO NOT INCLUDE the test classes and the Java library classes in the diagram.
    - http://draw.io/ → Create New Diagram → UML → Class Diagram
    - https://online.visual-paradigm.com/app/diagrams/ → Create New → Class Diagram
15. You are required to **generate the Javadoc** after you properly commented your code. Your Javadoc must include the documentations for the constructors, private methods, and public methods of all Java classes. Generate the Javadoc in a single folder and include it in the zip file to be submitted to Canvas. **Please double check your Javadoc** after you generated it by **opening the index.html file** in the folder, and ensure the descriptions are NOT EMPTY. You will lose points if any description in the Javadoc is empty. You will **lose 5 points** for not including the Javadoc.