

Name: _____

CST8284: OOP (in Java): Midterm: Part 2a

Programming: (25 marks)

Answer the following programming problem on separate sheets of blank paper. Your answer will be stapled to these sheets when you are finished.

Imagine that you've just been hired to implement a new savings/checking account system for a major bank. You will create a class hierarchy to organize the *savings* and *checking* accounts, and store each account as it is created using a suitable container in a managing class. (Note: You'll probably see similarities between this problem and the work you've completed for your *Actor* and *Army* classes.)

You can assume that you have access to a class called **Input**: a well-functioning set of methods that perform keyboard input. I have attached the JavaDoc overview of the available methods. You can also see how the **Input** class is used in the **main()** method. (Note: This **Input** class is constructed differently than the **Input** class you used for your assignments. Check the documentation and sample code to see the differences.)

Implement the methods using the names shown below. You will need to determine the return type for each method.

You do NOT need to write **import** statements.

Account Superclass (abstract)

Instance Fields

- **name**: A **String** to hold the account holder's name.
- **acctNum**: An **int** to store the number of the bank account. The account number must be a 5 digit number (one that ranges from 10000 to 99999).
- **balance**: A **double** number to track the money currently held in the account.

Methods

- **inputAllFields()**: This will call suitable **Input** class methods to establish values in the three fields listed above.
- **getInterestRate()**: This method can simply return 0.0, which is the default (or standard) interest rate. The method will be *over-ridden* in the **Savings** subclass. The interest rate will be an annual rate.
- **getFees()**: This method can return \$4.50 as the standard fee. This method will be *over-ridden* in the **Checking** subclass.
- **calculateMonthEnd()**: This will adjust *balance* by adding the monthly interest (which is calculated based on the value returned by the call to **getInterestRate()**), and subtracting the monthly fees (which is calculated based on the value returned by the call to **getFees()**).
- **toString()**: This returns a reference to a **String** object which assembles the data stored in the **Account** object.

(Note: Because of the limited time, you do not have to match the output shown in the sample run of the program. Keep it simple! Basic output of variable values is sufficient for full marks.)

Checking Subclass

Instance Field

Checking accounts will be charged a fee of 75 cents per transaction, thus we will need to track the number of account transactions.

- **numTransactions**: An **int** to track the number of transactions handled throughout the month.

Checking Methods

- **inputAllFields()**: This will prompt the user to input a value **numTransactions** and ensure that the superclass **inputAllFields()** method is called.
- **getFees()**: Checking accounts charge a monthly fee of \$6.00, plus the 75 cents per transaction. This calculated value is returned by this method.
- **toString()**: This method ensures that all fields of the superclass and this class are assembled in a readable **String** object.

Savings Subclass

There are no instance fields in **Savings** subclass, but there is an over-ridden implementation of one superclass method.

- **getInterestRate()**: Savings accounts pay an annualized interest rate of 2%.

Class to Manage Accounts

You must implement a managing class (somewhat like the *Army* class in your lab work). The class will be called **Bank**.

This class will implement a suitable **Collection** to hold references to **Account** objects. It will also track the number of accounts actually created.

The **Bank** class will also implement the three methods that are called in the **main()** method (shown on the next page). **addAccount()** will add a single account on each call, either **Savings** or **Checking** depending on the selection made by the user. **calculateMonthEnd()** will process all existing accounts, handling issues such as interest and account fees. The **displayAllAccounts()** method will display all existing accounts.

Main Routine

I have printed a complete implementation of the test class on the following page. Do not add any further functionality to **main()**. Do not reproduce **main()** on your answer sheet.

```
public class TestBank {
    public static void main(String[] args) {
        Bank badBank = new Bank("The Big Bad Bank");

        int menuChoice;
        do {
            System.out.println("\n1. Add Account");
            System.out.println("2. Perform Month End");
            System.out.println("3. Display All Accounts");
            System.out.println("0. Exit");
            menuChoice = Input.getInt("Choice:", 0, 3);
            System.out.println();
            switch (menuChoice) {
                case 1: badBank.addAccount(); break;
                case 2: badBank.calculateMonthEnd(); break;
                case 3: badBank.displayAllAccounts(); break;
            } // end switch
        } while (menuChoice != 0);
    }
}
```

Complete implementation of `main()`. Do not change.

```
1. Add Account
2. Perform Month End
3. Display All Accounts
0. Exit
Choice: (0-3): 1

1:Savings 2:Checking : (1-2): 2
Account Holder's Name: Bob Bowes
Account Number: (10000-99999): 34343
Opening Balance:(0.01-100000.00): 5000.00
Number of Transactions: (0-100): 13
```

Output before calculating the month end.

```
1. Add Account
2. Perform Month End
3. Display All Accounts
0. Exit
Choice: (0-3): 3
```

```
12121:Jim Smith Balance:100.0 Transactions:12
23232:Joan Rivers Balance:100000.0
34343:Bob Bowes Balance:5000.0 Transactions:13
```

```
1. Add Account
2. Perform Month End Calculations
3. Display All Accounts
0. Exit
Choice: (0-3): 2
```

Output after calculating the month end.

```
1. Add Account
2. Perform Month End
3. Display All Accounts
0. Exit
Enter Choice: (0-3): 3
```

```
12121:Jim Smith Balance:85.0 Transactions:12
23232:Joan Rivers Balance:100995.5
34343:Bob Bowes Balance:4984.25 Transactions:13
```

```
1. Add Account
2. Perform Month End
3. Display All Accounts
0. Exit
Choice: (0-3): 0
```

Memory Map: (4 marks)

Draw a diagram that shows the organization of memory which would result from the sequence shown above. Label your diagram with matching variable names and meaningful labels to convince me that you understand how memory is organized. Include suitable reference-to numbers.

Class Input Method Summary

static boolean	getBoolean (java.lang.String prompt) Retrieves a true / false value from the keyboard
static double	getDouble (java.lang.String prompt) Retrieves a <i>double</i> from the keyboard
static double	getDouble (java.lang.String prompt, double low, double high) Retrieves a <i>double</i> from the keyboard which is guaranteed to be between the specified ranges (inclusive)
static int	getInt (java.lang.String prompt) Retrieves an <i>int</i> from the keyboard.
static int	getInt (java.lang.String prompt, int low, int high) Retrieves an <i>int</i> from the keyboard which is guaranteed to be between the specified ranges (inclusive)
static String	getString (java.lang.String prompt) Retrieves a <i>String</i> from the keyboard where the text is delimited by the <Enter> key, that is the "\n" sequence.