SIT232 – OBJECT ORIENTED DEVELOPMENT

PROJECT 1 DUE 5:00PM, APRIL 29

SUBMISSION INSTRUCTIONS

You need to zip your entire project and submit the zip file in the provided assignment folder by 5:00pm April 29. Note that this is individual work, and thus you cannot share your solution with others.

UNIT LEARNING OUTCOMES

UL1. Apply object-oriented concepts including abstraction, encapsulation, inheritance, and polymorphism.

UL2. Solve programming problems using object-oriented techniques and the C# programming language.

PROJECT DESCRIPTION

You are required to develop a console application to manage customers and accounts using C#. Your application will include the following classes.

1. Customer

This class represents customers and has the following information

1.1. Attributes

- First name: Customer's first name. This information is required[1] and the attribute is a readable[2]/writeable[3] attribute.
- Last name: Customer's last name. This information is required and the attribute is a readable/writeable attribute.
- Address: Customer's address. This information is required and the attribute is a readable/writeable attribute.

[1] Required: this attribute must have a value
[2] The property is equipped with **get**
[3] The property is equipped with **set**

- Date of birth: Customer's date of birth. This information is required and the attribute is a readable/writeable attribute. You should use **DateTime**[4] for the datatype of this attribute. Customer must be at age of 16 or above. For the sake of simplicity, only the year of birth will be considered. For example, given that the current year is 2018, only customers whose year of birth is 2002 or prior can be maintained in the application. You can get the current year via **DateTime.Now**.
- Contact number: Customer's contact number. A contact number must contain exactly 10 digits. However, this attribute is not required and thus can be blank.
- Email: Customer's email. This attribute is not required and thus can be blank.
- List of accounts: Customer's accounts. **Account** class is described in section 2. You could use **List**[5] to maintain the list of accounts. This attribute is a read-only[6] attribute.

1.2. Methods

- Constructor: This constructor receives input parameters including first name, last name, address, date of birth, contact, and email. Note that if customer does not have a contact number or email address, those fields will be set to blank.
- Copy constructor: You need to implement a copy constructor. Please revisit Prac 4.1 for an example of copy constructors.
- Add account: This method adds an account to the list of accounts. This method should receive only an account as input parameter.
- Sum balance: This method returns the total balance of all accounts owned by the customer.
- ToString: This method returns a string including first name, last name, address, date of birth, contact, email, and total balance. Total balance is the sum of the balances of all accounts owned by the customer. The total balance is required to be presented in 1 decimal digit, e.g. 2,510.5 (see Figure 1 below). Please revisit Prac 4.1 for an example of ToString() method.

2. Account

This class represents accounts and has the following information.

2.1. Attributes

- ID: Account's ID. This attribute is required and is read-only. It cannot be set manually but generated automatically. In particular, the first account (in the system) will have ID = 1, accounts opened after that will have ID = 2, 3, etc. **Hint:** ID should be declared as **static** and its datatype should be unsigned int (**uint**) or unsigned long (**ulong**).

[4] DateTime: https://msdn.microsoft.com/en-us/library/system.datetime(v=vs.110).aspx
[5] List: https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx
[6] The property is equipped with **get** only. Please revisit Prac 4.1.

- Opened date: Account's opened date. This information is required and is read-only. The opened date of an account must be on or before the current date (obtained using **DateTime.Now**). **Hint:** Comparison of DateTime variables can be performed using **DateTime.Compare.**
- Closed date: Account's closed date. This attribute does not require a value if the account is still active and is set by the date the account is closed otherwise. This attribute is a read-only attribute. This attribute is set indirectly by calling method "Close" (described in Section 2.2).
- Active: Account's status. This attribute represents account's status that can be active or inactive. This is a read-only attribute. You could use **bool** for the datatype of this attribute. This attribute is set indirectly by calling "Close" method (described in Section 2.2).
- Balance: Account's balance. This attribute requires a value with 0 for default value. Balance must be equal or greater than 0. This is a readable/writable attribute.
- Owner: Account's owner. Account's owner is a Customer object. This attribute is read-only and is given via constructor's parameters.

2.2. Methods

- Constructors: There are two constructors
    - Constructor 1: This constructor receives input including an owner, an opened date, and an initial balance. In this constructor, the opened date and initial balance need to be validated (see description of opened date and balance in section 2.1). When an account is created its "Active" attribute needs to be set to true.
    - Constructor 2: This constructor has less parameters. In particular, it receives input including an owner and an initial balance. In this constructor, the opened date is set by the current date. Note that initial balance also needs to be validated. When an account is created its "Active" attribute needs to be set to true. Constructor 1 should be invoked in Constructor 2. See Prac 4.1 for an example of calling a constructor from another constructor.
- Close: This method closes the account. In the method, "Active" is updated and "Closed date" is set to the current date. Note that Close method is only applied to active accounts.
- Transfer: This method receives input as an account and amount of money and aims to transfer that amount from the current account to the input account. For example, let **a** and **b** be two accounts, **a.Transfer(b, 1000)** will transfer $1,000 from account **a** to account **b**. However, this method acts differently depending on the account types (see more details in Section 3 and 4). Note that Transfer method is only applied to active accounts.

- Calculate interest: This method calculates and results in the interest of the account. This method acts differently depending on the account types (see more details in Section 3 and 4).
- Update balance: This method is called when the balance needs to be updated. We assume that in the end of each month, the balance is re-calculated and updated based on the interest as follow, new balance = old balance + interest. Thus, this method will need to call **Calculate interest** to fulfill its task.
- ToString: This method returns a string including account's ID, opened date, balance, owner's name, and closed date (if the account has been closed). The balance is required to be presented in 1 decimal digit, e.g. 2,510.5 (see Figure 1 below)

There are two types of accounts: Type1Account and Type2Account.

3. Type1Account

This is a sub-class of **Account** and contains the following information.

3.1. Attributes

- Annual interest rate: The interest rate is fixed to 2.0 for all type 1 accounts and is a read-only attribute. **Hint**: this attribute should be **static**.

3.2. Methods

- Constructors: Like **Account** class, this class has two constructors with similar parameters.
- Deposit: This method receives an amount of money as input and adds this amount to the balance. Note that, this operation is only applied to active accounts and the deposit must be greater than 0.
- Withdraw: This method receives an amount of money as input and deducts this amount from the balance. Like "Deposit" method, the withdrawal is only applied to active accounts and the withdrawn money must be positive and not exceed the balance.
- Transfer: This method overrides the **Transfer** method declared in **Account** class, which transfers a given amount of money to a given account. The transferred amount of money must be greater than zero and cannot exceed the balance of the source account.
- Calculate interest: This method overrides the **Calculate interest** method declared in **Account** class. The interest can be calculated using the formula below,

  Interest = (Annual interest rate / 365 / 100) * nDays * Balance

  where nDays is the number of days since the first day of the current month to the current day. If the account is opened after the first day of the current month, then nDays is the number of days since its opened day to the current day. For example, given March 29th, 2018 as the current date and if the account was opened on

February 20th, 2018, then nDays = 29 – 1 = 28. If the account was opened on March 10th, 2018, then nDays = 29 – 10 = 19.

Hint: You could use **Timespan** class and **Subtract** method (of **DateTime**) to calculate difference between DateTime variables. For example, let **d1** and **d2** be two **DateTime** objects, the difference in months between **d1** and **d2** can be calculated as, **TimeSpan span = d1.Subtract(d2)**.

4. Type2Account

4.1. Attributes

- Monthly deposit: This type of account requires its owner to make a deposit monthly. Monthly deposit is the amount money that the owner deposits in the current month. This attribute is a readable/writable attribute. We assume that the owner makes this monthly.
- Annual interest rate: Like **Type1Account**, this interest rate is fixed for all type 2 accounts and is a read-only attribute. However, its value is higher and is set to 3.0.
- Deposit interest rate: This interest rate is fixed for all type 2 accounts. However, it is applied only to the Monthly deposit and is set to 4.0.

4.2. Methods

- Constructors: Like **Account** class, this class has two constructors with similar parameters.
- Transfer: This method overrides the **Transfer** method declared in **Account** class, which transfers a given amount of money to a given account. The transferred amount of money must be greater than zero and cannot exceed the balance of the source account. However, unlike **Type1Account**, a **Type2Account** can only transfer money to a **Type1Account** of the same owner. **Hint**: you could use **GetType()** and **typeof** for check the type of an account object. For example, let **a** be an account, we could check whether **a** is an instance of **Type2Account** by using this condition **a.GetType() == typeof(Type2Account)**.
- Calculate interest: This method overrides the **Calculate interest** method declared in **Account** class. The interest can be calculated using the formula below,

  Interest = (Annual interest rate / 365 / 100) * nDays * Balance
  + (Deposit interest rate / 365 / 100) * nDays * Monthly deposit

  where nDays is computed similarly as in the **Calculate interest** method of **Type1Account.**
- Update balance: This method calls the **UpdateBalance** method in **Account** class and then set the **Monthly deposit** to 0.

5. Main method

You are to implement a Main method to test your classes. The Main method must include the following step (you MUST keep the same order),

a. Creating 3 customers (**c1**, **c2**, and **c3**) with the following information
   c1: <First name: Arley, Last name: Praise, Address: 12 Hay Rd, Date of birth (DD/MM/YYYY): 02/10/1990, Mobile: 0412232116, Email: arleyp@gmail.com>
   c2: <First name: Joseph, Last name: Abot, Address: 4/1 Mandy Pl, Date of birth (DD/MM/YYYY): 11/05/1970, Mobile: 0413221624>
   c3: <First name: Rose, Last name: Magaret, Address: 30 Buxton St, Date of birth (DD/MM/YYYY): 06/07/1980, Email: rmt@yahoo.com>
b. Creating 5 accounts (a1, a2, a3, a4, and a5) with the following information
   a1 (Type1): <Owner: c1, opened date (DD/MM/YYYY): 01/02/2018, initial balance: $100>
   a2 (Type2): <Owner: c1, opened date (DD/MM/YYYY): 15/02/2018, initial balance: $5,000>
   a3 (Type1): <Owner: c2, opened date (DD/MM/YYYY): 20/03/2018>
   a4 (Type2): <Owner: c3, opened date (DD/MM/YYYY): 04/03/2018, initial balance: $2,000>
   a5 (Type2): <Owner: c3, initial balance: $3,000>. Note that when the opened date is not provided, it will be set to the current date.
c. Deposit -$200 to **a1**
d. Transfer $6,000 from **a2** to **a1**
e. Transfer $2,000 from **a2** to **a1**
f. Transfer $1,000 from **a1** to **a2**
g. Set the monthly deposit for **a2** to $200
h. Deposit $5,000 to **a3**
i. Withdraw $6,000 from **a3**
j. Transfer $1,000 from **a3** to **a1**
k. Transfer $500 from **a3** to **a2**
l. Set the monthly deposit for **a4** to $1,000
m. Transfer $100 from **a4** to **a1**
n. Set the monthly deposit for **a5** to $1,500
o. Transfer $500 from **a5** to **a4**
p. Calculate the interests of **a1**, **a2**, **a3**, **a4**, and **a5** and display them on the console window
q. Update balance of **a1**, **a2**, **a3**, **a4**, and **a5**
r. Close **a3** and **a5**
s. Display the information of **a1**, **a2**, **a3**, **a4**, and **a5** by calling ToString() method of these accounts
t. Display the information of **c1**, **c2**, and **c3** by calling ToString() method of these customers

Given the above customers and accounts and assume that the current date is 30/03/2018. Figure 1 shows the expected outputs of your program.

```
Deposit $5000 to a3 --
Withdraw $6000 from a3 -- Withdrawn amount exceed balance!
Transfer $1000 from a3 to a1 --
Transfer $500 from a3 to a2 --
Set the monthly deposit of a4 to $1000 --
Transfer $100 from a4 to a1 -- Cannot transfer to another account of a different customer!
Set the monthly deposit of a5 to $1500 --
Transfer $500 from a5 to a4 -- Cannot transfer to a Type 2 Account!

Calculate interest
Interest of a1: 3.3
Interest of a2: 11.4
Interest of a3: 1.9
Interest of a4: 7.1
Interest of a5: 0.0

Updating balance

Accounts' information
ID: 1     Opened Date: 1/02/2018    Balance: 2,103.3    Owner: Arley   Praise
ID: 2     Opened Date: 15/02/2018   Balance: 4,511.4    Owner: Arley   Praise
ID: 3     Opened Date: 20/03/2018   Balance: 3,501.9    Owner: Joseph  Abot       - Closed on 30/03/2018
ID: 4     Opened Date: 4/03/2018    Balance: 2,007.1    Owner: Rose    Magaret
ID: 5     Opened Date: 30/03/2018   Balance: 3,000.0    Owner: Rose    Magaret    - Closed on 30/03/2018

Customers' information
Name: Arley   Praise     Addres: 12 Hay Rd      DOB: 2/10/1990    Contact: 0412232116   Email: arleyp@gmail.com   Total balance: 6,614.7
Name: Joseph  Abot       Addres: 4/1 Mandy Pl   DOB: 11/05/1970   Contact: 0413221624   Email:                    Total balance: 3,501.9
Name: Rose    Magaret    Addres: 30 Buxton St   DOB: 6/07/1980    Contact:              Email: rmt@yahoo.com      Total balance: 5,007.1
```

Figure 1. An example of expected outputs

NOTE

1) Your program may generate different results depending on the date you run it.
2) You could format the outputs of your program similarly to Figure 1.
3) When a transaction/operation cannot be performed, e.g. withdrawing an amount that exceeds the balance, you could either write an error message (as shown in the 2nd line in Figure 1) or simply skip the transaction/operation.

OTHERS

1) You are to determine appropriate modifiers (public, private, etc.), data types (e.g. string, int, double, etc.) for attributes and inputs/outputs for methods of your classes.
2) Your implementation should reflect object-oriented principles such as abstraction, encapsulation, polymorphism.
3) Make sure that you have sufficient comments to enhance the readability of your code, organise your program reasonably and make it maintainable conveniently (e.g. avoiding hard-code and repeating code, naming variables and methods meaningfully, using static, constants reasonably).
4) Make your outputs in a reasonable and tidy format. You could refer to Figure 1 as an example.
5) If your program cannot be compiled, the maximum score you could get is 50% of the total mark (i.e. 7.5%).

END