Assignment #2 (4%)

Posted on Monday January 30, 2023

Due Date: Friday February 3, 2022, 11:59 PM

Late submission: 10% penalty per day until Monday February 6, 11:59 PM

Submit using the template to the Slate Drop box for Assignment#2 (available 2 days before the deadline-no email submission)

Assignment 2 covers topics of Encapsulation, Inheritance and Polymorphism (Module 1, 2, & 3)

Develop a program that models bank accounts as follow:

A superclass **BankAccount**, two subclasses **SavingsAccount**, **CheckingAccount** and a test program **TestBankAccount**.

Class **BankAccount** has the properties **accountNumber**, **name**, **balance**, **annualInterestRate**, and **dateCreated,** and methods **deposit** and **withdraw** funds.

1. A private **int** data field named **accountNumber** for the account (default **1000**).
2. A private **String** data field named **name** for the account holder's name (default blank).
3. A private **double** data field named **balance** for the account balance (default **0**).
4. A private **Date** data field named **dateCreated** that stores the date when the account was created.
5. A no-arg constructor that creates a default account.
6. A constructor that creates an account with the specified **accountNumber**, **name** (account holder's full name including first name middle initial and surname), and an initial **balance**.
7. The accessor and mutator methods for **accountNumber**, **name, balance.**
8. A method named **withdraw** that withdraws a specified amount from the account (balance –= amount).
9. A method named **deposit** that deposits a specified amount to the account (balance += amount).
10. A **toString()** method that uses String method format to return account information including: **accountNumber**, account holder's **name**, account **balance** (with 2 decimal point), and **dateCreated**.

Two subclasses of **BankAccount** are class **CheckingAccount** and class **SavingsAccount**.

11. **CheckingAccount** has a protected data field type integer for **overdraftLimit** for 6000 but a savings account cannot be overdrawn.
12. **CheckingAccount** has a constructor that invokes superclass constructor and sets accountNumber, name, balance data fields.
13. **SavingsAccount** has a private data field **double** named **annualInterestRate** that stores the current interest rate (default **0**). Assume that all SavingsAccounts have the same interest rate, therefore the data field should be declared static double.
14. **SavingsAccount** has a method named **getMonthlyInterestRate()** that returns the monthly interest rate.

15. **SavingsAccount** has a method named **getMonthlyInterest()** that returns the monthly interest amount.
16. **HINT:** The method **getMonthlyInterest()** is to return monthly interest amount on balance, not the interest rate. Monthly interest is calculated using: **getBalance() * monthlyInterestRate**.
17. **monthlyInterestRate** is: **annualInterestRate / 1200**.
18. Note **annualInterestRate** is a percentage, for example 4.5%. For this reason, it should be divided by 100 and by 12 for **monthlyInterestRate**.
19. Both account types **CheckingAccount** and **SavingsAccount** override **toString()** method defined in the superclass and append to that the statement: Account type Checking or Savings.

20. Driver Class **TestBankAccount** creates two bank accounts; an instance of **CheckingAccount** named **checking** and an instance of **SavingsAccount** named **savings**. with the following specifications:
21. Test program is the driver class **TestBankAccount** that creates two objects:
22. A **CheckingAccount** object named **checking** with the AccountNumber of 1001, account holder's name: John P Smith and a **balance** of 20,000.
23. A **SavingsAccount** object named **savings** with the AccountNumber of 1002, account holder's name: Janet E Holland and a **balance** of $10,000.
24. A withdrawal of 2,500 is made from the checking account
25. A deposit of 3000 is made to savings account.
26. Annual interest rate of savings account is set to 4.5%.
27. Monthly interest amount of the savings account is displayed.
28. Method accountInformation is called to display details of checking account.
29. Method accountInformation is called to display details of savings account.
30. **TestBankAccount** has a generic method **accountInformation** that has an argument of type supertype **BankAccount** as defined below. When the method is invoked and receives subtypes **checking** or **savings** it displays account information for **checking** or **savings** accounts using polymorphism (**Hint**: Refer to Module 3: Chapter 11- slides 39-45, ICE# 11.5 and ICE#11.6).
31. **accountInformation** is defined:

```
public static void accountInformation (BankAccount account) {
    System.out.println(account);
  }
```

Sample Program Execution and output:

```
accountNumber:  1001
Account holder's Name:  John P Smith
AccountBalance:  17500.00
Date Account Created:  Mon Jan 30 21:09:54 EST 2023
Account type: Checkings

accountNumber:  1002
Account holder's Name:  Janet E Holand
AccountBalance:  13000.00
Date Account Created:  Mon Jan 30 21:09:54 EST 2023
Account type: Savings

Monthly interest amount of savings account is: 48.75
```

Deliverable:

Use the submission template in MSWord and include the following sections for evaluation:

1. UML
2. source codes for 4 classes in the following order (Code must include start-up comment)
   1. *BankAccount*, 2. *CheckingAccount*, 3. *SavingsAccount*, 4. *TestBankAccount*
3. Screenshot of the program output

Evaluation:

1. UML 1 marks
   including individual classes and their relationships
2. Code & Functionality: 2 Marks
   Follow submission standards, coding standard and quality, use of encapsulation, inheritance, and polymorphism.
3. Program output: 1 Marks
   Total mark: 4

Penalty marks:

1. Penalty mark for Late submission 10% per day up to 3 days then mark of 0
2. Penalty mark for failing to follow submission standard 25% (1 mark)
3. Penalty mark for failing startup comment 25% (1 mark)
4. submissions is electronically examined for similarity and will not be accepted in cases of high similarity.