Seneca College
Applied Arts & Technology
SCHOOL OF COMPUTER STUDIES

JAC444 **Submission date:** **June 07, 2021**

Workshop 2

Description:
The following workshop lets you practice basic java coding techniques, creating classes, methods, inheritance, polymorphism.

Task 1:

Create an inheritance hierarchy that a bank might use to represent customers' bank accounts.

- All customers at this bank can deposit (i.e., credit) money into their accounts.
- All customers at this bank can withdraw (i.e., debit) money from their accounts.
- More specific types of accounts also exist.
    - Savings accounts: Earn interest on the money they hold.
    - Checking accounts: Charge a fee per transaction.

Account Class

- Create class Account that should include one private instance variable of type double to represent the account balance.
- The class should provide a constructor that receives an initial balance and uses it to initialize the instance variable with a public property.
- The property should validate the initial balance to ensure that it is greater than or equal to 0.0; if not, display an error message (Explore System.err.println).
- The class should provide two public methods.
    - Method Credit should add an amount to the current balance.
    - Method Debit should withdraw money from the Account and ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged, and the method should display the message "Debit amount exceeded account balance."
- The class should also provide a get accessor in property Balance that returns the current balance.

SavingsAccount Class

- Class SavingsAccount should inherit the functionality of an Account, but also include a double instance variable indicating the interest rate (percentage) assigned to the Account.
- SavingsAccount's constructor should receive the initial balance, as well as an initial value for the interest rate.
- SavingsAccount should provide public method CalculateInterest that returns a double indicating the amount of interest earned by an account.

- Method CalculateInterest should determine this amount by multiplying the interest rate by the account balance.

[*Note:* SavingsAccount should inherit methods Credit and Debit without redefining them.]

CheckingAccount Class

- Class CheckingAccount should inherit from base class Account and include a double instance variable that represents the fee charged per transaction.
- CheckingAccount's constructor should receive the initial balance, as well as a parameter indicating a fee amount.
- Class CheckingAccount should redefine methods Credit and Debit so that they subtract the fee from the account balance whenever either transaction is performed successfully.
- CheckingAccount's versions of these methods should invoke the base-class Account version to perform the updates to an account balance.
- CheckingAccount's Debit method should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance).

After defining the classes in this hierarchy, write an app that creates objects of each class and tests their methods. Add interest to the SavingsAccount object by first invoking its CalculateInterest method, then passing the returned interest amount to the object's Credit method. The tester class should be a different class.

Task 2:

Let us update the **Task1** and create an array of Account references to SavingsAccount and CheckingAccount objects. For each Account in the array, allow the user to specify an amount of money to withdraw from the Account using method Debit and an amount of money to deposit into the Account using method Credit. As you process each Account, determine its type. If an Account is a SavingsAccount, calculate the amount of interest owed to the Account using method CalculateInterest, then add the interest to the account balance using method Credit. After processing an Account, display the updated account balance obtained by using baseclass property Balance.

Continue to the next page…

Workshop Header

/***
Workshop #
Course:*<subject type> - Semester*
Last Name:*<student last name>*
First Name:*<student first name>*
ID:*<student ID>*
Section:*<section name>*
This assignment represents my own work in accordance with Seneca Academic Policy.
Signature
Date:*<submission date>*
***/

Code Submission Criteria:

Please note that you should have:

- Appropriate indentation.
- Proper file structure
- Follow java naming convention.
- Document all the classes properly
- Do Not have any debug/ useless code and/ or files in the assignment

Deliverables and Important Notes:

All these deliverables are supposed to be uploaded on the blackboard once done.

- You are supposed to submit a video/ detailed document of your running solution. **(40%)**
  - Things to be considered if you are uploading a video.
    - A reasonable length of video should be posted. 5%
    - Your video should show the running solution with different inputs. 5%
    - In a minute discuss the design of your solution. 10%
    - In a minute (max 3 to 5) discuss the important functions/ methods in your solution. 20%
    - If you are using in your solution concepts that are not discussed in the class then in a minute or two explain,
      - What is that concept?
      - Why did you use it?
      - How does it benefit your solution?
  - Things to be considered if you are uploaded the detailed document.
    - Should include **screen shots** of your output. 5%
    - Underneath each screen shot explain in 2 to 4 lines what is happening.

10%

- In 3 to 5 lines explain the design logic of your program. 10%
- Screen shots of important methods/ functions in your solution and discuss them underneath each screen shot (3 to 6 lines). 15%
- If you are using in your solution concepts that are not discussed in the class then in a minute or two explain,
    - What is that concept?
    - Why did you use it?
    - How does it benefit your solution?

- A word/ text file which will reflect on learning of your concepts in this workshop. (Also include the instructions on how to run your code, if required) **(30%)**
    - Should state your Full name and Id on the top of the file and save the file with your last name and id, like Ali_123456.txt
- Submission of working code. **(30%)**
    - Make sure your follow the "**Code Submission Criteria"** mentioned above.
    - You should zip your whole working project to a file named after your Last Name followed by the first 3 digits of your student ID. For example, **Ali123.**zip.
- Your marks will be deducted according to what is missing from the above-mentioned submission details.
- Late submissions would result in additional 10% penalties for each day or part of it.
- Remember that you are encouraged to talk to each other, to the instructor, or to anyone else about any of the assignments, but the final solution may not be copied from any source.