Vanier College
Computer Science Department

Programming 2

Assignment 2

Due Date: By 11:59pm Sunday March 16, 2025
Evaluation: 5% of final mark (see marking rubric at the end of handout)
Late Submission: none accepted
Demo: Students may need to demo their assignment if time allows. If a demo is required, submissions without a demo will be considered invalid. If the correct solution has been submitted but the student cannot properly present it, points may be deducted, or a zero may be assigned.

General Guidelines When Writing Programs:

Include the following comments at the top of your source codes :

```
// -------------------------------------------------------
// Assignment (include number)
// Written by: (include your name and student id)
// For "Programming 2" Section (include number) - Winter 2025
// -------------------------------------------------------
```

- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program.
- Display a welcome message.
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy to read format.
- End your program with a closing message so that the user knows that the program has terminated.

What to Submit:

- Make **one ZIP** file that includes all of the **.java files.**
- Submit the ZIP file via Omnivox.
  - Do not use the RAR (or some other) format!

Assignments not submitted to the correct location or not in the requested format will not be graded.

In this assignment, you will develop a Java application that models an Online Banking System. The system should support customers who can have different types of bank accounts (Savings and Checking), perform transactions, and handle insufficient balance scenarios using exception handling.

The program should use OOP concepts such as inheritance, abstract classes, polymorphism ,encapsulation, as well as exception handling (custom & built-in).

System Requirements

1. Class Structure and Relationships

- **Abstract Class: BankAccount**
  - Represents a generic bank account.
  - Attributes:
    - accountNumber: Unique identifier for the account.
    - balance: The current balance in the account.
  - Methods:
    - deposit(double amount): Deposits an amount to the account.
    - withdraw(double amount) throws InsufficientFundsException: Withdraws an amount from the account (throws exception if balance is insufficient).
    - displayAccountInfo(): Abstract method to be implemented in subclasses.

- **Subclass: SavingsAccount (inherits from BankAccount)**
  - Additional attribute:
    - interestRate: Interest rate applied to the balance.
  - Methods:
    - applyInterest(): Applies interest to the balance.
    - Overrides displayAccountInfo().

- **Subclass: CheckingAccount (inherits from BankAccount)**
  - Additional attribute:
    - overdraftLimit: Maximum overdraft allowed.
  - Methods:
    - Overrides withdraw() to allow overdrafts up to a limit.
    - Overrides displayAccountInfo().

- **Class: Customer**
  - Represents a customer in the banking system.
  - Attributes:
    - customerID: Unique identifier for the customer.
    - name: Name of the customer.

        - accounts: A list of BankAccount objects.

    - Methods:
        - addAccount(BankAccount account): Adds an account to the customer.
        - getAccount(String accountNumber): Retrieves a customer's account by number.
        - displayCustomerInfo(): Displays customer details including accounts.

- **Class: Bank**
    - Manages customers and their bank accounts.
    - Attributes:
        - customers: A list of Customer objects.
    - Methods:
        - addCustomer(Customer customer): Adds a customer to the bank.
        - findCustomerByID(String customerID): Finds a customer by their ID.

2. Exception Handling

- **Custom Exception: InsufficientFundsException**
    - Thrown when a withdrawal is attempted but the balance is insufficient.

3. Main Method with Menu-Driven Interaction

The main menu should allow users to:

1. Register a new customer.
2. Add a new bank account to a customer.
3. Display customer information.
4. Perform deposit transactions.
5. Perform withdrawals (handling overdrafts and exceptions).
6. Exit the program.

4. Sample Execution (User Input & Output)

Menu Display

```
===== Online Banking System =====
1. Register a new customer
2. Add an account to a customer
3. Display customer information
4. Deposit money
5. Withdraw money
6. Exit
Enter your choice:
```

Scenario 1: Register a New Customer

User Input:

```
Enter customer ID: C001
Enter customer name: Alice
```

Output:

```
Customer registered successfully!
```

Scenario 2: Add an Account (Savings)

User Input:

```
Enter customer ID: C001
Enter account type (1 for Savings, 2 for Checking): 1
Enter account number: 12345
Enter initial balance: 500
Enter interest rate: 2.5
```

Output:

```
Account added successfully!
```

Scenario 3: Add an Account (Checking)

User Input:

```
Enter customer ID: C001
Enter account type (1 for Savings, 2 for Checking): 2
Enter account number: 67890
Enter initial balance: 300
Enter overdraft limit: 100
```

Output:

```
Account added successfully!
```

Scenario 4: Display Customer Information

User Input:

```
Enter customer ID: C001
```

Output:

```
Customer: Alice (ID: C001)
Accounts:
- Savings Account (12345): Balance = $500.0, Interest Rate = 2.5%
- Checking Account (67890): Balance = $300.0, Overdraft Limit = $100.0
```

Scenario 5: Deposit Money

User Input:

```
Enter customer ID: C001
Enter account number: 12345
Enter deposit amount: 200
```

Output:

```
Deposited $200.0 into Savings Account (12345)
New Balance: $700.0
```

Scenario 6: Withdraw Money (Valid Transaction)

User Input:

```
Enter customer ID: C001

Enter account number: 67890

Enter withdrawal amount: 350
```

Output:

```
Withdrawn $350.0 from Checking Account (67890)

New Balance: -$50.0 (Within Overdraft Limit)
```

Scenario 7: Withdraw Money (Insufficient Funds - Exception Handling)

User Input:

```
Enter customer ID: C001

Enter account number: 12345

Enter withdrawal amount: 800
```

Output (Exception Thrown):

```
Error: Insufficient funds in Savings Account (12345)! Available balance:
$700.0
```

Evaluation Criteria for Assignment 2 / Demo (100 points)

Source Code		
Comments (11 pts.)		
Description of the program (authors, date, purpose)	2	pt.
Description of variables and constants	6	pt.
Description of the algorithm	3	pt.
Programming Style (4 pts.)		
Use of significant names for identifiers	1	pt.
Indentation and readability	2	pt.
Welcome Banner/Closing message	1	pt.
Classes Design, OOP Concepts, Exception Handling (50 pts.)		
Implementation of OOP (Inheritance, Abstraction, Polymorphism)	20	pts.
Correct Use of Exception Handling (Custom & Built-in)	20	
Proper use of required concepts	10	pts.
Main method (35pts.)		
Functional Menu System	10	pts.
Format/clarity/completeness/accuracy of output	15	
Proper use of required concepts	10	pts.
TOTAL	**100**	**pts.**