CSE1322L Assignment 9
Exception Handling

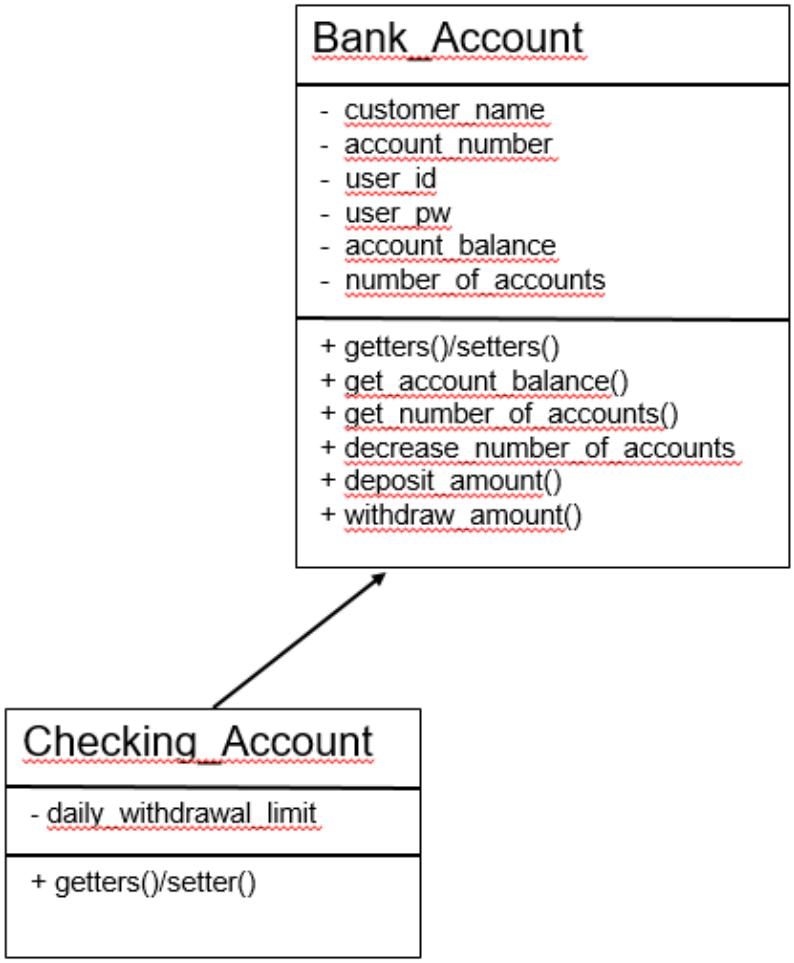
Overview

Topics

- Exception Handling
  (try, catch, throw(s), and user-defined exception subclasses)
- Inheritance / Polymorphism
- List (C#) / ArrayList (Java)
- Static variables/methods

Description

For this assignment, you will code a basic bank-account management system, which is depicted in the following UML (unified modeling language) diagram:



Although the above UML diagram depicts only one type of bank account (checking), the system can include several other account types, i.e. savings, credit card, money market, and certificate of deposit (CD).

Your implementation should include the following:

1. Create a List (C#) / ArrayList (Java) which can contain **all types of bank accounts**
2. Add new accounts to the List (C#) / ArrayList (Java)
3. Delete accounts from the List (C#) / ArrayList (Java)
4. Search the List (C#) / ArrayList (Java) to find a given customer's account
5. Update a given customer's account within the List (C#) / ArrayList (Java), i.e. make deposits and withdrawals
6. Print the account balance (and selected information) of a given customer's account

While performing the above actions, your implementation should use exception-handling functionality (try/catch, throw(s), and user-defined exception subclasses) to address the following errors/exceptions:

1. Invalid password format - passwords must be at least eight characters long and contain an at least one asterisk character ('*')
2. Negative dollar amounts – all dollar amounts must be positive
3. Insufficient funds – customers cannot withdraw more money than they have in their accounts
4. Customer account not found – no customer account exists that matches the provided user id and password

Your task

Define the following two Account classes:

1) Bank_Account Class
    a) Must have the following attributes:
        - Name: string, private, customer full name
        - Account id: int, private, unique account id (initialized via Number of accounts attribute)
        - Number of accounts: int, private, static, initialized with zero (0) and incremented each time an object is created
        - User id: string, private, customer defined login id
        - User password: string, private, customer defined login password

- Account balance: double, private, customer account balance

b) Must have an overloaded constructor which:
   - takes in a parameter of type string and assigns it to the Name attribute
   - takes in a parameter of type string and assigns it to the User id attribute
   - takes in a parameter of type string and assigns it to the User password attribute
   - assigns the value from Number of accounts to the Account id attribute (i.e. the creation of each object increments the Number of accounts attribute, which is then assigned to the Account id attribute)

c) Must have getter/setter methods for the Name, User id, User password, and Account balance attributes

d) Must have getter methods for the Account id and Number of accounts attributes

e) Must have a static method, which returns the value of the static attribute Number of accounts

f) Must have a static method, which returns no value, takes in no parameters, and decrements the static attribute Number of Accounts by one (1)

2) Checking_Account Class
   a) Must inherit Bank_Account class
   b) Must have the following attribute:
      - Daily withdrawal limit: double, private, the total amount of money a customer can withdraw within a day
   c) Must have an overloaded constructor which:
      - takes in a string parameter, Name
      - takes in a string parameter, User id
      - takes in a string parameter, User password
      - calls the parent/base overloaded constructor passing parameters Name, User id, User password
      - assigns a default value of 300.00 to the Daily withdrawal limit attribute
   d) Must have getter/setter methods for the Daily withdrawal limit attribute

Define the following four Exception classes:

I. InvalidPasswordFormatException – throw and catch this exception if an invalid password format is entered (note: passwords must be at least eight (8) characters long and must contain at least one asterisk (‘*’) character

II. NegativeDollarAmountException - throw and catch this exception if a negative dollar amount is entered

III. InsufficientFundsException - throw and catch this exception if the withdrawal amount is greater than the account balance

IV. CustomerAccountNotFoundException - throw and catch this exception if, when searching the Bank_Account List (C#) / ArrayList (Java), no account is found with the provided User id and User password

Define the following Driver class:

Your driver class should contain the following functionality:

1) Process Main Menu

   Use a loop to prompt the user with the following Main Menu and read in the user’s response:

   1 – Create An Account
   2 – Delete An Account
   3 – Make An Account Deposit
   4 – Make An Account Withdrawal
   5 – Check An Account Balance
   6 – Exit

   Enter Choice:

   - If the user enters 1, execute the functionality to create a new checking account
   - If the user enters 2, execute the functionality to delete a checking account
   - If the user enters 3, execute the functionality to make a deposit
   - If the user enters 4, execute the functionality to make a withdrawal
   - If the user enters 5, execute the functionality to check an account balance

- If the user enters 6, terminate the program
- If the user enters any character other than a 1, 2, 3, 4, 5, or 6, displays the following error message: Error: Please enter a valid choice (1 thru 6), and allows the user to reenter a valid choice.
- Hint: In Java, you may want to make Scanner a static object in the main class (i.e. *static Scanner input = new Scanner(System.in)*). This allows you to read in input from the console from inside you user-defined methods.

2) Create an account:
   - Prompt the user to enter the customer's name, user id, and user password
   - Ensure the user password meets the specified format (i.e. is at least eight (8) characters long and contains at least one asterisk ('*') character)
   - If the password does not meet the specified format, throw a InvalidPasswordFormatException error with an appropriate message and return to the Main Menu
   - If the password does meet the specified format, create a **Checking_Account** object, add it to the **Bank_Account** List (C#) / ArrayList (Java), and return to the Main Menu

3) Delete an account:
   - Prompt the user to enter the customer's user id and user password
   - Search the Bank_Account List (C#) / ArrayList (Java), using the customer's user id and user password, to ensure the customer's account exists
   - If the customer's account does not exist, throw a CustomerAccountNotFoundException error with an appropriate message and return to the Main Menu
   - If the customer's account does exist, remove the customer's account from the **Bank_Account** List (C#) / ArrayList (Java), reduce the number of accounts attribute by one, and return to the Main Menu

4) Make an account deposit:
   - Prompt the user to enter the customer's user id and user password
   - Search the Bank_Account List (C#) / ArrayList (Java), using the customer's user id and user password, to ensure the customer's account exists
   - If the customer's account does not exist, throw a CustomerAccountNotFoundException error with an appropriate message, and return to the Main Menu
   - If the customer's account does exist, prompt the user for a dollar amount

- If the dollar amount is not a positive amount, throw a NegativeDollarAmountException error with an appropriate message, and return to the Main Menu
- If the dollar amount is a positive amount, add the amount to the customer's account, and return to the Main Menu

5) Make an account withdrawal:
   - Prompt the user to enter the customer's user id and user password
   - Search the Bank_Account List (C#) / ArrayList (Java), using the customer's user id and user password, to ensure the customer's account exists
   - If the customer's account does not exist, throw a CustomerAccountNotFoundException error with an appropriate message and return to the Main Menu
   - If the customer's account does exist, prompt the user for a dollar amount
   - If the dollar amount is not a positive amount, throw a NegativeDollarAmountException error with an appropriate message and return to the Main Menu
   - If the dollar amount is a positive amount, check to ensure that the dollar amount entered is less than the account balance.
   - If the dollar amount entered is not less than the account balance, throw an InsufficientFundsException error with an appropriate message and return to the Main Menu
   - If the dollar amount entered is less than the account balance, deduct the dollar amount from the account balance and return to the Main Menu

6) Check an account balance:
   - Prompt the user to enter the customer's user id and user password
   - Search the Bank_Account List (C#) / ArrayList (Java), using the customer's user id and user password, to ensure the customer's account exists
   - If the customer's account does not exist, throw a CustomerAccountNotFoundException error with an appropriate message and return to the Main Menu
   - If the customer's account does exist, print the customer's name, account number, and account balance.
   - Check to see if the account is a checking account (hint: this involves polymorphism and object casting)
   - If the account is a checking account, print the label "Account Type: Checking", print the daily withdrawal amount, and return to the Main Menu

Sample Output:

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 9

Error: Please enter a valid choice (1 thru 6)

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 1

Enter Customer Name: Dexter Howard
Enter User ID: dexter001
Enter User Password: password*

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 5

Enter User ID: dexter001
Enter User Password: password*

Customer Name: Dexter Howard
Account Number: 1

Account Balance: 0
Account Type: Checking
Account Daily Withdrawal Limit: 300.00

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 3

Enter User ID: dexter001
Enter User Password: password*
Enter Amount: 1000.00

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 4

Enter User ID: dexter001
Enter User Password: password*
Enter Amount: 200.00

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance

Enter Choice: 5

Enter User ID: dexter001
Enter User Password: password*

Customer Name: Dexter Howard
Account Number: 1
Account Balance: 800.00
Account Type: Checking
Account Daily Withdrawal Limit: 300.00

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 1

Enter Customer Name: John Doe
Enter User ID: john2
Enter User Password: badpassword

Error: Must Enter a Valid Password
InvalidPasswordFormatException: Invalid Password Format

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 2

Enter User ID: dexter001
Enter User Password: badpassword*

Error: Must Enter a Valid User ID and Password
CustomerAccountNotFoundException: Customer Account Not Found

1 – Create An Account
2 – Delete An Account

3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 2

Enter User ID: baduserid
Enter User Password: password*

Error: Must Enter a Valid User ID and Password
CustomerAccountNotFoundException: Customer Account Not Found

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 3

Enter User ID: dexter001
Enter User Password: password*
Enter Amount: -500.00

Error: Must Enter a Positive Dollar Amount
NegativeDollarAmountException: Negative Dollar Amount

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 4

Enter User ID: dexter001
Enter User Password: password*

Enter Amount: 6000.00

Error: Must Withdraw an Amount Less Than Your Balance
InsufficientFundsException: Insufficient Funds

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 2

Enter User ID: dexter001
Enter User Password: password*

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice: 5

Enter User ID: dexter001
Enter User Password: password*

Error: Must Enter a Valid User ID and Password
CustomerAccountNotFoundException: Customer Account Not Found

1 – Create An Account
2 – Delete An Account
3 – Make An Account Deposit
4 – Make An Account Withdrawal
5 – Check An Account Balance
6 – Exit

Enter Choice:

Submitting your answer:

Please follow the posted submission guidelines here:
https://ccse.kennesaw.edu/fye/submissionguidelines.php

Ensure you submit before the deadline listed on the lab schedule for CSE1322L here:
https://ccse.kennesaw.edu/fye/courseschedules.php

Rubric:

- Bank_Account Class (12 points total)
    - Includes all attributes, with correct data types and modifiers (i.e. public, private, static) (2 points)
    - Includes overloaded constructor, which takes in parameter(s) and initializes attributes (2 points)
    - Includes getter/setter methods for each attribute (2 points)
    - Includes decreaseNumberOfAccounts methods (2 points)
    - Includes deposit methods (2 points)
    - Includes withdraw methods (2 points)

- Checking_Account Class (6 points total)
    - Inherits Employee class (2 points)
    - Includes all attributes, with correct data types and modifiers (i.e. public, private, static) (1 points)
    - Includes overloaded constructor, invokes parent/base constructor (2 points)
    - Includes getter/setter methods (1 points)

- Exception Classes (16 points total)
    - Defines InvalidPasswordFormatException class (4 points)
    - Defines NegativeDollarAmountException class (4 points)
    - Defines InsufficientFundsException class (4 points)
    - Defines CustomerAccountNotFoundException class (4 points)

- Driver program (66 points total)
    - Creates ArrayList (Java) / List (C#) of Faculty objects (3 points)

    - Presents menu and reads user input (total 9 points)
        - prompts user with correct menu and reads in user choice (1 point)
        - provides error message for invalid user input (1 points)
        - allows user to reenter choice in case of invalid user input (2 points)

- includes try/catch statements to process exceptions (5 points)

- Includes create account functionality (total 9 points)
  - prompt user to enter info (1 points)
  - validate user password (3 points)
  - throw InvalidPasswordFormatException (2 points)
  - create Checking_Account object add it to **Bank_Account** list (2 points)
  - return to the Main Menu as appropriate (1 points)

- Includes delete account functionality (total 9 points)
  - prompt user to enter info (1 points)
  - search list and locate customer account (2 points)
  - throw CustomerAccountNotFoundException (2 points)
  - remove object from **Bank_Account** list (2 points)
  - reduce the number of accounts attribute by one (1 points)
  - return to the Main Menu as appropriate (1 points)

- Includes make account deposit functionality (total 11 points)
  - prompt user to enter info (1 points)
  - search list and locate customer account (2 points)
  - throw CustomerAccountNotFoundException (2 points)
  - prompt the user for a dollar amount (1 points)
  - throw NegativeDollarAmountException (2 points)
  - add amount to the customer's account (2 points)
  - return to the Main Menu as appropriate (1 points)

- Includes make account withdrawal functionality (total 13 points)
  - prompt user to enter info (1 points)
  - search list and locate customer account (2 points)
  - throw CustomerAccountNotFoundException (2 points)
  - prompt the user for a dollar amount (1 points)
  - throw NegativeDollarAmountException (2 points)
  - throw InsufficentFundsException (2 points)
  - deduct amount from customer's account (2 points)
  - return to the Main Menu as appropriate (1 points)

- Includes check account balance functionality (total 12 points)
  - prompt user to enter info (1 points)
  - search list and locate customer account (2 points)
  - throw CustomerAccountNotFoundException (2 points)

- print account information (2 points)
- check for “checking” account object (2 points)
- print daily withdrawal amount (2 points)
- return to the Main Menu as appropriate (1 points)