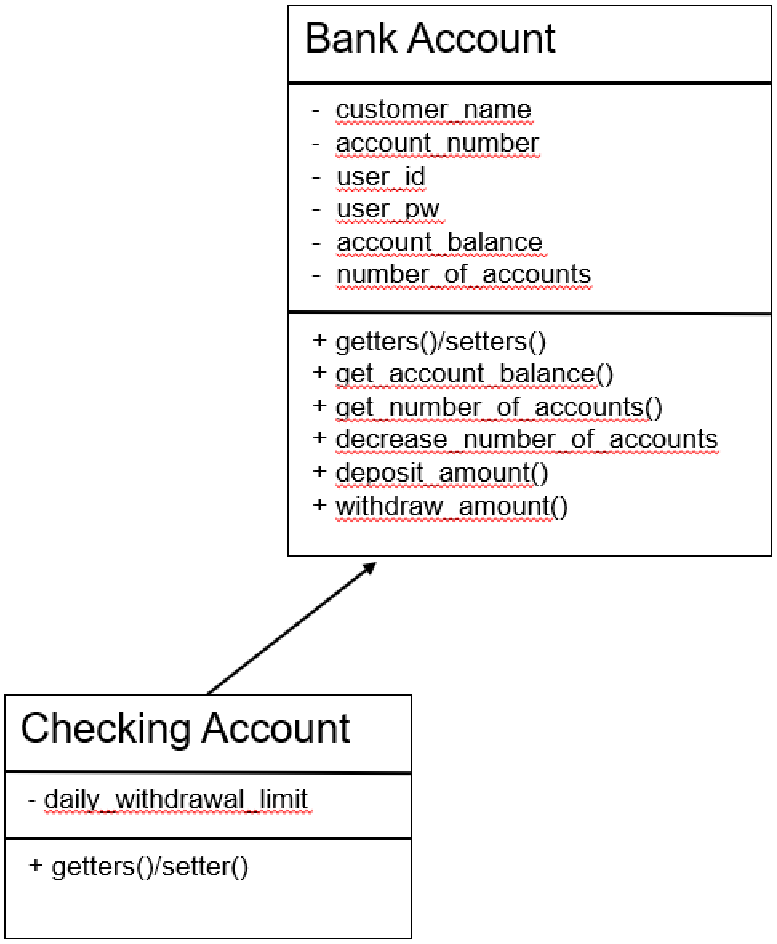
Group Assignment 3

Background

Topics
- Exception Handling (try, catch, throw(s), and user-defined exception subclasses)
- Inheritance / Polymorphism
- ArrayList
- Static variables
- Static 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:



Unified Modeling Language (UML) Diagram

Group Assignment 3

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 an ArrayList which can contain **all types of bank accounts**
2. Add new accounts to the ArrayList
3. Delete accounts from the ArrayList
4. Search the ArrayList to find a given customer's account
5. Update a given customer's account within the ArrayList, 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 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

Instructions

Define the following two 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 (Hint: increment the static attribute first and then assign its value to the account id attribute)
        - 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)

g) Must have a deposit_amount method, which returns no value, takes in a parameter of type double, and adds the parameter value to the account balance.

h) Must have a withdraw_amount method, which returns no value, takes in a parameter of type double, and substracts the parameter value from the account balance.

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

Group Assignment 3

<u>Define the following four Exception classes:</u>

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 ArrayList, no account is found with the provided User id and User password

Note: The fours exception classes above can be stored in their own file, i.e., BankExceptions.class

<u>Define the following Driver class:</u>

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 ArrayList, 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 ArrayList 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 ArrayList, 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 ArrayList 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 ArrayList 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 ArrayList 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

Submission Instructions

All four classes/subclasses below must be created, used, and submitted for a grade. Your .java files should be saved as .txt files and uploaded to the D2L dropbox before the deadline.

Group Assignment 3

1. BankAccount.txt
2. Checking.txt
3. BankExceptions.txt
4. BankAccountTest.txt

Note: points will be deducted if all four files are not submitted or if multiple files are combined in one document.

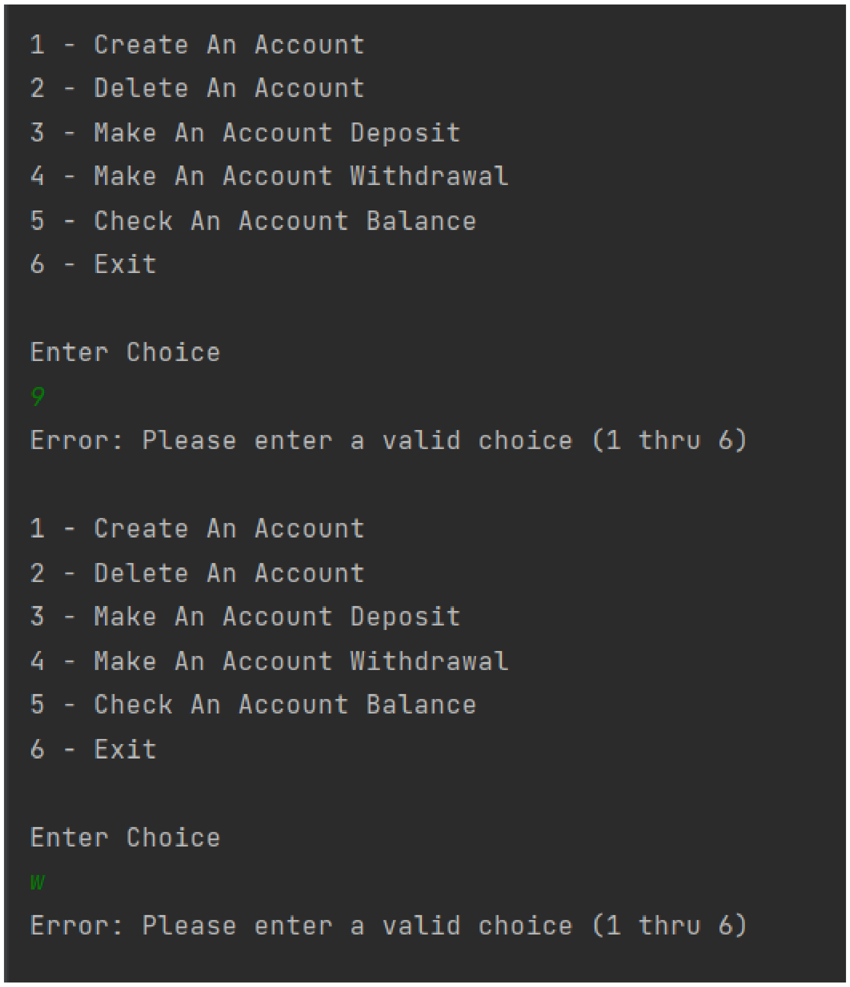
<u>Grading Criteria</u>

- Successfully compiles (5%)
- Indentation/spacing, comments, and naming (5%)
- Bank Account Class (12%)
- Checking Account Class (6%)
- Exception Classes (15%)
- Creates ArrayList of Faculty objects (2%)
- Presents menu and reads user input (7%)
- Includes create account functionality (8%)
- Includes delete account functionality (8%)
- Includes make account deposit functionality (10%)
- Includes make account withdrawal functionality (11%)
- Includes check account balance functionality (11%)

Group Assignment 3

Example Screenshots

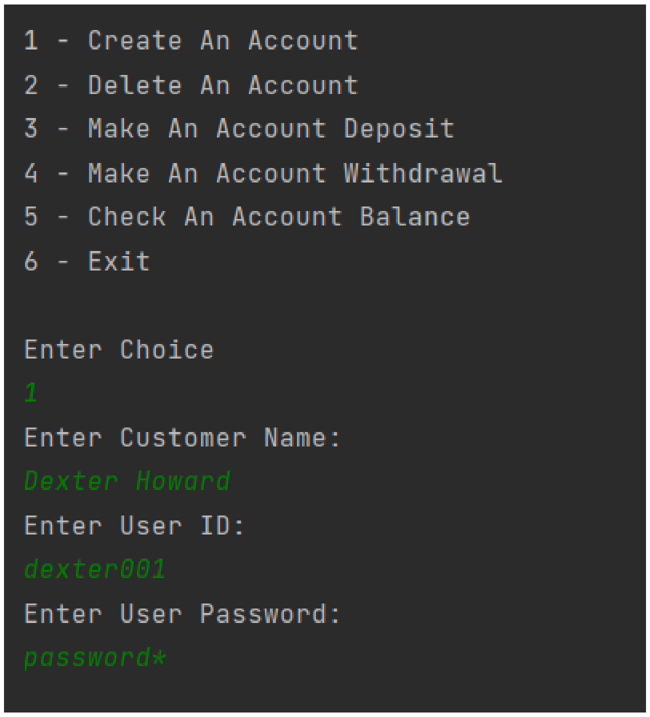
Screenshot 1 – Enter invalid numeric and non-numeric data



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

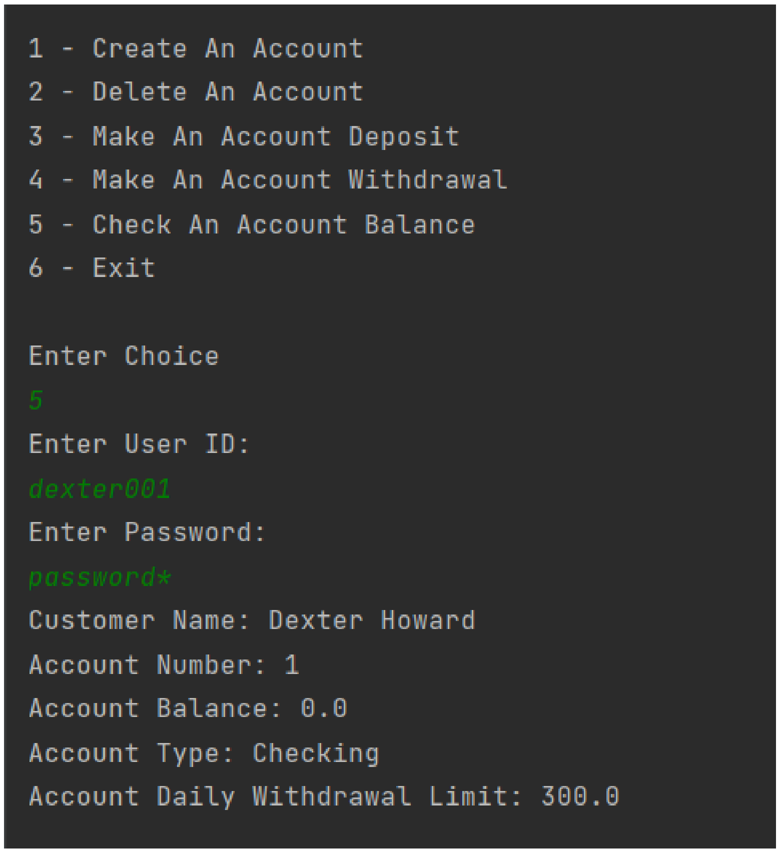
Group Assignment 3

Screenshot 2 – Create an account



```
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*
```

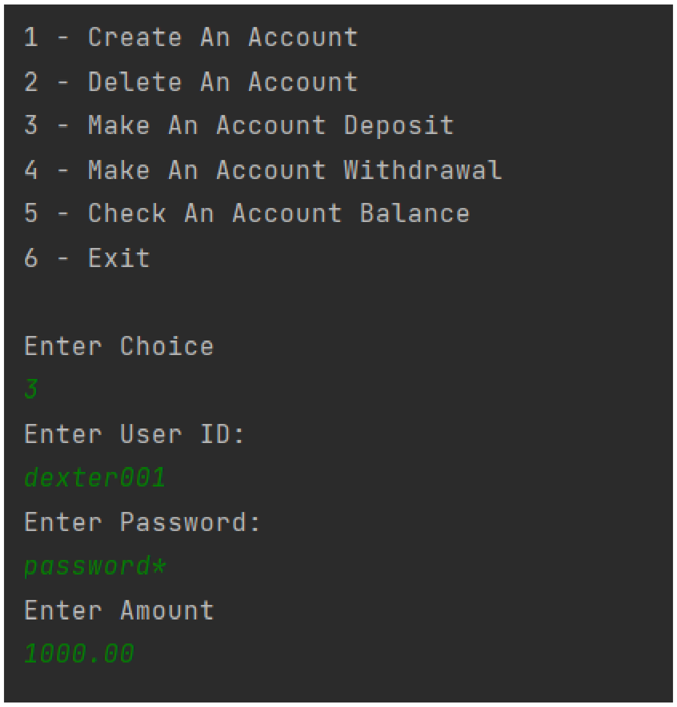
Screenshot 3 – Check an account balance



```
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 Password:
password*
Customer Name: Dexter Howard
Account Number: 1
Account Balance: 0.0
Account Type: Checking
Account Daily Withdrawal Limit: 300.0
```

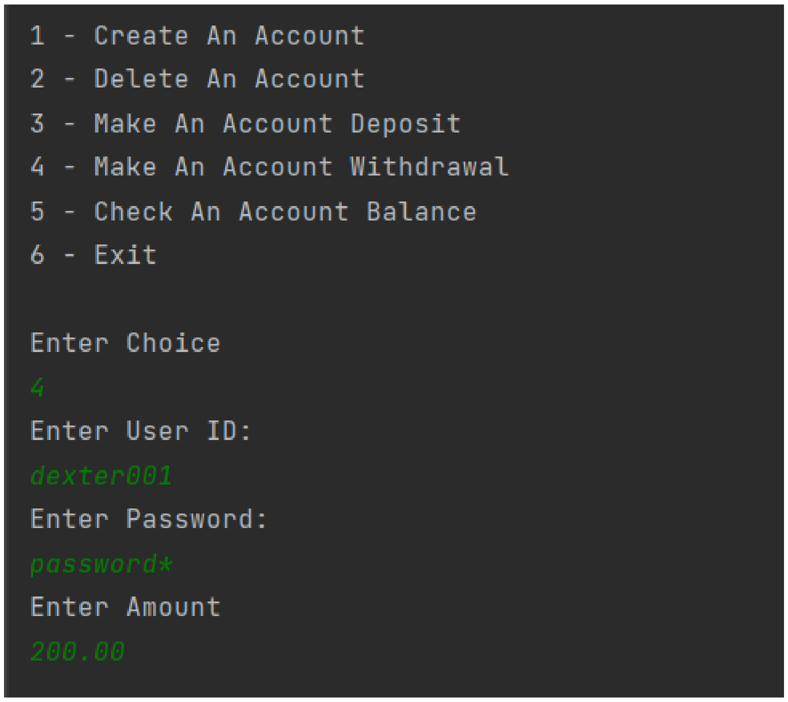
Group Assignment 3

Screenshot 4 – Make an account deposit



```
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 Password:
password*
Enter Amount
1000.00
```

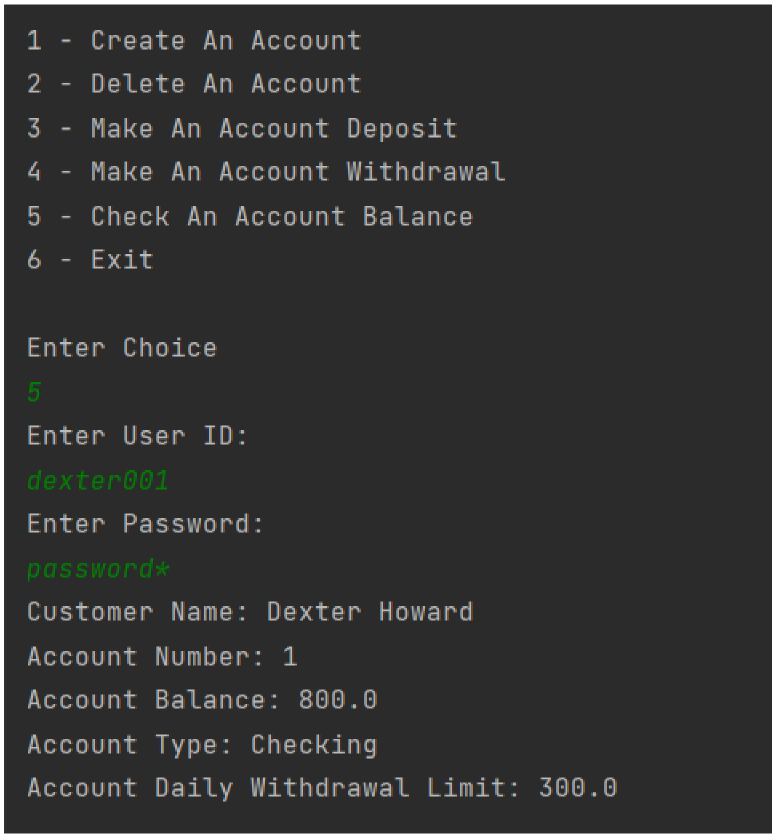
Screenshot 5 – Make an account withdrawal



```
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 Password:
password*
Enter Amount
200.00
```

Group Assignment 3

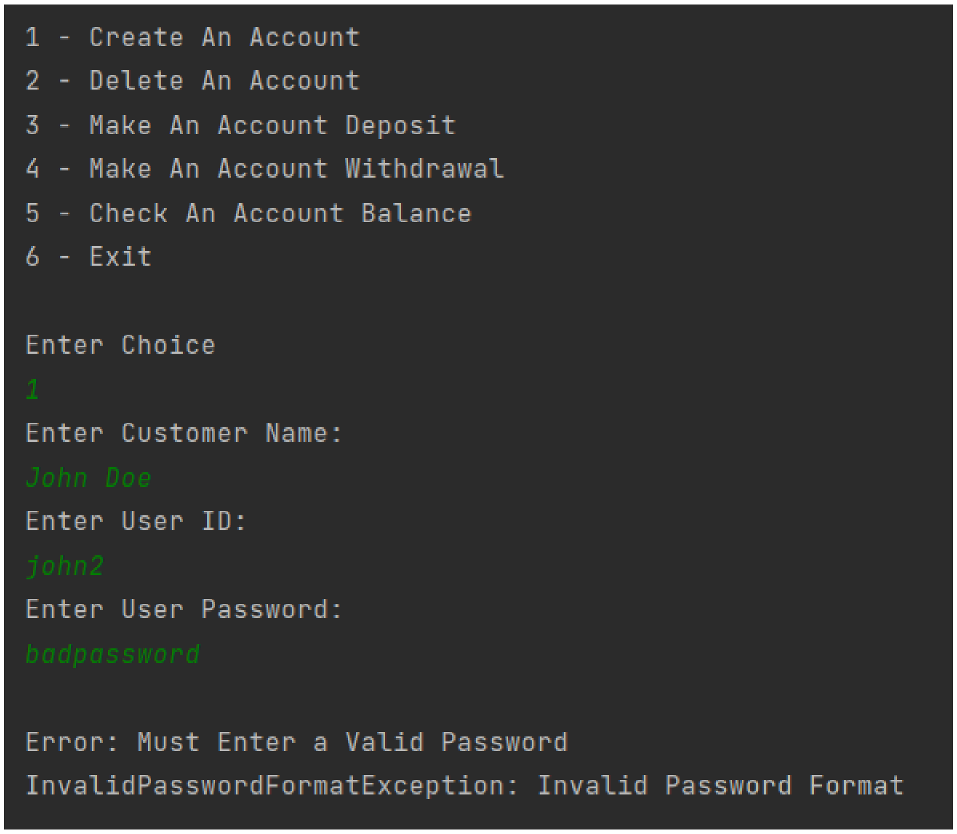
Screenshot 6 – Check an account balance



```
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 Password:
password*
Customer Name: Dexter Howard
Account Number: 1
Account Balance: 800.0
Account Type: Checking
Account Daily Withdrawal Limit: 300.0
```

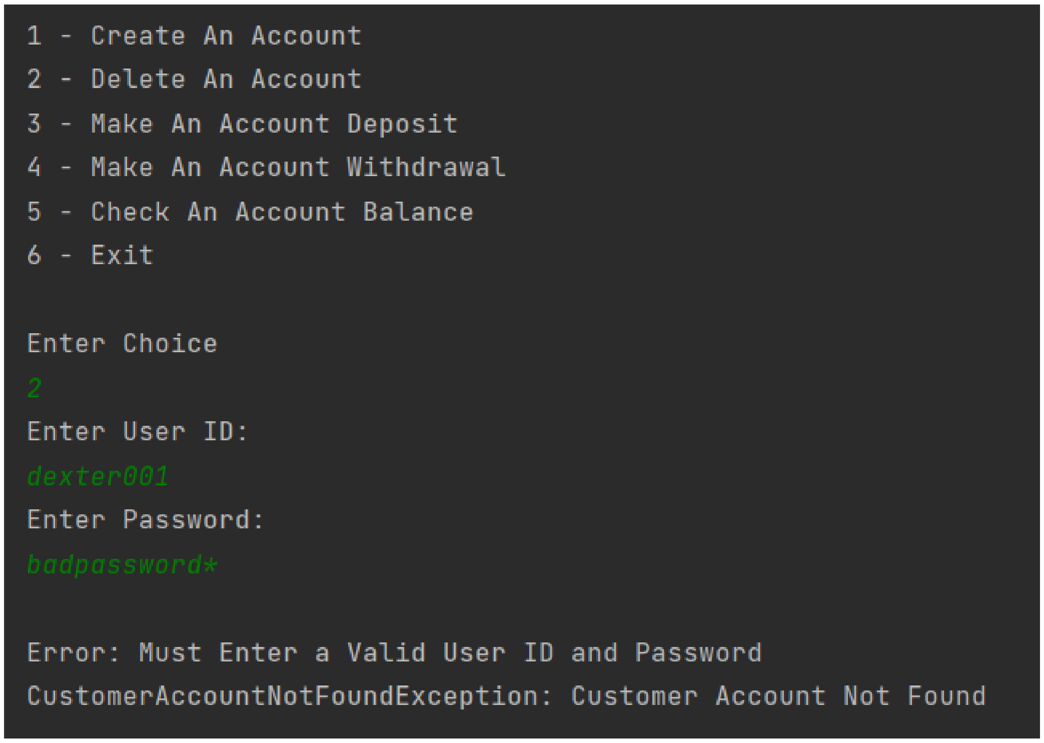
Group Assignment 3

Screenshot 7 – Enter an invalid password



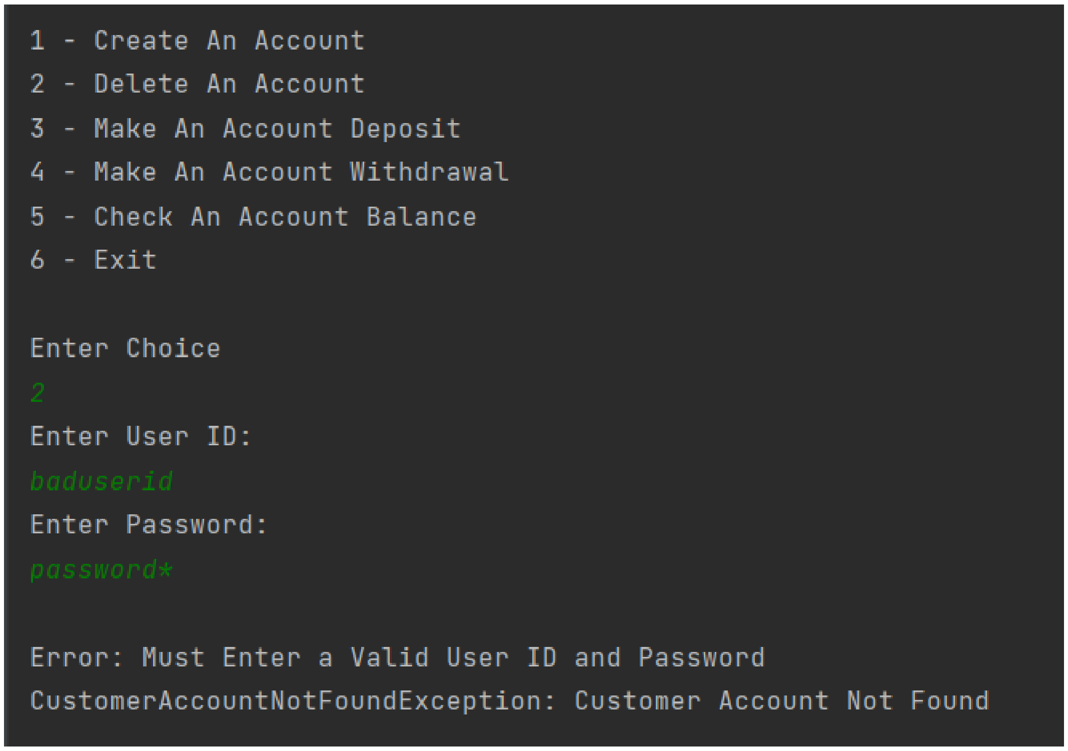
```
1 - Create An Account
2 - Delete An Account
3 - Make An Account Deposit
4 - Make An Account Withdrawal
5 - Check An Account Balance
6 - Exit

Enter Choice
1
Enter Customer Name:
John Doe
Enter User ID:
john2
Enter User Password:
badpassword

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

Screenshot 8 – Enter a non-existing account (wrong password)



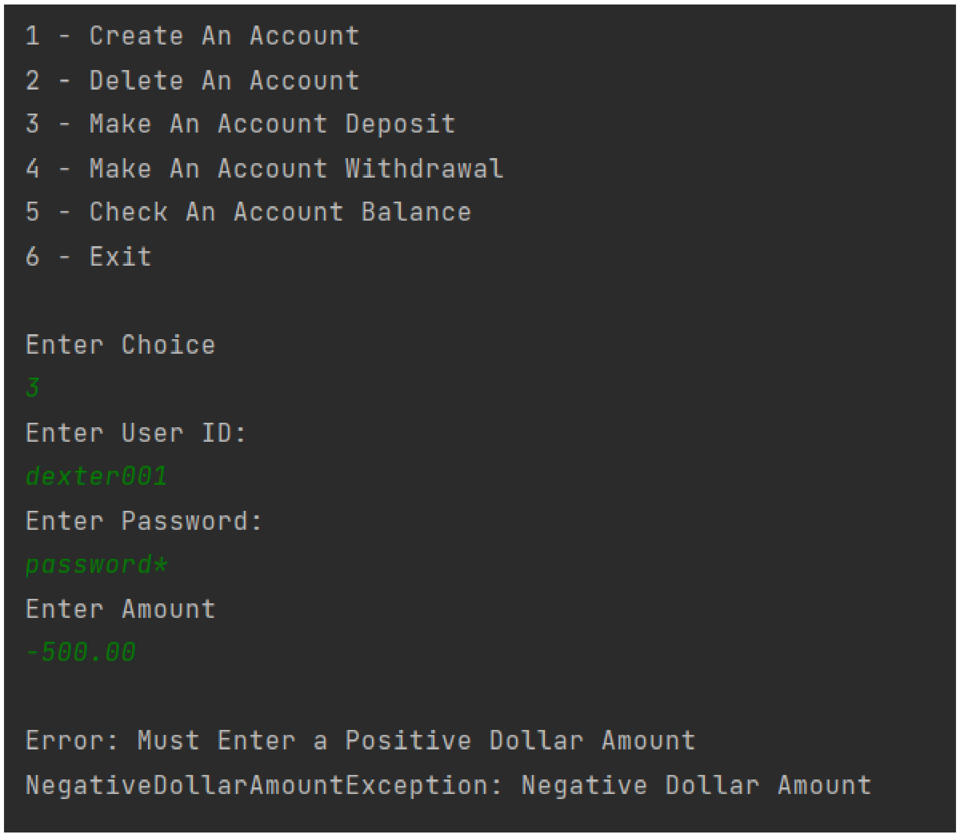
```
1 - Create An Account
2 - Delete An Account
3 - Make An Account Deposit
4 - Make An Account Withdrawal
5 - Check An Account Balance
6 - Exit

Enter Choice
2
Enter User ID:
dexter001
Enter Password:
badpassword*

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

Screenshot 9 – Enter a non-existing account (wrong user id)



```
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 Password:
password*

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

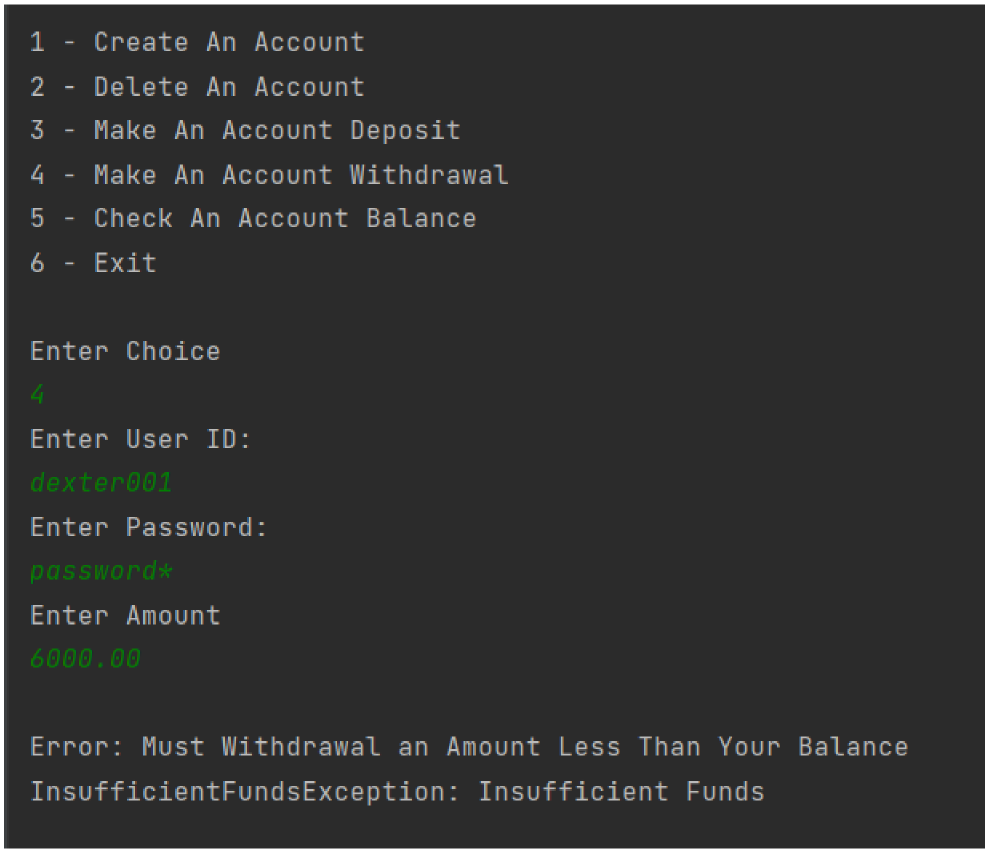
Screenshot 10 – Enter a negative dollar amount



```
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
3
Enter User ID:
dexter001
Enter Password:
password*
Enter Amount
-500.00

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

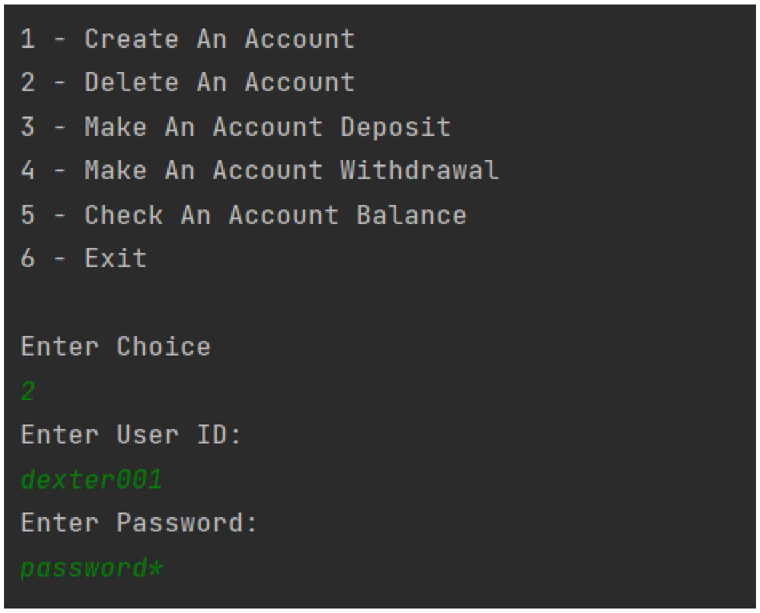
Group Assignment 3

Screenshot 11 – Attempt to withdrawal more money than user has



```
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 Password:
password*
Enter Amount
6000.00

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

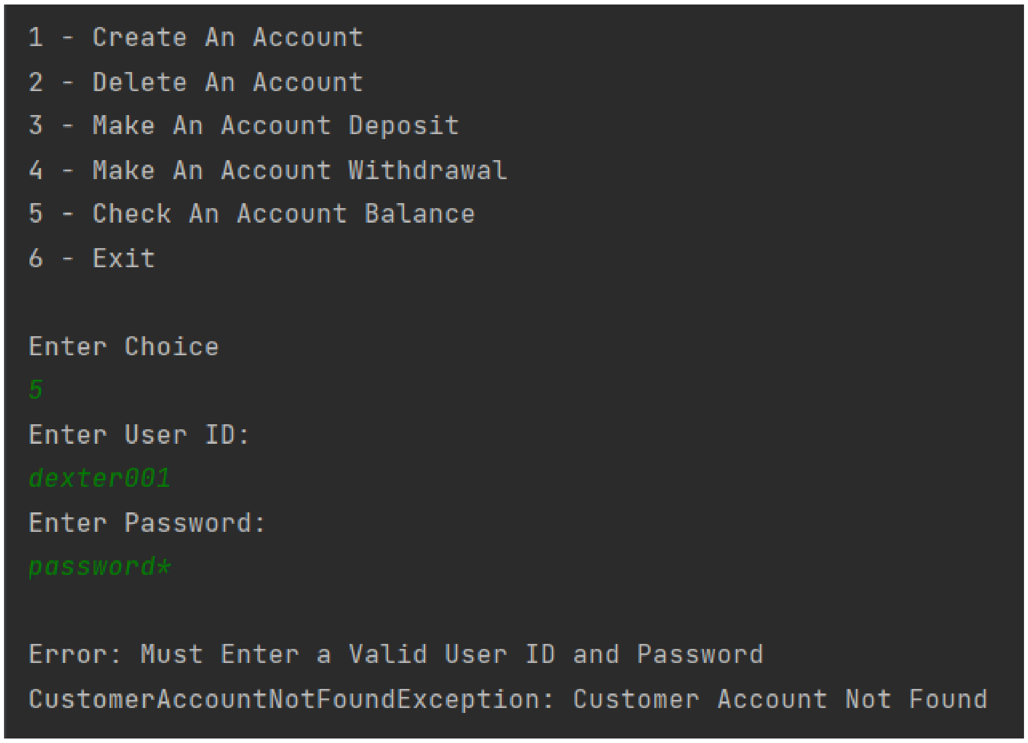
Screenshot 12 – Delete an account



```
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 Password:
password*
```

Group Assignment 3

Screenshot 13 – Attempt to access a non-existing account (account already deleted)



```
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 Password:
password*

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