Java Puzzle Ball MOOC
Lab 4: Lambda Expressions

Overview

Lambda expressions facilitate functional programming. They enable logic and functionality to be stored as a variable for later reference. Logic and functionality can be referenced and passed among methods as if it were a number. Lambda expressions are helpful in Java programs that need to detect mouse and keyboard events. This typically occurs as part of a program's graphical user interface (GUI). Lambda expressions are also very useful for filtering through a collection of many objects and performing actions for each object. Lab 4 gives you a GUI banking program. A number of account instances have already been created and added to a collection. This application contains buttons which are intended to work with account instances. Some of these buttons are implemented and working properly. Your goal is to implement the logic and functionality for the remaining buttons.

Tasks

Download the Lab 4 Start State. This is a zip (JPB_MOOC_Lab4.zip) containing a NetBeans project. This project is slightly different from the Lab 3 solution. Unzip the file and open the project in NetBeans.

Use the GUI application in conjunction with the NetBeans output window to test your work. All of your work will be done in the `ButtonController` class. This class implements the logic and functionality for the buttons in the program. It specifies the actions that are preformed when a button is pressed. Study the code for the working buttons in the `ButtonController` class to implement the remaining buttons. Each button and their intended functionality are listed below:

1. **Print Details**: Print details for each account belonging to the owner specified in the application's Account Owner search field. This text is case-sensitive.
2. **Close Accounts:** Close each account belonging to the owner specified in the application's Account Owner search field. This text is case-sensitive. This button is implemented for you.
3. **Deposit $200:** Deposit $200 into the account specified in the application's Account Number search field. This button is implemented for you.
4. **Withdraw $200:** Withdraw $200 from the account specified in the application's Account Number search field.
5. **Give Bonus:** Deposit a bonus $20 for each savings account with a balance of at least $20,000.
6. **Earn Interest:** Earn interest for each savings account. This button is implemented for you.

7. **Charge Fee:** Charge a $35 fee for each checking account with fewer than 1 transactions.
8. **Reset Transactions:** Reset transactions to 0 for each account.

When you see the same lambda expression logic repeated, this is an indication that it may be beneficial to store the repeated logic as a variable. The `ButtonController` class contains two fields you must use to store repeated logic:

1. **`matchAccountOwner`**: This logic checks to see which accounts in the collection have an accountOwner field which matches the application's Account Owner search field. The variable should be referenced 3 times by methods in this class.
2. **`matchAccountNumber`**: This logic checks to see which accounts in the collection have an accountNumber which matches the application's Account Number search field. The variable should be referenced 2 times by methods in this class.

Provide any other updates you feel are necessary for clarifying the output. For example, printing text to indicate a deposit is caused by earning a bonus.

Abstract Account

This class supports a new field to keep track of the number of account transactions. It also contains a number of special methods called **getter methods**. A getter method returns the value of a particular field. This is a design pattern known as **encapsulation**, which is much safer way of letting classes interact with each other. In general, it's dangerous to let classes access each other's fields directly. When you edit code in the `ButtonController` class and need to retrieve field values from an account, call the getter method rather than the field. You don't need to edit the `Account` class.

Checking Account

This inherits the fields and methods of the abstract `Account` class. You don't need to edit this class.

Savings Account

This inherits the fields and methods of the abstract `Account` class. You don't need to edit this class.

NewFXMain

JavaFX is a portion of the Java language which allows you to create GUI applications. Java Puzzle Ball is also programmed in JavaFX. The class `NewFXMain` replaces `TestClass` from the previous labs. It still contains a main method, along with other special components necessary for launching a GUI application. This class is responsible for detailing the visual components of the application. You don't need to edit this class, but you're welcome to explore it if you're curious.

This class also creates an ArrayList collection of all account instances used in this application. Because both Checking and Savings classes inherit from the abstract `Account` class, a collection of account instances may contain both Checking and Savings accounts. `CheckingAccount` and `SavingsAccount` instances are types of `Account`s.

Sample Output

```
New Account:
Savings Account #0
Account Owner: Duke
Balance: $100.0
Transactions: 0
Interest Rate: 0.02

New Account:
Savings Account #1
Account Owner: Damien
Balance: $20000.0
Transactions: 0
Interest Rate: 0.02

New Account:
Savings Account #2
Account Owner: Jessica
Balance: $50000.0
Transactions: 0
Interest Rate: 0.02

New Account:
Savings Account #3
Account Owner: Herbert
Balance: $500.0
Transactions: 0
Interest Rate: 0.02

New Account:
Checking Account #4
Account Owner: Duke
Balance: $0.0
Transactions: 0

New Account:
Checking Account #5
Account Owner: Jessica
Balance: $500.0
Transactions: 0

Savings Account #2
Account Owner: Jessica
Balance: $50000.0
Transactions: 0
Interest Rate: 0.02
```

```
Checking Account #5
Account Owner: Jessica
Balance: $500.0
Transactions: 0

CLOSING
Savings Account #2
Account Owner: Jessica
Balance: $50000.0
Transactions: 0
Interest Rate: 0.02

CLOSING
Checking Account #5
Account Owner: Jessica
Balance: $500.0
Transactions: 0

Deposit: $200.0
Savings Account #1
Account Owner: Damien
Balance: $20200.0
Transactions: 1
Interest Rate: 0.02

Bonus for high Savings Account balance.
Deposit: $20.0
Savings Account #1
Account Owner: Damien
Balance: $20220.0
Transactions: 2
Interest Rate: 0.02

Resetting Transactions to 0
Savings Account #0
Account Owner: Duke
Balance: $100.0
Transactions: 0
Interest Rate: 0.02

Resetting Transactions to 0
Savings Account #1
Account Owner: Damien
Balance: $20220.0
Transactions: 0
Interest Rate: 0.02
```

```
Resetting Transactions to 0
Savings Account #3
Account Owner: Herbert
Balance: $500.0
Transactions: 0
Interest Rate: 0.02

Resetting Transactions to 0
Checking Account #4
Account Owner: Duke
Balance: $0.0
Transactions: 0
```