Java Puzzle Ball
Nick Ristuccia

Lesson 4-3
Editing Java Code
Lab 4: Finish the Banking GUI Application

• The Lab Instructions are available on the Lesson 4 page of the MOOC.

• This is an enhanced version of the program you wrote in Lab 3.
  – The application works with many account instances.
  – The application is visual, and uses a GUI

• GUI means Graphical User Interface
  – The application is written in JavaFX, just like Java Puzzle Ball.
  – Lambda expressions detect button presses.
  – Lambda expressions implement the logic and functionality of working with accounts.

• The remaining part of this lesson will give you tips.
The Banking GUI Application

- The application contains 8 buttons.
  - Some of them work
  - Some of them don't

- Your goal is to implement the logic and functionality for the remaining buttons.
  - Do this in the `ButtonController` class.

- The Lab 4 instructions outline each button's functionality.
Demo

- Show the application in action
- Show how to use the application in conjunction with output window
- Show how to maximize the output window
The NewFXMain Class

• Replaces TestClass from Lab 3

• Still contains a main method, and has other special components necessary for launching a JavaFX GUI application

• It also creates many account instances

• The collection of these accounts is stored in an ArrayList

• An ArrayList is a special kind of class in Java
  – Because both Checking and Savings classes inherit from the abstract Account class, this ArrayList of Accounts may contain both Checking and Savings accounts
The **ButtonController Class**

- This is where you'll do all your editing for Lab 4.
- The field variable `accountList` stores the `ArrayList` of accounts.
- When you start implementing button functionality, `accountList` is what you'll need to create a stream of and filter.

```java
public class ButtonController{
    ArrayList<Account> accountList = new ArrayList<>();
    ...

    public void button3Pressed(){
        accountList.stream()
            .filter(a -> String.valueOf(a.getAccountNumber()).equals(numberSearchBar.getText()))
            .forEach(a -> a.deposit(200));
    }
    ...
}
The Dot Operator ( . )

• First, create an instance and name it with a reference variable.
  – In a stream, this is implicitly done for you. The ArrayList is defined to contain a specific object type.
  – In the stream, a is the name for any given Account instance.

```java
SavingsAccount a = new SavingsAccount("Duke", 100);
```

• Then, use the dot operator ( . ) to attempt accessing that instance's fields or methods:
  – Access field
    ```java
    a.accountNumber
    ```
  – Access method
    ```java
    a.getAccountNumber()
    ```
Encapsulation

• For reasons beyond the scope of this course, it's dangerous to allow different classes to freely access and edit each other's fields.

• Encapsulation makes fields private, preventing other classes from accessing or modifying fields in an unsafe way.

• Instead, use specially written...
  – Getter methods to read the value of a field
  – Setter methods to change the value of a field
    • You can think of deposit() and withdraw() as setters
    • They prevent negative and other inappropriate values
The Account Class

• All the getter and setter methods you'll need are in this class.
• These methods are shared by checking and savings accounts.
• Take a look, because there's some new fields and methods here:
  – Transactions
  – resetTransactions (safer to call this than to grant access to the transaction field)
• Button6 does something special, called casting, to access the earnInterest() method. This method is exclusive to the SavingsAccount subclass.

\[
a \rightarrow ((\text{SavingsAccount})a)\text{.earnInterest()}
\]
Comparing Strings

• When you filter, you'll need to compare values.
• For reasons beyond the scope of this course, using == on text (known as Strings) is not recommended. You might get the wrong result.
• This has to do with how Java stores Strings in memory.
• Instead, use the .equals() method.

– Bad:

\[
\text{a.getAccountType}() == "Savings Account"
\]

– Good:

\[
\text{a.getAccountType().equals("Savings Account"}"
\]
Comparing Numbers

- You can use () to organize logic.
- In the case of organizing logic or writing methods, be careful that parenthesis match the way you think they do. It's easy to mismatch, forget, or add an extra parenthesis somewhere.

<table>
<thead>
<tr>
<th>Logic Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>
Functional Programming

• 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.

• You've seen Consumer lambda expressions

• Lab 4 uses a two Predicate lambda expressions
  – Identify where logic is repeated in the ButtonController class.
  – Replace the field variables' null value with lambda logic.

Predicate<Account> matchAccountOwner = null;
Predicate<Account> matchAccountNumber = null;
Goodbye Message

• Thanks for playing!
• Wouldn't it be cool if there were more games and courses like this?
Lots More to Learn...

- What are those many other types of Lambdas? (mentioned in 4-2)
- What are other methods from the Streams API?

- You won't need to know these concepts for this course.
  - But if you're curious, Oracle as other courses where you can learn more.
  - You could also go for Oracle's industry-recognized certification exams.