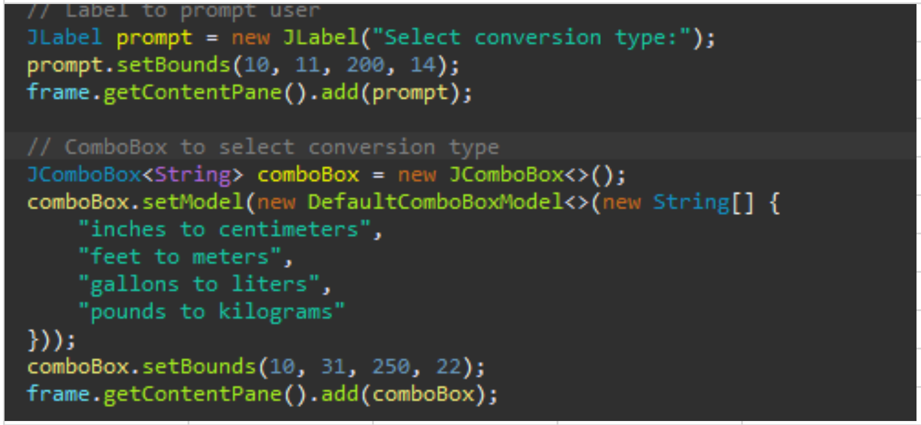
MetricConversion - Reflection

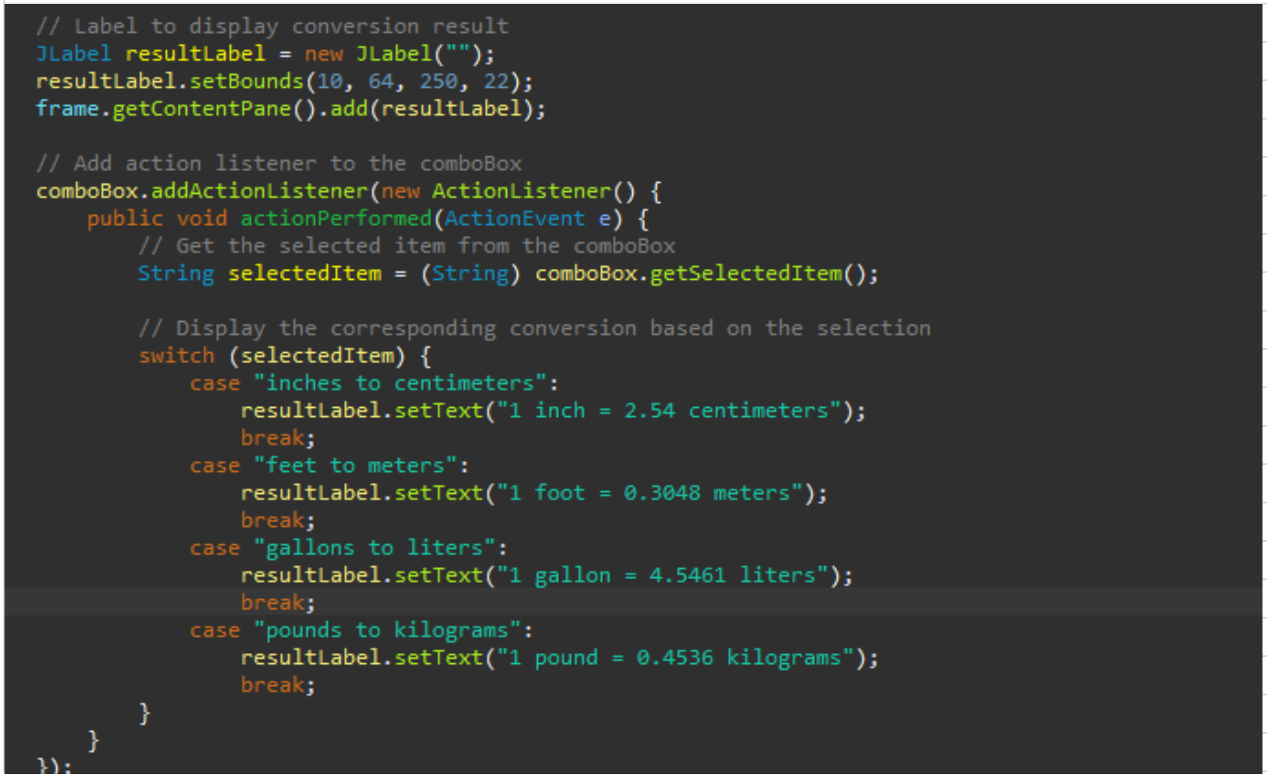
In my process, I first added a label with directions for the user, and a combo box to select the option of conversion:



```
// Label to prompt user
JLabel prompt = new JLabel("Select conversion type:");
prompt.setBounds(10, 11, 200, 14);
frame.getContentPane().add(prompt);

// ComboBox to select conversion type
JComboBox<String> comboBox = new JComboBox<>();
comboBox.setModel(new DefaultComboBoxModel<>(new String[] {
    "inches to centimeters",
    "feet to meters",
    "gallons to liters",
    "pounds to kilograms"
}));
comboBox.setBounds(10, 31, 250, 22);
frame.getContentPane().add(comboBox);
```

The combo box contains all of the measurement convertions available to the user.

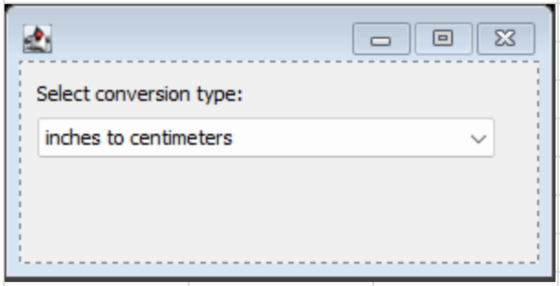
Then, I used an action listener to print to the label resultLabel based on what the user selected in the combo box.



```
// Label to display conversion result
JLabel resultLabel = new JLabel("");
resultLabel.setBounds(10, 64, 250, 22);
frame.getContentPane().add(resultLabel);

// Add action listener to the comboBox
comboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Get the selected item from the comboBox
        String selectedItem = (String) comboBox.getSelectedItem();

        // Display the corresponding conversion based on the selection
        switch (selectedItem) {
            case "inches to centimeters":
                resultLabel.setText("1 inch = 2.54 centimeters");
                break;
            case "feet to meters":
                resultLabel.setText("1 foot = 0.3048 meters");
                break;
            case "gallons to liters":
                resultLabel.setText("1 gallon = 4.5461 liters");
                break;
            case "pounds to kilograms":
                resultLabel.setText("1 pound = 0.4536 kilograms");
                break;
        }
    }
});
```

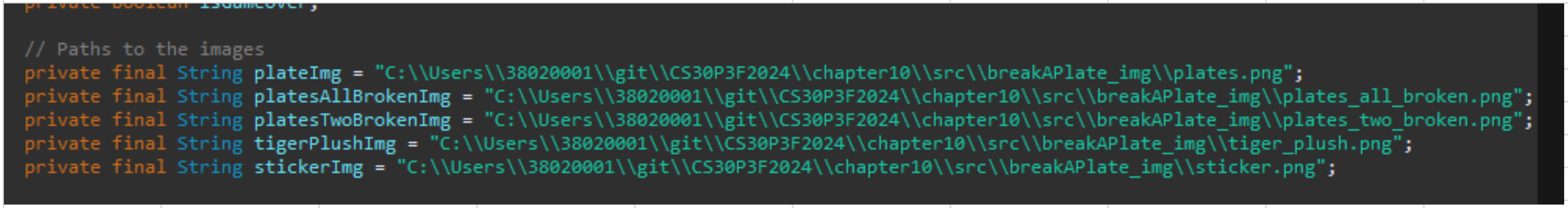
```
});
```

This is what the complete design looks like:



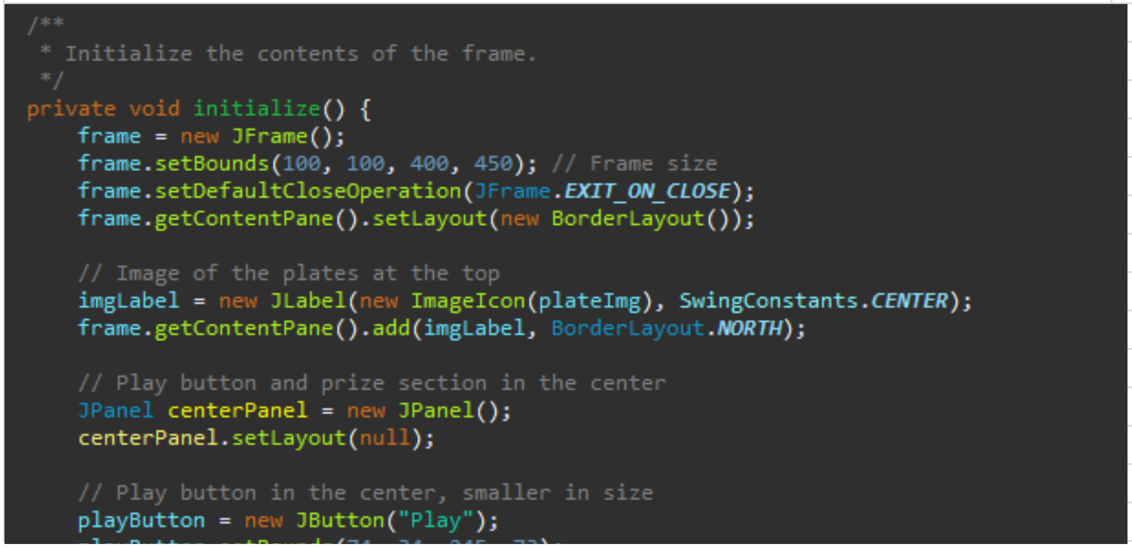
BreakAPlate - Reflection

In my process, I first set all the paths to all the images I used in the GUI.

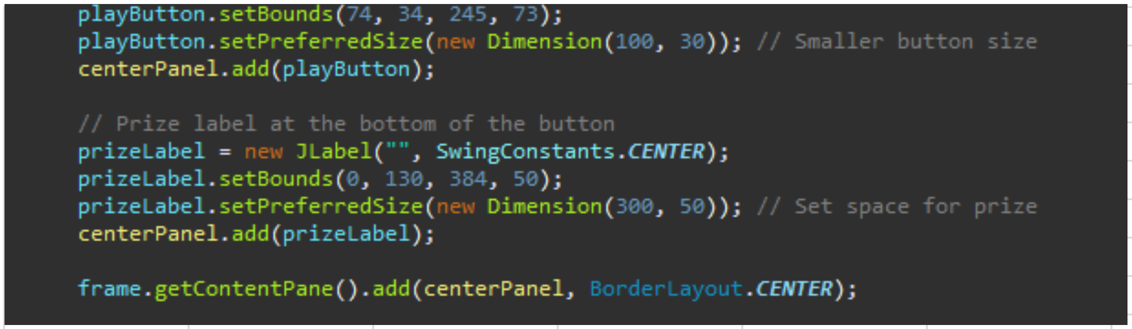


```
// Paths to the images
private final String plateImg = "C:\\Users\\38020001\\git\\CS30P3F2024\\chapter10\\src\\breakAPlate_img\\plates.png";
private final String platesAllBrokenImg = "C:\\Users\\38020001\\git\\CS30P3F2024\\chapter10\\src\\breakAPlate_img\\plates_all_broken.png";
private final String platesTwoBrokenImg = "C:\\Users\\38020001\\git\\CS30P3F2024\\chapter10\\src\\breakAPlate_img\\plates_two_broken.png";
private final String tigerPlushImg = "C:\\Users\\38020001\\git\\CS30P3F2024\\chapter10\\src\\breakAPlate_img\\tiger_plush.png";
private final String stickerImg = "C:\\Users\\38020001\\git\\CS30P3F2024\\chapter10\\src\\breakAPlate_img\\sticker.png";
```

The next components I added were the play button, the area for the plates, and the prize labels.

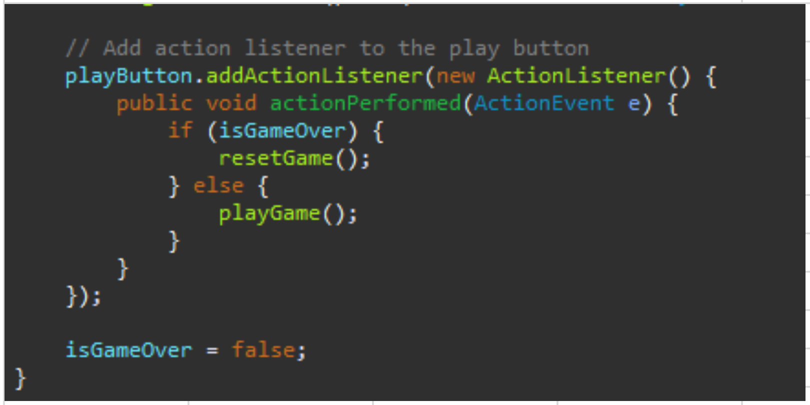


```
/**
 * Initialize the contents of the frame.
 */
private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 400, 450); // Frame size
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(new BorderLayout());

    // Image of the plates at the top
    imgLabel = new JLabel(new ImageIcon(plateImg), SwingConstants.CENTER);
    frame.getContentPane().add(imgLabel, BorderLayout.NORTH);

    // Play button and prize section in the center
    JPanel centerPanel = new JPanel();
    centerPanel.setLayout(null);

    // Play button in the center, smaller in size
    playButton = new JButton("Play");
```



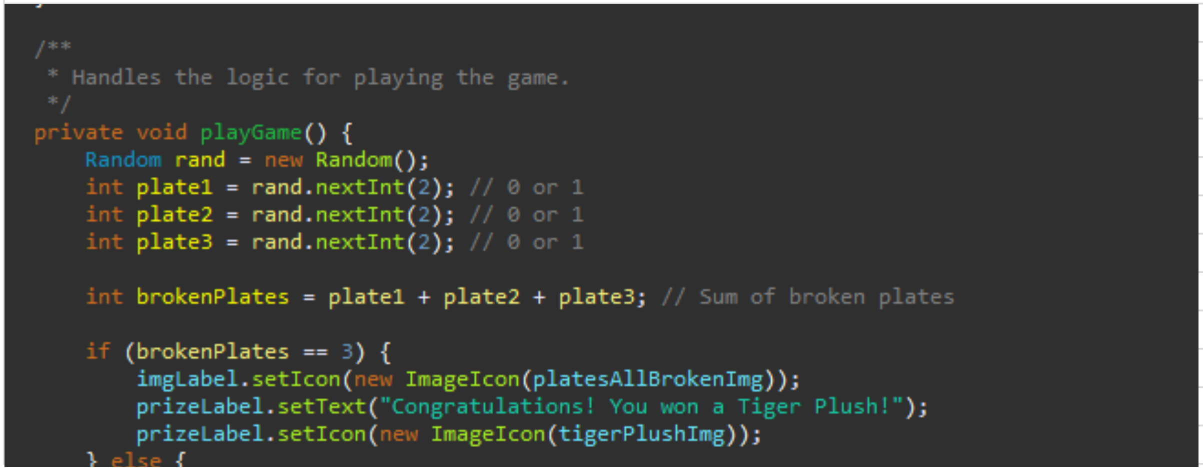
```java
        playButton.setBounds(74, 34, 245, 73);
        playButton.setPreferredSize(new Dimension(100, 30)); // Smaller button size
        centerPanel.add(playButton);

        // Prize label at the bottom of the button
        prizeLabel = new JLabel("", SwingConstants.CENTER);
        prizeLabel.setBounds(0, 130, 384, 50);
        prizeLabel.setPreferredSize(new Dimension(300, 50)); // Set space for prize
        centerPanel.add(prizeLabel);

        frame.getContentPane().add(centerPanel, BorderLayout.CENTER);
```

Then, I use functions to allow the user to play the game.

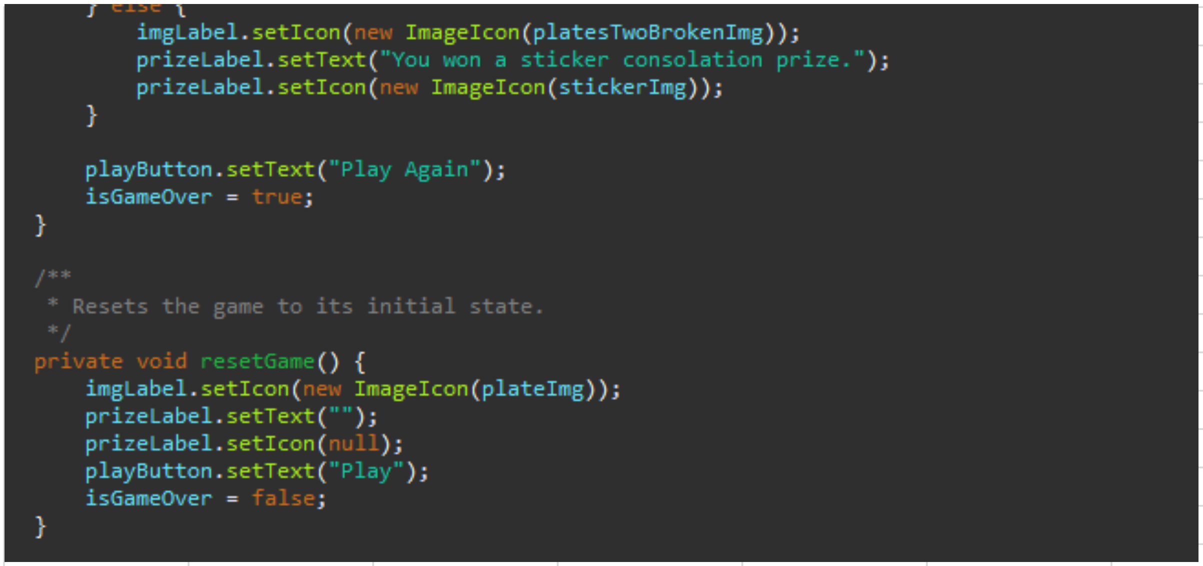


```java
    // Add action listener to the play button
    playButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if (isGameOver) {
                resetGame();
            } else {
                playGame();
            }
        }
    });

    isGameOver = false;
}
```

The functionality of the game includes a randomizer, and the display of the prize based on the result of the randomizer.



```java
  /**
   * Handles the logic for playing the game.
   */
  private void playGame() {
      Random rand = new Random();
      int plate1 = rand.nextInt(2); // 0 or 1
      int plate2 = rand.nextInt(2); // 0 or 1
      int plate3 = rand.nextInt(2); // 0 or 1

      int brokenPlates = plate1 + plate2 + plate3; // Sum of broken plates

      if (brokenPlates == 3) {
          imgLabel.setIcon(new ImageIcon(platesAllBrokenImg));
          prizeLabel.setText("Congratulations! You won a Tiger Plush!");
          prizeLabel.setIcon(new ImageIcon(tigerPlushImg));
      } else {
```

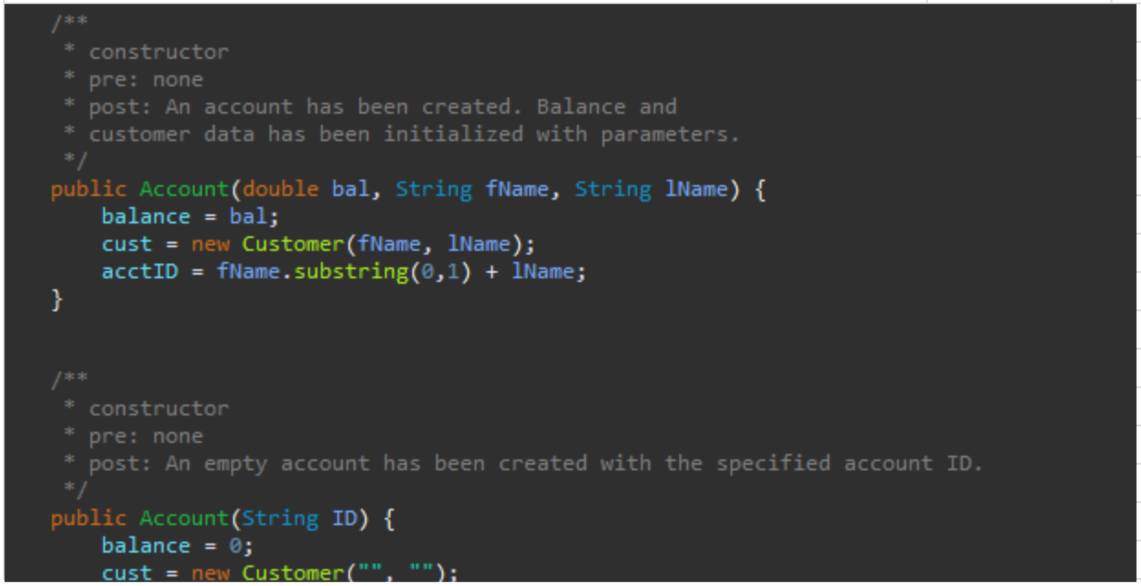


```
    } else {
        imgLabel.setIcon(new ImageIcon(platesTwoBrokenImg));
        prizeLabel.setText("You won a sticker consolation prize.");
        prizeLabel.setIcon(new ImageIcon(stickerImg));
    }

    playButton.setText("Play Again");
    isGameOver = true;
}

/**
 * Resets the game to its initial state.
 */
private void resetGame() {
    imgLabel.setIcon(new ImageIcon(plateImg));
    prizeLabel.setText("");
    prizeLabel.setIcon(null);
    playButton.setText("Play");
    isGameOver = false;
}
```

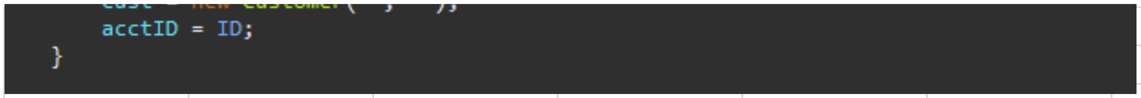
LocalBankGUI - Reflection

The functionality of the LocalBank program is based around three classes: Acccount, Bank, and Customer.

The Account class includes functions and constructors to manage account functions:

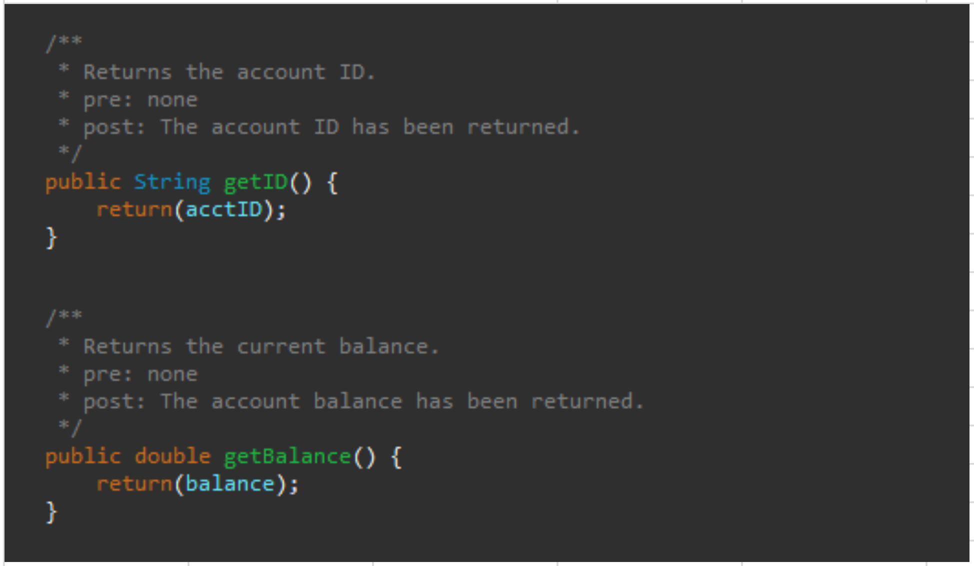


```
/**
 * constructor
 * pre: none
 * post: An account has been created. Balance and
 * customer data has been initialized with parameters.
 */
public Account(double bal, String fName, String lName) {
    balance = bal;
    cust = new Customer(fName, lName);
    acctID = fName.substring(0,1) + lName;
}


/**
 * constructor
 * pre: none
 * post: An empty account has been created with the specified account ID.
 */
public Account(String ID) {
    balance = 0;
    cust = new Customer("", "");
```



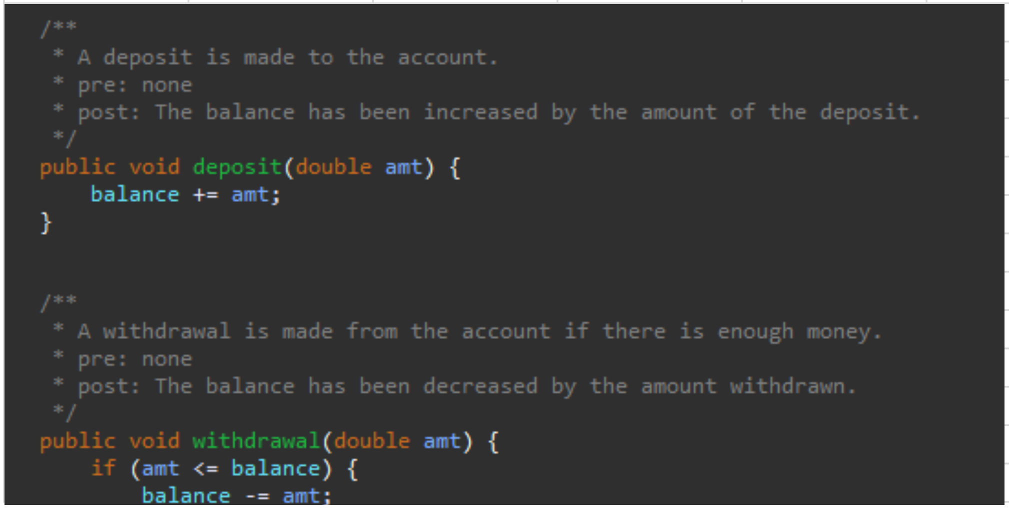
```
        acctID = ID;
    }
```

Above: constructors for creations of a user account.

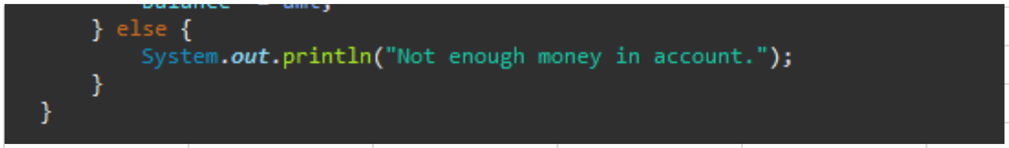


```
    /**
     * Returns the account ID.
     * pre: none
     * post: The account ID has been returned.
     */
    public String getID() {
        return(acctID);
    }


    /**
     * Returns the current balance.
     * pre: none
     * post: The account balance has been returned.
     */
    public double getBalance() {
        return(balance);
    }
```

Functions for returning user balance and/or account ID



```
    /**
     * A deposit is made to the account.
     * pre: none
     * post: The balance has been increased by the amount of the deposit.
     */
    public void deposit(double amt) {
        balance += amt;
    }


    /**
     * A withdrawal is made from the account if there is enough money.
     * pre: none
     * post: The balance has been decreased by the amount withdrawn.
     */
    public void withdrawal(double amt) {
        if (amt <= balance) {
            balance -= amt;
```

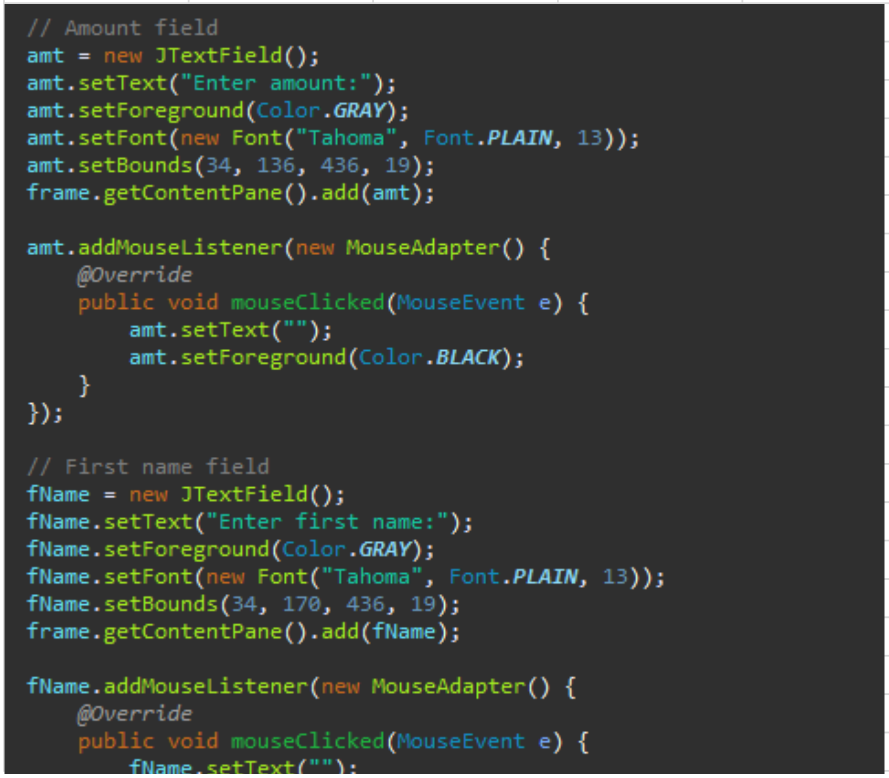


```
    } else {
        System.out.println("Not enough money in account.");
    }
}
```

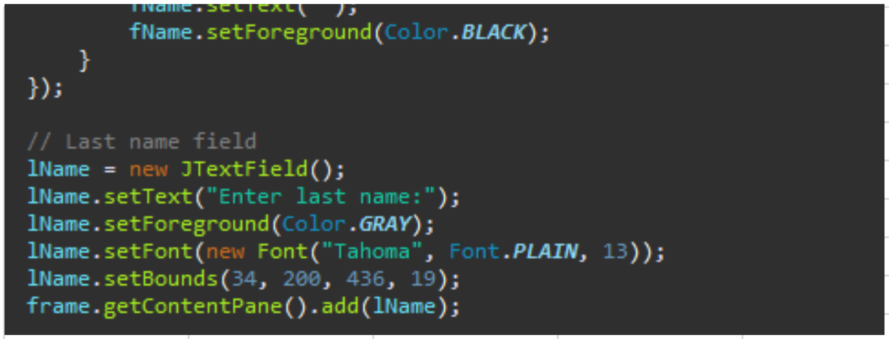
Functions for deposit and withdrawal.

The Bank class manages system functions such as:
- Adding a new account to the bank's accounts
- Deleting existing accounts
- Performing transactions on existing accounts
- Displaying account information

Originally, I tried to manage the whole program within one class - splitting them up into the separate three classes helped me better manage my code.

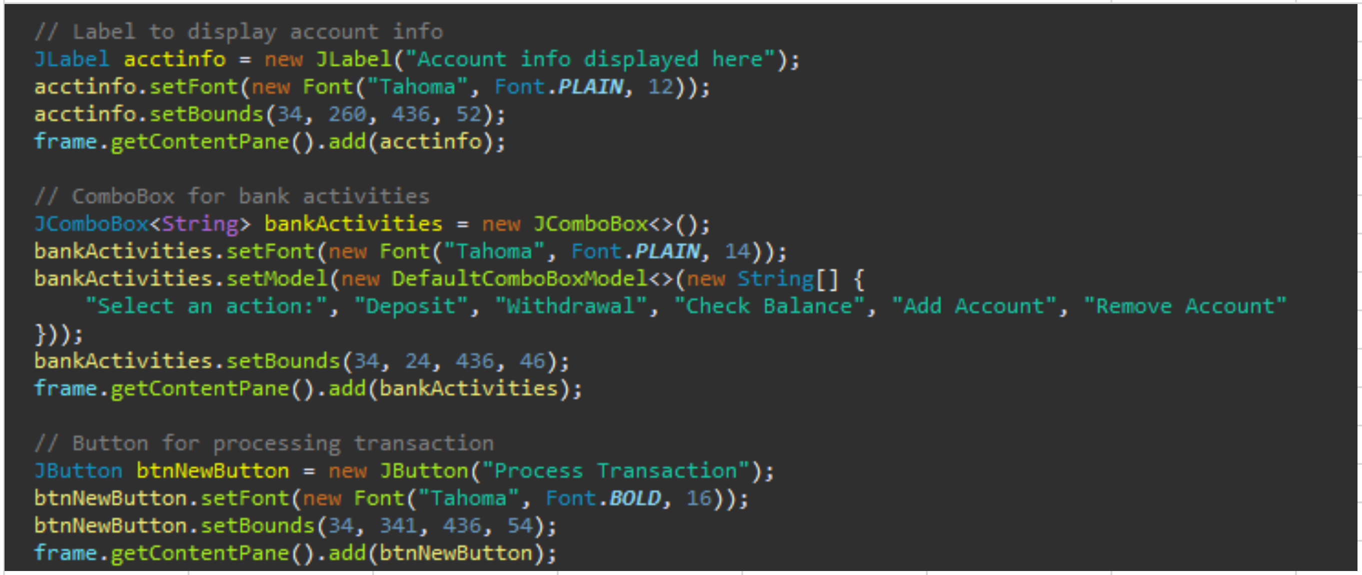


```
// Amount field
amt = new JTextField();
amt.setText("Enter amount:");
amt.setForeground(Color.GRAY);
amt.setFont(new Font("Tahoma", Font.PLAIN, 13));
amt.setBounds(34, 136, 436, 19);
frame.getContentPane().add(amt);

amt.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        amt.setText("");
        amt.setForeground(Color.BLACK);
    }
});

// First name field
fName = new JTextField();
fName.setText("Enter first name:");
fName.setForeground(Color.GRAY);
fName.setFont(new Font("Tahoma", Font.PLAIN, 13));
fName.setBounds(34, 170, 436, 19);
frame.getContentPane().add(fName);

fName.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        fName.setText("");
```



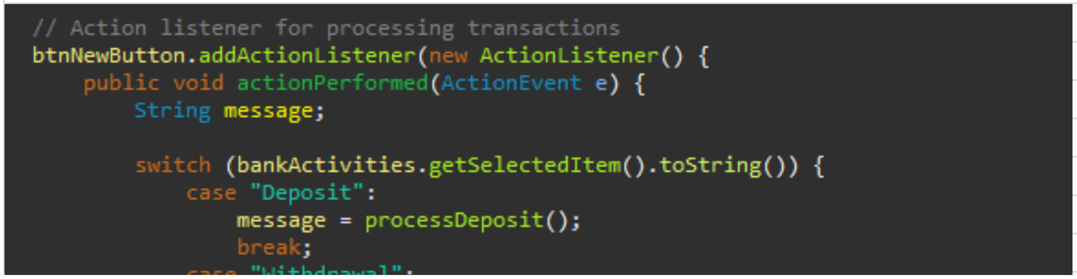
```
        fName.setText("");
        fName.setForeground(Color.BLACK);
    }
});

// Last name field
lName = new JTextField();
lName.setText("Enter last name:");
lName.setForeground(Color.GRAY);
lName.setFont(new Font("Tahoma", Font.PLAIN, 13));
lName.setBounds(34, 200, 436, 19);
frame.getContentPane().add(lName);
```

The LocalBankGUI class manages the GUI elements, such as the text fields, buttons, etc.

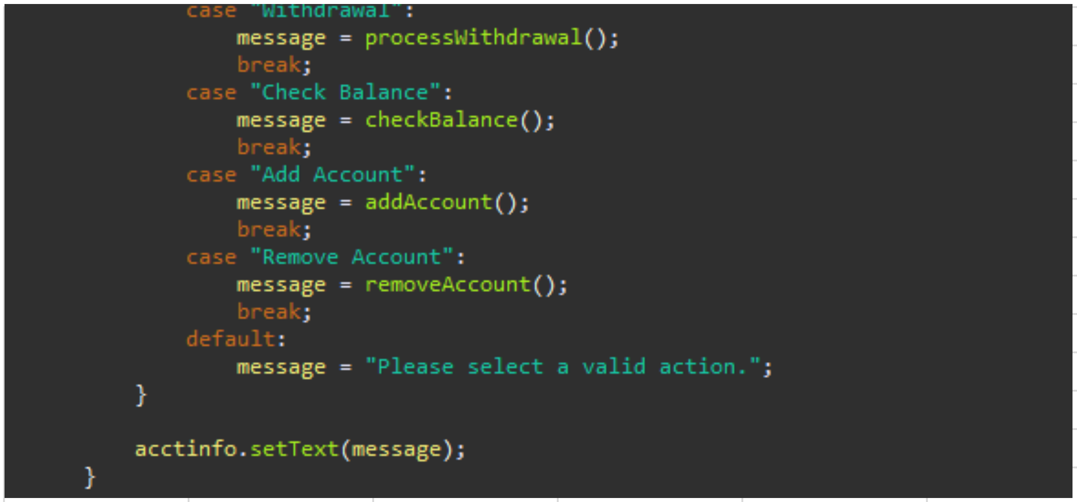


```
// Label to display account info
JLabel acctinfo = new JLabel("Account info displayed here");
acctinfo.setFont(new Font("Tahoma", Font.PLAIN, 12));
acctinfo.setBounds(34, 260, 436, 52);
frame.getContentPane().add(acctinfo);

// ComboBox for bank activities
JComboBox<String> bankActivities = new JComboBox<>();
bankActivities.setFont(new Font("Tahoma", Font.PLAIN, 14));
bankActivities.setModel(new DefaultComboBoxModel<>(new String[] {
    "Select an action:", "Deposit", "Withdrawal", "Check Balance", "Add Account", "Remove Account"
}));
bankActivities.setBounds(34, 24, 436, 46);
frame.getContentPane().add(bankActivities);

// Button for processing transaction
JButton btnNewButton = new JButton("Process Transaction");
btnNewButton.setFont(new Font("Tahoma", Font.BOLD, 16));
btnNewButton.setBounds(34, 341, 436, 54);
frame.getContentPane().add(btnNewButton);
```

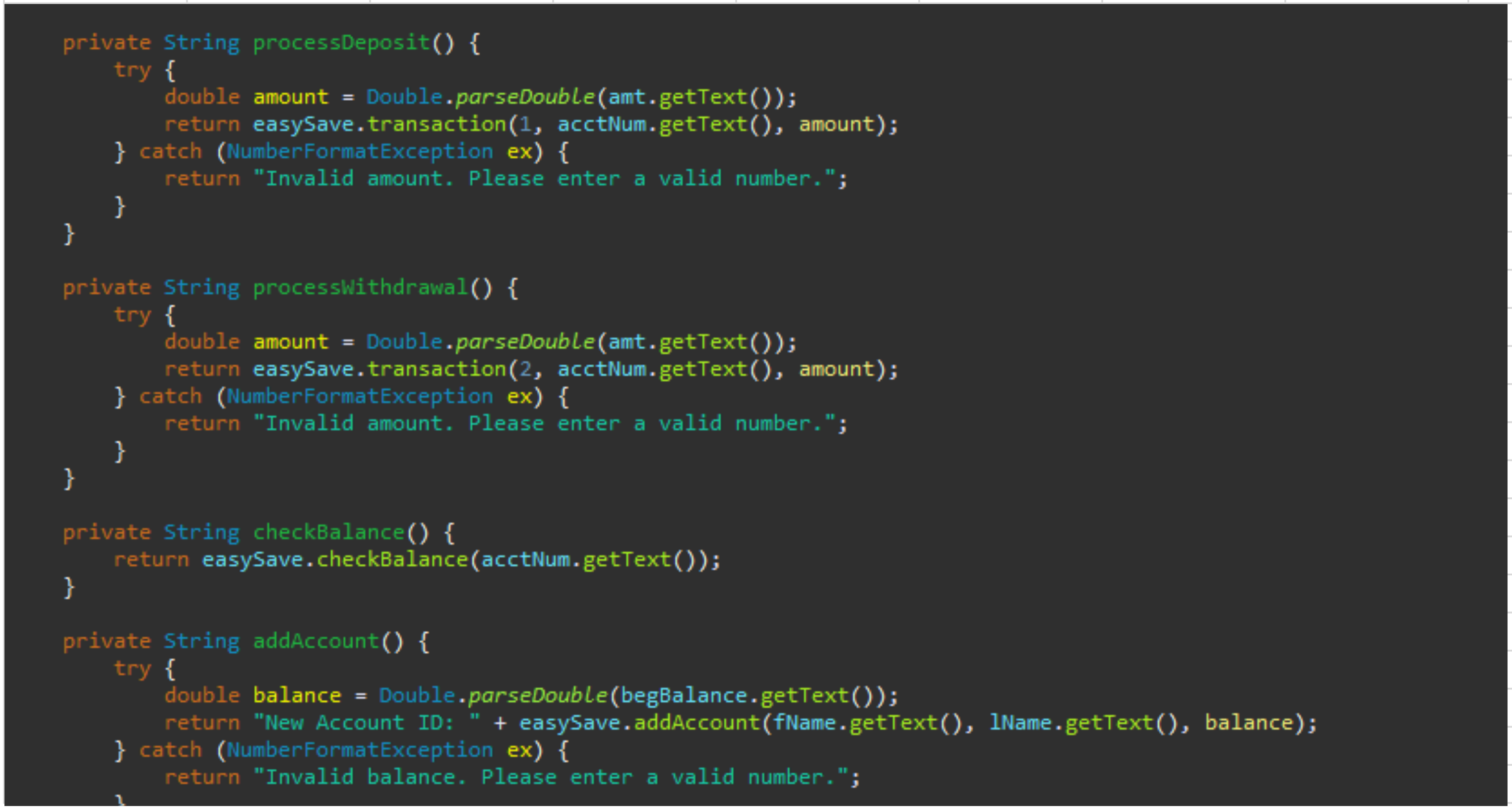
The action listener in this class receives what the user chooses within the combo box and performs a respective function.



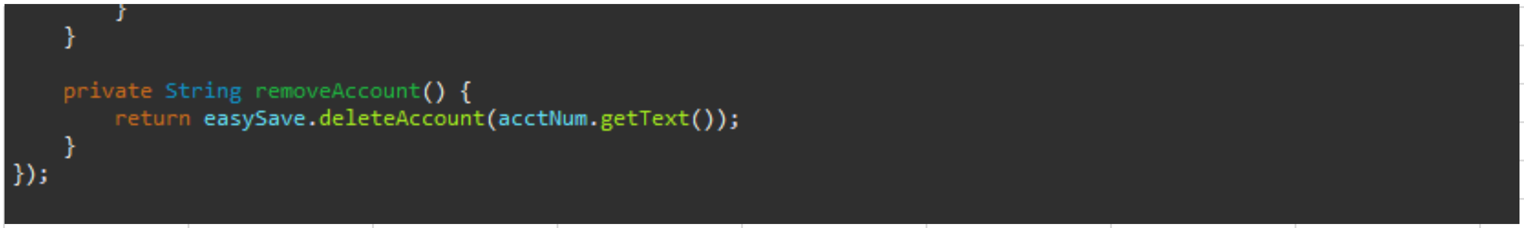
```
// Action listener for processing transactions
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String message;

        switch (bankActivities.getSelectedItem().toString()) {
            case "Deposit":
                message = processDeposit();
                break;
            case "Withdrawal":
```



```
            case "Withdrawal":
                message = processWithdrawal();
                break;
            case "Check Balance":
                message = checkBalance();
                break;
            case "Add Account":
                message = addAccount();
                break;
            case "Remove Account":
                message = removeAccount();
                break;
            default:
                message = "Please select a valid action.";
        }

        acctinfo.setText(message);
    }
```



```
    private String processDeposit() {
        try {
            double amount = Double.parseDouble(amt.getText());
            return easySave.transaction(1, acctNum.getText(), amount);
        } catch (NumberFormatException ex) {
            return "Invalid amount. Please enter a valid number.";
        }
    }

    private String processWithdrawal() {
        try {
            double amount = Double.parseDouble(amt.getText());
            return easySave.transaction(2, acctNum.getText(), amount);
        } catch (NumberFormatException ex) {
            return "Invalid amount. Please enter a valid number.";
        }
    }

    private String checkBalance() {
        return easySave.checkBalance(acctNum.getText());
    }

    private String addAccount() {
        try {
            double balance = Double.parseDouble(begBalance.getText());
            return "New Account ID: " + easySave.addAccount(fName.getText(), lName.getText(), balance);
        } catch (NumberFormatException ex) {
            return "Invalid balance. Please enter a valid number.";
        }
```



```
        }
    }

    private String removeAccount() {
        return easySave.deleteAccount(acctNum.getText());
    }
});
```