Faculty of Engineering & Applied Science

SOFE4640U Mobile Application Development

Assignment 1 - Equated Monthly Installment Calculator

Mitchell Hicks

100707709

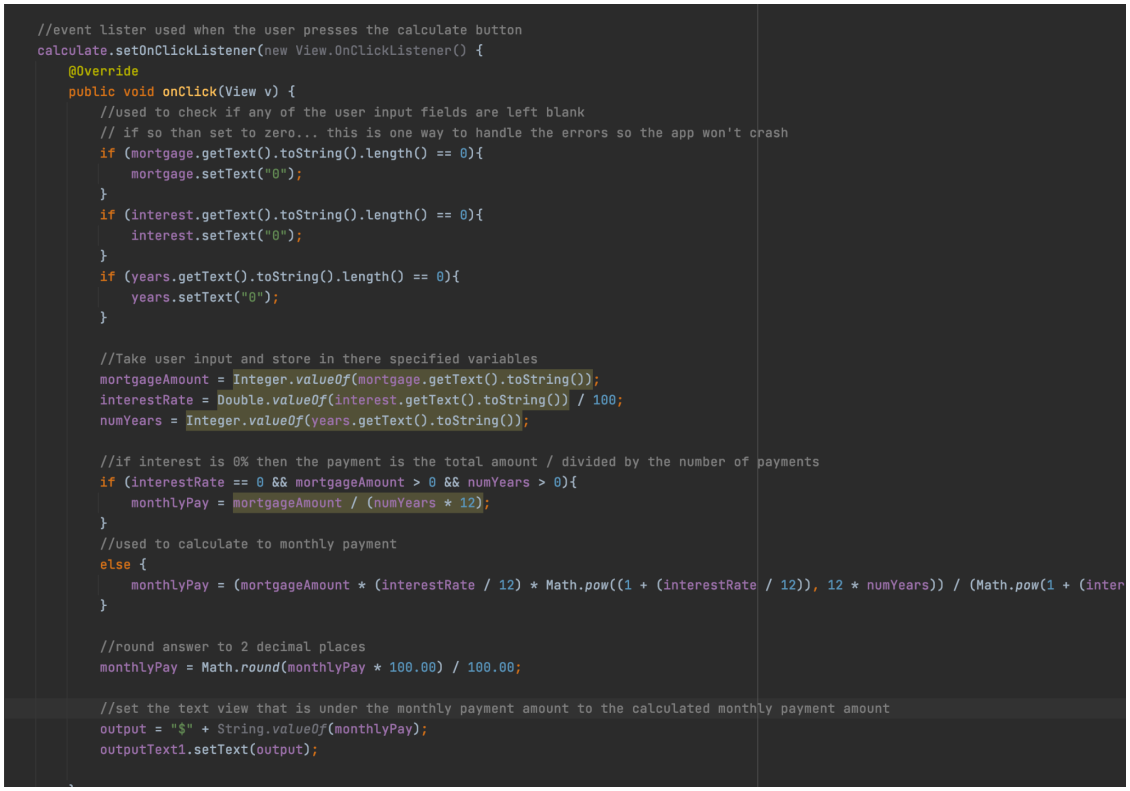
Assignment Overview

The requirements for assignment 1 was to create an equated monthly installment (EMI) calculator that makes it easy for users to accurately calculate the amount of money they will owe on a monthly basis to cover a specified mortgage amount at a given interest rate that's amortized over a number of years.

My assignment covers the above requirements by implementing a calculator that is accurate, easy to use, sleek, and is designed dynamically (meaning the design of the app will automatically change depending on the device).

Intents

In my EMI app there are many intents that are implemented. The main intent used in my application is when the calculate button is clicked, this action is then picked up by a function in main_activity.java, which then uses 3 intents to get the information the user inputted in the edit text boxes. We then perform an arithmetic operation on the data and use an intent to display it in our main textview header.



```
//event lister used when the user presses the calculate button
calculate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //used to check if any of the user input fields are left blank
        // if so than set to zero... this is one way to handle the errors so the app won't crash
        if (mortgage.getText().toString().length() == 0){
            mortgage.setText("0");
        }
        if (interest.getText().toString().length() == 0){
            interest.setText("0");
        }
        if (years.getText().toString().length() == 0){
            years.setText("0");
        }

        //Take user input and store in there specified variables
        mortgageAmount = Integer.valueOf(mortgage.getText().toString());
        interestRate = Double.valueOf(interest.getText().toString()) / 100;
        numYears = Integer.valueOf(years.getText().toString());

        //if interest is 0% then the payment is the total amount / divided by the number of payments
        if (interestRate == 0 && mortgageAmount > 0 && numYears > 0){
            monthlyPay = mortgageAmount / (numYears * 12);
        }
        //used to calculate to monthly payment
        else {
            monthlyPay = (mortgageAmount * (interestRate / 12) * Math.pow((1 + (interestRate / 12)), 12 * numYears)) / (Math.pow(1 + (inter
        }

        //round answer to 2 decimal places
        monthlyPay = Math.round(monthlyPay * 100.00) / 100.00;

        //set the text view that is under the monthly payment amount to the calculated monthly payment amount
        output = "$" + String.valueOf(monthlyPay);
        outputText1.setText(output);
```

Layouts

For my mobile application I only used one layout and that was a constraint layout. The main reason I used the constraint layout was because it helped me avoid using nested layouts, as well the constraint layout causes better performance. Given the simplicity of the assignment I only needed to create one layout and didn't require nested layouts.

Views

My EMI calculator app uses 9 different views (views = layout components (ex. text boxes, buttons, etc.)), 5 of which are considered "textview" views, 1 is a "button" view and the other 3 are "editview" views. The 5 textviews are used to display text headers that are used to describe the editviews. The 3 editviews are used to prompt the user to input the 3 numbers needed to calculate their EMI. The editviews take in the mortgage principal amount, the interest rate on the mortgage and the number of years the mortgage is amortized over. Lastly the calculate button, once clicked is used as a function to calculate the EMI for the given results. All of these were altered properly in either the strings.xml (for text values) or colors.xml (for different colors), no views were hardcoded. This helps reusability and functionality of the code.