

Introduction:

The Mortgage Calculator app helps users calculate mortgage-related values based on their input. This paper provides an overview of the steps to develop the app, including setting dependencies, creating the user interface using XML, and implementing the app's functionality in the MainActivity Java file.

I. Setup Dependencies:

Prior to operation, several dependencies required updating within the project configuration to allow for functionality. These dependencies include adding the MPAndroidChart library by including "implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'" in the dependencies section of the build.gradle (Module: app) file. Changes to the settings.gradel file include "android.enableJetifier=true" property to the gradle.properties file, 'mavenCentral()' with 'jcenter()', and a new repository was added with the URL "https://jitpack.io" in the repositories section of the settings.gradle file. Finally, the project was synced with the Gradle files to ensure proper configuration.

II. XML Files:

Themes and colors using the Material3 color builder located at https://material.io/theme-builder and manually transferred into the project files. As this is not a Kotlin project, the Jetpack Compose system could not apply themes automatically. Theme colors were added to components manually in order to apply the theme.

The activity\_main.xml file defines the user interface of the Mortgage Calculator app. The layout utilizes a ConstraintLayout as the root view, containing a ScrollView to enable content scrolling. Inside the ScrollView, a LinearLayout was used to hold all the views. This LinearLayout wraps a ConstraintLayout to organize the individual components of the app.

Title TextView: A TextView displaying the title "Welcome to the Mortgage Calculator" was added to provide a clear heading for the app.

Input Fields: EditText fields allow the user to enter the mortgage's purchase price, down payment, and interest rate.

Length Option Toggle: A SwitchMaterial component allows users to toggle between manual length selection and predefined options (10, 20, or 30 years). A TextView located to the right of the toggle button explains the current setting to the user. As the user switches between the two, the system resets all displayed results, but the user inputs remain to allow the user to recalculate their previous request quickly.

Slider: A Slider component allows the user to select the length of the mortgage. It has a range from 0 to 30 years with increments of 5.

Calculation Buttons: Two buttons, "Calculate" and "Clear," were included to calculate the mortgage and reset the input fields, respectively.

Result Table: A TableLayout Is used to display the calculated mortgage results. The table consists of several TableRows with TextViews for titles and calculated values for different mortgage lengths.

Line Graph: An MPAndroidChart LineChart component visualizes the mortgage results. By default, the line chart is hidden (GONE). The system checks the selected length option once the user clicks the calculate button. It will either set graph 1 to VISIBLE in the case of user-defined length or set graphs 1, 2, and 3 to VISIBLE in the case of system defined in order to create and display the graphs for 10, 20, and 30-year mortgages.

III. MainActivity Implementation (MainActivity.java):

The MainActivity class serves as the entry point for the app's functionality. It handles user input, performs calculations, and updates the UI accordingly.

Importing Dependencies: Several necessary dependencies were imported, including classes for handling user interface elements, charting, and formatting.

View References: The function findViewById() sets all required references to the various views defined in the activity\_main.xml file.

Initialization and Configuration: The TableLayout and LineChart components are hidden (GONE) by default. The Slider's step size, minimum, and maximum values are set to default.

Event Listeners: The length option toggle button (SwitchMaterial) and the calculate and clear buttons contain event listeners. The calculate and clear buttons listener calls the calculate and clear functions, respectively. The toggle button changes the boolean optionChecked to match the checked status of the toggle, later setting the length array as the value of the slider or as {10, 20, 30}.

Calculation and Display (continued):

The values entered by the user are retrieved from the EditText fields. If the user has selected a specific mortgage length option, the corresponding value is stored in an array of size 1. Otherwise, an array with predefined values of 10, 20, and 30 years is created.

The main activity then finds the length of the lengthArray and uses this to determine if the user has set a length or if the system default is used. It then uses a switch case to pass the input data to the setupCalculate function.

There, the system calls the CalculateMonthly class and passes in the data, where it processes it into an array of "Monthly" objects that contain the total monthly payment, interest payment, and principal payment of each month in the length of the mortgage. It uses this array of objects to set the table layout information with the monthly payments, as well as passing the array into the GetLineData class, where it processes it into a LineData object, then passed into a LineDataSet, then finally used in the 'setData()' function on its corresponding chart to build the data within the chart.

The system displays these results in the corresponding sections of the TableLayout, and the setCharts. If a user-defined length is selected, only the first two rows of the table will contain values. However, if the active length set is the system-defined 10, 20, and 30-year option, all four rows would be populated with values.

Line Chart Visualization:

The LineChart from the MPAndroidChart library visually represents the mortgage results. The app calculates the total payments for each year and stores them in a LineData object. The system then passes the LineData object into the chart to be displayed. Various UI settings, such as colors and labels, are configured for the chart by calling the ChartSetup class to ensure proper visual presentation.