# Loan Calculator using JavaFX

Jeffrey Wan

John Hopkins University

## Author Note

I guess an author's note goes here sometimes? I'd like to thank Stride consulting for paying for this Intro to Java Course. It was actually pretty great to learn how to code in Java even though I work as a full time software engineer. This has made reading Java and even Scala so much easier. Thanks to my classmates and my professor at Hopkins for helping me through this course as well.

Abstract

This program is a Loan Calculator that allows a user to input an interest rate, number of periods, and loan amount and outputs a monthly payment and total payment. It's written using the JavaFX library, using a GridPane, TextFields, and a calculate button. The code is written using OOP and encapsulation in mind with an attention to reusable code.

## Design and Analysis

This is a loan calculator written in Java using the JavaFX library. It allows a user to input an interest rate, loan amount, and number of years and outputs the monthly payment and total payment. The assumption that the calculator makes is that there are 12 compounding periods in a year.

At first I wrote this program using a FlowPane but found it much harder to design the actual application window than using a GridPane. I ultimately wound up using a GridPane that is broken down into 6 rows and 2 columns. It is composed of three labels, three text fields, one button, and one additional label to hold error responses when the user enters invalid input. The user is not allowed to submit negative numbers or blank fields when submitting the form.

The key class here is the CalculateHandler which implements the EventHandler<ActionEvent> interface. It is a listener which is listening to calculate button for clicks. When clicked, the listener grabs values from the form, calculates the payments, and sets the text of the payment fields after performing an interest rate calculation.

## Alternatives Considered

Originally, I attempted to build this using a FlowPane. However, I couldn't force the labels and text fields and button into their appropriate sections of the panel. Using a GridPane was much easier than using a FlowPane.

Also, I almost implemented this without using a guard close to protect against invalid user input. Without it, the user can enter in negative numbers or blanks which caused the program to either crash when attempting to convert empty strings to integers or calculate invalid payment amounts when negative numbers were entered. The guard clause prevents invalid calculations from being performed and the error message label guides the user into entering valid inputs.

## Lessons Learned

I think the biggest lesson learned here was the utility in using guard clauses in my code. Without it, invalid user inputs have to be handled. However, if a guard clause is used and error messages are displayed to the user, it greatly simplifies the rest of the program. Without it, the programmer has to handle a wider variety of user inputs, some of which may be even logically incorrect for the calculations at hand.