# The Final Countdown: Java Mini Project 3

Allison Juliette Snipes

Department of Computer Science, Johns Hopkins University

EN 605.201.81: Introduction to Programming Using Java

Dr. Sidney Rubey

August 15, 2020

## Abstract

The purpose of Mini Project Three was to provide the student an opportunity to design, and implement, an object oriented Graphical User Interface (GUI) to simulate a loan calculator. The application largely depends on input from the user, mathematical calculations provided by a secondary class, and carefully planned getter and setter methods. While this application was simple, it provided students the chance to apply fundamental concepts integral to Java programming.

## Program Design

My program was coded in the Eclipse Integrated Development Environment (IDE) using the Java 8 Standard Edition Development Kit (JDK), and GitHub to manage version control. No application builders were utilized; as per the instructions given in the project's guidelines. The program is designed to rely on: one main class, a method to initiate the GUI framework, a start method that runs the GUI, an event handler, a GUI close method, and a second class to perform the necessary mathematical calculations for the program.

It is important to note that the application's code begins by importing controls, layouts, and classes via extending from JavaFx's Application class. The main class is rather simple and relies on only two lines of code. The first line of code serves to provide simple error handling, and prints to the console (if the method executes successfully). The second line of code is a launch method that is responsible for running the application. It depends on a sole argument, args, that is a subclass of the Application class.

While some programmers might deem this next step unnecessary, my design includes both stop and init GUI methods. These two methods are utilized to identify bugs when the program starts and ends. Without including them troubleshooting the application is nearly

impossible for a beginner programmer! While testing the program, it will alert the programer of successful executions of methods or tasks' completions.

Within the start method is the program's grid layout, stage, scene, rootNode, labels, textfields, buttons, and event handler. Within the first three textfields the user is only permitted to enter numerical values, otherwise the program will quit prematurely. The user is not permitted to input anything into the last two textfields as this is where the loan calculations will be displayed. The application's event handler is triggered when the calculate button is pressed, which prompts the class to perform mathematical calculations of the user's input.

**Analysis**

There are countless alternative design approaches that would produce similar results, however the version I chose made the most sense to me and provided fewer errors. For instance, while including the console statements at each milestone of the program is not needed—it is certainly recommended for beginner programmers! Using these statements is an asset to identify where errors reside, assists in rectifying those issues, and demonstrates best troubleshooting practices.

Secondly, setting the set editable method as true for the last two textfields could have been implemented instead of setting it to false. However, it may of been confusing to the user (if they should provide input into these empty textfields). Consequently, this would overcomplicate the application and have unintentional consequences. When in doubt, programmers should aim to have the users' experience in mind (by keeping their applications simple).

Lastly, I chose to have my event handler reside within the same file as my main class. I was able to implement this design by not specifying an access modifier on the secondary class. As a result, the class automatically defaults to a package protected access modifier. This allows

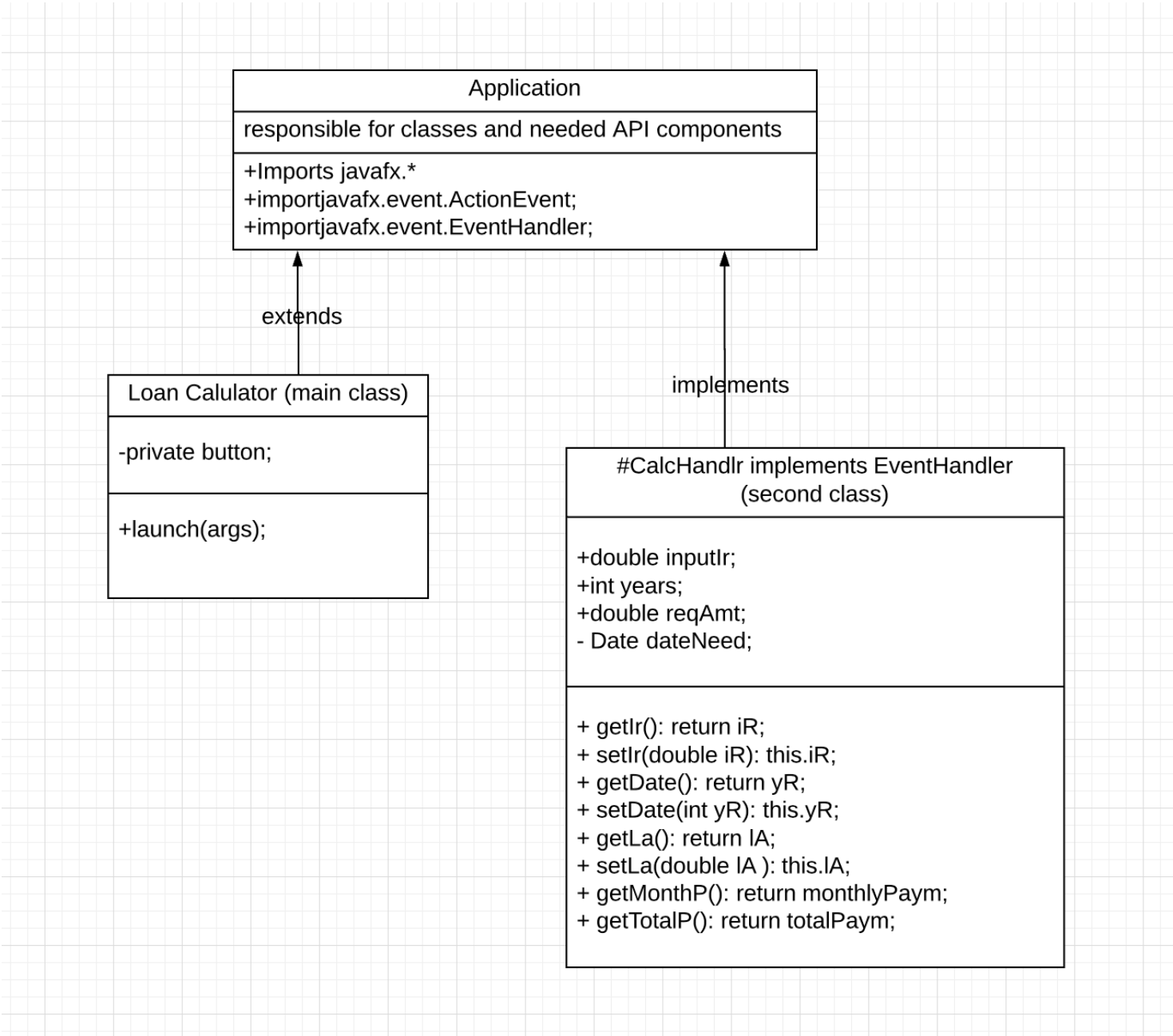
the class to be protected, and prevents other elements of the code from accessing it (unless it is within the same package or inherited). I chose this design feature as it helped me overcome logic errors, and runtime errors in my initial program. As I become more experienced in in Java, I will explore the possibility of including higher level features and separate class files.

## Reflection

When I reflected on my experience in the seminar, I realize that I have come a long way in my coding journey. At the beginning of the term, I did not know how to confidently plan or execute an assignment that required multiple classes. I am grateful to now grasp: when I should create a separate class to perform functions, how to extend and import classes, and how to better troubleshoot my code.

## Conclusion

In closing, this was my favorite Mini Project for the course! It offered the most straightforward implementation of skills and logic discussed during this seminar. Although is was a rather simple and rudimentary, for seasoned programmers, it is no small feat to the novice. I was able to grasp concepts and skills I previously grappled with earlier in the semester. I am hopeful and excited to continue on my coding journey, and look forward to building on my knowledge developed during this summer term.


Application
responsible for classes and needed API components
+Imports javafx.*
+importjavafx.event.ActionEvent;
+importjavafx.event.EventHandler;
extends
implements
Loan Calulator (main class)
-private button;
+launch(args);
#CalcHandlr implements EventHandler
(second class)
+double inputIr;
+int years;
+double reqAmt;
- Date dateNeed;
+ getIr(): return iR;
+ setIr(double iR): this.iR;
+ getDate(): return yR;
+ setDate(int yR): this.yR;
+ getLa(): return lA;
+ setLa(double lA ): this.lA;
+ getMonthP(): return monthlyPaym;
+ getTotalP(): return totalPaym;