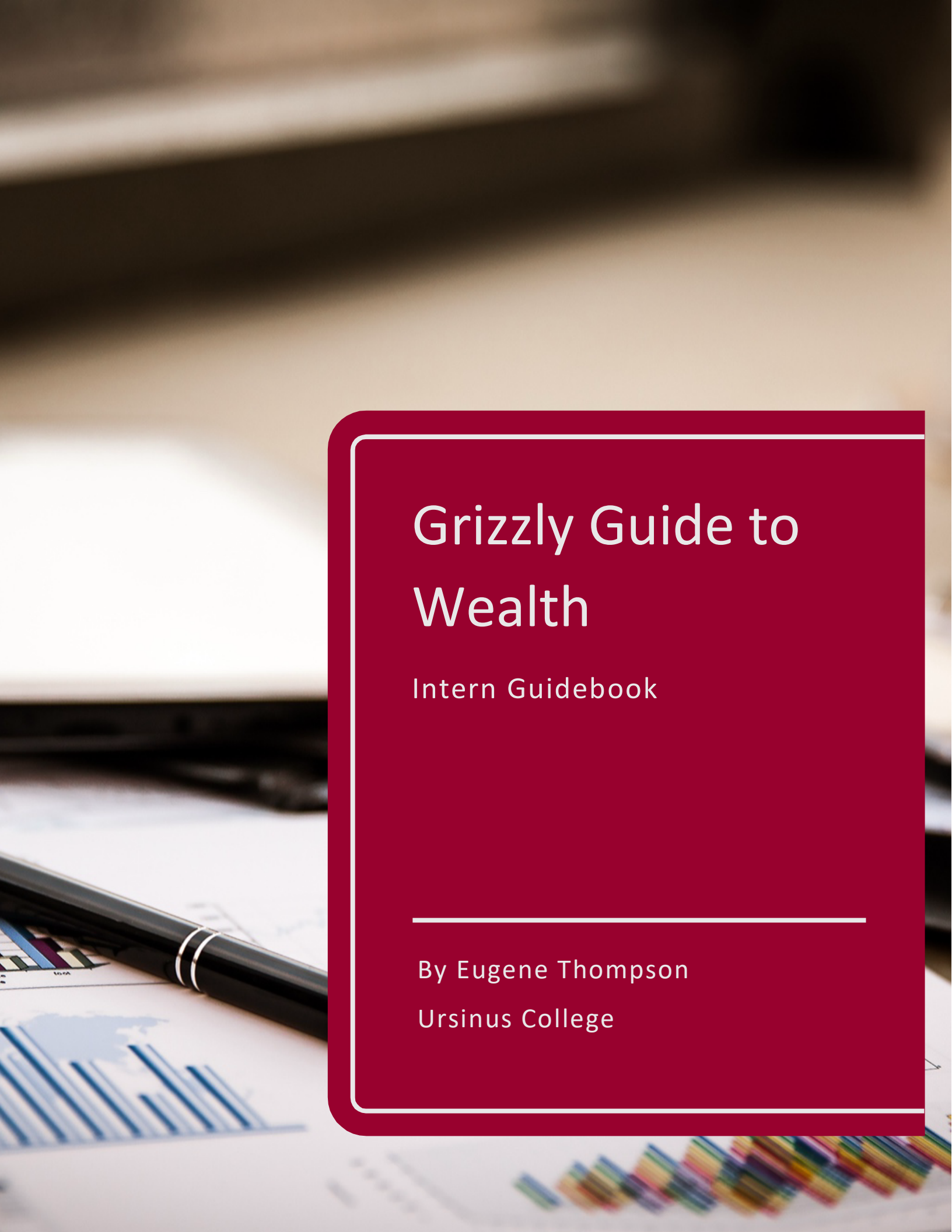


# Grizzly Guide to Wealth

Intern Guidebook

By Eugene Thompson

Ursinus College

# Table of Contents

**Introduction** ..... **4**

*Purpose of the Document* ..... *4*

*Scope of the Document* ..... *4*

**Calculators Architecture Overview** ..... **5**

*Annuity Calculator* ..... *5*

Methods ..... 5

Event Listeners ..... 8

*Expense Calculator* ..... *10*

Methods ..... 10

Event Listeners ..... 13

*Emergency Savings Calculator* ..... *14*

Methods ..... 14

Event Listeners ..... 16

*Future Value Calculator* ..... *18*

Methods ..... 18

Event Listeners ..... 20

*Loan Repayment Calculator* ..... *21*

Methods ..... 21

Event Listeners ..... 22

*Risk Tolerance Questionnaire* ..... *23*

Methods ..... 23

Event Listeners ..... 26

**Future Ideas** ..... **27**

*Introduction of Future Ideas* ..... *27*

*Students' Ideas* ..... *28*

*Faculty and Staff Ideas* ..... *29*

**Resources..... 30**

*Style Guide & External Resources.....30*

*How To Integrate Calculators into Canvas .....30*

# Figures

Figure 1: Annuity Calculator Methods Diagram ..... 5

Figure 2: Annuity Calculator Event Listeners Diagram ..... 8

Figure 3: Expense Calculator Methods Diagram ..... 10

Figure 4: Expense Calculator Event Listeners Diagram..... 13

Figure 5: Emergency Savings Calculator Methods Diagram ..... 14

Figure 6: Emergency Savings Calculator Event Listeners Diagram ..... 16

Figure 7: Future Value Calculator Methods Diagram ..... 18

Figure 8: Future Value Calculator Event Listeners Diagram..... 20

Figure 9: Loan Repayment Calculator Schema Diagram..... 21

Figure 10: Risk Tolerance Questionnaire Methods Diagram ..... 23

Figure 11: Risk Tolerance Questionnaire Event Listeners Diagram..... 26

# Introduction

## Purpose of the Document

This document identifies the various functional elements of the GGTW financial calculators and describes how they are implemented. This document should serve as a guideline for good design practices and a guide to future features. This document also serves as a basis for understanding the financial calculator codebase. Lastly, this document is meant to be a living document, feel free to make edits and changes and expand to the feature ideas, and mark off ones that are completed or create new sections for new features!

## Scope of the Document

Including code documentation, this document lays out all the features that either faculty, staff, or students have suggested in the previous internship. Lastly, this document provides resources that future interns need for this program for good practices in design to maintain the Ursinus College brand and additional tools that were helpful.

# Calculators Architecture Overview

## Annuity Calculator

### Methods

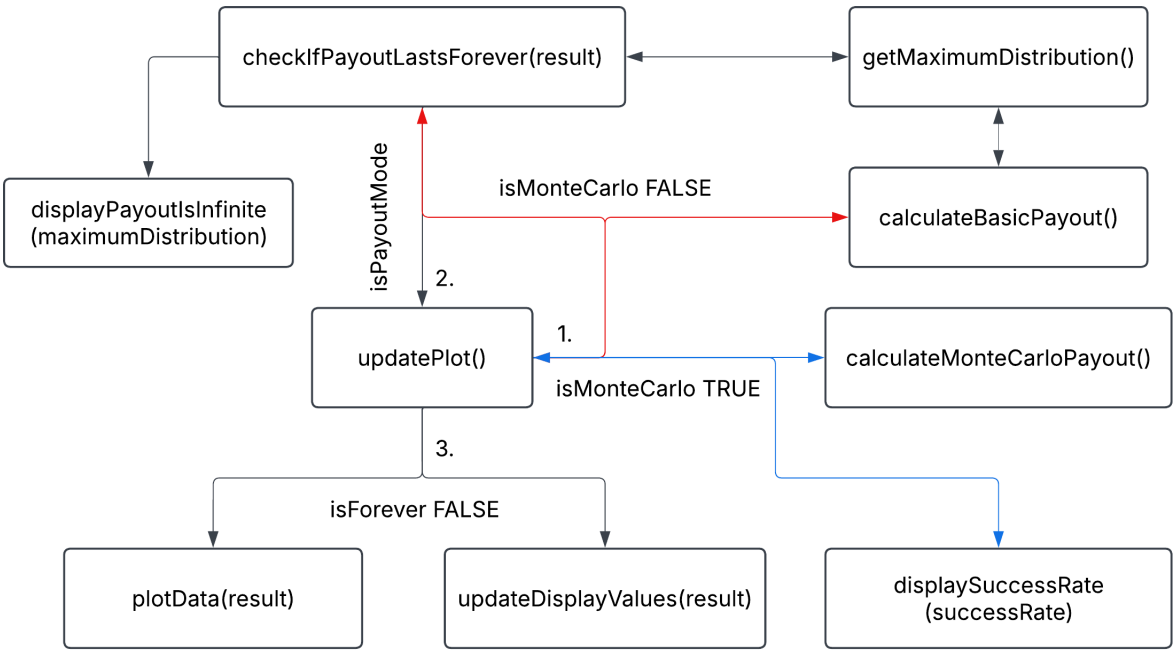



*Figure 1: Annuity Calculator Methods Diagram*

- calculateBasicPayout() return {balanceValues, intValues, maxYear, recommendedPayout}
    - Pre-conditions: Checks if the calculator is in Payout mode or not.
    - Post-conditions: Returns the balance over time as an array (balanceValues), the interest over time as an array (totalInterest), the max year that the annuity lasts, and the recommended payout for the annuity.
- calculateMonteCarloPayout() return {balanceValues, intValues, maxYear, successRate, recommendedPayout}
    - Pre-conditions: None
    - Post-conditions: Returns the balance over time as an array (balanceValues), the interest over time as an array (totalInterest), the

max year that the annuity lasts, the success rate of the 100 trials done, and the recommended payout for the annuity.

- displaySuccessRate(rate) return isAlwaysSuccess
    - Pre-conditions: Check if the Success Rate is at 100%
    - Post-conditions: Displays the success rate if the rate is below 100% and returns a boolean labeled “isAlwaysSuccess” (true if rate is equal to 100, false otherwise)
- updateDisplayValues(result)
    - Pre-conditions: None
    - Post-conditions: Displays a withdraw message based on the Payout Mode
- getMaximumDistribution()
    - Pre-conditions: None
    - Post-conditions: Return the maximum distribution that the annuity can payout over periods until age 100.
- checkIfPayoutLastsForever(result) return isInfinite
    - Pre-conditions: Checks if the payout amount is less than or equal to the maximum distribution that can be given in the period.
    - Post-conditions: Return if the payout amount can be paid out infinitely or not as a boolean.
- displayPayoutIsInfinite(maximumDistribution)
    - Pre-conditions: Check if the payout is under the age limit, or if it’s at the limit, and if the annuity will last forever, and check if the annuity lasts above age 100.
    - Post-conditions: Displays if the annuity lasts forever, if not it displays how long the annuity lasts up with how many years.
- plotData(result)
    - Pre-conditions: None.
    - Post-conditions: Displays the plot based on the payout option and the calculation mode.
- updatePlot()

- Pre-conditions: Check if the calculator is in Monte Carlo mode or what Calculation Mode is on or off.
- Post-conditions: Displays plot if the payout does not last forever, and shows the appropriate text needed for the user

# Event Listeners

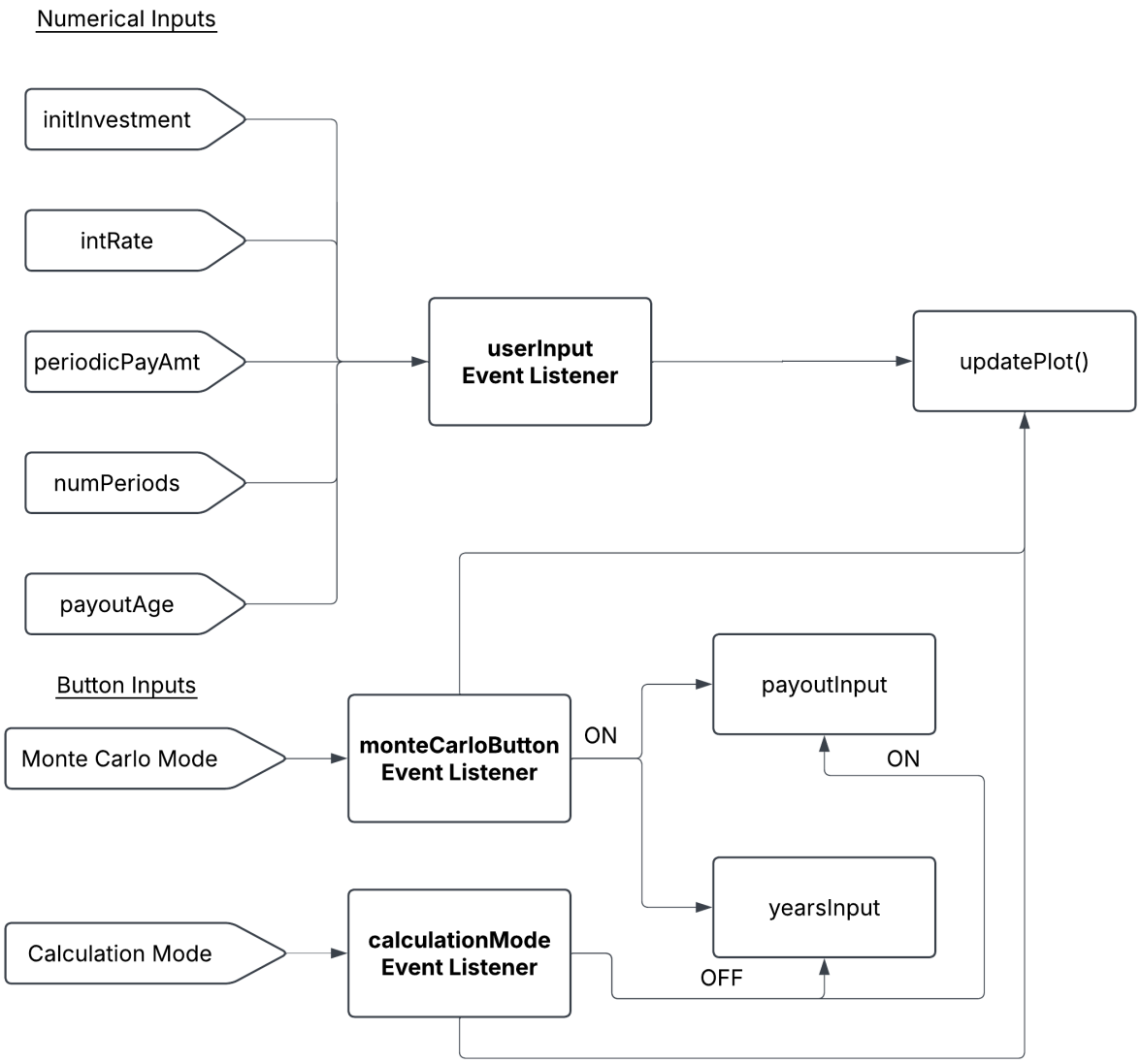



*Figure 2: Annuity Calculator Event Listeners Diagram*

- userInput
    - Pre-conditions: Check if the input is from the number input and checks if the number input has a value greater than or equal to 0, and that the input has at least one number
    - Post-conditions: If the input is a valid numerical input, then it updates the plot updatePlot().
- monteCarloButton

    - Pre-conditions: Checks if the calculation mode is checked if the button is unchecked.
    - Post-conditions: Displays both the year and payout inputs if Monte Carlo is selected, Otherwise, toggles the two inputs based on what calculation mode is selected. Regardless, it updated the plot through updatePlot().
- calculationMode
    - Pre-conditions: Checks if the Monte Carlo switch is on or off
    - Post-conditions: Display the payout input if the button is in the “on” position, and the years input if the button is in the “off” position

# Expense Calculator

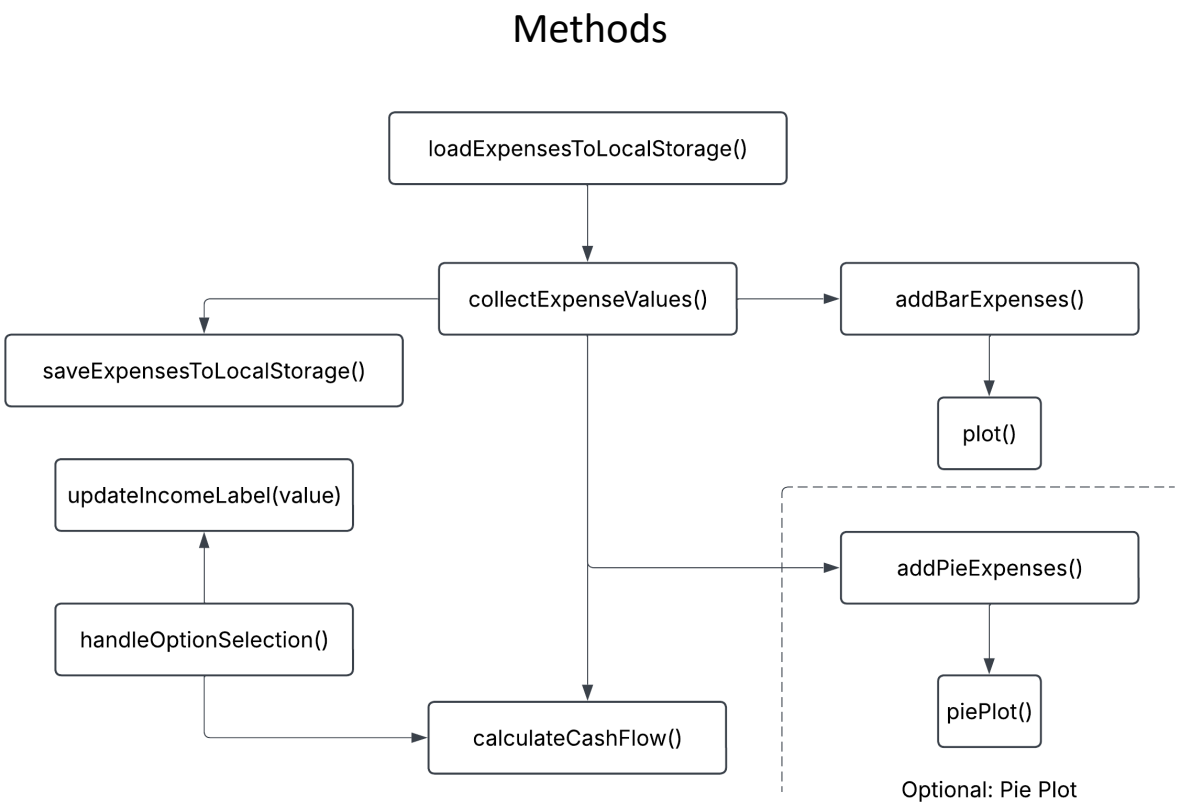



*Figure 3: Expense Calculator Methods Diagram*

- handleOptionSelection(e)
    - Pre-conditions: Check if the tagName is “LI” (referring to the Month, Semester, and Yearly Expense timeframe in the list element).
    - Post-conditions: Displays the drop-down menu and sets the timeframe to the input, and update income labels and multipliers and calculates cash flow. If the income is available, then it plots the expenses.
- updateIncomeLabel(timeframe)
    - Pre-conditions: Checks if the timeframe is monthly, semester, or yearly, or a default value.
    - Post-conditions: Changes the income label or month multipliers, or semester multipliers for expenses.

- collectExpenseValues()
    - Pre-conditions: None.
    - Post-conditions: Returns a dictionary with the id of the expense, the value, the category, and the name.
- addBarExpenses(data) return data
    - Pre-conditions: None.
    - Post-conditions: Returns plot data with the addition of every expense included.
- addPieExpenses() return [values, labels, colors]
    - Pre-conditions: None.
    - Post-conditions: Returns values of the expenses, savings, and remaining income, alongside labels and colors.
- piePlot()
    - Pre-conditions: None
    - Post-conditions: Displays a pie plot with the remaining income, savings, and expenses.
- plot()
    - Pre-conditions: None
    - Post-conditions: Displays a bar graph with the income, savings, and expenses.
- calculateCashFlow()
    - Pre-conditions: Checks if the timeframe has been specified and checks the amount of “netIncome” left after expenses.
    - Post-conditions: Displays how much the user has spent relative to income, and if the user overspends, it gives a suggestion to cut expenses
- saveExpensesToLocalStorage()
    - Pre-conditions: None.
    - Post-conditions: Saves the expense data into the user’s local storage in their browser
- loadExpensesFromLocalStorage()

- Pre-conditions: Checks if expenses are saved in the browser’s local storage
- Post-conditions: Loads the expenses into the user input

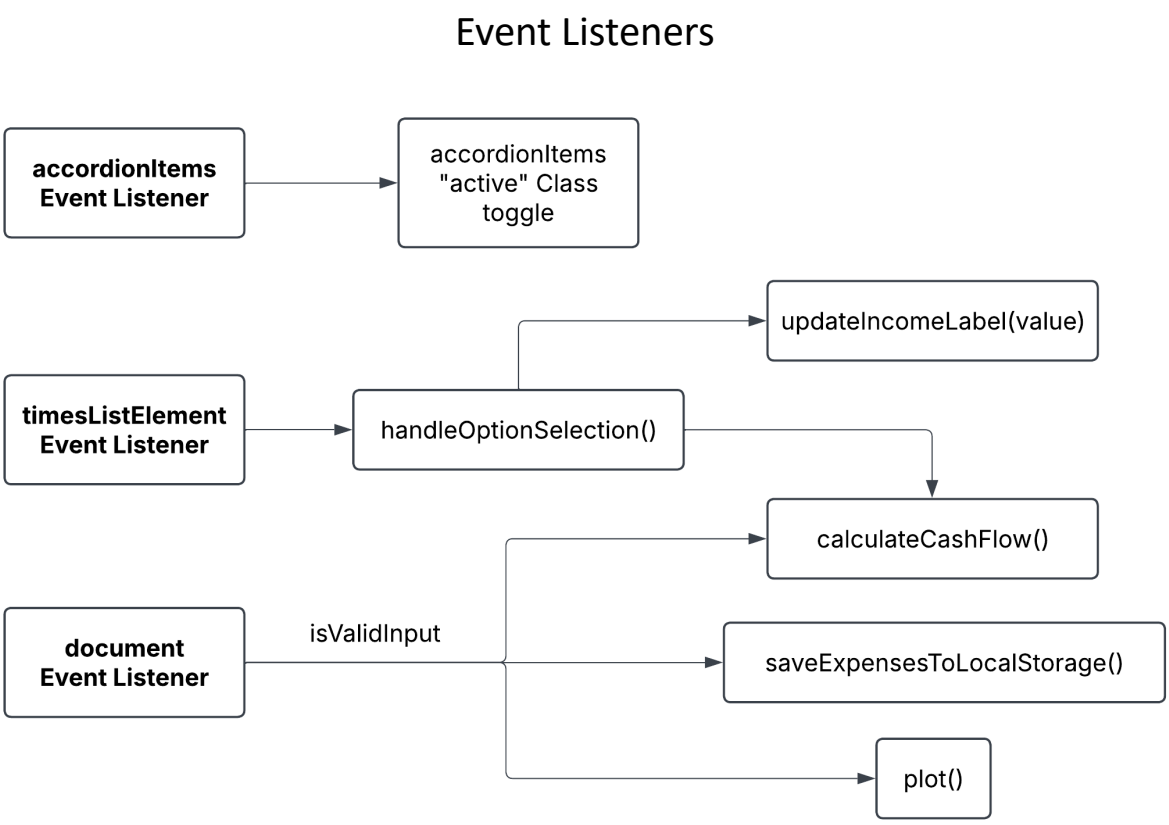



*Figure 4: Expense Calculator Event Listeners Diagram*

- accordionItems
  - Pre-conditions: None.
  - Post-conditions: Closes all the other accordion items and shows only the clicked item.
- timesListElement (refers to the Timeframe Element)
  - Pre-conditions: Checks if the user clicks on the Timeframe input.
  - Post-conditions: Handles the click through handleOptionSelection() function
- document
  - Pre-conditions: Check if the input is from the number input and checks if the number input has a value greater than or equal to 0, and that the input has at least one number.
  - Post-conditions: If the input is valid, then it saves the value to local storage, calculates the cash flow, and plots the new data.

# Emergency Savings Calculator

## Methods

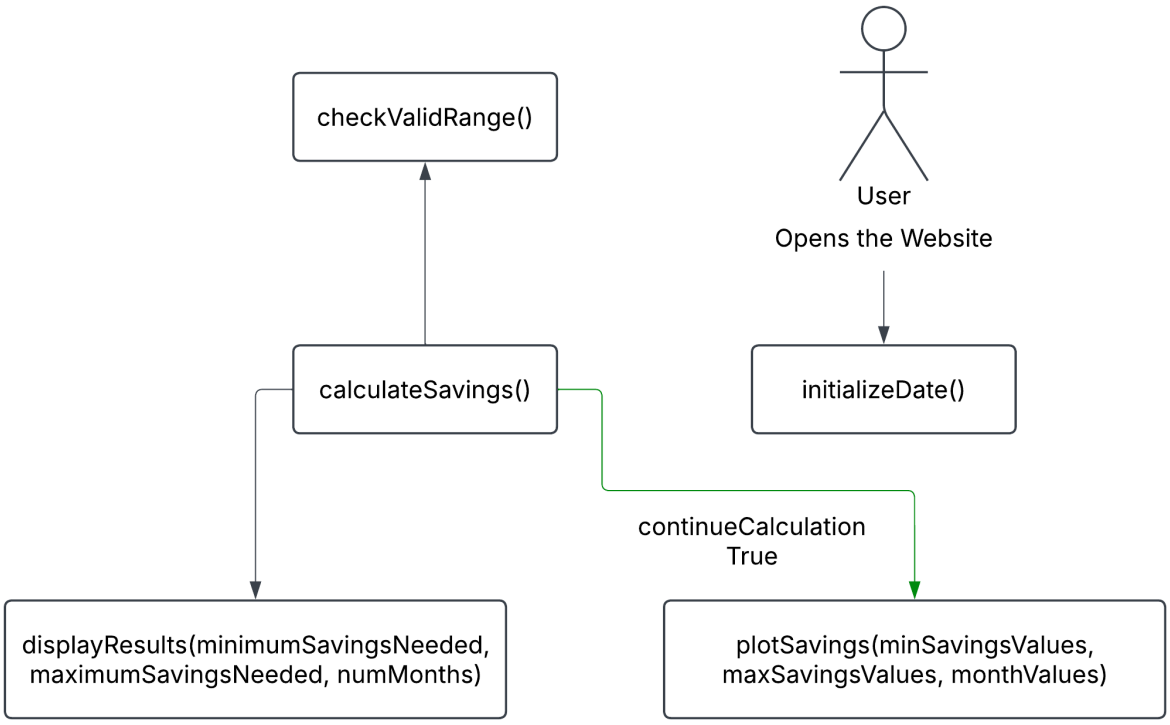



*Figure 5: Emergency Savings Calculator Methods Diagram*

- checkValidRange() return isValid
    - Pre-conditions: Checks if the starting month and year are less than the goal month and year or if the dates are initialized, and if there's the same date or there's an error (goal month and year are before the start month and year)
    - Post-conditions: Displays an error message if any of the pre-conditions are triggered and returns if the date range is valid.
- initializeDate()
    - Pre-conditions: None.
    - Post-conditions: Initializes the year inputs based on the current year (so you don't have to update the year every year).
- calculateSavings()

    - Pre-conditions: Checks if the calculator should continue plotting the results based on if the date range is valid or not
    - Post-conditions: Plots the savings over time if the date range is correct. Regardless, it will display how much to save for three and six months of expenses.
- displayResults(minimumSavingsNeeded, maximumSavingsNeeded, numMonths)
    - Pre-conditions: Checks if the number of months in between the two date ranges is not 0
    - Post-conditions: Displays how much the user should save in total and only shows how much they need to save every month if there is a valid date range.
- plotSavings(minSavingsValues, maxSavingsValues, monthValues)
    - Pre-conditions: None.
    - Post-conditions: Displays a line graph with the minimum and maximum savings targets with the range highlighted in green.

## Event Listeners

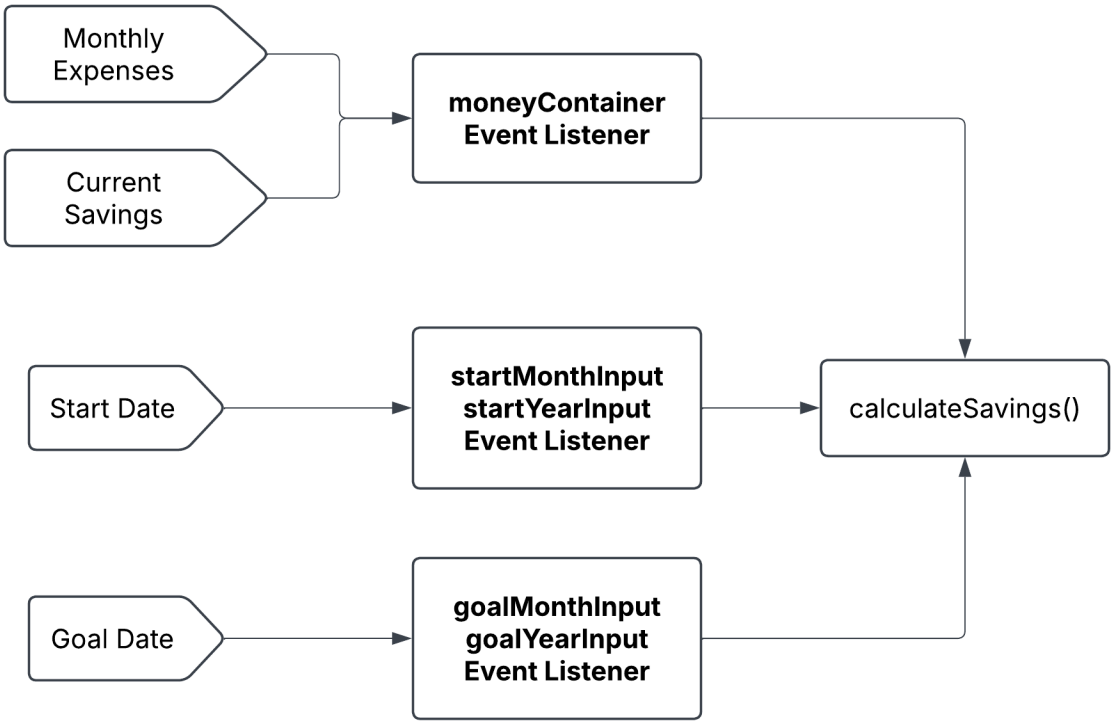



*Figure 6: Emergency Savings Calculator Event Listeners Diagram*

- moneyContainer
  - Pre-conditions: Check if the number input has a value greater than or equal to 0, and that the input has at least one number.
  - Post-conditions: Calls the calculateSavings() function if the input is valid.
- startMonthInput
  - Pre-conditions: None.
  - Post-conditions: Calls the calculateSavings() function.
- startYearInput
  - Pre-conditions: Check if the input is the current year or if the month is before the current month.
  - Post-conditions: Adjusts the start month input values based on the year and current month. Then, it calls the calculateSavings() function.
- goalMonthInput

    - Pre-conditions: None.
    - Post-conditions: Calls the calculateSavings() function.
- goalYearInput
    - Pre-conditions: Check if the input is the current year or if the month is before the current month.
    - Post-conditions: Adjusts the goal month input values based on the year and current month. Then, it calls the calculateSavings() function.

# Future Value Calculator

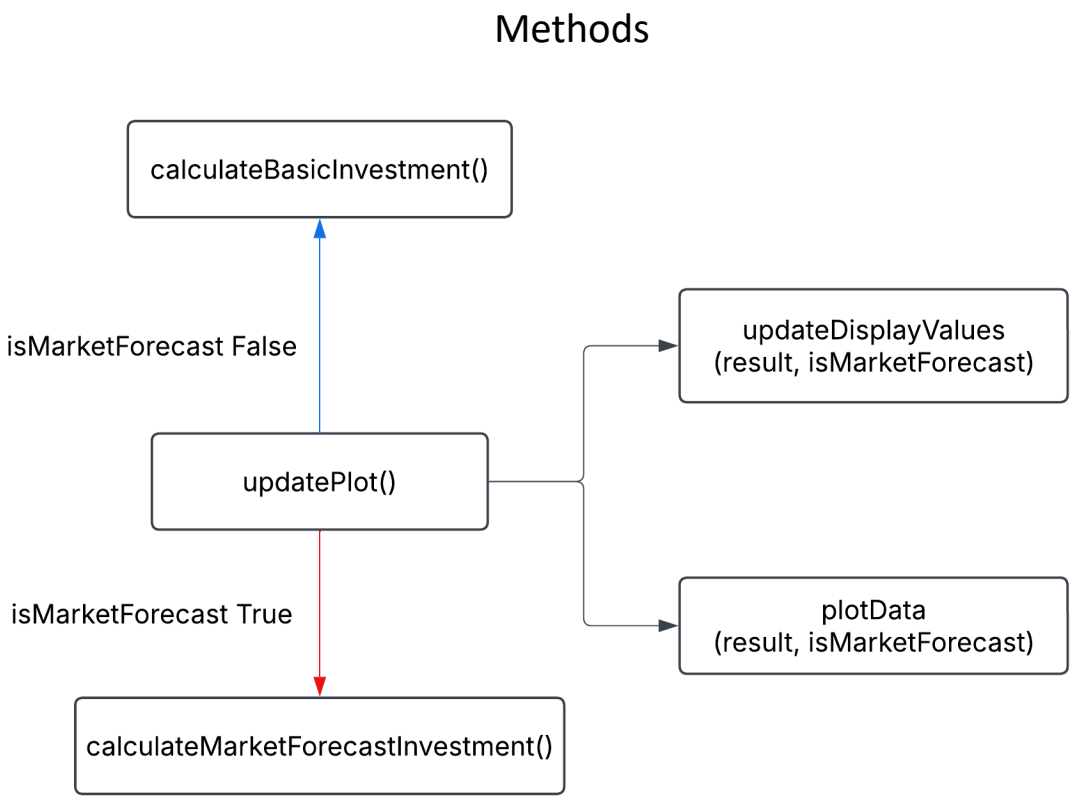



*Figure 7: Future Value Calculator Methods Diagram*

- calculateBasicInvestment()
  - Pre-conditions: None
  - Post-conditions: Returns the balance and interest over periods specified by the user.
- calculateMarketForecastInvestment()
  - Pre-conditions: None
  - Post-conditions: Returns the balance and interest over periods specified by the user.
- updateDisplayValues(result, isMarketForecast)
  - Pre-conditions: Checks if the calculator is in Market Forecast mode or not.

    - Post-conditions: If the calculator is in the Market Forecast mode, then it displays the average values for the ending balance and interest, otherwise it shows total.
- plotData(result, isMarketForecast)
    - Pre-conditions: Checks if the calculator is in Market Forecast mode or not.
    - Post-conditions: Shows a plot based on year for Market Forecast mode, and period for when the mode is turned off.
- updatePlot()
    - Pre-conditions: Checks if the mode is the basic mode or forecast mode.
    - Post-conditions: If it's the basic mode it calculates the basic investment calculations, otherwise it calculates using a market forecast based on Monte Carlo.

Event Listeners

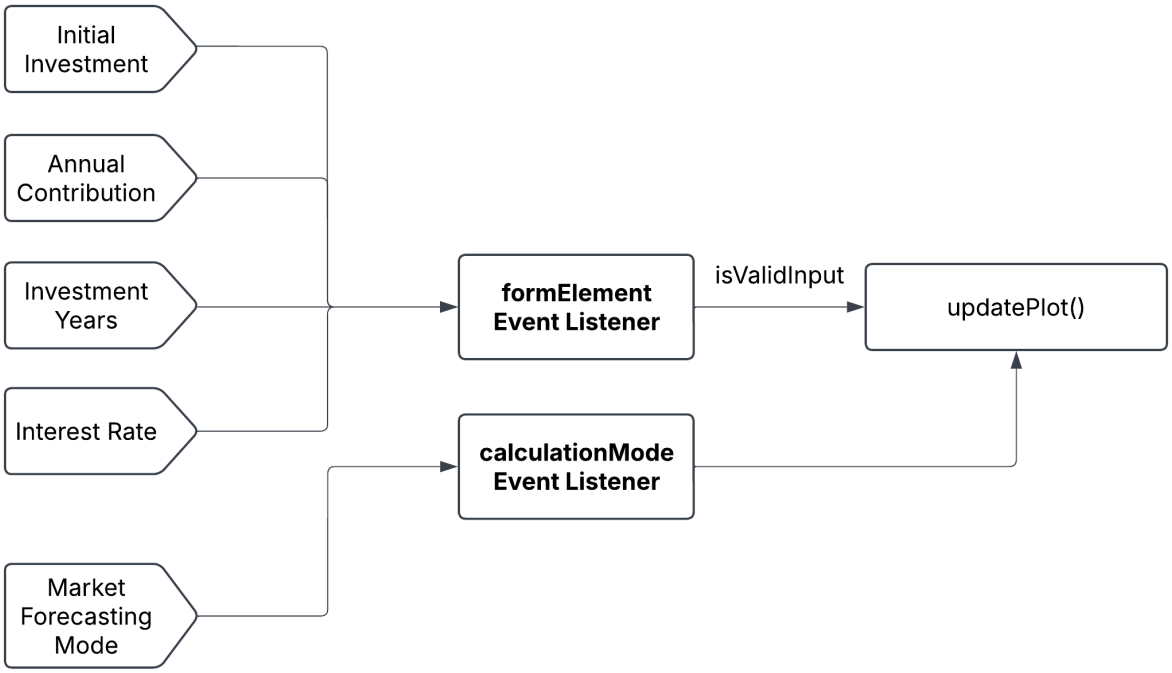



*Figure 8: Future Value Calculator Event Listeners Diagram*

- formElement
    - Pre-conditions: Check if the input is from the number input and checks if the number input has a value greater than or equal to 0, and that the input has at least one number. It also checks if the year is a valid year and rounds the value of the year.
    - Post-conditions: If the input is valid, it continues to the updatePlot() function.
- calculationMode
    - Pre-conditions: None.
    - Post-conditions: It calls the updatePlot() function to handle the input.

# Loan Repayment Calculator

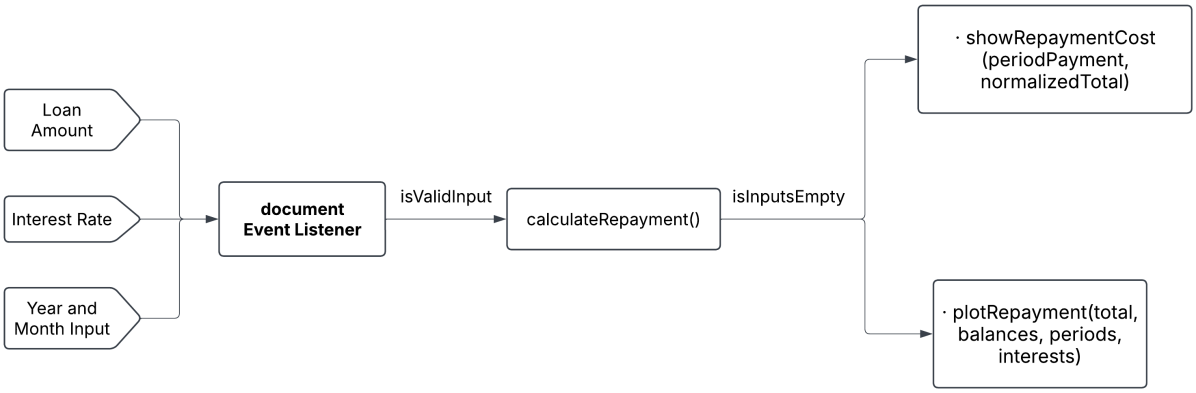



*Figure 9: Loan Repayment Calculator Schema Diagram*

## Methods

- calculateRepayment()
  - Pre-conditions: Checks if the inputs are empty
  - Post-conditions: Calls showRepaymentCost and plotRepayment after calculating repayment and if the inputs are not empty.
- showRepaymentCost(periodPayment, normalizedTotal)
  - Pre-conditions: Checks if both the periodPayment and the normalizedTotal are numbers
  - Post-conditions: Shows a summary if the periodPayment and the normalizedTotal are numbers
- plotRepayment(total, balances, periods, interests)
  - Pre-conditions: Checks if both the periodPayment and the normalizedTotal are numbers
  - Post-conditions: Shows the plot if the periodPayment and the normalizedTotal are numbers

## Event Listeners

- document
    - Pre-conditions: Check if the input is from the number input and checks if the number input has a value greater than or equal to 0, and that the input has at least one number.
    - Post-conditions: Calls the calculateRepayment() function if the input is valid

# Risk Tolerance Questionnaire

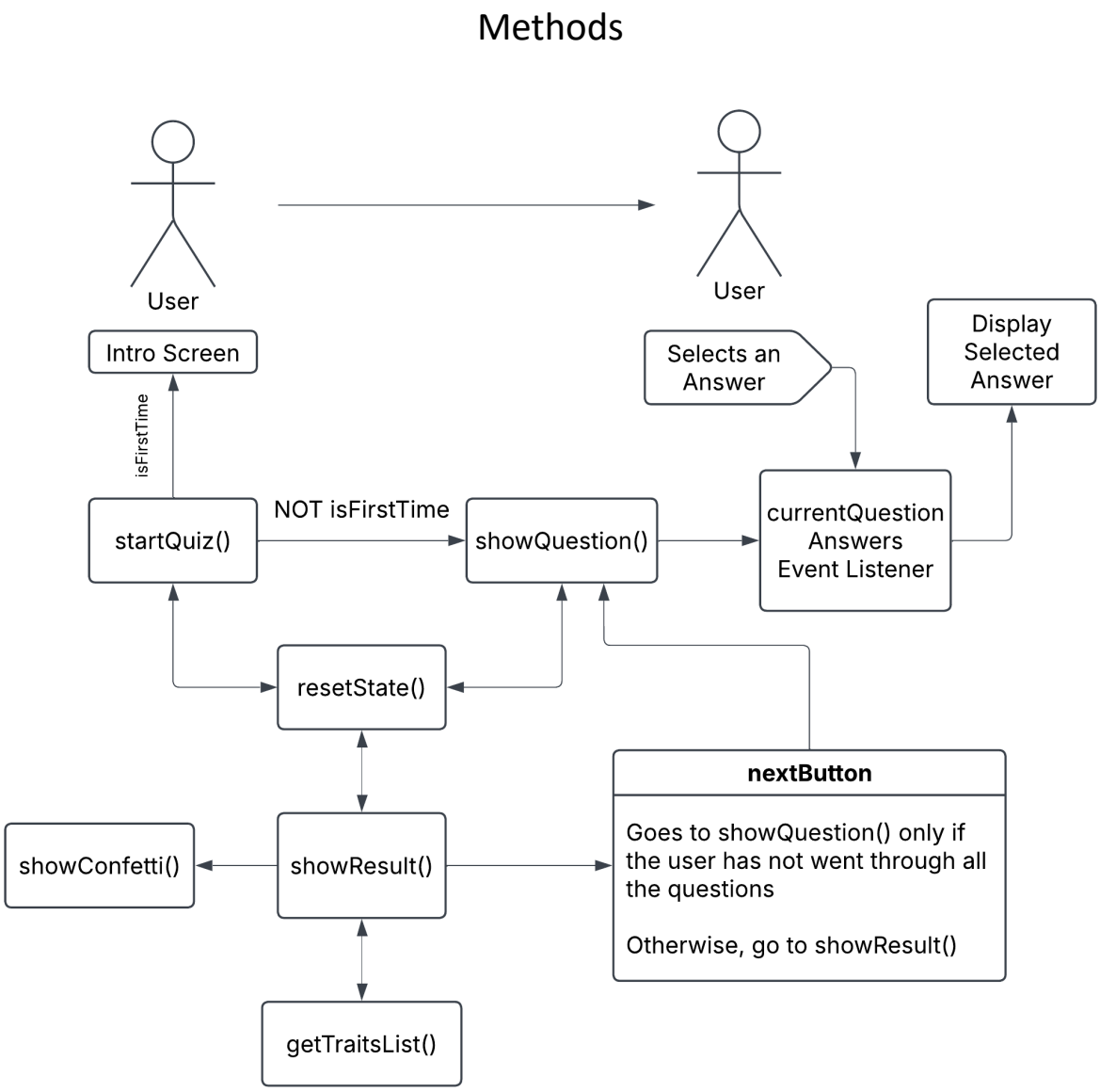



*Figure 10: Risk Tolerance Questionnaire Methods Diagram*

- startQuiz()
    - Pre-conditions: Check if the user is starting the quiz for the first time or not
    - Post-conditions: Show the introduction info screen if it's the first time, otherwise, skips to the first question in the quiz
- resetState()

    - Pre-conditions: None
    - Post-conditions: Displays the next button if the user is at the introduction, otherwise it doesn't. It also removes all the answers (buttons) on the question.
- showQuestion()
    - Pre-conditions: None
    - Post-conditions: Displays the question and the question choices for each button and makes them interactable through EventListener.
- selectAnswer(e)
    - Pre-conditions: None
    - Post-conditions: Removes the "selected" class on the button if a button was already pressed and has the user-selected button with the "selected" class. Regardless, the point total is adjusted in the overall calculation.
- showResult()
    - Pre-conditions: Checks if the score is within the three investment risk ranges: Conservative, Moderate, and Aggressive.
    - Post-conditions: Displays the results of the user's risk tolerance based on the score ranges.
- getTraitsList() return String
    - Pre-conditions: None
    - Post-conditions: Returns a HTML formatted bullet point for each trait of the Investor's Tolerance Level
- showConfetti()
    - Pre-conditions: None
    - Post-conditions: Displays confetti on screen based on the Ursinus colors specified in the Visual System.
- handleNextButton()
    - Pre-conditions: Checks if the user is at the introduction page, if not then it continues to the next question, otherwise it just displays the question and hides the introduction elements from the user.

    - Post-conditions: Displays the next question and shows the result if the user has gone through all the questions.
- handlePrevButton()
    - Pre-conditions: Check if the user is not in the introduction screen and that the current question is not the first question.
    - Post-conditions: Displays the previous question.

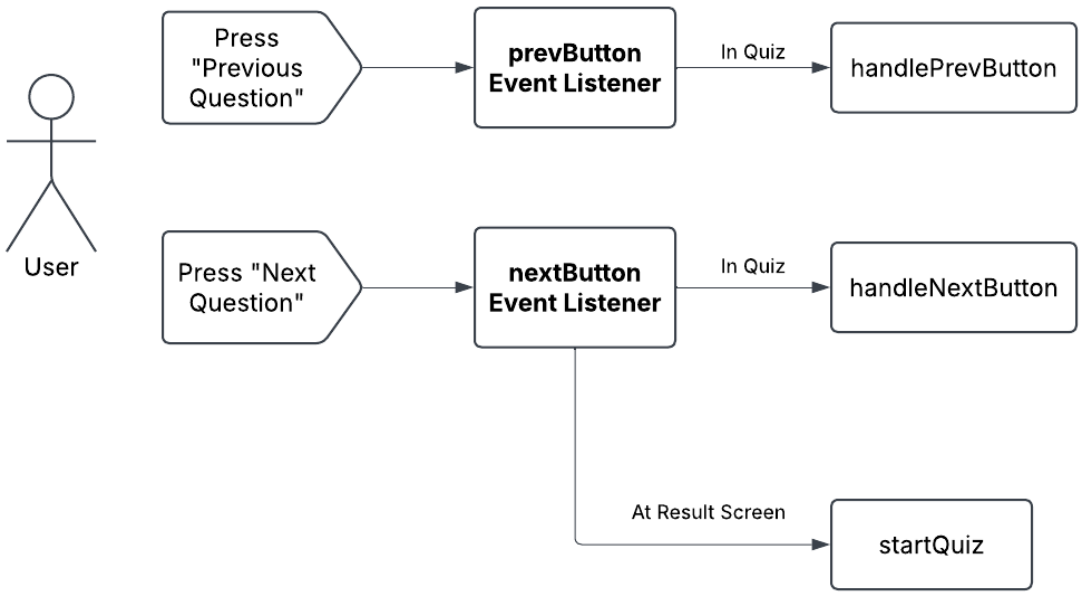



*Figure 11: Risk Tolerance Questionnaire Event Listeners Diagram*

- nextButton
  - Pre-conditions: Checks if the user is finished with the quiz or not.
  - Post-conditions: Handles the button accordingly, if the user is at the end of the quiz, then it sets the “isFirstTime” variable to false, otherwise it handles the button to the handleNextButton function.
- prevButton
  - Pre-conditions: Checks if the user is finished with the quiz or not.
  - Post-conditions: Handles the button according to handlePrevButton.

# Future Ideas

## Introduction of Future Ideas

This section primarily shows all the ideas of Students, Faculty and Staff that were collected as of Spring 2025, this is the part that should dynamically change across interns. Feel free to strikethrough any ideas that have been implemented and add new ideas.

Since this program is campus-wide, it’s a great idea while you are doing this internship to ask around to fellow students, and any faculty or staff that you know for suggestions on how to further improve the program!

## Students' Ideas

- Club or Peer Coaches and Program Associates to help students with Financial Literacy.
- Archiving Excess Canvas Pages through the Canvas API.
- Theming the Landing Page with Ursinus Style in Mind.
- Improving Calculators:
    - Add Cookies/Local Storage to save previous entries
        - User Accounts Tied to Canvas Accounts
    - Using a graphing library that would be look like D3 or Apache eCharts…
- Implementing YellowDig for Social Engagement:
    - https://www.yellowdig.co/post/8-tips-for-high-quality-open-conversations
- Using a Search Engine Database like:
    - Chroma: https://trychroma.com
        - Logistics: Integrating a Server-Based Database on a Static Website like Canvas…
    - Canvas Smart Search: https://community.canvaslms.com/t5/Smart-Search-Feature-Preview/Smart-Search-FAQ/ta-p/604415
        - Logistics: Waiting for the Feature to get out of "Feature Preview"

## Faculty and Staff Ideas

- Combining the Loan Repayment and Future Value Calculator to determine which option is better when making a large purchase[1]
- Adding an additional payment during the Loan Repayment calculator to see how much it changes accordingly.
- Social Engagement Platform:
    - Students list their future life goals and aspirations and frame financial goals around them…
    - The platform should be moderated by compassionate administrators who can link students to a financial expert when needed.
        - "Preferred Professionals" (attorneys, accountants, financial advisors, etc.)
    - Private channels should be available for students who have sensitive financial questions as well.
- Prioritizing Financial Wellness
    - Make the platform feel less "Finance Bro," and more Financial Self-Care.
    - Make elements feel welcoming to students, not too intimidating, and prioritize having thoughtful use of information and graphics.

[1] This suggestion is more in the realm of financial planning, and would likely be hard to integrate carefully, as this requires professional advice.

# Resources

## Style Guide & External Resources

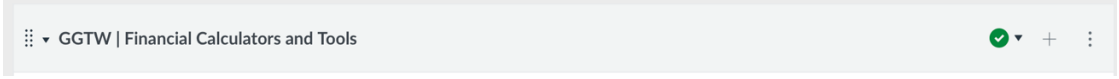
The link to Ursinus' Brand Hub is here!

It is worth noting that this entire document is using the fonts that are specified in the visual system.

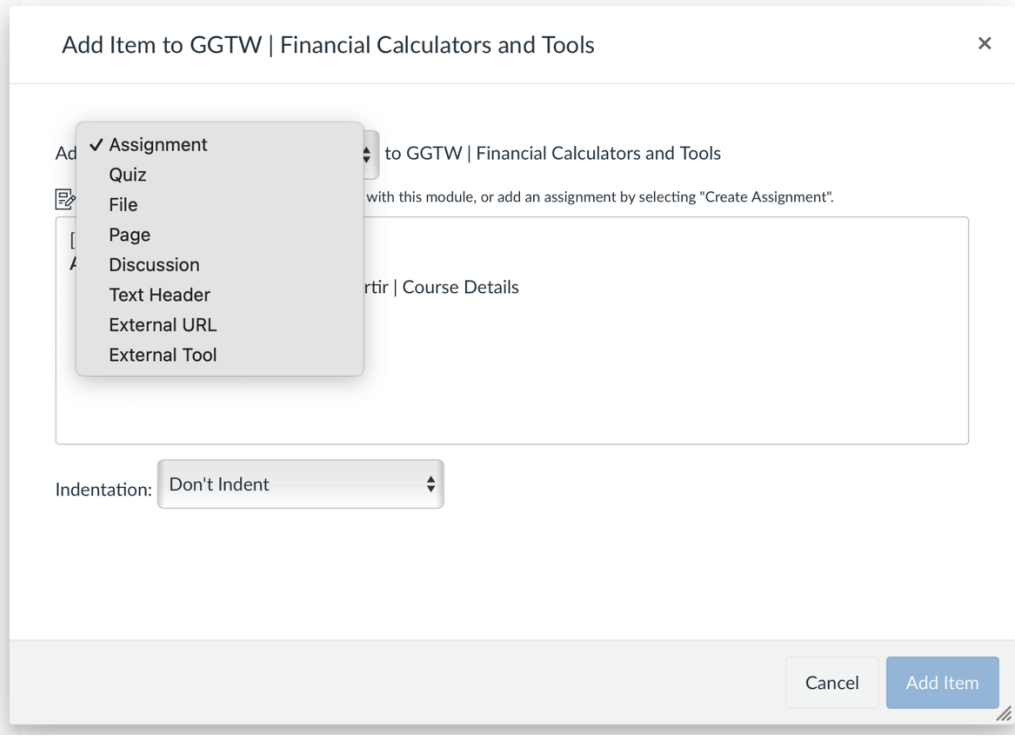
For Plotly used in the Calculators, please check out the documentation here!

## How To Integrate Calculators into Canvas

1. Go to GGTW under the Files Tab and then upload the respective .html file into the folder labeled "Eugene's Calculators"
2. If you are adding a new calculator, you need to create a new Canvas page
    a. First, right-click the html file and copy the link, it will look something like this:
        https://ursinus.instructure.com/files/2522126/download?download_frd=1
        i. You want to remove everything from the end up to the first instance of download, so it should look something like this: https://ursinus.instructure.com/files/2522126/download
        ii. Once you get that, you should copy it to your clipboard!
    b. Second, go to Modules, then go to the respective header you wish to put the calculator under (usually this is under "GGTW | Financial Calculators and Tools)
    c. Then, you will see something like this:

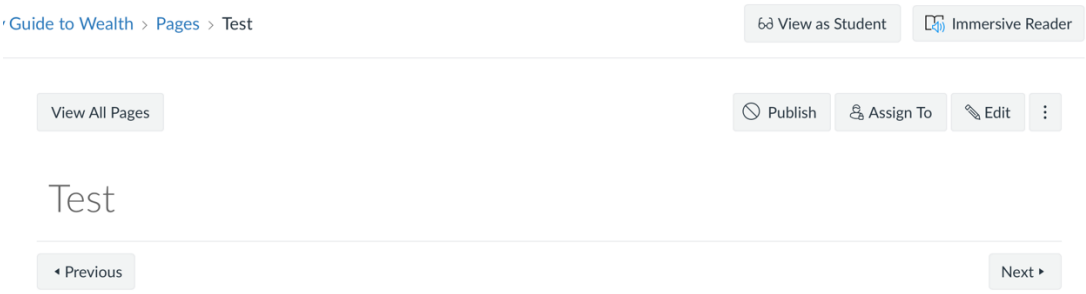


d. Click the '+' button next to the checkmark, then go to the drop down and select 'Page'

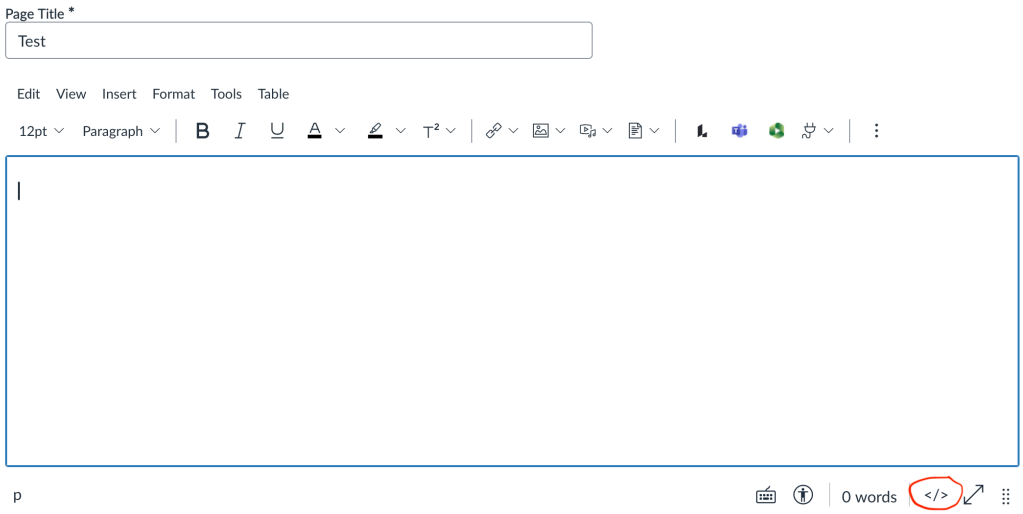



e. After selecting 'Page,' click on "[ Create Page]" and then put your Page Name, and set Indentation to "Indent 1 Level"

f. Next, click "Add Item" and go to the page you just created, you will see something like this:



g. Click “Edit”, then click on the button circled in red below:




h. Paste the following code in, making sure to change the src link and the height specification accordingly:

```
<iframe src="<Insert your Calculator Link Here>"
width="100%" height="<Adjust to 1000 or higher>"></iframe>
```

3. Otherwise, if you are not adding a new calculator, simply click “Replace” when prompted after uploading the file.