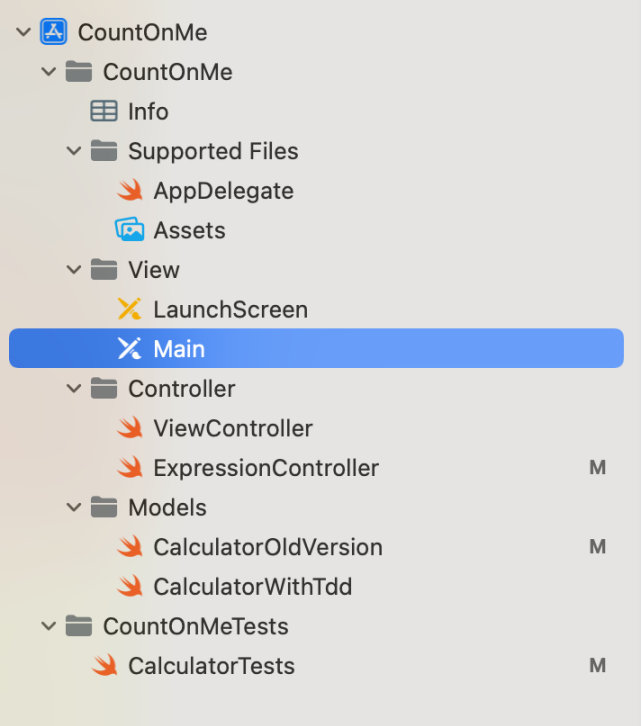
# Rapport Projet 5 - OpenClassRoom CountOnMe

Bar Marc-Antoine 10/2022

## 1. Technologie

- Xcode 14
- Github  / git
- Iterm2 + oh-my-zsh
- https://www.hackingwithswift.com/
- swiftLint
- lien gitHub https://github.com/bluholm/Openclassroom-countOnMe

## 2. Architecture




## 3. Le controlleur

```
//
//  ViewController.swift
//  SimpleCalc
//
//  Created by Vincent Saluzzo on 29/03/2019.
//  Copyright (c) 2019 Vincent Saluzzo. All rights reserved.
//



import UIKit

class ViewController: UIViewController {

    private var calculator = CalculatorTdd()


    @IBOutlet weak var textView: UITextView!
    @IBOutlet var numberButtons: [UIButton]!

    // MARK: - Life cycle methods
```

```
    override func viewDidLoad() {
        super.viewDidLoad()

        textView.text = ""
    }

    // MARK: - Users action

    @IBAction func tappedNumberButton(_ sender: UIButton) {

        guard let numberText = sender.title(for: .normal) else {
            return
        }

        if expressionHaveResult {
            textView.text = ""
        }

        if numberText == "0" && !divisionByzero{
            let alertVC = UIAlertController(title: "Error!", message: "Division  by 0
impossible", preferredStyle: .alert)
            alertVC.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
            self.present(alertVC, animated: true, completion: nil)
        } else {
            textView.text.append(numberText)
        }

    }

    @IBAction func tappedAdditionButton(_ sender: UIButton) {
        if canAddOperator && isFirstoperator {
            textView.text.append(" + ")
        } else {
            alertOperatorAlreadyExist()
        }

    }

    @IBAction func tappedSubstractionButton(_ sender: UIButton) {
        print("\(elements.count)")
        if canAddOperator && isFirstoperator {
            textView.text.append(" - ")
        } else {
            alertOperatorAlreadyExist()
        }

    }


    @IBAction func tappedMultiplyButton(_ sender: UIButton) {
        if canAddOperator && isFirstoperator {
            textView.text.append(" * ")
        } else {
            alertOperatorAlreadyExist()
        }
    }
```

```
    @IBAction func tappedDivideButton(_ sender: UIButton) {
        if canAddOperator && isFirstoperator {
            textView.text.append(" / ")
        } else {
            alertOperatorAlreadyExist()
        }
    }

    @IBAction func tappedEqualButton(_ sender: UIButton) {
        guard expressionIsCorrect else {
            let alertVC = UIAlertController(title: "Error!", message: "please write a
good operation", preferredStyle: .alert)
            alertVC.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
            return self.present(alertVC, animated: true, completion: nil)
        }

        guard expressionHaveEnoughElement else {
            let alertVC = UIAlertController(title: "Error !", message: "not enough
elements", preferredStyle: .alert)
            alertVC.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
            return self.present(alertVC, animated: true, completion: nil)
        }

        guard let result = calculator.calculateAll(elements).first else { return }
        textView.text.append(" = \(result)")
    }

    @IBAction func tappedResetButton(_ sender: UIButton) {
        textView.text = ""
    }

    // MARK: - Custom Methods

    private func alertOperatorAlreadyExist() {
        let alertVC = UIAlertController(title: "Error!", message: "operator error",
preferredStyle: .alert)
        alertVC.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
        self.present(alertVC, animated: true, completion: nil)
    }

}
```

les autres fichiers du contrôleur seront vus lors de la présentation orale .

# 4. La vue

**Les contraintes :**

**Les stacks Views**

# 5. Le Modèle

## L’ancien modèle :

```
/
//  Calcul.swift
//  CountOnMe
//
//  Created by Marc-Antoine BAR on 2022-09-18.
//  Copyright (c) 2022 Vincent Saluzzo. All rights reserved.


import Foundation

class Calculator {

    ///function  calculate the result
    func getResultTotalOperation(_ elements: [String]) -> [String] {

        var copyOfElements = elements
        copyOfElements = getResultOfPriorityOperations(elements, "*")
        copyOfElements = getResultOfPriorityOperations(copyOfElements, "/")
        copyOfElements = getResultsOfMinorOperations(operations: copyOfElements)
        return copyOfElements
    }

    ///execution of division & multiplier first
    private  func getResultOfPriorityOperations(_ operations: [String],_ operatorSign:
String) -> [String] {

        var operationsTemp = operations
        while operationsTemp.contains(operatorSign){
            var  temporaryResult: Double
            let indexOfMultiplier = operationsTemp.firstIndex(of: operatorSign)!
            let left = Double(operationsTemp[indexOfMultiplier-1])!
            let right = Double(operationsTemp[indexOfMultiplier+1])!

            if operatorSign == "*" {
                temporaryResult = left * right
            } else {
                temporaryResult = left / right
                temporaryResult = round(temporaryResult * 100) / 100
            }
            operationsTemp = returnResultOfOperation(operations: operationsTemp,result:
temporaryResult,index: indexOfMultiplier)
        }
        return operationsTemp
    }

    ///reduce minus to let elements with only the good information ( plus and minus )
    private func getResultsOfMinorOperations(operations: [String]) -> [String]{

        var operationsTemp = operations
        // Iterate over operations while an operand still here
```

```
        while operationsTemp.count > 1 {
            let left = Double(operationsTemp[0])!
            let operand = operationsTemp[1]
            let right = Double(operationsTemp[2])!

            let result: Double
            switch operand {
            case "+": result = left + right
            case "-": result = left - right
            default: result = 0
            }

            operationsTemp = Array(operationsTemp.dropFirst(3))
            operationsTemp.insert("\(result)", at: 0)
        }
        return operationsTemp
    }

    /// delete 3 cols A-B-C and replace by the result of operations a A op C
    private func returnResultOfOperation(operations: [String],result: Double,index:
Int)->[String]{

        var operationsResult = operations
        operationsResult.remove(at: index)
        operationsResult.insert("\(result)", at: index)
        operationsResult.remove(at: index+1)
        operationsResult.remove(at: index-1)
        return operationsResult
    }
}
```

codé en full TDD :

```
/
//  CalculatorTdd.swift
//  CountOnMe
//
//  Created by Marc-Antoine BAR on 2022-09-22.
//  Copyright (c) 2022 Vincent Saluzzo. All rights reserved.
//

import Foundation

class CalculatorTdd {

    func additionByArray(_ newElement: [String]) -> [String] {
        let result: [String]
        let firstNumber = Double(newElement[0])!
        let secondNumber = Double(newElement[2])!
        let calcul = firstNumber+secondNumber
```

```
        result = ["\(calcul)"]
        return result
    }

    func substractByArray(_ newElement: [String]) -> [String] {
        let result: [String]
        let firstNumber = Double(newElement[0])!
        let secondNumber = Double(newElement[2])!
        let calcul = firstNumber-secondNumber
        result = ["\(calcul)"]
        return result
    }

    func multiplyByArray(_ newElement: [String]) -> [String] {
        let result: [String]
        let firstNumber = Double(newElement[0])!
        let secondNumber = Double(newElement[2])!
        let calcul = firstNumber*secondNumber
        result = ["\(calcul)"]
        return result
    }

    func divideByArray(_ newElement: [String]) -> [String] {
        let result: [String]
        let firstNumber = Double(newElement[0])!
        let secondNumber = Double(newElement[2])!
        let calcul = firstNumber/secondNumber
        result = ["\(calcul)"]
        return result
    }

    func findPositionOfOperation(_ newElement: [String], with operation:
String) -> Int? {
        return newElement.firstIndex(of: operation)
    }

    func extractByArray(_ newElement: [String], at index: Int) ->
[String] {
        var result: [String] = []
        result.append(newElement[index-1])
        result.append(newElement[index])
        result.append(newElement[index+1])
        return result
    }

    func reduce(_ newElement: [String]) -> [String] {
```

```
        var result: [String] = []
        let operation = newElement[1]
        switch operation {
        case "+":
            result = additionByArray(newElement)
        case "-":
            result = substractByArray(newElement)
        case "*":
            result = multiplyByArray(newElement)
        case "/":
            result = divideByArray(newElement)
        default:
            break
        }
        return result
    }

    func remove(_ newElement: [String], at index: Int) -> [String] {
        var copyOfInput = newElement
        copyOfInput.remove(at: index+1)
        copyOfInput.remove(at: index)
        copyOfInput.remove(at: index-1)
        return copyOfInput
    }

    func calculateOneOperation(_ newElement: [String], at index: Int) ->
[String] {
        var copyOfInput = newElement
        var result = extractByArray(copyOfInput, at: index)
        result = reduce(result)
        copyOfInput = remove(copyOfInput, at: index)
        copyOfInput.insert(contentsOf: result, at: index-1)
        return copyOfInput
    }

    func calculatorWithOnePriorityOperator(_ newElement: [String], with
operation: String) -> [String] {
        var copyOfInput = newElement
        while findPositionOfOperation(copyOfInput, with: operation) != 0
{
            guard let  wheretoCalculate =
findPositionOfOperation(copyOfInput, with: operation)
                else { return copyOfInput }
            copyOfInput = calculateOneOperation(copyOfInput, at:
wheretoCalculate)
        }
```

```
        return copyOfInput
    }

    func calculatorWithOneMinorOperator(_ newElement: [String]) ->
[String] {
        var copyOfInput = newElement
        while copyOfInput.count != 1 {
            copyOfInput = calculateOneOperation(copyOfInput, at: 1)
        }
        return copyOfInput
    }

    func calculateAll(_ newElement: [String]) -> [String] {
        var copyInputs = newElement
        copyInputs = calculatorWithOnePriorityOperator(copyInputs, with:
"*")
        copyInputs = calculatorWithOnePriorityOperator(copyInputs, with:
"/")
        copyInputs = calculatorWithOneMinorOperator(copyInputs)
        return copyInputs
    }
}
```

## 6. Les difficultés rencontrées

- Lecture du code au début => décomposition avant de comprendre la logique
- les premiers test unitaires
- Un déclic et tout est devenu amusant

## 7. Mon retour

- Projet le plus instructif jusqu'à maintenant
- Comprendre le MVC et surtout la façon d'aller chercher des informations
    i. jump définition
    ii. OPT + func
- Je vois bien mieux comment je m'y prendrais pour refaire le même exercice .

Merci à mon mentor Ali pour l'accompagnement ! un vrai soutien .