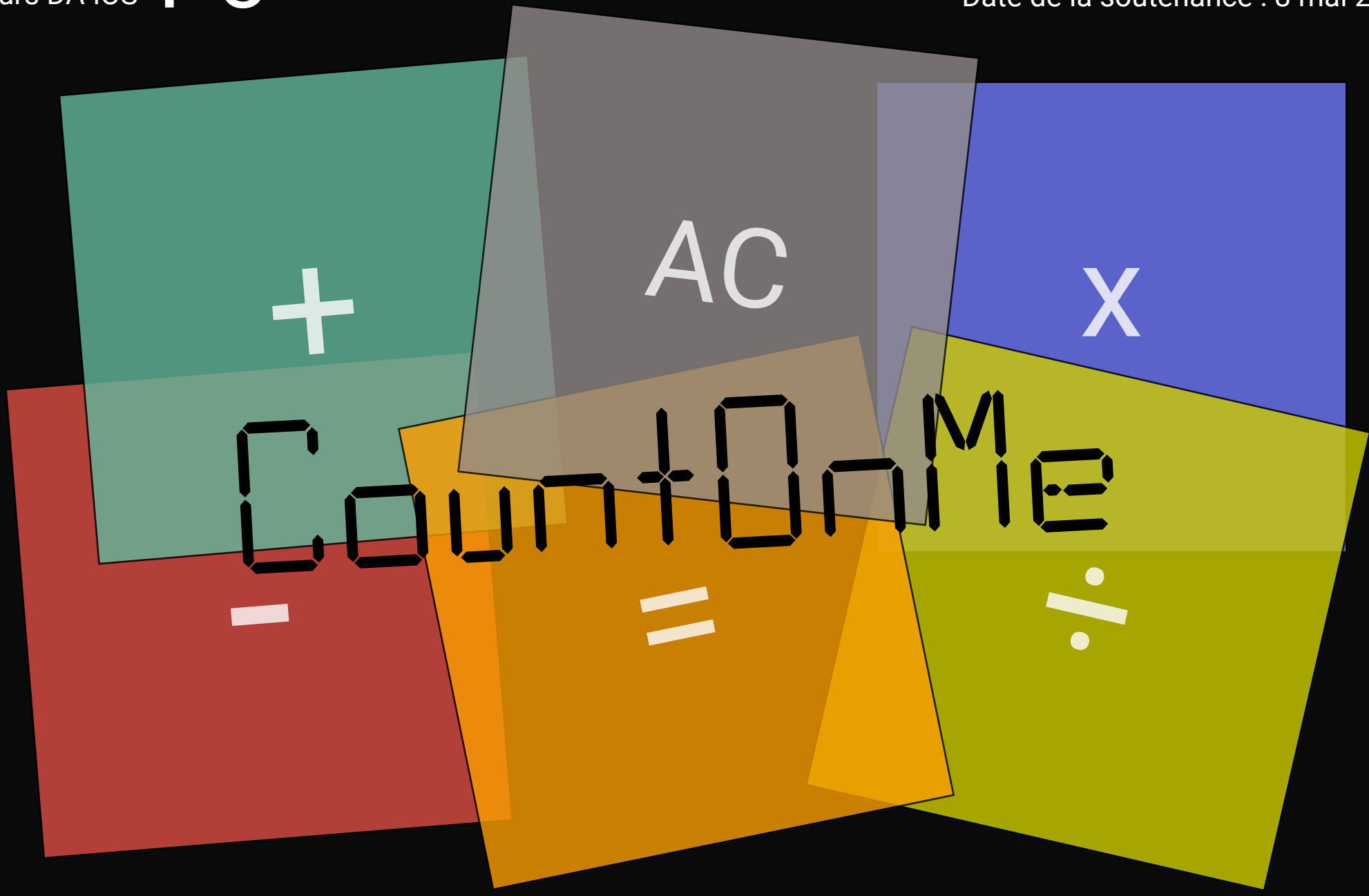
OpenClassrooms
Parcours DA-iOS P5

Projet présenté par Anthony Scott
Date de la soutenance : 8 mai 2020



DOCUMENT D'ARCHITECTURE

PRESENTATION DU PROJET

- reprise de projet existant (calculatrice)
- modifié en Swift 5.1 (Xcode 11.4.1)
- supporté par toutes les tailles d'iPhone, en mode Portrait
- en anglais

github.com/anthonymfscott/CountOnMe

ARCHITECTURE DE L'APPLICATION : MODELE

Expression.swift

```
enum Sign

class Expression
        input: String
        elements: [String]
        isCoherent: Bool
        hasEnoughElements: Bool
        containsEqualsSign: Bool
        canAddPoint: Bool
        result: String

        simplify(_:) -> String
```

Le Modèle de ce projet consiste en une classe unique, que j'ai appelée Expression. C'est ici que se passent la vérification de l'expression entrée par l'utilisateur d'une part, et les opérations arithmétiques d'autre part.

L'utilisation d'une énumération (*Sign*) permet de traiter les opérations en une liste de cas exhaustive.

La propriété *input* est la base à partir de laquelle toute la logique de cette classe découle ; il s'agit de ce que l'utilisateur a entré via l'interface. À partir de cette chaîne de caractères est créé un tableau qui en isole chaque partie sensée par index (*elements*).

isCoherent, *hasEnoughElements*, *containsEqualsSigns* et *canAddPoint* sont des propriétés calculées qui renvoient toutes un booléen dont la valeur sera utile au sein du Contrôleur.

result est la propriété au sein de laquelle toutes les opérations arithmétiques se passent, en tenant compte de la priorité des opérations et de l'impossibilité de la division par zéro.

Enfin, la méthode *simplify(_:)* est appelée en fin de calcul de la variable *result*. Elle permet de supprimer le point et les zéros qui le suivent si le résultat de l'opération est un entier et non un décimal.

ARCHITECTURE DE L'APPLICATION : VUE

Main.storyboard

La Vue contient ce seul fichier, dans lequel l'entièreté de l'interface a été créée.

On constatera l'ajout de quatre UIButtons par rapport au projet initial :
- « AC » qui permet à l'utilisateur de remettre les données de calcul à zéro ;
- « . » qui lui permet d'entrer des nombres décimaux ;
- « x » qui lui permet d'entrer une multiplication ;
- « ÷ » qui lui permet d'entrer une division.

L'existence de Stack Views a rendu l'ajout de ces boutons très facile.

Les contraintes, également faciles à modifier grâce aux Stack Views, ont été re-pensées au plus simple pour permettre un affichage adéquat dans toutes les tailles d'iPhone (en mode Portrait).

ARCHITECTURE DE L'APPLICATION : CONTROLEUR

ViewController.swift

enum Error

class ViewController
- o textView: UITextView!
- o deleteButton: UIButton!
- expression: Expression!
- viewDidLoad()
- o numberButtonTapped(_:)
- o signButtonTapped(_:)
- o equalsButtonTapped(_:)
- o deleteButtonTapped(_:)
- o decimalButtonTapped(_:)
- add(_:)
- reset()
- showErrorMessage(_:)

Le Contrôleur contient lui aussi un fichier unique : le ViewController.

L'énumération *Error* liste les cas possibles d'erreur présentés dans le projet initial. Ils seront vérifiés plusieurs fois dans la classe (via les propriétés calculées de la classe Expression vues précédemment), et appelleront la méthode *showErrorMessage(_:)* au besoin. Celle-ci affiche un UIAlertController spécifique à chaque cas d'erreur.

Les outlets *textView* et *deleteButton* permettent de référer les objets de la Vue auxquels ils se rapportent, et d'en modifier les propriétés.

expression est une instance de la classe Expression. Par son biais se fera la communication entre le Contrôleur et le Modèle. C'est au sein de *viewDidLoad()* qu'elle est initialisée.

numberButtonTapped(_:), *signButtonTapped(_:)*, *equalsButtonTapped(_:)*, *deleteButtonTapped(_:)* et *decimalButtonTapped(_:)* sont des actions. De manière générale, elles commencent par vérifier les propriétés d'*expression* et appellent *showErrorMessage(_:)* en cas d'erreur. Ensuite, en fonction des cas, elles appellent les méthodes *add(_:)* ou *reset()* pour mettre à jour le Modèle (via la propriété *input* d'*expression*) et la Vue (via la propriété *text* de *textView*).

equalsButtonTapped(_:) est plus particulièrement la méthode dans laquelle on récupère la propriété *result* d'*expression*, qui sera à son tour ajoutée à la propriété *text* de *textView*.

TESTS UNITAIRES

ExpressionTests.swift

```
class ExpressionTests
    expression: Expression!

    setUp()
    testAddition()
    testSubtraction()
    testMultiplication()
    testDivision()
    testDivisionToFloat()
    testDivisionByZero()
    testMultipleOperations()
    testMultipleOperationsIncludingDivisionByZero()
    testExpressionIsCoherent()
    testExpressionDoesNotHaveEnoughElements()
    testExpressionContainsEqualsSign()
    testSignIsWrong()
    testOperationsPriority()
    testAddPoint()
```

Le fichier de tests unitaires de ce projet est simple.

setUp() initialise la propriété *expression* pour éviter la redondance du code par la suite.

À part cela et les méthodes de test en tant que telles, il ne m'a pas semblé utile d'implémenter d'autres fonctionnalités.

Ce fichier teste le Modèle avec une couverture de 100%.