Assignment II: Calculator Brain (Калькулятор с мозгами)

Цель:

Вы начнете это задание с усовершенствования вашего калькулятора Calculator из Задания 1 с включением всех изменений, сделанных на лекции (то есть program Property List и управления доступом). Это последнее задание, когда вам нужно печатать код с лекции, а не использовать copy / paste.
Мы собираемся расширить немного его возможности, позволяя ему вводить “переменные” и поддерживая Undo.

Обязательно посмотрите ниже раздел подсказок (**Hints**)!
Также проверьте последние изменения в секции **Оценка**, чтобы убедиться, что вы понимаете, что собираются оценивать в вашем Задании.

Материалы

- У вас должно быть успешно выполненное Задание 1. Задание 2 выполняется на его основе.
- Вам необходимо посмотреть видео Лекции 3 и сделать такие же изменения с кодом вашего Задания 1.

Обязательные пункты задания

1. Все изменения, сделанные с калькулятором Calculator на лекции, должны быть внесены в код вашего Задания 1. Это включает как **var program**, так и правильную установку доступа (access control) всех методов и свойств. Сделайте полученный Calculator полностью функционирующим прежде, чем вы приступите к выполнению остальных обязательных заданий. И, как и в прошлый раз, "печатайте" изменения в коде, а не используйте copy / paste.

2. Ничего не меняйте в non-private API в CalculatorBrain и продолжайте использовать **Dictionary<String,Operation>** в качестве основной внутренней структуры данных.

3. Ваш пользовательский интерфейс (UI) должен быть всегда синхронизирован с вашей Моделью (CalculatorBrain).

4. Наделите ваш CalculatorBrain способностью вводить "переменные". Сделайте это путем реализации следующего API в вашем CalculatorBrain ...

```
func setOperand (variableName: String)
var variableValues: Dictionary<String, Double>
```

Это должно делать точно то, что говорят названия : первая функция должна вводить "переменную" как операнд (например, **setOperand("x")** будет использовать переменную с именем x), а вторая переменная позволит пользователю CalculatorBrain API установить любое значение для любой переменной (например, **brain.variableValues ["x"] = 35.0**). Ваш CalculatorBrain должен поддерживать любое число переменных. Вы можете предполагать,что никакое имя переменной не совпадает с символом операции.

5. Результат **var result** должен теперь правильно отражать значения "переменных" (из словаря **variableValues**) всякий раз, когда используются "переменные" (как в прошлом, так и в будущем, см. 8e ниже). Если у "переменной" нет значения в словаре, то используйте **0.0** как ее значение. Если словарь **variableValues** меняется, то **result** должен изменяться, отражая новые значения для "переменных".

6. Ваше описание **var description** должно продолжать работать правильно и должно показывать имя "переменной" (а не ее значение) всякий раз, когда ее вводят.

7. Свойство **var program**, добавленное на лекции, также нуждается в изменении для поддержания "переменных".

8. Добавьте две новых кнопки на UI калькулятора Calculator: **→M** и **M**. Не приносите в жертву никакие кнопки с требуемыми операциями из Задания 1 (хотя вы можете добавить больше операций, если хотите). Эти две кнопки будут соответственно устанавливать и получать переменную в **CalculatorBrain**, называемую **M**.
   - **→M** устанавливает значение переменной **M** в **brain** в текущее значение на **display** и

показывает на **display** результат **result**, полученный из **brain**.

- →**M** не должна выполнять **setOperand**.
- Нажатие **M** должно **setOperand("M")** в **brain** и затем показывать на **display** результат **result**, полученный из **brain**.
- →**M** и **M** являются механизмом **C**ontroller, а не механизмом **M**odel (хотя они оба используют концепцию "переменных" в **M**odel).
- Это не выдающаяся кнопка "memory" на нашем калькуляторе, но она является хорошим инструментом для тестирования, правильно ли работает концепция "переменных", реализованная выше.
- Примеры …
- **9 + M = √** ⇒ **description** имеет вид **√(9+M)**, **display** показывает 3, так как **M** не установлена (то есть равна 0)
- **7 →M** ⇒ **display** теперь показывает 4 (квадратный корень 16), **description** все еще показывает **√(9+M)**
- **+14 =** ⇒ **display** показывает 18, **description** теперь **√(9+M)+14**

9 . Убедитесь, что кнопка **C** вашего Задания 1 работает правильно в этом задании. В дополнение она должна убирать любые значения "переменной" **M** из **variableValues Dictionary** в CalculatorBrain (а не устанавливать в 0 или какое-то другое значение). Это позволит вам тестировать случай "неустановленной" переменной.

10 . Добавьте "Undo" к вашему калькулятору: в дополнительном пункте Задания 1 вы добавляли кнопку "backspace", если пользователь ввел неверную цифру. Теперь мы говорим о комбинации "backspace" и реального "Undo" в единой кнопке. Если пользователь находится в середине ввода числа, то это кнопка работает как "backspace". Если пользователь не находится в середине ввода числа, то должно выполняться Undo последней вещи, которая была выполнена в **CalculatorBrain**. Не отменяйте запоминания значений переменной (но ДЕЛАЙТЕ отмену установки "переменной" как операнда).

11 .На этой неделе другой критерий **Оценки**, так что убедитесь, что вы посмотрели эту секцию внизу.

Подсказки (Hints)

1. Даже если пользователи **API** класса **CalculatorBrain** вводят “переменную” с помощью метода с именем **setOperand**, нет причин, чтобы внутреняя реализация “переменных” в классе **CalculatorBrain** не использовала механизма “операций”, чтобы заставить это работать (возможно, это хорошая идея, так как внутри класса **CalculatorBrain** достаточно большая инфраструктура для управлеия операциями).

2. Пределы этого Задания подобны тем, которые были для Задания на прошлой неделе (то есть оно может быть выполнено полностью с помощью нескольких дюжин (12) строк кода и если это требует более 100 кода, то, возможно, вы где-то пошли неверным путем).

3. Некоторые вещи (подобные **π** или результат другого выражения) несколько хитроумно запоминать в **M** после того, как вы ввели выражение, которое вы хотите оценить (например, ввели cos(**M**), а затем пытаетесь установить **M** в **π**). Как только вы реализуете **Undo**, вы сможете это сделать (путем “undoing” этого выражения, полученного просто нажатием **π** для вычисления значения, которое вы хотите запомнить в **M**).

4 . Не забудьте подумать о вашем **CalculatorBrain** как о повторно используемом классе: будет более гибко (и к этому нет причин) разрешить использовать программистам **public API** для раздельного очищения **brain** от значений переменных в **brain** (хотя ваша кнопка **C** делает оба этих действия).

5. Рассмотрите возможность использования **Optional chaining** (Optional цепочек) при реализации **displayValue**.

Что нужно изучать

Это частичный список концепций, в которых это задание увеличивает ваш опыт работы или продемонстрирует ваши знания.

- Array
- Value Semantics
- Property Observer
- Property List
- String Manipulation
- Setting Dictionary Values
- Other Assorted Swift Language Features

Оценка (Evaluation)

Во всех заданиях требуется написание качественного кода, на основе которого строится приложение без ошибок и предупреждений (without warnings or errors), следовательно вы должны тестировать полученное приложение (application) до тех пор, пока оно не начнет функционировать правильно согласно поставленной задачи.

Приведем наиболее общие соображения, по которым задание может быть отклонено:

- Приложение не создается .
- Приложение не создается без предупреждений .
- Один или более пунктов в разделе **Обязательные пункты задания** не выполнены.
- Не понята фундаментальная концепция задания .
- Код - небрежный или тяжелый для чтения (например, нет отступов и т.д.).
- Приложение может заканчиваться аварийно (например, при попытке “развернуть” **Optional**, когда его значение равно **nil** !).
- Ваше решение тяжело (или невозможно) прочитать и понять из-за отсутствия комментариев, из-за плохого наименования методов и переменных, из-за непонятной структуры и т.д.
- Ваш пользовательский интерфейс беспорядочен. Все элементы должны быть выравнены и отделены соответствующими “зазорами”, чтобы выглядеть хорошо.
- Private API установлено неправильно.

Часто студенты спрашивают: “Сколько комментариев кода нужно сделать?” Ответ - ваш код должен легко и полностью быть понятным любому, кто его читает. Вы можете предполагать, что читатель знает SDK, но вам не следует предполагать, что он уже знает решение проблемы.

Дополнительные задания (Extra Credit)

Мы постарались создать Дополнительные задания так, чтобы они расширили ваши познания в том, что мы проходили на этой неделе. Попытка выполнения по крайней мере некоторых из них приветствуется в плане получения максимального эффекта от этого курса. Есть несколько подсказок для выполнения дополнительных заданий на следующей странице

1. Ваш Калькулятор должен собщать об ошибках. Например, √ из отрицательного числа или деление на нуль. Есть несколько способов "обнаружения" этих ошибок (может быть добавить ассоциированное значение к **Unary/ BinaryOperation** вариантам, которые являются функциями, которые обнаруживают ошибку, или, возможно, заставить функцию, которая ассоциируется с **Unary/ BinaryOperation** возвращать кортеж как с результатом **result**, так и с ошибкой **error** (если она есть) или ???). То, как вы будете сообщать о любых обнаруженных ошибках пользователям **CalculatorBrain API**, потребует от вас некоторых изменений **API**, но не заставляйте пользователей **CalculatorBrain API** иметь дело с ошибками, если они этого не хотят (то есть позвольте **C**ontrollers, которые хотят показывать **errors**, делать это, а другим, которые не хотят показывать ошибки, позвольте иметь дело просто с **NaN** или +∞, появляющимися на вашем **UI**). Другими словами, **не изменяйте никакие из существующих методов и свойств в non-private API CalculatorBrain для того, чтобы поддерживать эти возможности** (вместо этого добавьте методы / свойства, если это необходимо).