

# • progress bar

```

Aspect UIKit
class ViewController: UIViewController {
    @IBOutlet weak var questionLabel: UILabel!
    @IBOutlet weak var trueButton: UIButton!
    @IBOutlet weak var falseButton: UIButton!
    @IBOutlet weak var resultLabel: UILabel!
    @IBOutlet weak var progressBar: UIProgressView!
    let quiz = [
        Question("A slug's blood is green.", at: "True",
            trueAnswer: "A slug's blood is green.", falseAnswer: "False"),
        Question("The surface area of two human lungs is approximately 70 square meters.", at: "True",
            trueAnswer: "The surface area of two human lungs is approximately 70 square meters.", falseAnswer: "False"),
        Question("In West Virginia, USA, if you accidentally hit an animal with your car, you must stop and help it across the road.", at: "True",
            trueAnswer: "In West Virginia, USA, if you happen to drive by some house of an animal and accidentally exciting the animal, it is considered to be dead.", falseAnswer: "False"),
        Question("It is illegal to pee in the Ocean in Portugal.", at: "True",
            trueAnswer: "You can't leave a row down stairs but yet we urinate.", falseAnswer: "False"),
        Question("Bees are originally yellow, then turn red after their 4th larval moulting.", at: "True",
            trueAnswer: "Bees are originally yellow, then turn red after their 4th larval moulting.", falseAnswer: "False"),
        Question("The loudest sound produced by any animal is 188 decibels and it's made by a male African elephant.", at: "False",
            trueAnswer: "The loudest sound produced by any animal is 188 decibels and it's made by a male African elephant.", falseAnswer: "False"),
        Question("No piece of square dry paper can be folded in half more than 7 times.", at: "False",
            trueAnswer: "Chocolate affects a dog's heart and nervous system; a few ounces are enough to kill a small dog.", falseAnswer: "False")
    ]
    var questionNumber = 0
    override func viewDidLoad() {
        super.viewDidLoad()
        updateUI()
    }
    // UIButton button에 기록
    @IBAction func answerButtonTapped(_ sender: UIButton) {
        let userAnswer = sender.currentTitle // True, False
        let actualQuestion = quiz[questionNumber]
        let actualAnswer = actualQuestion.answer
        if userAnswer == actualAnswer {
            sender.backgroundColor = UIColor.green
        } else {
            sender.backgroundColor = UIColor.red
        }
        if questionNumber + 1 < quiz.count {
            questionNumber += 1
            updateUI()
        } else {
            questionNumber = 0
            userAnswer = "Progress"
        }
    }
    Timer.scheduledTimer(timersInterval: 0.2, target: self, selector:
        #selector(updateUI), userInfo: nil, repeats: false)
}

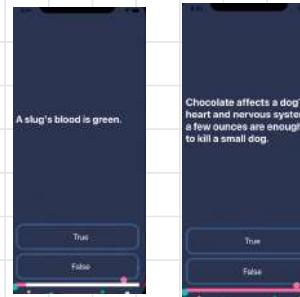
```

이제 맨 위에 `progressBar`를 넣어주면 됩니다.

#progressBar를 `UIProgressView()`로 바꿔주세요.

— 이번이 `NSTimer`이 아님 두번째 `progressBar` 응용인데,  
내가 느낀건 `progressBar`는 굉장히 간단한 수 있으나,  
현재 진행상황 / 정답의 양 or 개수 등 두 가지에 대응 해야되는  
이해가 필요 하다는 것이다.  
이번에는 `currentQuestionNumber` / `currentQuestion`을  
간단 했다.  
문제를 맞추면 툭리고 Timer의 `interval`은 0.1  
일어나는 `updateUI`의 코드를 작성해 `progressBar`가  
상승 하게 하면 된다.

## • 결과

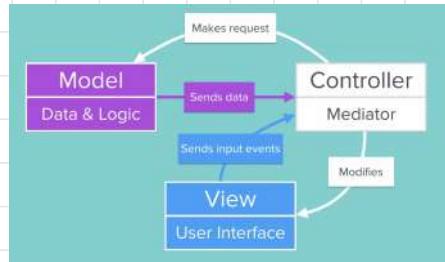
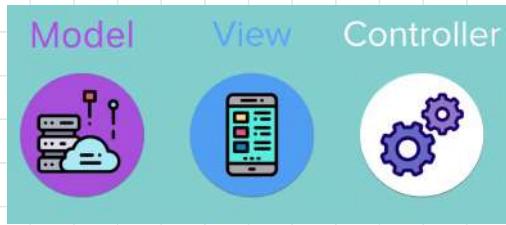


# Quizzer - MVC pattern

현재 Quizzer의 코드는 하나의 파일에만 위치하며 작동한다.

그렇게 되면 가독성이 떨어질 뿐만 아니라 유지보수하기 힘든다.

그러니 사용되는 design pattern들 중 하나인 **MVC pattern**을 사용해 보자.



## • M - Model

- Application의 정보, data를 나타낸다.  
database, 처음에 정의하는 상수(constant), 초기값, 변수(var) 등을 뜻한다.  
또한 이러한 data 정보들의 가공을 책임지는 component를 말한다.

### - Model의 구조

1. 사용자가 편집하기 원하는 모든 data를 가지고 있어야 한다.

즉, 화면 안의 네모박스에 글자를 표시된다면, 네모박스의 화면 위치 정보, 네모박스의 크기정보, 글자내용, 글자의 위치, 글자의 표면 정보 등을 가지고 있어야 하는 것이다.

2. View나 controller에 대해서 어떤 정보도 알지 않아야 한다

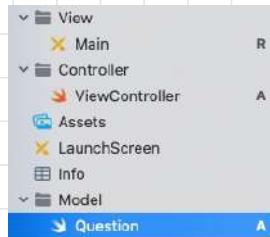
data 변경이 일어났을 때 Model에서 화면 UI를 직접 조정해서 수정할 수 있도록 View를 참조하는 내부 속성값을 가지면 안된다.

3. 변경이 일어나면, 변경 통지에 대해서 처리방법을 구현 해야 한다.

• Model의 속성 중 text 정보가 변경이 된다면, 이벤트를 발생시키거나 누군가에게 전달해야 하며, 누군가 Model을 변경하도록 요청하는 이벤트를 발생을 때 이를 수신하는 처리방법을 고민해야 한다. 또한 Model은 재사용 가능해야 하여 다른 인터페이스에서도 변경하지 않아야 한다.

# Quizler의 Model

~~Quizler의 Model~~



↳ Quizler의 Model은

Question: 문제와 정답 정보를 저장하는 구조체,

QuizBrain: Question의 정보를 두대로  
그리고 사용하는 data 할당,  
정답 정보를 하였다.



```
//# parameter 외에 다른 parameter 이름을 넣거나 _ 를 넣을 수 있다. _ 는 외부에서 프로퍼티의 아무것도 만들거나 아니면 아무거나 넣고 할 수 있다.
//# _ : parameter // external parameter name이라고 하고 이 대신 밖에서 쓰는 parameter name이고 checkAnswer 메소드 안에서 쓰이는 걸
internal parameter name이 된다.
```

```
var questionNumber = 0
```

```
// _ : external parameter name, userAnswer : internal parameter name
func checkAnswer(_ userAnswer: String) {
    if userAnswer == quizQuestions[questionNumber].text {
        print("Correct!")
    } else {
        print("Incorrect!")
    }
}
```

- ViewController() 있던 변수와 method는

QuizBrain struct에 가지고 온 모습.



```
@IBAction func answerButtonPressed(_ sender: UIButton) {
    let userAnswer = sender.currentTitle // True, False
    quizBrain.checkAnswer(userAnswer)
```

- 보내자니 checkAnswer에 userAnswer가  
넣었으나, 하지만 optional 오류가 발생할수는 있어  
그 이유는 userAnswer는 optional String이기  
checkAnswer는 actual String을 expect  
하고 있기 때문이다.



```
let userAnswer = sender.currentTitle // True, False
quizBrain.checkAnswer(userAnswer)
```

- currentTitle은 force-unwrapping 해주어야  
해결 하였다.



```
@objc func updateUI() {
    questionLabel.text = quizBrain.getQuestionText()
    progressBar.progress = quizBrain.getProgress()
```

```
trueButton.backgroundColor = UIColor.clear
falseButton.backgroundColor = UIColor.clear
```

- questionLabel.text = quiz[questionNumber].text

↓  
QuizBrain의 Method

progressBar.progress =  
Float((questionNumber + 1)) /  
Float(quiz.count)

↓  
QuizBrain의 Method

```
func nextQuestion() {
    if questionNumber + 1 < quiz.count {
        questionNumber += 1
    } else {
        questionNumber = 0
    }
}
```

```
func getQuestionText() -> String {
    // quiz[questionNumber].text
    return quiz[questionNumber].text
}
```

```
func getProgress() -> Float {
    Float(questionNumber + 1) / Float(quiz.count)
}
```

## • V-View

- input text, checkbox 항목 등과 같은 user interface 요소를 나타낸다.  
다시 말해 data의 입력, 그리고 보여주는 출력을 담당한다.  
data를 기반으로 사용자들이 볼 수 있는 화면이다.

### • View의 구조

1. Model이 가지고 있는 정보를  
다른 저장해서는 안된다.



. 화면에 글자를 표시하기 위해 Model이 가지고 있는  
정보를 전달 받게 된 템데, 그 정보로 유지하기 위해서  
임의의 View 내부에 저장하면 안된다.  
단순히 네오박스를 그리라는 명령을 받으면,  
화면에 표시하기만 하고, 그 화면을 그릴 때 필요한  
정보들은 저장하면 안된다.

2. Model이나 controller와 같이  
다른 구성요소를 몰라야 한다.



. Model과 같은 자기 자신의 모든 빼고는  
다른 모스는 참조하거나 어떻게 동작하는지  
알아서는 안된다.  
View는 data를 받으면 화면에 표시하는 역할임.

3. 변경이 일어나면 변경되는 대로  
처리방법을 구현해야만 한다.



. Model과 같이 변경이 일어났을 때 이를  
누군가에게 변경을 알려 주어야 하는 방법을 구현해야 한다.  
View에서는 화면에서 사용자가 화면에 표시된 내용을 변경하기  
되면 이를 Model에게 전달하여 Model을 변경해야 할 것이다.  
또한, 재사용 가능하게끔 설계를 해야하며 다른 정보들을  
표현할 때 쉽게 설계를 해야 한다.

### • Xcode의 View



- Xcode에서 설계 뷰에서는  
View의 규칙들이 크게 상관이  
없는 것 같다.

## • C - Controller

- data와 UserInterface 요소들은 있는 대화역할을 한다.  
즉, 사용자가 data를 클릭하고, 수정하는 것에 대한  
“이벤트”들을 처리하는 부분을 뜻한다.

### . Controller의 규칙

- 1. Model이나 View에 대해서  
알고 있어야 한다.

• Model이나 View는 서로의 존재를 모르고,  
변경을 외부로 알리고, 수신하는 방법만 가지고  
있는데, 이를 Controller가 중재 하기 위해  
Model과 그에 관련된 View의 대해  
알고 있어야 한다.

- 2. Model이나 View의 변경을  
모니터링 해야 한다.

• Model이나 View의 변경 통지를 받으면 이를 해석해서  
각각의 구성요소에게 통지를 해야 한다.  
또한, application의 Main Logic은 Controller가  
담당하게 된다.

# QuizBrain의 Controller

X QuizBrain을 MVC pattern으로 재설계하고  
그 후에 challenge로 선택지가 3개 있는 Question으로  
바꾸는 걸 했는데, 그걸 dataSource를 캡처를 이용해서  
Challenge 버전으로 처리 한다.

```

class ViewController: UIViewController {
    @IBOutlet weak var scoreLabel: UILabel!
    @IBOutlet weak var questionLabel: UILabel!
    @IBOutlet weak var progressBar: UIPercentDrivenInteractiveTransition!
    @IBOutlet weak var firstButton: UIButton!
    @IBOutlet weak var secondButton: UIButton!
    @IBOutlet weak var thirdButton: UIButton!

    var buttonArray: [UIButton] = [UIButton]()
    var quizBrain = QuizBrain()

    override func viewDidLoad() {
        super.viewDidLoad()
        buttonArray = [firstButton, secondButton, thirdButton]
        updateUI()
    }

    // 설정적인 button의 기능 구현
    @IBAction func answerButtonPressed(_ sender: UIButton) {
        let userAnswer = sender.currentTitle! // True, False
        let userGotItRight = quizBrain.checkAnswer(userAnswer)

        if userGotItRight == true {
            sender.backgroundColor = UIColor.green
        } else {
            sender.backgroundColor = UIColor.red
        }

        quizBrain.nextQuestion()

        Timer.scheduledTimer(timeInterval: 0.2, target: self, selector: #selector(updateUI), userInfo: nil, repeats: false)
    }

    @objc func updateUI() {
        updateAnswerButton()
        questionLabel.text = quizBrain.getQuestionText()
        progressBar.progress = quizBrain.getProgress()
        scoreLabel.text = "Score : \(quizBrain.getScore())"
        clearButton()
    }

    func clearButton(){
        firstButton.backgroundColor = UIColor.clear
        secondButton.backgroundColor = UIColor.clear
        thirdButton.backgroundColor = UIColor.clear
    }

    func updateAnswerButton(){
        for i in 0..<buttonArray.count{
            buttonArray[i].setTitle(quizBrain.quiz[quizBrain.questionNumber].answer[i], for: .normal)
        }
        ↳; 각각 button의 title을 question에 맞게 변경
    }
}

```

① 버튼 3개에 같은 선택지를 어떤 방법으로 넣을까 하니까  
Array를 이용하게 됐다

② View가 load되는 즉시 배열의  
button을 설정한다.

③ button의 초기값.

```

import Foundation

struct Question{
    let text: String
    let answer: Array<String>
    let correctAnswer : String

    init(q: String, a: Array<String>, correctAnswer: String){
        text = q
        answer = a
        self.correctAnswer = correctAnswer
    }
}

```

→ Model  
- Question

```

import Foundation
struct QuizBrain{
    var quiz: [Question]
    var score: Int = 0

    // An array of questions come, questioner : internal parameter name
    // inside func checkAnswer, questioner: string) -> Bool {
    // questioner: internal parameter name, used in this method.
    // If userAnswer == quiz[questionNumber].correctAnswer
    // return true
    // else {
    //     return false
    // }

    func getScore() -> Int{
        return score
    }

    mutating func nextQuestion(){
        if questionNumber + 1 < quiz.count {
            questionNumber += 1
        } else {
            questionNumber = 0
            questionNumber += 1
        }
    }

    func getQuestionText() -> String{
        return quiz[questionNumber].text
    }

    func getProgress() -> Float{
        let progress = Float(questionNumber+1)/Float(quiz.count)
        return progress
    }
}

```

→ Model  
- QuizBrain

# Quizzer - Multiple choice Udemy & Me's Code Review

## • Udemy's code

### . Model - Question

#### - Udemy

```
import Foundation

struct Question {
    let text: String

    //Multiple choice questions have multiple answers, an Array of Strings would work for our quiz data.
    let answers: [String]
    //Look at the data in the quiz array, there is a seperate string that is the correctAnswer.
    let rightAnswer: String

    //The initializer needs to be updated to match the new multiple choice quiz data.
    init(q: String, a: [String], correctAnswer: String) {
        text = q
        answers = a
        rightAnswer = correctAnswer
    }
}
```

#### - Me

```
import Foundation

struct Question{
    let text: String
    let answer: Array<String>
    let correctAnswer : String

    init(q: String, a: Array<String>, correctAnswer: String){
        text = q
        answer = a
        self.correctAnswer = correctAnswer
    }
}
```

- 다른 점은 띄어쓰기.  
나의 correctAnswer 보다 rightAnswer CT 가 정한 이름 같고  
self는 쓰면 안 되나보다

# Model 1 - QuizBrain

## Udemy

```

import Foundation

struct Question {
    var questionNumber = 0
    var score = 0
}

let quiz = [
    Question(id: "Q1", question: "Which is the largest organ in the human body?", answers: ["Heart", "Skin", "Large Intestine"], correctAnswer: "Skin"),
    Question(id: "Q2", question: "What is the capital of France?", answers: ["Paris", "London", "Berlin"], correctAnswer: "Paris"),
    Question(id: "Q3", question: "What is the letter in the 9th line name stand for?", answers: ["General Aviation Time", "General Marine Time"], correctAnswer: "General"),
    Question(id: "Q4", question: "The French word for 'heart', as ["Coeur", "Cœur"], correctAnswer: "Cœur"),
    Question(id: "Q5", question: "In past times, what would a gentleman need to take his pocket?", answers: ["Handkerchief", "Watch"], correctAnswer: "Watch"),
    Question(id: "Q6", question: "What is the French word for 'water', as ["Eau", "Wasser", "Wasser"], correctAnswer: "Eau"),
    Question(id: "Q7", question: "What colour of these colours is NOT featured in the logo for Google?", answers: ["Green", "Orange", "Blue"], correctAnswer: "Orange"),
    Question(id: "Q8", question: "What is the English word for 'waterloo'?", answers: ["Australia", "Australia", "Australia"], correctAnswer: "Australia")
]

var questionNumber = 0
var correctAnswers = 0
var totalQuestions = quiz.count
var score = 0

var getProgress: Float {
    return Float(correctAnswers) / Float(totalQuestions)
}

func getScore() {
    let percentage = Set.allElements.filter { $0.correctAnswer == correctAnswers }.count
    print("You got \(percentage) questions right!")
}

func getQuestionIndex() -> Int {
    return Int.random(in: 0..

```

→ d 별은 놓쳤다...  
 Answer 빠져있을 때  
 Method는 Model이 아님.  
 주석은 놓쳤다.

## Me

```

import Foundation

struct Question {
    // Qn ID & question + 1
    let qnId = 1
    let question = "What is the largest organ in the human body?", as: ["Heart", "Skin", "Large Intestine"], correctAnswer: "Skin"
    let answer = "Skin"
    let hint = "The largest organ in the human body is the skin, which covers about 1.5 square meters and weighs about 2 kilograms." as String
    let difficultyLevel = "Easy"
    let category = "Health"
}

let quiz = [
    Question(qnId: 1, question: "What is the largest organ in the human body?", answers: ["Heart", "Skin", "Large Intestine"], correctAnswer: "Skin", difficultyLevel: "Easy", category: "Health"),
    Question(qnId: 2, question: "What is the capital of France?", answers: ["Paris", "London", "Berlin"], correctAnswer: "Paris", difficultyLevel: "Easy", category: "Geography"),
    Question(qnId: 3, question: "What is the letter in the 9th line name stand for?", answers: ["General Aviation Time", "General Marine Time"], correctAnswer: "General", difficultyLevel: "Easy", category: "Time"),
    Question(qnId: 4, question: "The French word for 'heart', as ["Coeur", "Cœur"], correctAnswer: "Cœur", difficultyLevel: "Easy", category: "Language"),
    Question(qnId: 5, question: "In past times, what would a gentleman need to take his pocket?", answers: ["Handkerchief", "Watch"], correctAnswer: "Watch", difficultyLevel: "Easy", category: "History"),
    Question(qnId: 6, question: "What is the French word for 'water', as ["Eau", "Wasser", "Wasser"], correctAnswer: "Eau", difficultyLevel: "Easy", category: "Language"),
    Question(qnId: 7, question: "What colour of these colours is NOT featured in the logo for Google?", answers: ["Green", "Orange", "Blue"], correctAnswer: "Orange", difficultyLevel: "Easy", category: "Technology"),
    Question(qnId: 8, question: "What is the English word for 'waterloo'?", answers: ["Australia", "Australia", "Australia"], correctAnswer: "Australia", difficultyLevel: "Easy", category: "History"),
    Question(qnId: 9, question: "What is the German word for 'water', as ["Wasser", "Wasser", "Wasser"], correctAnswer: "Wasser", difficultyLevel: "Easy", category: "Language"),
    Question(qnId: 10, question: "What is the English word for 'waterloo'?", answers: ["Australia", "Australia", "Australia"], correctAnswer: "Australia", difficultyLevel: "Easy", category: "History")
]

var questionNumber = 0
var correctAnswers = 0
// external parameter name, questionNumber : internal examiner name
matching func checkAnswer(_ userAnswer: String) -> Bool {
    if userAnswer == correctAnswers {
        correctAnswers += 1
        return true
    } else {
        return false
    }
}

func getProgress() -> Int {
    return correctAnswers
}

var getQuestionIndex() -> Int {
    if questionNumber + 1 < quiz.count {
        questionNumber += 1
        return questionNumber
    } else {
        questionNumber = 0
        return 0
    }
}

func getQuestion() -> Question {
    return quiz[getQuestionIndex()]
}

func getScore() -> String {
    return String(correctAnswers)
}

func getProgress() -> Float {
    let progress = Float(correctAnswers) / Float(totalQuestions)
    return progress
}

```

# • Controller - View Controller

```

@IBOutlet weak var questionLabel: UILabel!
@IBOutlet weak var progressLabel: UILabel!
//Add another button and a corresponding outlet.
@IBOutlet weak var thridButton: UIButton!
@IBOutlet weak var scoreLabel: UILabel!
@IBOutlet weak var thridLabel: UIButton!
@IBOutlet weak var correctLabel: UILabel!
//New button needs to be linked to this IBAction too.
@IBAction func answerButtonPressed(_ sender: UIButton) {
    let userAnswer = sender.currentTitle!
    let userGetRight = quizBrain.getQuestionUserAnswer(userAnswer)
    if userGetRight {
        sender.backgroundColor = UIColor.green
    } else {
        sender.backgroundColor = UIColor.red
    }
    quizBrain.nextQuestion()
    Timer.scheduledTimer(timeInterval: 0.2, target: self, selector: #selector(updateUI), userInfo: nil, repeats: false)
}

@objc func updateUI() {
    questionLabel.text = quizBrain.getQuestionText()
    //Need to return the answers and update the button titles using the setTitle method.
    let answerButtons = quizBrain.getAnswers()
    choice1.setTitle(answerButtons[0], for: .normal)
    choice1.setTitle(answerButtons[1], for: .normal)
    choice1.setTitle(answerButtons[2], for: .normal)
    choice2.setTitle(answerButtons[3], for: .normal)
    progressLabel.progress = quizBrain.getProgress()
    scoreLabel.text = "Score: \(quizBrain.getScore())"
    choice1.backgroundColor = UIColor.clear
    choice2.backgroundColor = UIColor.clear
    choice3.backgroundColor = UIColor.clear
    //Third button needs to be reset too.
    thridLabel.backgroundColor = UIColor.clear
}

```

→ 022 Ako? 3  
만화로 생각해.

```

class ViewController: UIViewController {
    @IBOutlet weak var questionLabel: UILabel!
    @IBOutlet weak var questionLabel: UILabel!
    @IBOutlet weak var progressLabel: UILabel!
    @IBOutlet weak var firstButton: UIButton!
    @IBOutlet weak var secondButton: UIButton!
    @IBOutlet weak var thirdButton: UIButton!
    @IBOutlet weak var correctLabel: UILabel!
    @IBOutlet weak var thridLabel: UIButton!
    var quizBrain = QuizBrain()
    override func viewDidLoad() {
        super.viewDidLoad()
        buttonArray = [firstButton, secondButton, thirdButton]
        quizBrain = QuizBrain()
        let buttonArray: [UIButton] = [firstButton] + [secondButton] + [thirdButton]
        var userAnswer = QuizBrain()
        override func viewDidLoad() {
            super.viewDidLoad()
            buttonArray = [firstButton, secondButton, thirdButton]
            quizBrain = QuizBrain()
        }
        //만화로 생각해 2부
        @IBAction func answerButtonPressed(_ sender: UIButton) {
            let userAnswer = sender.currentTitle! // True, False
            let userGetRight = quizBrain.checkAnswers(userAnswer)
            if userGetRight == true {
                sender.backgroundColor = UIColor.green
            } else {
                sender.backgroundColor = UIColor.red
            }
            quizBrain.nextQuestion()
            Timer.scheduledTimer(timeInterval: 0.2, target: self, selector: #selector(updateUI), userInfo: nil, repeats: false)
        }

        @objc func updateUI() {
            updateLabels()
            questionLabel.text = quizBrain.getQuestionText()
            progressLabel.progress = quizBrain.getProgress()
            scoreLabel.text = "Score: \(quizBrain.getScore())"
            clearButton()
        }

        func clearButton() {
            firstButton.backgroundColor = UIColor.clear
            secondButton.backgroundColor = UIColor.clear
            thirdButton.backgroundColor = UIColor.clear
        }

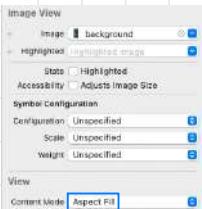
        func updateLabels() {
            for i in 0...buttonArray.count {
                buttonArray[i].setTitle(quizBrain.quizBrain.operationNumber1.answer[i], for: .normal)
            }
        }
    }
}

```

→ 022 Ako? 3  
만화로 생각해.  
Controller와 연결.

# Destini

## 1. Background



- stackView setting

• Distribution: Fill Proportionally  
 비율을 맞춰 StackView를 채워주는 것.  
 Stack View OK는 유연성이 따라 사용 가능한  
 공간을 활용다. (View's intrinsic content size)  
 • spacing: 20  
 간격

## 4. Apply MVC

### • Model - Story

```

import Foundation

struct Story {
  var title: String
  var choice1: String
  var choice2: String
  var unscied: String

  init(title: String, choice1: String, choice2: String) {
    self.title = title
    self.choice1 = choice1
    self.choice2 = choice2
  }
}
  
```

• Story 구조체를 만들어 사용하게  
필요한 data format 정의 담고 있다.

### • Model - StoryBrain

```

enum StoryBrain {
  case story1
  case story2
  case story3
  case story4
  case story5
  case story6
  case story7
  case story8
  case story9
  case story10
}
  
```

- StoryBrain 구조체에 Story를 이용해 Stories Analysis를 만들었다.

- auto layout에 대한 이해가 부족해서 AUTO layout이 적용이 안된 project로 시작했다.
- 배경이 디자인하지 않았아서 Content Mode를 Aspect Fit으로 했다.

## 2. View & constraints



• button의 간격은 저 정도로 설정 하면 좋았지만, Stack View는 적용 시키고 button 사이 간격이 조정이 안되서 다시 unembed하고 조정을 다시 stack view로 만들었다.

\* StackView 상태에서도 조정 가능성이 좋았다.



## 3. IBoutlet - button, label

```

@IBOutlet weak var storyText: UILabel!
@IBOutlet weak var choiceButtons: UIButton!
@IBOutlet weak var choice2Buttons: UIButton!

let story1 = "You see a rock in the road."
let choice1 = "Take a left."
let choice2 = "Take a right."

override func viewDidLoad() {
  storyText.text = story1
  choiceButtons.setTitle(choice1, for: .normal)
  choice2Buttons.setTitle(choice2, for: .normal)
  super.viewDidLoad()
}
  
```

• button과 label은 연결해 테스트 해 봤다.

## Controller - View Controller

```

import UIKit

class ViewController: UIViewController {
  @IBOutlet weak var storyText: UILabel!
  @IBOutlet weak var choiceButtons: UIButton!
  @IBOutlet weak var choice2Buttons: UIButton!

  var storyBrain = StoryBrain()

  override func viewDidLoad() {
    storyText.text = storyBrain.story1
    choiceButtons.setTitle(storyBrain.choice1, for: .normal)
    choice2Buttons.setTitle(storyBrain.choice2, for: .normal)
    super.viewDidLoad()
  }
}
  
```

- StoryBrain StoryBlaine 이용해 View와 Model에 연결시키는 logic을 구현했다.



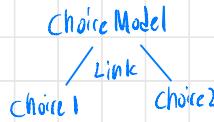
## 5. IBAction



```
IBAction func choiceModel(sender: UIButton) {
    if sender == choice1Button {
        storyText.text = storyBrain.stories[1].title
        choice1Button.setTitle(storyBrain.stories[1].choice1, for: .normal)
        choice2Button.setTitle(storyBrain.stories[1].choice2, for: .normal)
    } else {
        storyText.text = storyBrain.stories[2].title
        choice1Button.setTitle(storyBrain.stories[2].choice1, for: .normal)
        choice2Button.setTitle(storyBrain.stories[2].choice2, for: .normal)
    }
}
```

- 각 button을 누를 때마다 새로운 선택지가 나오게 한다.

↳



· 모식도 (diagram)



## 6. Model (StoryBrain) - Next Story function

```
mutating func nextStory() {
    if storyNumber+1 < stories.count {
        storyNumber += 1
    } else {
        storyNumber = 0
    }
}
```

```
func nextStory() {
    if storyNumber+1 < stories.count {
        storyNumber += 1
    } else {
        storyNumber = 0
    }
}
```

View - Viewcontroller



- 다음 스토리로 넘어가기 위해 StoryBrain에서 사용한 method를 만들었다.

처음에 .count 셀프인 애를 만들었지만

.count는 앞 그대로 개수를 세주는 걸로 대체해  
1부터 시작한다.

- Update UIText 필드를 적용하지 않는다.  
적용 viewDidLoad를 재설정하는 흐름이 안된다.

또한 스크린은 못 짜았지만

choice 1 → storyNumber += 1

choice 2 → storyNumber += 2 가 되기

구성했지만, 첫번째 가지는 작동이 잘되다가  
두번째 클릭 시부터 choice 1 을 누르면 각  
choice 2로 인식한다.

## • Model - StoryBrain

```

mutating func nextStory(userChoice: String) {
    if userChoice == stories[storyNumber].choice1 {
        print("choice1 : \(storyNumber)")
        storyNumber += 1
        print("choice1 : \(storyNumber)")
    } else {
        print("choice2 : \(storyNumber)")
        storyNumber += 2
        print("choice2 : \(storyNumber)")
    }
}

func getStory() -> String {
    return stories[storyNumber].title
}

func getChoice1() -> String {
    return stories[storyNumber].choice1
}

func getChoice2() -> String {
    return stories[storyNumber].choice2
}

```

• 무엇이 문제인가?

## 5 step approach to Debug our App

통해 문제를 해결하자



What did you expect your code to do?



What happened instead?



What does your expectation depend upon?



How can we test the things our expectations depend on?

- 누르는 Button에 따라 storyNumber의 증가 수가 다르고, updateUI를 통해 다음 story로 넘어가는가?

- 누르는 button에 따라 story가 달라지거나 button마다 증가하는 storyNumber도 다르다

그것이 기대로 반영되지 않을 뿐더러, updateUI로 최초 배포를 제외하고는 실행이 안된다

### • Model - StoryBrain

```

mutating func nextStory(userChoice: String) {
    if userChoice == stories[storyNumber].choice1 {
        print("choice1 : \(storyNumber)")
        storyNumber += 1
        print("choice1 : \(storyNumber)")
    } else {
        print("choice2 : \(storyNumber)")
        storyNumber += 2
        print("choice2 : \(storyNumber)")
    }
}

```

### Controller ~ ViewController

```

@IBAction func increment(_ sender: UITapGestureRecognizer) {
    if let storyboard = storyboard as? MainStoryboard {
        storyboard.instantiateViewController(withIdentifier: "ViewController") as! ViewController
    }
}

@IBAction func decrement(_ sender: UITapGestureRecognizer) {
    if let storyboard = storyboard as? MainStoryboard {
        storyboard.instantiateViewController(withIdentifier: "ViewController") as! ViewController
    }
}

@IBAction func updateUI(_ sender: UIButton) {
    storyText.text = choice1_2.text!
    // 선택 내용 그대로로 story에 들어오게 되어,
    // 그걸로 nextStory에서 이미 다음 story로 넘어온 내용과
    // 당연히 다르게 처리로 각성을 얻는 것이다.
    UpdateUI의 초기화 과정이다
}

```

- 각각의 button과 nextStory에서 증가하는 StoryNumber의

로그를 적어보았다.

여기 updateUI가 호출의 안도니 문제나 생기는 것이 코드를 통해 확인되었다.



- Viewcontroller에 타이머를 추가한 모습

- 앞서 한 project Quizzer에서 updateU2를  
이용해 흐름 했는지 살펴봤다.

- `Quizzler`에서는 전달해온 오답률을 누울 때마다  
일시저장을 사용해 효과를 주기 위해 `TimeIntervel`을 사용했는데,  
`TimeIntervel`의 `property` 중 `Selectors`는 개별시간가지 설정하는  
`timeInterval` 후에 실행할 Function을 넣어줫다.

그리하여 보다やすく `updateUI`를 넣고,  
`Objeactive-C`의 문제에 @objc까지 앞에  
붙여 주어 `updateUI`가 정상적으로 반응할 수 있어 `도입` ct.

- ReadMe에서 여러 Story가 있는 내용을 가져와 적용시킨 모습이다.

Struct에는 2개의 property가 적용된 모습을

볼 수 있는데, StoryNumber이 증가수를 걸친 대체주는 *But* 이런,

Destination property  $\equiv$  StoryNumber on HFZ CHERI b6E41

더 깔끔하고 훤한 상태 진짜 가능하다.  
if-else문으로 클릭되는 button에 따라 start 빠르게 이동되는 기준의 logic을  
사용하면, 코드가 자세히지는 것 뿐만 아니라 여러 가지 사용이 가능하다.

## - Model - Story Brain

```
mutating func nextStory(userChoice: String) {
```

```
if userChoice == stories[storyNumber].choice1 {  
    storyNumber = stories[storyNumber].choice1Destination  
} else {  
    storyNumber = stories[storyNumber].choice2Destination  
}
```

- Next Story on Destination property  $\frac{2}{2}$

# 적용시킨 모습

# Destini - Code Review with Angela's

## • Angela - ViewController

```

@IBAction func choiceMade(_ sender: UIButton) {
    storyBrain.nextStory(userChoice: sender.currentTitle)
    updateUI()
}

func updateUI() {
    storyLabel.text = storyBrain.getStoryTitle()
    choice1Button.setTitle(storyBrain.getChoice1(), for: .normal)
    choice2Button.setTitle(storyBrain.getChoice2(), for: .normal)
}

```

## • updateUI() 헷갈리기

- Timer도 쓸 필요가 있다.
- updateUI는 function이다.
- 그냥 호출하면 되는 것이다.
- 단, button 클릭 시마다 내용이 반영되는 것이다.
- IBAction choiceMade에서 호출은 해주면 된다.
- 아직 간단한 것이다.
- 구현 수록 간단한 방법부터 천천히 생각해보자.**

- Timer가 있다.  
 또, 41 choiceMade에 있는 sender가 누군가에 따라  
 구분하는 if-else문도 있다.  
 다시 사용하지만 choiceMade 아래의 sender 구분은 왜인가  
 있다.  
 왜? → 그 가능은 바로 StoryBrain의  
 nextStory에서 하고 있기 때문

## • Angela - StoryBrain

```

mutating func nextStory(userChoice: String) {

    let currentStory = stories[storyNumber]
    if userChoice == currentStory.choice1 {
        storyNumber = currentStory.choice1Destination
    } else if userChoice == currentStory.choice2 {
        storyNumber = currentStory.choice2Destination
    }
}

```

## • Me - StoryBrain

```

mutating func nextStory(userChoice: String) {

    if userChoice == stories[storyNumber].choice1 {
        storyNumber = stories[storyNumber].choice1Destination
    } else {
        storyNumber = stories[storyNumber].choice2Destination
    }
}

```

- 크게 다른점은 있지만, 나는 상수나 변수 선언을  
 습관화해야 할 것이다. Angela의 코드가 더 가독성이 있다.
- \* 반복되는 코드는 상수, 변수 선언을 하여 효율적으로 코드하자.
- Make variables or constants when it comes to Repetitive**