```swift
//
//  Haiku.swift
//  HaikuBox
//
//  Created by Yu Liu on 2015-12-28.
//

import Foundation

/*
Markers that indicates the formatting of the haikus
*/

let haikuReplaceMarker = ["-n", "-v", "-adj", "-adv"]
let haikuNewlineMarker = ","

/*
Type assignments
*/

typealias haikuWordSet = [[String]]
typealias haikuListItem = (String, haikuWordSet)

enum haikuWordTypes: Int {
    case noun = 0
    case verb = 1
    case adjective = 2
    case adverb = 3
}

// MARK: Functions

/*
Joins a list of strings into one
*/
func join(split: [String]) -> String {
    var joinedString = ""
    for word in split {
        joinedString.appendContentsOf(word)
        joinedString += " "
    }
    return joinedString
}

/*
Returns a ascending or flat counter based on the length of word types
*/
func counter(countUp: Bool = false) -> [Int] {
    var array: [Int] = []

    for count in 0..<haikuReplaceMarker.count {
        if countUp == true {
            array.append(count)
        } else {
            array.append(0)
        }
    }

    return array
}

// MARK: Classes

class Haiku {

    // MARK: Properties

    let template: String
    let original: haikuWordSet

    // MARK: Methods

    init(withTemplate template: String, andWordSet original: haikuWordSet) {
        self.template = template
        self.original = original
    }

    /*
    Use a tuple as argument that contains the template and original
    */
    convenience init(withCombinedData data: haikuListItem) {
        self.init(withTemplate: data.0, andWordSet: data.1)
    }

    /*
    Combines several haiku tuples into one
    */
    convenience init(withSeparateLineItems items: [haikuListItem]){
        var template: String = ""
        var wordset: haikuWordSet = []

        for _ in 0..<haikuReplaceMarker.count {
            wordset.append([])
        }

        for item in items {
            template += item.0
            template += haikuNewlineMarker
            template += " "
            for x in counter(true) {
                if item.1[x].count == 0 {
                    continue
                }
                wordset[x].appendContentsOf(item.1[x])
            }
        }

        self.init(withTemplate: template, andWordSet: wordset)
    }

    /*
    Replaces the the marks in the template with new set
    */
    func replace(withWordSet newset: haikuWordSet) -> String {

        var splitTemplate = template.componentsSeparatedByString(" ")
        var wscount = counter()
        var temp = ""
        var markerLength = 0

        for (count, word) in splitTemplate.enumerate() {
            for i in counter(true) {
                if word.hasPrefix(haikuReplaceMarker[i]) {
                    markerLength = haikuReplaceMarker[i].characters.count
                    temp = word

                                    temp.removeRange(temp.startIndex..<temp.
                                    startIndex.advancedBy(markerLength))
                    splitTemplate[count] = newset[i][wscount[i]]
                    splitTemplate[count] += temp
                    wscount[i] += 1
                    break
                }
            }
        }

        return join(splitTemplate)
    }

    /*
    Replaces one specific word in the haiku template, but keep the other words
    */
    func replace(withOneWord newWord: String, ofType type: haikuWordTypes,
             atIndex index: Int) -> String {

        var newWordSet = original
        if index < 0 || newWordSet.count < type.rawValue || newWordSet[type.
                  rawValue].count <= index {
            return "An error has occured"
        }

        newWordSet[type.rawValue][index] = newWord
        return replace(withWordSet: newWordSet)
    }

    /*
    Returns the template and wordset of one item
    */
    func lineItem(lineNo: Int) -> haikuListItem {

        var newWordSet: haikuWordSet = [[],[],[],[]]
        var wscount = counter()
        let lines = template.componentsSeparatedByString(haikuNewlineMarker)

        for (currentLine,line) in lines.enumerate() {
            for word in line.componentsSeparatedByString(" ") {
                for i in counter(true) {
                    if word.hasPrefix(haikuReplaceMarker[i]) {
                        if wscount[i] == original[i].count {
                            print("Index Error",template, original)
                        }
                        if currentLine == lineNo {
                            newWordSet[i].append(original[i][wscount[i]])
                        }
                        wscount[i] += 1
                    }
                }
            }
        }

        return (lines[lineNo], newWordSet)
    }

    /*
    Returns a random number within the range of the Array for the specified list
    */
    func randIndex(type: haikuWordTypes) -> Int? {
        let count = original[type.rawValue].count

        if count == 0 {
            return nil
        }
        let x = random() % original[type.rawValue].count
        return x
    }

    func atRandIndex(type: haikuWordTypes) -> String? {
        let ri = randIndex(type)
        if ri == nil {
            return "===Error==="
        }
        return original[type.rawValue][ri!]
    }
}

class HaikuManager {

    // MARK: Properties

    var managedHaikus: [Haiku]
    var currentHaiku: Haiku?
    var currentWord: String?
    var type: haikuWordTypes = .noun
    var lastRandom = -1

    // MARK: Main Methods

    init() {
        let time = UInt32(NSDate().timeIntervalSinceReferenceDate)
        srandom(time)
        managedHaikus = []
    }

    /*
    Loads the haikus into memory
    */
    func loadAll() {
        addHaikus(getHaikuList())
    }

    /*
    A separated method for setting the type of word used to generate haikus
    */
    func setType(type: haikuWordTypes) {
        self.type = type
        currentHaiku = nil
    }

    /*
    Returns a haiku string, with a random word replaced by the argument word
    */
    func oneWord(word: String) -> String {

        var items: [haikuListItem] = []
        for i in 0...2 {
            let h = randomHaiku()
            items.append(h.lineItem(i))
        }

        let newHaiku = Haiku(withSeparateLineItems: items)
        if newHaiku.randIndex(type) == nil {
            print("No word of type",type)
            return oneWord(word)
        }

        currentHaiku = newHaiku
        let replaced: String

        if !word.stringByTrimmingCharactersInSet(NSCharacterSet.
                    whitespaceCharacterSet()).isEmpty {
            replaced = newHaiku.replace(withOneWord: word, ofType: type, atIndex:
                        newHaiku.randIndex(type)!)
        } else {
            replaced = newHaiku.replace(withWordSet: newHaiku.original)
        }
        print(word, " - ", replaced)

        return formatLines(replaced)
    }

    func formatLines(lines: String) -> String {
        let splitLines = lines.componentsSeparatedByString(haikuNewlineMarker)
        var result = ""
        for i in splitLines {
            var temp = i.stringByTrimmingCharactersInSet(NSCharacterSet.
                        whitespaceCharacterSet())
            if temp == "" {
                break
            }
            temp.replaceRange(temp.startIndex...temp.startIndex, with: String
```

```swift
                            (temp[temp.startIndex]).uppercaseString)
            result += temp + "\n"
        }
        return result
    }

    // MARK: Helper method

    /*
    Adds a list of haiku to the existing list
    */
    func addHaikus(newHaikus: [haikuListItem]) {
        for newHaiku in newHaikus {
            managedHaikus.append(Haiku(withCombinedData: newHaiku))
        }
    }

    /*
    Select a random haiku from the managedHaikus list
    */
    func randomHaikuId() -> Int {
        let randId = random() % managedHaikus.count
        if randId == lastRandom {
            return randomHaikuId()
        }
        lastRandom = randId
        return randId
    }

    /*
    Select a haiku from the list with the id
    */
    func getHaikuById(id: Int) -> Haiku {
        return managedHaikus[id]
    }

    /*
    Combination of randomHaikuId and getHaikuByid
    */
    func randomHaiku() -> Haiku {
        return getHaikuById(randomHaikuId())
    }
}
```