TP 3 - Protocols

Adrien Humilière 22/04/2019

With the current situation, we will use repl.it to code this lab. Please use the **Share** button to send me the final project's URL. Also, link it if you have any question for me.

1 Bank Vault

Create a BankVault class:

```swift
class BankVault {
    let name: String
    let address: String
    var amount: Double = 0.0

    init(name: String, address: String) {
        self.name = name
        self.address = address
    }
}
```

It has three instance properties. name which is a constant of type String, address is a constant of type String and amount which is a variable of type Double set to a default value of 0.0.

It also has an initializer which takes in two arguments, both of type String which assigns those passed values to the name and address properties.

(1) - Create a protocol named `ProvideAccess`. This protocol should require any class/struct that conforms to it to implement a function called `allowEntryWithPassword` which takes in one argument called `password` of type `[Int]` (an array of Ints). The function should return a `Bool`.

(2) - Below where you created the `ProvideAccess` protocol - create an extension on the `BankVault` class. In that extension, it should conform to the `ProvideAccess` protocol. In your implementation of the `allowEntryWithPassword(_:)` function, you should adhere to the following rules:

- If the person calling on this function doesn't enter in any digits (empty array), then we should return false.
- If the total numbers of digits entered (items in the array) exceed 10, then we should return false.

- In order to allow access (return true), every other digit (starting with the first digit) must be even. The first digit entered is the first element in the array.

2 Ships

Create a new file, `Ships.swift`. We're going to create various classes in this file, but in a slightly different way.

Protocol-Oriented Programming in Swift - I suggest watching this video on the subject.

You're going to create various classes with a Protocol-Oriented Programming approach.

(1) - In the `Ships.swift` file, create an enum called `Direction` that has four cases. `north`, `south`, `east` and `west`.

(2) - Below the `Direction` enum, create a new protocol called `SteerAbility`. This protocol should require any class/struct conforming to it to provide a property called direction of type `Direction` that is both gettable and settable. It should also require that the class/struct conforming to it should provide a function called `steerInDirection(_:)`. This function should take in an argument called direction which is of type `Direction`. Mark this function as mutating, because in its implementation (which will we will do soon), it will change the value of one of its properties and because it's doing that we need to mark it as mutating. This only affects structs and not classes, but we don't know who will be implementing this protocol, it could be both structs and classes so we are required to mark it as mutating no matter what.

(3) - Below this newly made Protocol - create an extension on the `SteerAbility` protocol.

Like this:

```swift
extension SteerAbility {

}
```

We've just extended a protocol.

In that extension, create a mutating function (the exact one you're asking classes or structs to implement if they were to conform to the `SteerAbility` protocol.) This function should be called `steerInDirection(_:)`. It should have one argument called direction of type `Direction`. It should also be marked as a mutating. You're going to now implement this function. In its implementation, you should set the direction property to equal the direction argument to this function.

```swift
self.direction = direction
```

The reason you have to write `self.direction` is because the argument label matches the name of the instance property.

(4) - Still in the `Ships.swift` file, below where you just created the extension on `SteerAbility`, create a new class called `SailBoat` which conforms to the `SteerAbility` protocol. In conforming to this protocol, you might think that you need to provide both the direction instance property and the `steerInDirection(_:)` function but you're only required to provide the direction property because the `steerInDirection(_:)` has a default implementation. The implementation of the `steerInDirection(_:)` function you provided above within the extension has given us this default implementation.

So provide only an instance property called direction of type `Direction` with a default value of `.north`. Since all properties (only one in this case) are set to an actual value, you aren't required to provide an initializer.

(5) - Below the `SailBoat` class, create a class called `RowBoat` which also conforms to the `SteerAbility` protocol. Just like in the above instruction, only provide a direction property with a default value of `.north`.

(6) - Create a class called `SteamBoat` that also conforms to the `SteerAbility` protocol. Just like above, you should only provide a direction property with a default value of `.north`. You should not implement the `steerInDirection(_:)` as this already has a default implementation.