SWIFT BEST PRACTICES

EXTENSIONS – DIFFICULTY: EASY

INTRODUCTION

Swift's extensions allow you to add or modify functionality in existing data types, which by itself is a common feature of many languages. However, Swift adds two features that make extensions an indispensable feature of the language:

1. You can extend *protocols* as well as concrete types, which means you can add functionality to a whole range of types at once.
2. You can add *constraints* to your extensions, ensuring your modifications apply only to a specific subset of targets.

Combined, these help make extensions indispensable parts of the Swift language.

THE PROBLEM

Many methods we write don't need to belong to a specific type – we just place them there because that's where they are used. This adds clutter, making it more difficult to understand all the functionality the type offers.

Sometimes there's no better place for code, but a lot of the time you'll find you can move the code to an extension on a different type. For example, this view controller has a method clamps one integer between minimum and maximum values:

```
class ViewController {
    func clamp(number: Int, low: Int, high: Int) -> Int {
        if (number > high) {
            return high
        } else if (number < low) {
            return low
        }

        return number
    }
}
```

Putting it inside the view controller makes it harder to test, and harder to re-use. Worse, it works only on **Int**, as opposed to other varieties of integer.

THE SOLUTION

It's usually a good idea to spin off these orphan methods into extensions, either on a concrete type or a protocol. In the case of the **clamp()** method, we could extend **BinaryInteger** so that all integer types benefited, while also allowing other parts of our code to use the functionality easily:

```
extension BinaryInteger {
    func clamp(low: Self, high: Self) -> Self {
        if (self > high) {
            return high
        } else if (self < low) {
            return low
        }

        return self
    }
}
```

THE CHALLENGE

Our app has a lot of code inside the **viewDidLoad()** method of **ViewController**. It's mostly responsible for locating a JSON file in our app bundle, loading it into a **Data** instance, then attempting to decode it into some sort of object. This is a very common thing to want to do, so can you spin that off into an extension on a different type?

You can choose any type you like, but you might find **Bundle** a good choice.

You don't need to make the method generic, but it becomes much more useful if you do. I've included a suggested function signature on the following page in case you're not sure how to do this, but I strongly recommend you try it yourself before looking ahead.

Still here?

OK, this should help you get started:

```
extension Bundle {
  func decode<T: Decodable>(_ type: T.Type, from file: String) -> T {
      // ...
  }
}
```