8 JUNE 2018 / DESIGN PATTERNS

Swift prototype design pattern

The prototype design pattern is used to create clones of a base object, so let's see some practical examples written in Swift.

This is also a creational design pattern, it is useful when you have a very basic configuration for an object and you'd like to give (clone) those predefined values to another one. Basically you're making clones from a prototype objects. 😊😊😊

This approach has some benefits, one is for example that you don't have to subclass, but you can configure clones individually. This also means that you can remove a bunch of boilerplate (configuration) code if you are going to use prototypes. 🤔

Prototype.swift 714 Bytes on GitLab

```
class Paragraph {

    var font: UIFont
    var color: UIColor
    var text: String

    init(font: UIFont = UIFont.systemFont(ofSize: 12),
         color: UIColor = .darkText,
         text: String = "") {

        self.font = font
        self.color = color
        self.text = text
    }

    func clone() -> Paragraph {
        return Paragraph(font: self.font, color: self.color, text: self.text)
    }
}

let base = Paragraph()

let title = base.clone()
title.font = UIFont.systemFont(ofSize: 18)
title.text = "This is the title"

let first = base.clone()
first.text = "This is the first paragraph"

let second = base.clone()
second.text = "This is the second paragraph"

```

As you can see the implementation is just a few lines of code. You only need a default initializer and a clone method. Everything will be pre-configured for the prototype object in the init method and you can make your clones using the clone method, but that's pretty obvious at this point... 🤐

Let's take a look at one more example:

Prototype.swift 714 Bytes on GitLab

```
class Paragraph {

    var font: UIFont
    var color: UIColor
    var text: String

    init(font: UIFont = UIFont.systemFont(ofSize: 12),
         color: UIColor = .darkText,
         text: String = "") {

        self.font = font
        self.color = color
        self.text = text
    }

    func clone() -> Paragraph {
        return Paragraph(font: self.font, color: self.color, text: self.text)
    }
}

let base = Paragraph()

let title = base.clone()
title.font = UIFont.systemFont(ofSize: 18)
title.text = "This is the title"

let first = base.clone()
first.text = "This is the first paragraph"

let second = base.clone()
second.text = "This is the second paragraph"

```

The prototype design pattern is also helpful if you are planning to have snapshots of a given state. For example in a drawing app, you could have a shape class as a proto, you can start adding paths to it, and at some point at time you could create a snapshot from it. You can continue to work on the new object, but this will give you the ability to return to a saved state at any point of time in the future. 🎉

That's it about the prototype design pattern (in Swift | in a nuthsell). 🐿

External sources

- Prototype pattern
- Prototype pattern in Swift
- What's the point of the Prototype design pattern?

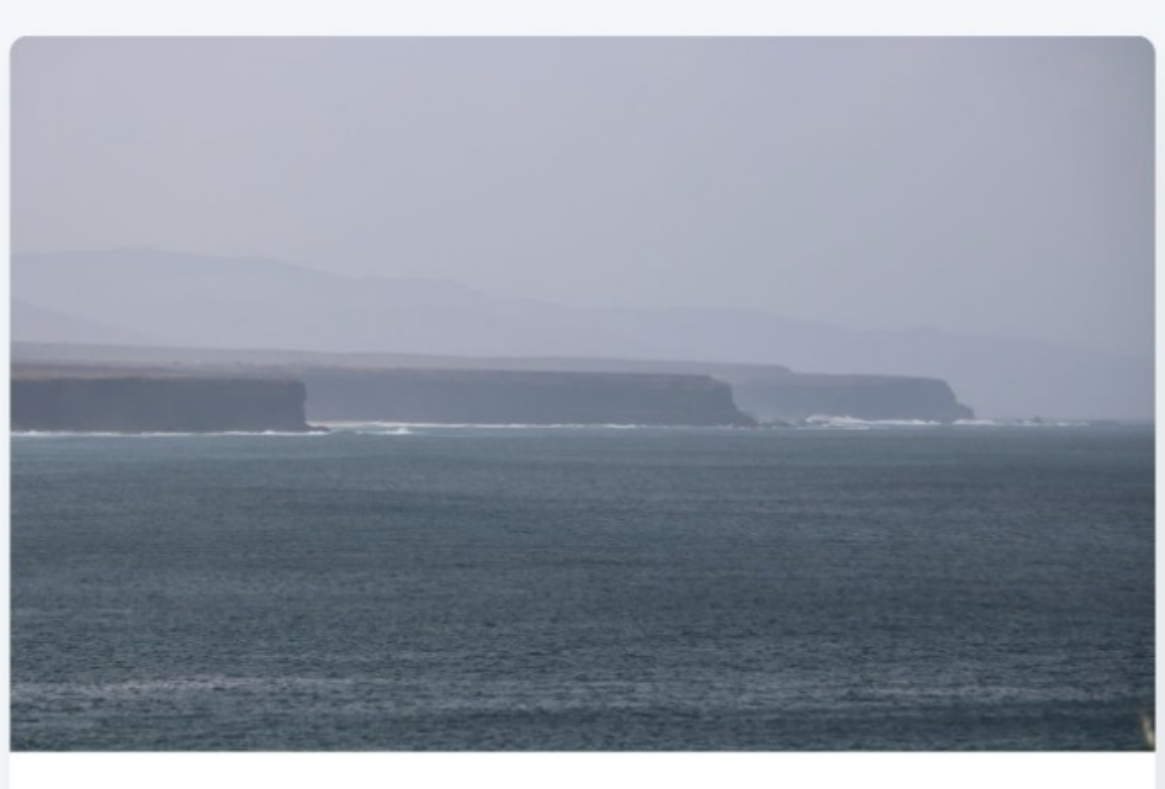
Tibor Bödecs
Read more posts by this author.

Read More

— The.Swift.Dev. —
Design patterns

- Lazy initialization in Swift
- Swift object pool design pattern
- Swift facade design pattern

See all 18 posts →



DESIGN PATTERNS

Swift command design pattern

This time I'm going to show you a behavioral pattern. Here is a little example of the command design patten written in Swift.

1 MIN READ

DESIGN PATTERNS

Comparing factory design patterns

Learn what's the difference between static factory, simple factory, factory method and abstract factory using the Swift language.

1 MIN READ