



Published in Dev Genius

You have **2** free member-only stories left this month.

[Sign up for Medium and get an extra one](#)



Israel Josué Parra Rosales

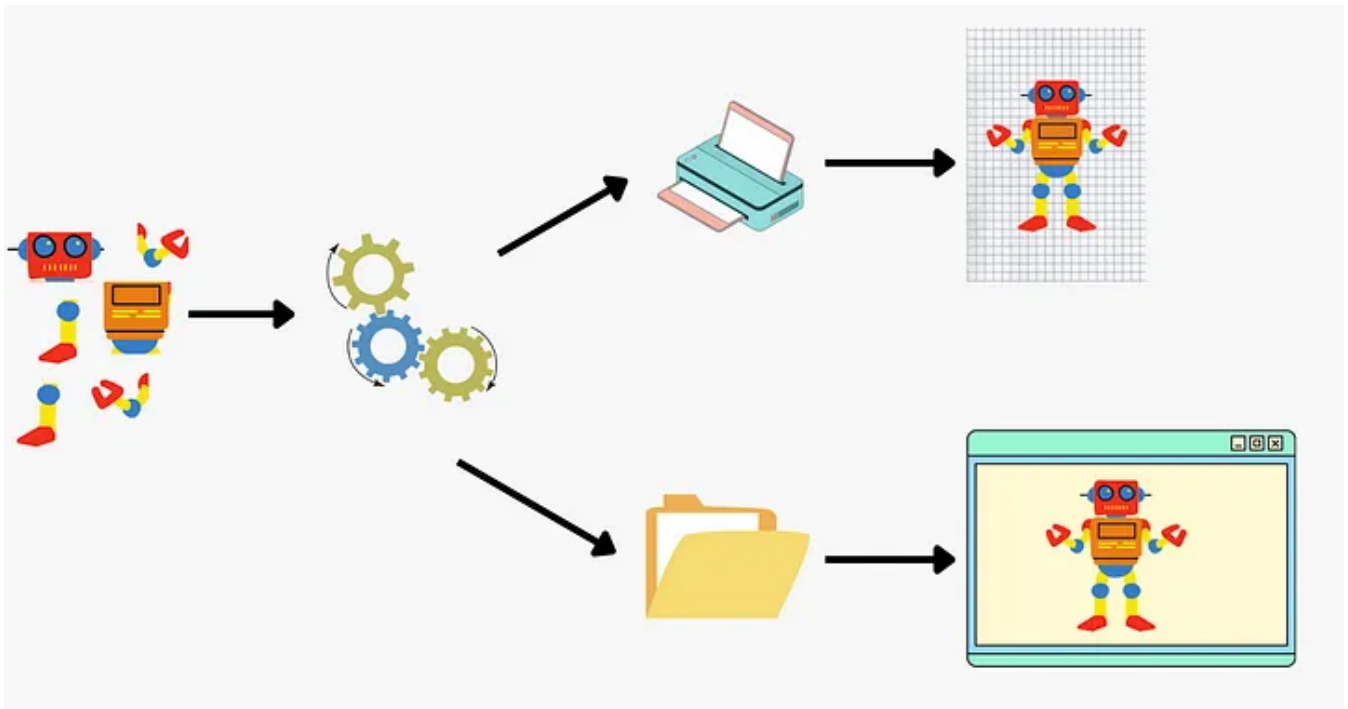
Follow

Jan 4 · 3 min read · ✨ · [Listen](#)

Save



Strategy Pattern in Go



The Strategy Pattern is a behavioral design pattern. This design pattern allows you to change the behavior of an object at run time without any change in the class of that object.

As we can read this pattern is easy to understand, now let us have an example to know how to implement it.

For this example let us define two structs “printer” and “display”, and also we going to define an interface “outputter” that going to be implemented by both structs.

Outputter interface defines two methods one to know the destination and another one to send the image.

First, let us take a look at “printer” struct and the interface methods implemented by it. The destination is a string to simulate a printer machine, the “sendImage” method will send the image to the printer machine and the “getDestination” will take the destination value. On the other hand “display” the destination is a string to simulate the application name and the “sendImage” method will show the image on the screen.

```
1  package main
2
3  import "fmt"
4
5  type outputter interface {
6      sendImage()
7      getDestination() string
8  }
9
10 type printer struct {
11     destination string
12 }
13
14 func (p printer) sendImage() {
15     fmt.Println("printing image")
16 }
17
18 func (p printer) getDestination() string {
19     return p.destination
20 }
21
22 type display struct {
23     destination string
24 }
25
26 func (d display) sendImage() {
27     fmt.Println("drawing image in screen")
28 }
29
30 func (d display) getDestination() string {
31     return d.destination
32 }
```

Now is time to see the strategy pattern in action, for this example let us keep it simple. In *line 34* we can see defined the “draw” function that takes as a param the

interface type, that going to help us execute the methods from “printer” and “display” and change the behavior of the draw function.

```
34 func draw(output outputter) {
35     fmt.Println("Preparing destination: ", output.getDestination())
36     output.sendImage()
37 }
38
39 func main() {
40     p := printer{destination: "printer machine 007"}
41     d := display{destination: "image viewer"}
42
43     draw(p)
44     fmt.Println("-----")
45     draw(d)
46 }
```

Output:

```
Preparing destination: printer machine 007
printing image
-----
Preparing destination: image viewer
drawing image in screen
```

Additional Information

After learning how this pattern could be implemented with Golang and before ending this article let me give you some bonus information, I will mention a couple of pros and cons and how this pattern could be related to other patterns.

Relationship with other patterns



- Decorator allows you to change the structure of an object, while Strategy allows you to change its innards.

- State and Strategy Patterns are similar to each other because both patterns allow changing the behavior and their main differences are the following:
On one hand Strategy Pattern makes these objects completely independent and unknown to each other, in the other hand, State Pattern does not restrict dependencies between particular states, allowing them to alter the state of the context at will.

Pros and Cons



- Helps to modify in run time the algorithm defined for an object.
- Allows isolating the implementation details of an algorithm from the code that uses it.
- Helps to apply the Open-closed principle. You can introduce new strategies without having to change the existing context.



- If you only have a couple of algorithms that rarely change, there is no real reason to overcomplicate the program with new classes and interfaces that come with the pattern.
- At the time to execute the strategies the developer must know the differences in order to choose the correct one.

Note:


This article is related to “Desing Patters in Golang” series that you can find listed here <https://blog.devgenius.io/desing-patters-in-golang-1b03de4fb89>


Sign up for DevGenius Updates

By Dev Genius

Get the latest news and update from DevGenius publication [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

 Get this newsletter

Open in app 

Sign up

Sign In



About

Help

Terms

Privacy

Get the Medium app

