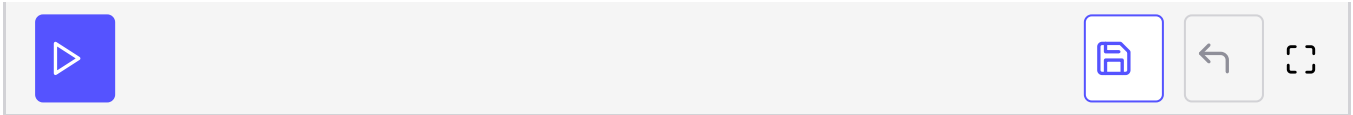
Solution Review: Make a Simple Interface

This lesson discusses the solution to the challenge given in the previous lesson.

```
package main
import (
        "fmt"
)

type Simpler interface {  // interface implementing functions called on Simple struct
        Get() int
        Set(int)
}

type Simple struct {
        i int
}

func (p *Simple) Get() int {
        return p.i
}

func (p *Simple) Set(u int) {
        p.i = u
}

func fI(it Simpler) int {     // function calling both methods through interface
        it.Set(5)
        return it.Get()
}

func main() {
        var s Simple
        fmt.Println(fI(&s))  // &s is required because Get() is defined with a receiver type
}
```



Implementing a Simple Interface

In the code above, from **line 6** to **line 9**, we define the interface `Simpler` by specifying the signature of the functions `Get` and `Set`. The `Get()` function returns an integer, and the `Set(int)` function takes an integer as a parameter. Note that because these are only function descriptions, we do not have to specify variables' names here.

The type `Simple` itself is defined from **line 11** to **line 13**. It contains a field `i` of type *int*. In order for `Simple` to implement the interface `Simpler`, it has to implement the functions `Get` and `Set` as methods, working on a receiver argument `p` of type `*Simple` (see **line 15** and **line 19**). The `Get` will return the value of field `i` of `p`, as shown at **line 16**. The `Set` will take an integer `u` and change `p.i` to this value, as shown at **line 20**.

Our test function `fI` (from **line 23** to **line 26**) will take a parameter `it` of type `Simpler` and will call both methods of `Simpler`. The parameter `it` will have to contain an integer field at the very least (`it` could contain additional fields as well). We will first call `Set` (**line 24**) to give the integer field an arbitrary value **5**, and then return the value of the integer field with a call to `Get` (**line 25**).

Finally, in `main()` we define a variable `s` of type `Simple` at **line 29**. Then, at **line 30**, we pass a reference to `s` (`&s`) as a parameter to the test function `fI`. `&s` is required because `Get()` is defined with a receiver type pointer `*Simple`. The function `fI` first sets `s.i` to **5**, and then gets the value of `s.i`, which is printed out.

That's it about the solution. In the next lesson, you'll see how an interface can be embedded inside another interface.