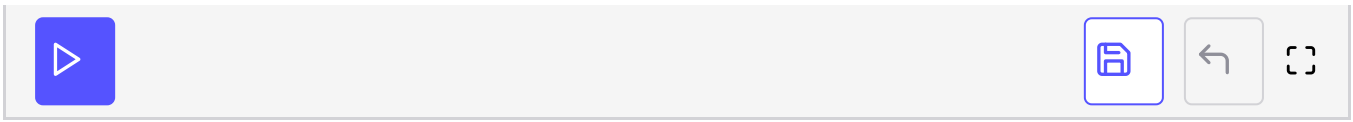
Solution Review: Advancing the Simple Interface

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

```
package main
import (
        "fmt"
)

type Simpler interface {  // interface declaring Simple methods
        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
}

type RSimple struct {
        i int
        j int
}

func (p *RSimple) Get() int {
        return p.j
}

func (p *RSimple) Set(u int) {
        p.j = u
}

func fI(it Simpler) int { // switch cases to judge whether it's Simple or RSimple
        switch it.(type) {
        case *Simple:
                        it.Set(5)
                        return it.Get()
        case *RSimple:
                        it.Set(50)
                        return it.Get()
        default:
                        return 99
        }
        return 0
}
```

```
}

func main() {
        var s Simple
        fmt.Println(fI(&s))  // &s is required because Get() is defined with the type pointer
        var r RSimple
        fmt.Println(fI(&r))
}
```



Advancing a Simple Interface

As we stated in the discussion of the previous challenge, we can make other types that implement the interface `Simpler`; the only condition is that they have at least one integer field. We illustrate this here by defining a type `RSimple` (see implementation from **line 23** to **line 26**), which has two integer fields `i` and `j`. We define the `Get` and `Set` methods on `RSimple` in the exact same way as for `Simple` (see implementation from **line 28** to **line 34**).

In our test function `fI`, we want to differentiate between a parameter of type `Simple` (which we will give a value of 5) and a parameter of type `RSimple` (which we will give a value of 50). To do this, we'll make use of the type switch:

- At **line 38**, we have our *first* case. If the type is `*Simple`, we'll set the value **5** via `Set` and return it via `Get`.
- At **line 41**, we have our *second* case. If the type is `*RSimple`, we'll set the value **50** via `Set` and return it via `Get`.
- At **line 44**, we have our *default* case. If none of the above cases are true, it'll simply return **99**.

Because the methods work on pointers to the struct types (`*Simple` and `*RSimple`), we have to use pointer types in the cases as well.

In our `main()` function, we call `fI` with a reference to a variable `r` of type `RSimple` at **line 54**, just as we did for type `Simple` before.

That's it for the solution. In the next lesson, you'll study how to implement interfaces.