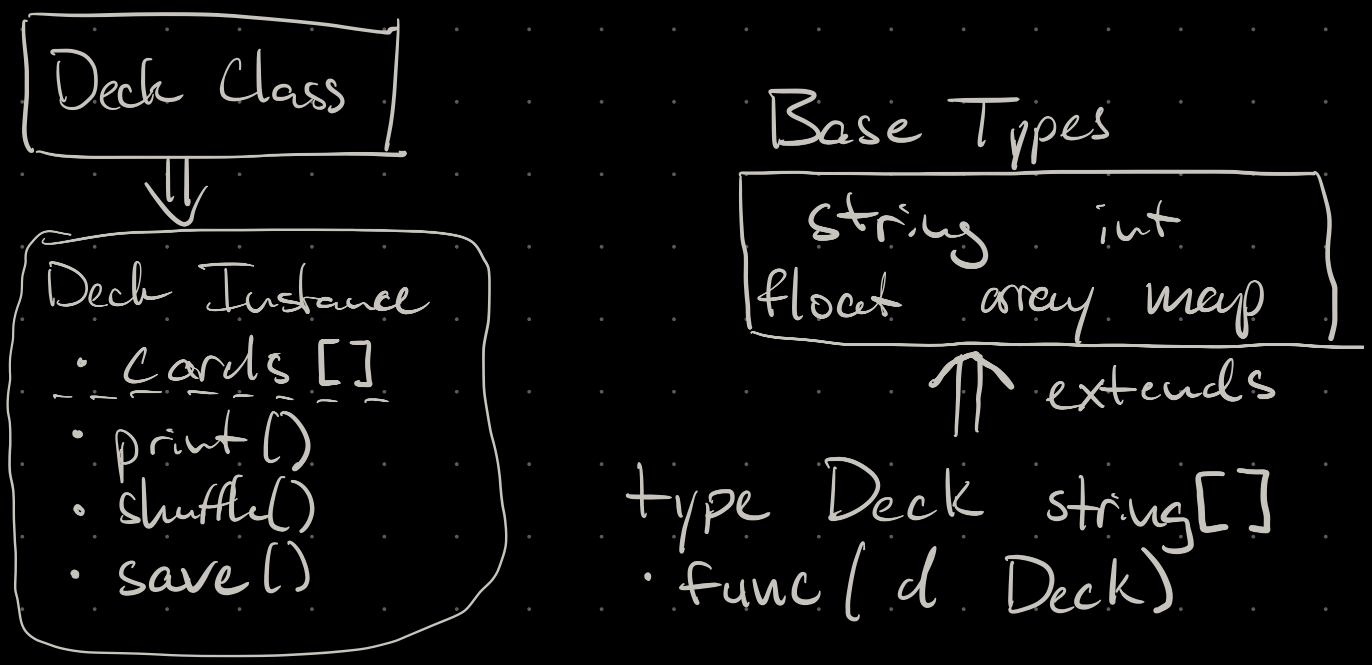
Go: The Complete Developer's Guide

~ chapter 18

Cards project

Deck OO approach ⟶ Go approach



Deck Class

Deck Instance
- cards []
- print()
- shuffle()
- save()

Base Types

string int
float array map

extends

type Deck string[]
- func (d Deck)

"create a new type that extends the base types and create funcs with deck as a receiver"

Receivers

func $\boxed{\text{(d deck)}}$ print() {}

func print() {} with a (d deck) Receiver

→ any variable w/ type 'deck' gets access to the print method

Value Types	Reference Types
int	slice
float	map
string	channel
bool	pointer
struct	function

use pointers to modify the underlying

- no overloading function names

create a new type called "bot"

```
type bot interface {
    getGreeting() string
}
```

if a type in this program implements a getGreeting function that returns a string, then you are a member of the "bot" type and may call our new printGreeting() function

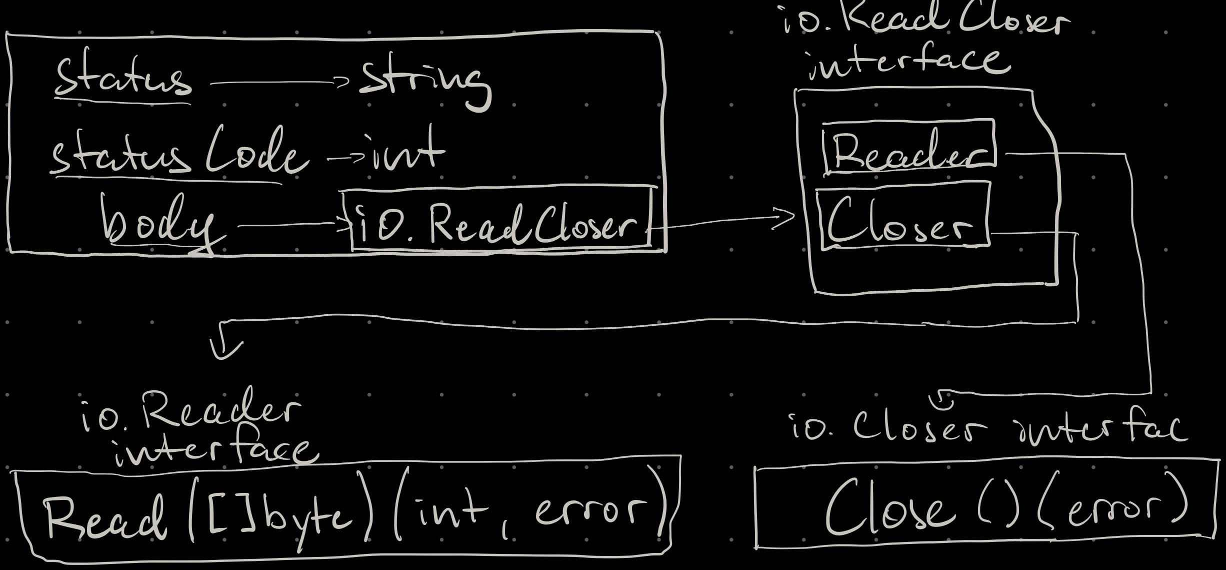
```
func printGreeting(b bot) {}
```

In the bot program

Concrete Types	Interface Types
map struct	
int string	bot
englishBot	

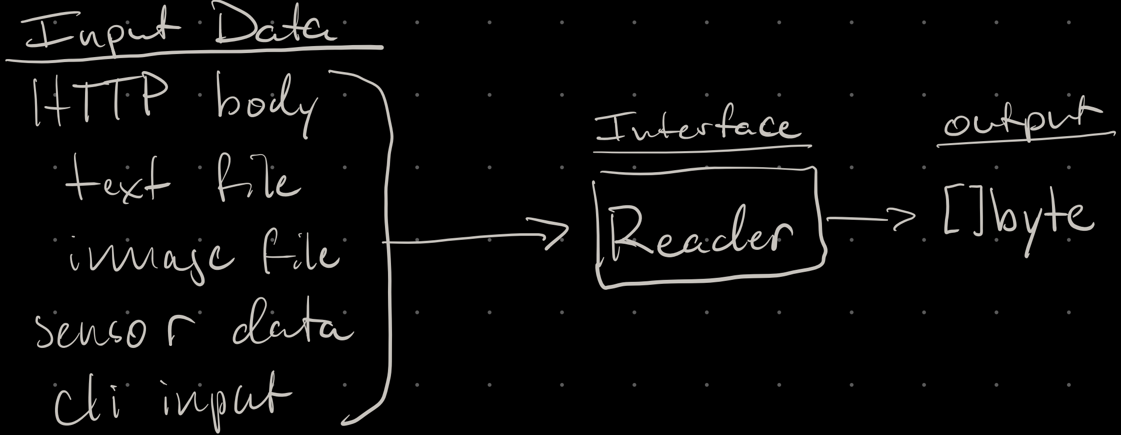
- interfaces are <u>not</u> <u>generic</u> <u>types</u>
- interfaces are <u>implicit</u> - no "implements"
- interfaces are a contract to manage types
- interfaces are tough - need to understand how to read them
  - not a requirement of the language to write your own, which requires some experience

net. http Responese

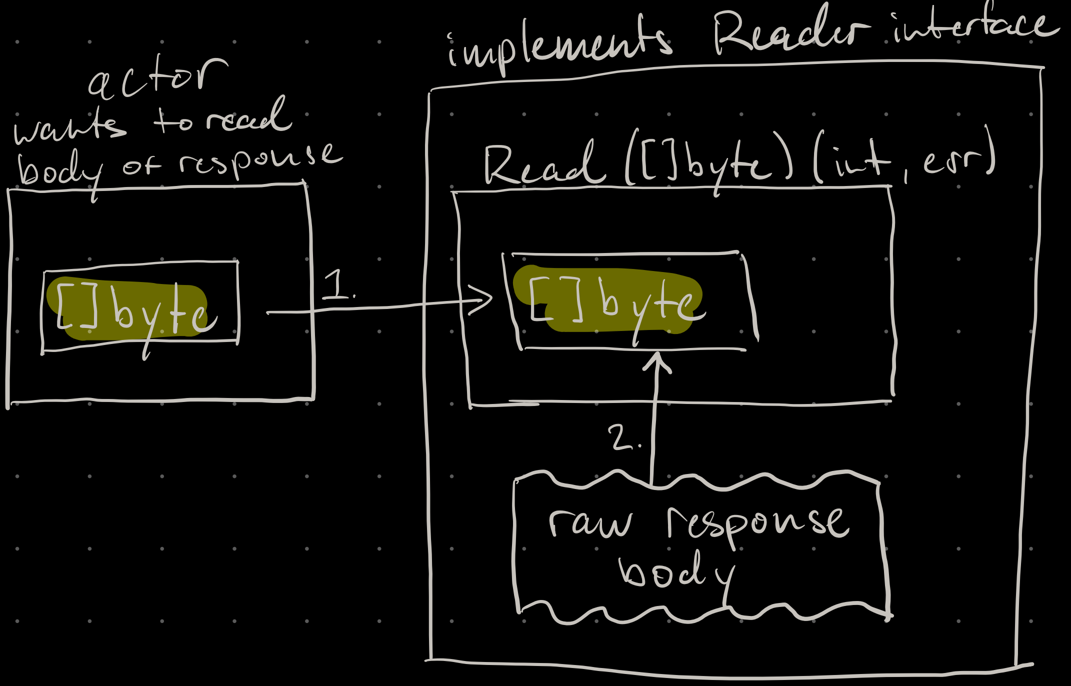


- interfaces can be embedded

consider print() ...

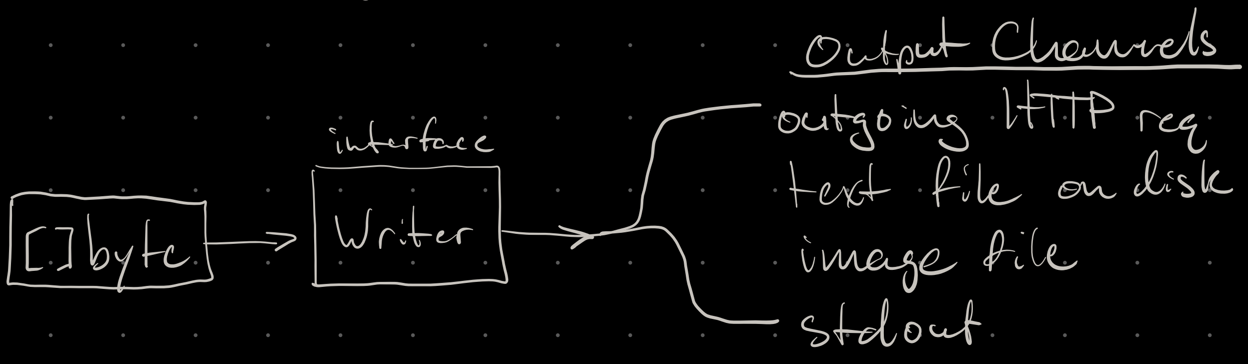


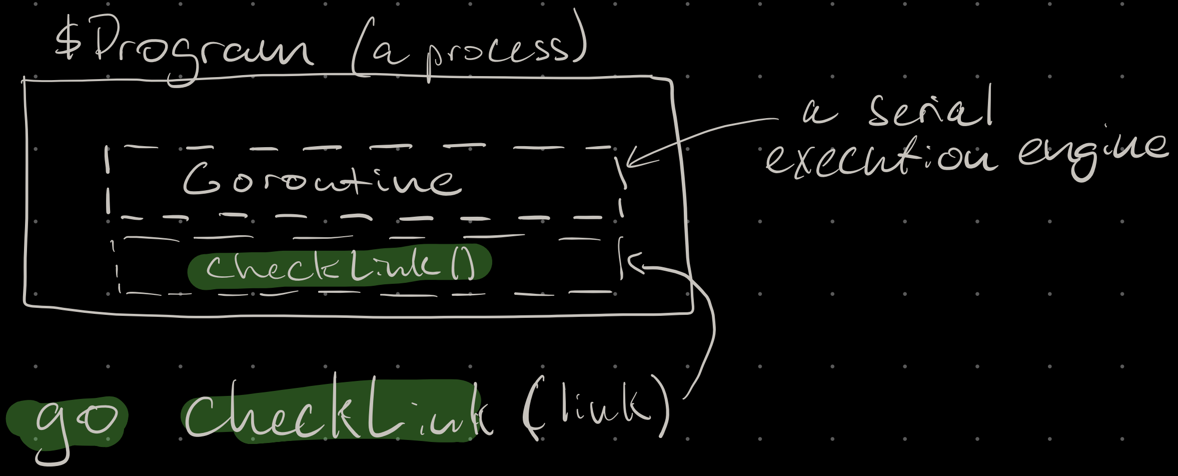
how the Read interface works



1) we create a []byte, pass it to Read()

2) Read() takes the data from the raw response body and injects/pushes it into that slice - ultimately modifying the underlying variable




$Program (a process)
Goroutine
checkLink()
a serial
execution engine
go checkLink(link)