Go brain teasers

EXERCISE YOUR MIND

```
1 package main
2
3 import (
4       "fmt"
5 )
6
7 func main() {
8       var π = 22 / 7.0
9       fmt.Println(π)
10 }
```

WILL THIS CODE COMPILE? WHAT WILL IT PRINT?

25 MIND BENDING TEASERS & SOLUTIONS

MIKI TEBEKA

Copyright

Copyright © 2020, 353solutions LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

A Job to Do

job.go

```
1  package main
2
3  import (
4      "fmt"
5  )
6
7  type Job struct {
8      State string
9      done  chan struct{}
10 }
11
12 func (j *Job) Wait() {
13     <-j.done
14 }
15
16 func (j *Job) Done() {
17     j.State = "done"
18     close(j.done)
19 }
20
21 func main() {
22     ch := make(chan Job)
23     go func() {
24         j := <-ch
25         j.Done()
26     }()
27
28     job := Job{"ready", make(chan struct{})}
29     ch <- job
30     job.Wait()
31     fmt.Println(job.State)
32 }
```

Try to guess what the output is before moving to the next page.

This code will print: `ready`

At first glance, it looks like the code is OK. You're using a pointer receiver in the `Job` struct methods. The fact that the call to `Wait` returned tells you that the channel was closed.

The problem is with the definition of `ch`. It is a channel of `Job`, not `*Job`, which means that when you send the variable `job` over the channel, you actually send a copy of it. A channel in Go is a "pointer like" type, so even though there is a copy of `job` inside the goroutine, `j.done` points to the same channel `job.done` is pointing to.

Strings in Go are not "pointer like". When you call `j.Done()` inside the goroutine, you change the value of the `State` field in the goroutine copy of `job`. This change is not reflected in the `job` variable declared in line 28.

The solution is to make `ch` type `*Job`

job_ptr.go

```go
package main

import (
    "fmt"
)

type Job struct {
    State string
    done  chan struct{}
}

func (j *Job) Wait() {
    <-j.done
}

func (j *Job) Done() {
    j.State = "done"
    close(j.done)
}

func main() {
    ch := make(chan *Job)
    go func() {
        j := <-ch
        j.Done()
    }()

    job := Job{"ready", make(chan struct{})}
    ch <- &job
    job.Wait()
    fmt.Println(job.State)
}
```

Further Reading

- There is no pass-by-reference in Go
- Channel types in the Go specification
- Go Concurrency Patterns: Pipelines and cancellation
- Channels in the Go tour