GO on the toilet

episode #4

<u>Go is kind of object-oriented.</u>

```
type HealthCheck struct {
  Interval   time.Duration
  Healthy    bool
}
func (hc HealthCheck) isHealthy() bool {
  return hc.Healthy
}
```

Above you have what you could call a class. It is a struct though with function defined on it.

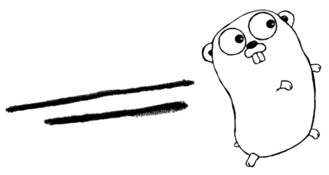
```
check := HealthCheck{
  Interval: 2*time.Second,
  Healthy: true,
}
check2 := &HealthCheck{1*time.Second, true}
```

Look-n-feel of the struct with function defined on it is class-like. You use it as object just like in other languages. Above there are two definitions. First is more verbose (it is called **field:value initialization**), in the second initialization is implicit (**value**) initialization.

There is also ampersand (&) symbol. It means that **`check`** is of type **HealthCheck** and **`check2`** is of type **"pointer to HealthCheck"**. It matters a lot, but let's not go into the details today. I will just say that you don't have to care about pointers when accessing fields/functions:

```
check.isHealthy()   // this gives true
check.Healthy       // this also gives true
check2.isHealthy()  // gives also true
check2.Healthy      // surprise, surprise it gives true
```

By the way did you know that there is no private/public keywords in go? If struct/function/variable starts with capital letter, you can use it outside of the package, if not - you can't! ¯_(ツ)_/¯



Bathroom magazine with golang trivia, examples and patterns
Inspired by original Google's "Testing on the Toilet"