Our tests will no longer compile because we are trying to pass in a `Counter` rather than a `*Counter`. To solve this I prefer to create a constructor which shows readers of your API that it would be better to not initialise the type yourself.

```
func NewCounter() *Counter {
        return &Counter{}
}
```

Use this function in your tests when initialising `Counter`.

Wrapping up

We've covered a few things from the sync package

- `Mutex` allows us to add locks to our data
- `Waitgroup` is a means of waiting for goroutines to finish jobs

When to use locks over channels and goroutines?

We've previously covered goroutines in the first concurrency chapter which let us write safe concurrent code so why would you use locks? The go wiki has a page dedicated to this topic; Mutex Or Channel

> A common Go newbie mistake is to over-use channels and goroutines just because it's possible, and/or because it's fun. Don't be afraid to use a sync.Mutex if that fits your problem best. Go is pragmatic in letting you use the tools that solve your problem best and not forcing you into one style of code.

Paraphrasing:

- **Use channels when passing ownership of data**
- **Use mutexes for managing state**