Day 25: The null object design pattern

Another easy one - Java used to only have `null` for objects, such that when that object was referenced, a `NullPointerException` would be thrown. Scala aims to instead create an object capable of expressing missingness, which Scala calls `Option[_]`. This object is an instance of a monad, with monadic flows defined around `Some(v)` and `None` - with this, Scala developers can express complex workflows involving the possibility of missingness, without having a referential-transparency-breaking exception thrown for `null`.

The code example provided in the book discusses a queue that polls on an interval, therefore yielding the possiblity of missingness. This is expressed via an `Option` type, like: `def getMessage(): Option[Message] = Option(queue.poll()).map { case number => Message(number) }`.

This is interesting only in this sense: `Option(null)` == `None`! Scala makes that transformation for you, which is very useful. So now, when `getMessage` is called, you can `map` or `foreach` over the return and your missingness is automatically handled. Thanks, monads! Or in this case, functors.