The Object Orientation in Go

This lesson summarizes how object orientation is handled by Go and upto which extent object orientation can be handled by Go.

Summary on object orientation

Let us summarize what we have seen about object orientation in Golang so far: Go has no classes, but instead, it has loosely coupled types and their methods to implement interfaces. The three important aspects of OO-languages are encapsulation, inheritance, and polymorphism. How are they realized in Go?

- **Encapsulation (data hiding)**: in contrast to other OO languages where there are four or more access-levels, Go simplifies this to only two levels:
  - *package scope*: object is only known in its package, it starts with a lowercase letter
  - *exported*: object is visible outside of its package because it starts with an uppercase letter. A type can only have methods defined in its package.
- **Inheritance**: via composition that is embedding of 1 (or more) type(s) with the desired behavior (fields and methods); multiple inheritance is possible through embedding various types.
- **Polymorphism**: via interfaces in which a variable of a type can be assigned to a variable of any interface it implements. Types and interfaces are loosely coupled; again, multiple inheritance is possible through implementing various interfaces. Go's interfaces aren't a variant on Java or C# interfaces, they're much more. They are independent and are key to large-scale programming and adaptable, evolutionary design.

So in effect, we can say that Go implements all important object-oriented features.

Even without classes and inheritance, Go has managed so well to be an object-oriented language. There is a quiz in the next lesson for you to solve.