Object-Oriented Go

Simple and Fast Object Orientation

Ian Harris

December 14, 2018

1 Go is Weakly Object-Oriented

So, Go language is object-oriented, but let's say, weakly object-oriented. So, what I mean by that is that it implements objects but maybe they have fewer features than you would see in another object-oriented language like Python or Java or C++ or something like that.

1.1 Review of Object Oriented Language Features

So, I'm just going to summarize what object-oriented languages are, what object-oriented programming is very briefly just so I can highlight some of the differences in Go's implementation.

1.1.1 Code Organization

So, object-oriented programming, it's really for code organization, in my mind is for code organization. So, you organize your code by encapsulating your code. You group together data and functions that are related to each other.

1.1.2 Self-Defined Types

So, this is essentially what a type is. So, what object-oriented programming and lets you do is essentially create a user-defined type, one that's specific for whatever application it is that you're building. So, typical types, you have integers, floats, that sort of thing. Those are generic. You can use those in any kind of program. But when you're making a specific application, you might like to have a type that is specific to that application domain. You can create that. Object-oriented programming allows you to create that.

1.1.3 Data and Methods

So, if you think about a type, your standard type, like an integer, each integer it has data, the number, the value of the number, and then it has functions, the functions that you can apply to the data. So, you can apply addition,

subtraction, multiplication, that sort of thing. So, a type has data and has set of functions that you can apply to the data. So, object-oriented programming is the same idea. You're creating types. But they are more complicated, they can be more complicated, and they are specific to your application.

Example: As an example, let's say you're making an application and it's going to do geometry, some kind of geometric operations in three dimensions. So, many of the functions that you're going to write are going to operate on points. You're going to have this idea of points because you're doing 3D geometry. Each point is going to have some data associated with it, specifically x, y, and z coordinates. Maybe you want to put other data in there, but at least, you're going to have that x, y, z coordinates because in 3D. Also points are going to have a set of functions, a set of functions that you can use to operate on points. So, distance to origin, maybe every point has a distance to its origin. You can have a function that computes that. Quadrant, it tells you what quadrant. Maybe it returns what quadrant the point is in, something like this. There are a lot of functions that you can imagine that work on points. So, if I think about this idea of points, I got a bunch of data that I want all related together. So, I want the x, and the y, and the z variables for a particular point, I want them to be somehow associated together. Also, I'd like these functions that operate on points to be associated with that too.

1.1.4 Class as Definition of Type

So, in typical object-oriented languages, you have this idea of a class. Now, I should note right now, Go does not use this term class. But I bring it up because it's used in other object-oriented languages and so people might be familiar with this idea. So, this class, it defines that may I make a point class, and the class defines all the data that would be inside a point. So, x, y, and z, maybe they're all floating point value, something like that. The class would define all the functions that you would have that are associated with points, distance to origin, quadrant, and so on. So, that will be the class.

1.1.5 Object as Instantiation of Class

Then an object is an instantiation of the class. So, I can make my point class but then I might have many different actual points with actual data values inside. So, if I have a triangle, I've got three points with three sets of x, y, and z values. So, I might have this point class that's sort of a general template of what point should have, but then the point objects, my three-point object will have actual values for x, and y, and z and so on. So, that's the idea, that's the terminology that you normally see use, classes and objects which are instances of that class.

1.2 Go's Object-Oriented Features

So, different languages have this. Java is very popular, Java, Python, C++, and so on.

Structs: Now Go, we don't use this term class, instead they use structs. Now structs, actually, this goes back to C and probably before that. But the idea of a struct is a struct is just the data. So, the different types of data that you want to associate together. So, like with our point class, you'd have a point struct and it would have an x, and a y, and a z, maybe they're all three floating points. So, just the data are related together.

Structs with Methods: But also you can associate methods or functions with those structs. So, the struct ends up being like what you would call a class in a normal object-oriented language. So, you got the structs that had some data, some fields of data associated with them plus some methods that you want to define.

Simplified Features of Go: Now, Go's implementation of structs is simplified compared to traditional implementation of classes. So, you don't have inheritance, you don't have constructors, and you don't have generics, none of those. Now, this, one can argue it makes it easier to code, also it makes it efficient to run. So, it typically runs faster. But it can make it easier to code unless you like those features. Now, if you'd like inheritance, and generics, and constructors. Then, you can see this is a disadvantage. But Go is different. It has objects but is different than traditional object-oriented implementation and a linear object-oriented implementation.