Part 27: Composition Instead of Inheritance - OOP in Go

04 SEPTEMBER 2017

Welcome to tutorial no. 27 in Golang tutorial series.

Go does not support inheritance, however it does support composition. The generic definition of composition is "put together". One example of composition is a *car*. A car is composed of wheels, engine and various other parts.

Composition by embedding structs

Composition can be achieved in Go is by embedding one struct type into another.

A blog post is a perfect example of composition. Each blog post has a title, content and author information. This can be perfectly represented using composition. In the next steps of this tutorial we will learn how this is done.

Let's first create the `author` struct.

```
package main

import (
    "fmt"
)

type author struct {
    firstName string
    lastName  string
    bio       string
}

func (a author) fullName() string {
    return fmt.Sprintf("%s %s", a.firstName, a.lastName)
}
```

In the above code snippet, we have created a `author` struct with fields `firstName`, `lastName` and `bio`. We have also added a method `fullName()` with the `author` as receiver type and this returns the full name of the author.

The next step would be to create the `post` struct.

```
type post struct {
    title     string
    content   string
    author
}

func (p post) details() {
    fmt.Println("Title: ", p.title)
    fmt.Println("Content: ", p.content)
    fmt.Println("Author: ", p.author.fullName())
    fmt.Println("Bio: ", p.author.bio)
}
```

The post struct has fields `title`, `content`. It also has an embedded anonymous field `author`. This field denotes that `post` struct is composed of `author`. Now `post` struct has access to all the fields and methods of the `author` struct. We have also added `details()` method to the `post` struct which prints the title, content, fullName and bio of the author.

Whenever one struct field is embedded in another, Go gives us the option to access the embedded fields as if they were part of the outer struct. This means that `p.author.fullName()` in line no. 11 of the above code can be replaced with `p.fullName()`. Hence the `details()` method can be rewritten as below,

```
func (p post) details() {
    fmt.Println("Title: ", p.title)
    fmt.Println("Content: ", p.content)
    fmt.Println("Author: ", p.fullName())
    fmt.Println("Bio: ", p.bio)
}
```

Now that we have the `author` and the `post` structs ready, let's finish this program by creating a blog post.

```
package main

import (
    "fmt"
)

type author struct {
    firstName string
    lastName  string
    bio       string
}

func (a author) fullName() string {
    return fmt.Sprintf("%s %s", a.firstName, a.lastName)
}

type post struct {
    title   string
    content string
    author
}

func (p post) details() {
    fmt.Println("Title: ", p.title)
    fmt.Println("Content: ", p.content)
    fmt.Println("Author: ", p.fullName())
    fmt.Println("Bio: ", p.bio)
}

func main() {
    author1 := author{
        "Naveen",
        "Ramanathan",
        "Golang Enthusiast",
    }
    post1 := post{
        "Inheritance in Go",
        "Go supports composition instead of inheritance"
        author1,
    }
    post1.details()
}
```

Run in playground

The main function in the program above creates a new author in line no. 31. A new post is created in line no. 36 by embedding `author1`. This program prints,

```
Title:  Inheritance in Go
Content:  Go supports composition instead of inheritance
Author:  Naveen Ramanathan
Bio:  Golang Enthusiast
```

Embedding slice of structs

We can take this example one step further and create a website using a slice of blog posts :).

Let's define the `website` struct first. Please add the following code above the main function of the existing program and run it.

```
type website struct {
        []post
}
func (w website) contents() {
    fmt.Println("Contents of Website\n")
    for _, v := range w.posts {
        v.details()
        fmt.Println()
    }
}
```

When you run the program above after adding the above code, the compiler will complain with the following error,

```
main.go:31:9: syntax error: unexpected [, expecting field name or embedded type
```

This error points to the embedded slice of structs `[]post`. The reason is that it is not possible to anonymously embed a slice. A field name is required. So let's fix this error and make the compiler happy.

```
type website struct {
        posts []post
}
```

I have added the field name `posts` to the slice of post `[]post`.

Now let's modify the main function and create a few posts for our new website.

The complete program after modifying the main function is provided below,

```
package main

import (
    "fmt"
)

type author struct {
    firstName string
    lastName  string
    bio       string
}

func (a author) fullName() string {
    return fmt.Sprintf("%s %s", a.firstName, a.lastName)
}

type post struct {
    title   string
    content string
    author
}

func (p post) details() {
    fmt.Println("Title: ", p.title)
    fmt.Println("Content: ", p.content)
    fmt.Println("Author: ", p.fullName())
    fmt.Println("Bio: ", p.bio)
}

type website struct {
 posts []post
}
func (w website) contents() {
    fmt.Println("Contents of Website\n")
    for _, v := range w.posts {
        v.details()
        fmt.Println()
    }
}

func main() {
    author1 := author{
        "Naveen",
        "Ramanathan",
        "Golang Enthusiast",
    }
    post1 := post{
        "Inheritance in Go",
        "Go supports composition instead of inheritance"
        author1,
    }
    post2 := post{
        "Struct instead of Classes in Go",
        "Go does not support classes but methods can be
        author1,
    }
    post3 := post{
        "Concurrency",
        "Go is a concurrent language and not a parallel
        author1,
    }
    w := website{
        posts: []post{post1, post2, post3},
    }
    w.contents()
}
```

Run in playground

In the main function above, we have created an author `author1` and three posts `post1`, `post2` and `post3`. Finally we have created the website `w` in line no. 62 by embedding these 3 posts and displayed the contents in the next line.

This program will output,

```
Contents of Website

Title:  Inheritance in Go
Content:  Go supports composition instead of inheritance
Author:  Naveen Ramanathan
Bio:  Golang Enthusiast

Title:  Struct instead of Classes in Go
Content:  Go does not support classes but methods can be
added to structs
Author:  Naveen Ramanathan
Bio:  Golang Enthusiast

Title:  Concurrency
Content:  Go is a concurrent language and not a parallel one
Author:  Naveen Ramanathan
Bio:  Golang Enthusiast
```

This brings us to the end of this tutorial. Have a great day. Please leave your feedback and comments.

Next tutorial - Polymorphism

Naveen Ramanathan

iOS developer at dietco.de and Golang enthusiast.

Share this post

About

For any queries/suggestions, please contact us at **naveen[at]golangbot[dot]com**

Follow Us

Newsletter

Join Our Newsletter

Signup for our newsletter and get the **Golang tools cheat sheet for free**.

Email*

Subscribe