RX-M / go Private

Branch: master go / modules / methods-and-interfaces / rx-m-go-lab-methods-and-interfaces.md Find file Copy path

ronaldpetty updated methods-and-interfaces lab, normalized file names largely afb9e77 on Sep 9

3 contributors

118 lines (71 sloc) 2.94 KB

RX-M Cloud Native Consulting

Go

Methods and Interfaces

Go does not have classes. However, you can define methods on types. A method is a function with a special receiver argument. The receiver appears in its own argument list between the func keyword and the method name.

1. Create a Moto struct type

Create a working directory for this lab:

```
user@ubuntu:~/go/src/lab-functions$ cd

user@ubuntu:~$
```

```
user@ubuntu:~$ mkdir -p ~/go/src/lab-methods-interfaces

user@ubuntu:~$
```

```
user@ubuntu:~$ cd ~/go/src/lab-methods-interfaces

user@ubuntu:~/go/src/lab-methods-interfaces$
```

Now create a program that declares a Moto type with the following fields:

- Make
- Model
- MPG
- Price

Create an instance of type Moto initialized with a literal and test your program.

2. Add a Dump() method

Create a method called "Dump" that displays all of the field values of the Moto variable it is invoked on.

Create two distinct instance of type Moto initialized with separate literals and test your program by using the Dump() method on each Moto instance. Verify the output you receive.

3. Add a Discount() method

Create a Discount() method for the Moto type. Each time Discount() is called on a Moto variable it should reduce the price of that Moto instance by 10%. Test your Discount() method and verify the discount is working using the Dump() method.

4. Binding receivers

Examine the following Go program:

```
user@ubuntu:~/go/src/lab-methods-interfaces$ vi interface.go
user@ubuntu:~/go/src/lab-methods-interfaces$ go run interface.go

package main

import "fmt"

func x(a []func()) {
        for _, f := range a {
                f()
        }
}

func main() {
        var a [2]func()
        a[0] = func() { fmt.Println("One") }
        a[1] = func() { fmt.Println("Two") }
        x(a[:])
}
user@ubuntu:~/go/src/lab-methods-interfaces$
```

What does this program do?

Add the `x()` function to your Moto program (without modification to x()).

Now create an array of Dump() calls such that each Dump() call is invoked on a different instance of Moto (make sure you have at least two Moto instances).

> Hint: You will need to bind the Dump() method receiver to each of the Moto instances in turn as discussed in the section on "Binding" methods in class.

Pass the array to `x()` and run your program. The `x()` function should call each of the functions in the slice it receives, printing out the fields of each of your Motos.

5. Challenge: Dump Interface

Create a Dump Interface that has one method, Dump(), which matches the signature of the Dump method defined on Moto above. Change the x() function to accept a slice of Dump interfaces. Now invoke x with an array of 2 or more Motos. The updated x() function should invoke the Dump() method on each of the Dump interface compatible objects, displaying their fields.

Congratulations you have completed the lab!!