Golang Programming

Methods. Composing Types by struct Embedding

Where to Find The Code and Materials?

https://github.com/iproduct/coursego

Methods

```
type Vertex struct {
	X, Y float64
}

func (v Vertex) Abs() float64 {
	return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func main() {
	v := Vertex{3, 4}
	fmt.Println(v.Abs())
	abs := Vertex.Abs
	fmt.Println(abs(v))
}
```

Methods on Non Structs

```
type MyFloat float64

func (f MyFloat) Abs() float64 {
	if f < 0 {
		return float64(-f)
	}
	return float64(f)
}

func main() {
	f := MyFloat(-math.Sqrt2)
	fmt.Println(f.Abs())
}
```

Methods on Non Structs

```go
type Role int

const (
    User Role = 1 << iota
    Manager
    Admin
    RoleMask = (1 << (iota)) - 1
)

func (r Role) String() string {
    switch r {
    case User:
        return "User"
    case Manager:
        return "Manager"
    case Admin:
        return "Admin"
    default:
        return "Invalid role"
    }
}
```

```go
// Status type
type Status int

// User statuses enum
const (
    Registered Status = iota
    Active
    Disabled
)

// Returns string representation of the Role
func (r Status) String() string {
    switch r {
    case Registered:
        return "Registered"
    case Active:
        return "Active"
    case Disabled:
        return "Disabled"
    default:
        return "Invalid status"
    }
}
```

Methods on Non Structs

```
type Vertex struct {
        X, Y float64
}

func (v Vertex) Abs() float64 {
        return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func (v *Vertex) Scale(f float64) {
        v.X = v.X * f
        v.Y = v.Y * f
}

func main() {
        v := Vertex{3, 4}
        v.Scale(10)
        fmt.Println(v.Abs())
}
```

Methods Are Just Functions

```
type Vertex struct {
        X, Y float64
}

func Abs(v Vertex) float64 {
        return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func Scale(v *Vertex, f float64) {
        v.X = v.X * f
        v.Y = v.Y * f
}

func main() {
        v := Vertex{3, 4}
        Scale(&v, 10)
        fmt.Println(Abs(v))
}
```

Methods and Pointer Indirection

```
type Vertex struct {
        X, Y float64
}
func (v *Vertex) Scale(f float64) {
        v.X = v.X * f
        v.Y = v.Y * f
}
func ScaleFunc(v *Vertex, f float64) {
        v.X = v.X * f
        v.Y = v.Y * f
}
func (v Vertex) Abs() float64 {
        return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
func AbsFunc(v Vertex) float64 {
        return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

```
func main() {
        // Pointer receiver methods
        v := Vertex{3, 4}
        v.Scale(2)
        ScaleFunc(&v, 5)

        p := &Vertex{4, 3}
        p.Scale(5)
        ScaleFunc(p, 2)
        fmt.Println(v, p)

        // Value receiver methods
        fmt.Println(v.Abs())
        fmt.Println(AbsFunc(v))

        fmt.Println(p.Abs())
        fmt.Println(AbsFunc(*p))
}
```

Methods: Value and Pointer Receivers

```
type ByteSlice []byte

func (slice ByteSlice) Append(data []byte) []byte {
    return append([]byte(slice), data...)
}

func (slice *ByteSlice) AppendPointer(data []byte) {
    *slice = append([]byte(*slice), data...)
}

func (slice *ByteSlice) Write(data []byte) (n int, err error) {
    *slice = append([]byte(*slice), data...)
    return len(data), nil
}

func main() {
       var b ByteSlice
       fmt.Fprintf(&b, "This hour has %d days\n", 7)
       fmt.Printf("%v", b)
}
```

Choosing Value or Pointer Receiver

There are two reasons to use a pointer receiver:

- The first is so that the method can modify the value that its receiver points to.
- The second is to avoid copying the value on each method call. This can be more efficient if the receiver is a large struct, for example.
- In general, all methods on a given type should have either value or pointer receivers, but not a mixture of both.
- More about selectors and method expressions: https://golang.org/ref/spec#Selectors

Method Receivers and Interfaces

```
type Abser interface {
	Abs() float64
}

func main() {
	var a Abser
	f := MyFloat(-math.Sqrt2)
	v := Vertex{3, 4}

	a = f  // MyFloat implements Abser
	fmt.Println(a.Abs())
	a = &v // *Vertex implements Abser

	// Vertex do not implement Abser
	//a = v

	fmt.Println(a.Abs())
}
```

```
type MyFloat float64

func (f MyFloat) Abs() float64 {
    if f < 0 {
        return float64(-f)
    }
    return float64(f)
}

type Vertex struct {
    X, Y float64
}

func (v *Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

Methods with Nil Receivers

```
// Path represents a sequence of Vertices. A nil Path represents empty sequence.
type Path []Vertex
func (p *Path) Distance() (dist float64) {
        dist = 0
        if *p == nil || len(*p) == 0 {
                return 0
        }
        v1 := (*p)[0]
        var v2 Vertex
        for i := 1; i < len(*p); i++ {
                v2 = (*p)[i]
                dist += v1.Distance(v2)
                v1 = v2
        }
        return
}
func main() {
        var path Path
        path = Path{{1, 1}, {4, 5}, {4, 1}, {1, 1}}
        fmt.Println("Perimeter = ", path.Distance())
}
```

Composing structs by Type Embedding

```
type ColorVertex struct {
    Vertex
    Color color.RGBA
}

func main() {
    green := color.RGBA{0, 255, 0, 255}
    yellow := color.RGBA{255, 255, 0, 255}
    cv1 := ColorVertex{Vertex{2, 3}, green}
    cv2 := ColorVertex{Vertex{6, 6}, yellow}
    fmt.Println(cv1.Distance(cv2.Vertex)) // 5
    cv1.Scale(4)
    cv2.Scale(4)
    fmt.Println(cv1.Distance(cv2.Vertex)) // 20
    // cv1.Distance(cv2) // no cv1.Distance(ColorVertex)

}
```

```
type Vertex struct {
    X, Y float64
}

func (v Vertex) Distance(o Vertex) float64 {
    return math.Hypot(o.X-v.X, o.Y-v.Y)
}

func (v *Vertex) Scale(f float64) {
        v.X = v.X * f
        v.Y = v.Y * f
}
```

Composing structs by Pointer Type Embedding

```
type ColorVertexP struct {
        *Vertex
        Color color.RGBA
}

cvp1 := ColorVertexP{&Vertex{2, 7}, green}
cvp2 := ColorVertexP{&Vertex{5, 3}, yellow}
fmt.Println(cvp1.Distance(*cvp2.Vertex)) // "5"
cvp1.Vertex = cvp2.Vertex
cvp1.Scale(3)
fmt.Println(*cvp1.Vertex, *cvp2.Vertex) // {15 9} {15 9}
```

Method Values and Expressions

```
a := Vertex{2, 7}
b := Vertex{5, 3}

distance := Vertex.Distance // method expression
fmt.Println(distance(a, b)) // 5
fmt.Printf("%T\n", distance) // func(main.Vertex, main.Vertex) float64

scale := (*Vertex).Scale // method expression
scale(&a, 2)
fmt.Println(a)              // {4 14}
fmt.Printf("%T\n", scale) // func(*main.Vertex, float64)

scaleB := (&b).Scale  // method value
fmt.Printf("%T\n", scaleB) // func(float64)
scaleB(2)
fmt.Printf("Sacling b with factor 2: b now is %f\n", b) //{10 6}

distanceFromA := a.Distance // method value
fmt.Printf("%T\n", distanceFromA) // func(*Vertex, float64)
fmt.Printf("Distance from A of B is %f\n", distanceFromA(b)) //10
```

Encapsulation

- State encapsulation:

```
type IntSet struct {
     words []uint64
}
```

- No state encapsulation:

```
type IntSet []uint64
```

Examples

- IntSet
- PriorityQueue
- HttpServer

Homework 2 (More Methods of the Path)

Extend the Path type from slide 12 by adding:

- two more fields: strokeColor and fillColor
- methods: Scale(f float64), Translate(vector Vertex), Rotate(angle float64)
- methods: Add(v Vertice, position int), Remove(v Vertice, position int)
- method: Draw() image.Image – implement the Image interface and return it

```
package image
type Image interface {
    ColorModel() color.Model // should return color.RGBAModel.
    Bounds() Rectangle // should return a image.Rectangle - e.g. image.Rect(0, 0, w, h)
    At(x, y int) color.Color // should return color.RGBAModel– e.g. color.RGBA{v,v,255,255}
}
```

- method: EncodePNG(file *os.File) error - using: png.Encode(f File, i Image)
- function main(), that should demonstrate the use of all above methods. It should create new file in the directory of the project and encode the resulting Image (with drawn Path) into it.

Recommended Literature

- The Go Documentation - https://golang.org/doc/
- The Go Bible: Effective Go - https://golang.org/doc/effective_go.html
- David Chisnall, *The Go Programming Language Phrasebook*, Addison Wesley, 2012
- Alan A. A. Donovan, Brian W. Kernighan, *The Go Programming Language*, Addison Wesley, 2016
- Nathan Youngman, Roger Peppé, *Get Programming with Go*, Manning, 2018
- Naren Yellavula, *Building RESTful Web Services with Go*, Packt, 2017

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD