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())
}
```

Exercise 1: More Methods of the Path

Extend the Path type from previous slide with more methods:

- Scale(factor float64)
- Translate(vector Vertice)
- Rotate(angle float64)
- Draw() Image
- Add(vertice Vertice, position int)
- Remove(vertice Vertice, position int)

```
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}
}
```

- Implement the following function to encode the Image as PNG file in main()

```
func EncodeImagePNG(m image.Image, file *os.File) error {
        return png.Encode(file, m)
}
```

Composing Types by struct Embedding

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

func main() {
        //!+main
        red := color.RGBA{255, 0, 0, 255}
        blue := color.RGBA{0, 0, 255, 255}
        var p = ColoredVertex{Vertex{1, 1}, red}
        var q = ColoredVertex{Vertex{5, 4}, blue}
        fmt.Println(p.Distance(q.Vertex)) // 5
        p.ScaleBy(2)
        q.ScaleBy(2)
        fmt.Println(p.Distance(q.Vertex)) // 10
        // p.Distance(q) // compile error: cannot use q (ColoredVertex) as Vertex
}
```

```
func (p Vertex) Distance(q Vertex) float64 {
        dX := q.X - p.X
        dY := q.Y - p.Y
        return math.Sqrt(dX*dX + dY*dY)
}

func (p *Vertex) ScaleBy(factor float64) {
        p.X *= factor
        p.Y *= factor
}
```

Composing Types by struct Embedding

```
red := color.RGBA{255, 0, 0, 255}
blue := color.RGBA{0, 0, 255, 255}

//!+indirect
type ColoredVertex struct {
        *Vertex
        Color color.RGBA
}

p := ColoredVertex{&Vertex{1, 1}, red}
q := ColoredVertex{&Vertex{5, 4}, blue}
fmt.Println(p.Distance(*q.Vertex)) // "5"
q.Vertex = p.Vertex                // p and q now share the same Vertex
p.ScaleBy(2)
fmt.Println(*p.Vertex, *q.Vertex) // "{2 2} {2 2}"
```

Method Values and Expressions

```
p := Vertex{1, 2}
q := Vertex{4, 6}

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

scale := (*Vertex).ScaleBy
scale(&p, 2)
fmt.Println(p)            // "{2 4}"
fmt.Printf("%T\n", scale) // "func(*Vertex, float64)"
```

Encapsulation

- State encapsulation:

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

- No state encapsulation:

```
type IntSet []uint64
```

Examples

- IntSet
- PriorityQueue
- HttpServer

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