Golang Programming

Interfaces. Http handlers. Routing with ServeMux. Sorting. Templates. JSON

Where to Find The Code and Materials?

https://github.com/iproduct/coursego

Interfaces as Contracts

- Interfaces in Go are *abstractions (generalizations)* of the concrete types' behaviors. They specify the *contract* that *concrete types* implement.
- While *type embedding* effectively achieves *non-virtual inheritance*, *interfaces* in Go allow for *virtual inheritance*.
- Structurally typed *interfaces* provide *runtime polymorphism* through *dynamic dispatch*.
- Go *interfaces are satisfied implicitly* – *duck typing*. *Substitutable* impl.
- An *interface type* specifies a *method set* called its *interface*.
- A *variable of interface type* can store a *value* of any *type* with a method set that is any *superset of the interface*. Such type *implements* the interface.
- The value of an *uninitialized variable* of interface type is *nil*.

SOLID Design Principles of OOP

- Single responsibility principle - a class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.
- Open–closed principle - software entities should be open for extension, but closed for modification.
- Liskov substitution principle - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- Interface segregation principle - Many client-specific interfaces are better than one general-purpose interface.
- Dependency inversion principle - depend upon abstractions, not concretions.

Example 1: Interface Writer and fmt.Fprintf()

```
package io
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

```
type ByteSlice []byte  // implements Writer

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, "Interfaces in Go are %s\n", "useful")
     fmt.Printf("%v", b) //[73 110 116 101 114 102 97 99 101 115 32 105 110 32 71
                         // 111 32 97 114 101 32 117 115 101 102 117 108 10]
}
```

Example 2: Interface Writer and fmt.Fprintf()

```
package io
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

```
type ByteCount int  // implements Writer

func (c *ByteCount) Write(p []byte) (int, error) {
    *c += ByteCounter(len(p))
    return len(p), nil
}

func main() {
    var c ByteCount
    fmt.Fprintf(&c, "Interfaces in Go are %s\n", "useful")
    fmt.Printf("%v\n", c) //Result: 28
}
```

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)
}
```

Example 3: Interface File - Multiple Implementations

```
type File interface {
    Read([]byte) (int, error)
    Write([]byte) (int, error)
    Close() error
}
type MyFile struct { /*...*/ } // Implementation of File interface
func (p MyFile) Read(b []byte) (n int, err error)   { return /*...*/ }
func (p MyFile) Write(b []byte) (n int, err error)  { return /*...*/ }
func (p MyFile) Close() error                       { return /*...*/ }

type OtherFile struct { /*...*/ } // Another implementation of File interface
func (o *OtherFile) Read(b []byte) (n int, err error)   { return /*...*/ }
func (o *OtherFile) Write(b []byte) (n int, err error)  { return /*...*/ }
func (o *OtherFile) Close() error                       { return /*...*/ }

var f File
f = MyFile{} //or &MyFile{}

fmt.Printf("%T", f)
f = &OtherFile{}
fmt.Printf("%T", f)
```

Interface Types

```
type Reader interface {
        Read(p []byte) (n int, err error)
}

type Closer interface {
        Close() error
}

type ReadWriter interface {
        Reader
        Writer
}

type File interface {
        Reader
        Writer
        Closer
}
```

Interface Types – Errors: Selector Duplication

```
type ReadWriter interface {
        Read(b Buffer) bool
        Write(b Buffer) bool
}
type Locker interface {
        Lock()
        Unlock()
}
type File interface {
        ReadWriter  // same as adding the methods of ReadWriter
        Locker      // same as adding the methods of Locker
        Close()
}
type LockedFile interface {
        Locker
        File        // illegal: Lock, Unlock not unique
        Lock()      // illegal: Lock not unique
}
```

Interface Types – Errors: Recursive Embeding

```
// illegal: Bad cannot embed itself
type Bad interface {
        Bad
}

// illegal: Bad1 cannot embed itself using Bad2
type Bad1 interface {
        Bad2
}
type Bad2 interface {
        Bad1
}
```

Interface Conformance and Duck Typing

```
// Declaring io.Writer interface implementation by *bytes.Buffer
var _ io.Writer = (*bytes.Buffer)(nil)

// Interface satisfaction
var w Writer
w = os.Stdout           // *os.File has method Write()
w = new(bytes.Buffer) // *bytes.Buffer has method Write()
w = time.Hour           // compile time error: no method Write()
w.Write([]byte("abcd"))

var f File
f = os.Stdout
w = f
f = w  // compile time error: w has no methods Read() and Close()
```

Maps of Interfaces as Keys and Values

- The comparison operators == and != must be fully defined for operands of the key type.
- If the key type is an interface type, these comparison operators must be defined for the dynamic key values; failure will cause a run-time panic.
- *Examples:*

```
var m1 map[*Writer]struct{ x, y float64 }
var m2 map[string]interface{}
```

Embedding Interfaces in Structs

```
type Bouncer interface {
        Bounce()
}
type Football struct {
        Bouncer
}

type Ball struct {
        Radius   int
        Material string
}
func (b Ball) Bounce() {
        fmt.Printf("Ball bouncing ... %v\n", b)
}

func main() {
        // Interfaces embedding
        fb := Football{Ball{Radius: 5, Material: "leather"}}
        fb.Bounce()
}
```

Interface Values

- Under the hood, interface values can be thought of as a tuple of a *value* and a *concrete type*: ***(value, type)***

- An interface *value* holds a value of a specific underlying *concrete type*.

- *Calling a method* on an interface value executes the *method of the same name on its underlying type*.

Interface Values Example

```go
package main

import (
    "fmt"
    "math"
)

type I interface {
    M()
}

type T struct {
    S string
}

func (t *T) M() {
    fmt.Println(t.S)
}
```

```go
type F float64

func (f F) M() {
    fmt.Println(f)
}

func main() {
    var i I

    i = &T{"Hello"}
    describe(i) //(&{Hello}, *main.T)
    i.M() //Hello

    i = F(math.Pi)
    describe(i) //(3.141592653589793, main.F)
    i.M() //3.141592653589793
}

func describe(i I) {
    fmt.Printf("(%v, %T)\n", i, i)
}
```

Interface Values with nil Underlying Values

```go
package main

import "fmt"

type I interface {
    M()
}

type T struct {
    S string
}

func (t *T) M() {
    if t == nil {
        fmt.Println("<nil>")
        return
    }
    fmt.Println(t.S)
}
```

```go
func main() {
    var i I

    var t *T
    i = t
    describe(i) //(<nil>, *main.T)
    i.M()       //<nil>

    i = &T{"hello"}
    describe(i) //(&{hello}, *main.T)
    i.M()       //hello
}

func describe(i I) {
    fmt.Printf("(%v, %T)\n", i, i)
}
```

Nil Interface Values

```
package main

import "fmt"

type I interface {
	M()
}

func main() {
	var i I
	describe(i)
	i.M()  // (<nil>, <nil>)
		// panic: runtime error: invalid memory address or nil pointer dereference
		// [signal 0xc0000005 code=0x0 addr=0x0 pc=0x49c106]
}

func describe(i I) {
	fmt.Printf("(%v, %T)\n", i, i)
}
```

The Empty Interface

- The interface type that specifies zero methods is known as the empty interface: *interface{}*
- An empty interface may hold values of any type. (Every type implements at least zero methods.)
- Empty interfaces are used by code that handles values of unknown type. For example, fmt.Print takes any number of arguments of type interface{}.

```
func main() {
	var i interface{}
	describe(i) // (<nil>, <nil>)

	i = 42
	describe(i) // (42, int)

	i = "hello"
	describe(i) // (hello, string)
}
```

Type Assertions

- A type assertion provides access to an interface value's underlying concrete value.

  t := i.(T)

- This statement asserts that the interface value i holds the concrete type T and assigns the underlying T value to the variable t.
- If i does not hold a T, the statement will trigger a panic.
- To test whether an interface value holds a specific type, a type assertion can return two values: the underlying value and a boolean value that reports whether the assertion succeeded.

  t, ok := i.(T)

Type Assertion Examples

```
import "fmt"

func main() {
	var i interface{} = "hello"

	s := i.(string)
	fmt.Println(s) // hello

	s, ok := i.(string)
	fmt.Println(s, ok) // hello true

	f, ok := i.(float64)
	fmt.Println(f, ok) // 0 false

	f = i.(float64) // panic
	fmt.Println(f)  // panic: interface conversion: interface {} is string, not float64
}
```

Type Switches

- A type switch is a construct that permits several type assertions in series.
- A type switch is like a regular switch statement, but the cases in a type switch specify types (not values), and those values are compared against the type of the value held by the given interface value.

```
switch v := i.(type) {      // same syntax as a type assertion i.(T), but type T is
                            // replaced with the keyword type.
case T:
    // here v has type T
case S:
    // here v has type S
default:
    // no match; here v has the same type as i
}
```

Interface Conversion using Type Switches/Assertions

```
type Stringer interface {
        String() string
}

var value interface{} // Value provided by caller.

func String() {
        switch str := value.(type) {          // type switch
        case string:
                return str
        case Stringer:
                return str.String()
        }

        if str, ok := value.(string); ok {    // The same as above with type assertion
                return str
        } else if str, ok := value.(Stringer); ok {
                return str.String()
        }
}
```

Sorting by implementing sort.Interface (1)

```
import ("fmt"; "sort")

type Sequence []int

// Methods required by sort.Interface.
func (s Sequence) Len() int {
        return len(s)
}
func (s Sequence) Less(i, j int) bool {
        return s[i] < s[j]
}
func (s Sequence) Swap(i, j int) {
        s[i], s[j] = s[j], s[i]
}

// Copy returns a copy of the Sequence.
func (s Sequence) Copy() Sequence {
        copy := make(Sequence, 0, len(s))
        return append(copy, s...)
}
```

Sorting by implementing sort.Interface (2)

```
// Method for printing - sorts the elements before printing.
func (s Sequence) String() string {
        s = s.Copy()
        sort.Sort(s)
        return fmt.Sprint([]int(s))
}

func main() {
        s := Sequence{54, 12, 3, 17, 29, 77, 22, 34, 2, 3, 91, 22, 5}
        fmt.Println(s) //[2 3 3 5 12 17 22 22 29 34 54 77 91]
}
```

Sorting using sort.IntSlice().Sort() - Shorter

```
type Sequence []int

// Copy returns a copy of the Sequence.
func (s Sequence) Copy() Sequence {
	copy := make(Sequence, 0, len(s))
	return append(copy, s...)
}

// String method sorts elements and returns them as string
func (s Sequence) String() string {
	s = s.Copy()
	sort.IntSlice(s).Sort()
	return fmt.Sprint([]int(s))
}
```

Example: Sort solar planets by multiple criteria

https://github.com/iproduct/coursego/tree/master/sorting/sort-keys

Interface http.Handler

```
type Handler interface {
        ServeHTTP(ResponseWriter, *Request)
}
```

```
type HandlerFunc func(ResponseWriter, *Request)
func (f HandlerFunc) ServeHTTP(
                w ResponseWriter, req *Request) {
        f(w, req)
}
```

```
func hello(w http.ResponseWriter, req *http.Request) {
        fmt.Fprintf(w, "hello\n")
}
func headers(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "%s %s %s\n", r.Method, r.URL, r.Proto)
        fmt.Fprintf(w, "Host = %q\nRemoteAddr = %q\n\n ", r.Host, r.RemoteAddr)
        for name, headers := range r.Header {
                for _, h := range headers {
                        fmt.Fprintf(w, "%v: %v\n", name, h)
                }
        }
}
func main() {
        http.HandleFunc("/hello", hello)
        http.HandleFunc("/headers", headers)
        http.ListenAndServe(":8080", nil)
}
```

Example: HTTP QR Link Generator

HTML Template [https://golang.org/pkg/html/template/]

```
const templateStr = `
<html>
<head>
<title>QR Link Generator</title>
</head>
<body>
{{if .}}
<img src="http://chart.apis.google.com/chart?chs=300x300&cht=qr&choe=UTF-8&chl={{.}}" />
<br>
{{.}}
<br>
<br>
{{end}}
<form action="/" name=f method="GET"><input maxLength=1024 size=70
name=s value="" title="Text to QR Encode"><input type=submit
value="Show QR" name=qr>
</form>
</body>
</html>
`
```

Text Templates [https://golang.org/pkg/text/template/]

```
import ("html/template"; "log"; "os")
const textTempl =
`{{len .}} issues:
{{range .}}----------------------------------------
ID: {{.ID}}
Title: {{.Title | printf "%.64s"}}
{{end}}`

func main() {
        tmpl := template.New("report")
        tmpl, err := tmpl.Parse(textTempl)
        if err != nil {
                log.Fatal("Error Parsing template: ", err)
                return
        }
        err1 := tmpl.Execute(os.Stdout, goBooks)
        if err1 != nil {
                log.Fatal("Error executing template: ", err1)
        }
}
```

```
3 books:

----------------------------------------

ID: fmd-DwAAQBAJ

Title: Hands-On Software Architecture with Golang

----------------------------------------

ID: o86PDwAAQBAJ

Title: Learn Data Structures with Golang

----------------------------------------

ID: xfPEDwAAQBAJ

Title: From Ruby to Golang
```

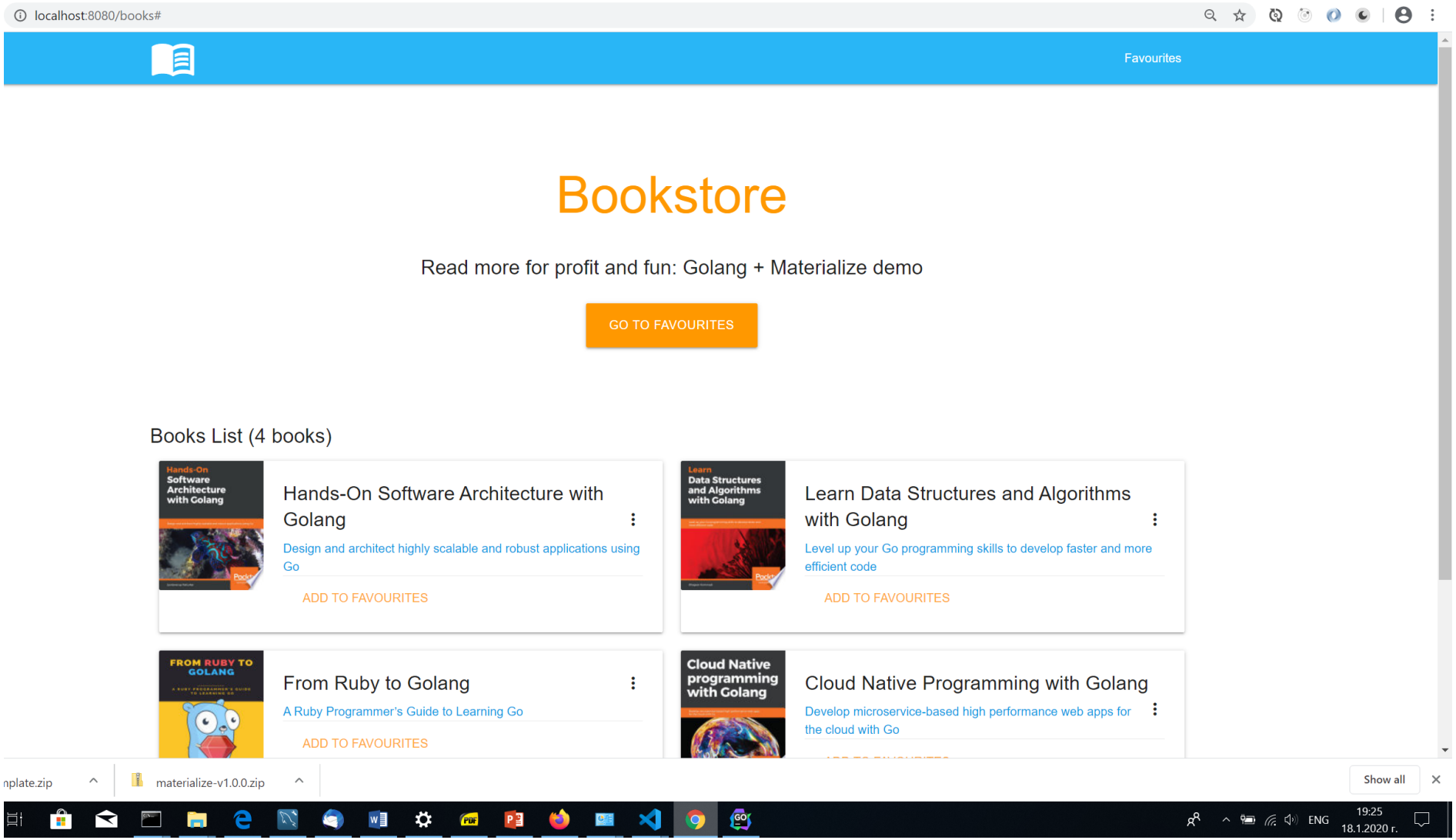
JSON Marshalling and Unmarshalling

```
// Structs --> JSON
data, err := json.Marshal(goBooks)
if err != nil {
        log.Fatalf("JSON marshaling failed: %s", err)
}
fmt.Printf("%s\n", data)

// Prettier formatting
data, err = json.MarshalIndent(goBooks, "", "     ")
if err != nil {
        log.Fatalf("JSON marshaling failed: %s", err)
}
fmt.Printf("%s\n", data)

// JSON -> structs
var books []Book
if err := json.Unmarshal(data, &books); err != nil {
        log.Fatalf("JSON unmarshaling failed: %s", err)
}
fmt.Println("AFTER UNMARSHAL:\n", books)
```

Example: Bookstore Web App



https://github.com/iproduct/coursego/tree/master/http/bookstore

Routing with http.ServeMux (1)

```
type database map[string]Book


func (db database) list(w http.ResponseWriter, req *http.Request) {
        for _, book := range db {
                fmt.Fprintf(w, "%s: %s - $%6.2f\n", book.ID, book.Title, book.RetailPrice)
        }
}


func (db database) price(w http.ResponseWriter, req *http.Request) {
        id := req.URL.Query().Get("id")
        if book, ok := db[id]; ok {
                fmt.Fprintf(w, "$%6.2f\n", book.RetailPrice)
        } else {
                w.WriteHeader(http.StatusNotFound) // 404
                fmt.Fprintf(w, "no book with ID: %q\n", id)
        }
}
```

Routing with http.ServeMux (2)

```
var db database = make(map[string]Book, 10)

func init() {
	for _, book := range goBooks {
		db[book.ID] = book
	}
}

func main() {
	mux := http.NewServeMux()
	mux.HandleFunc("/list", db.list)
	mux.HandleFunc("/price", db.price)
	log.Fatal(http.ListenAndServe("localhost:8080", mux))
}
```

Exercise: Implement additional Bookstore handlers

- Extend the Bookstore web application (from previous slide) with following functionality:
  - In the /books main page, present an HTML button <Add to Favorites> next to each book, that submits the selected book to the server, and server ads it to Favorites collection, and <Details> button that presents the book details in a corresponding page (*/book?id=<bookID>*).
  - */favorites* web page that presents the books added to Favorites, and shows a <Remove> button next to each favorite book, that removes it from Favorites, as well as a <Details> button that presents the book details in a corresponding page, and <All Books> button redirecting to /books.
  - */book?id=<bookID>* page that presents full details about the book together with <Add to Favorites>/<Remove>, and < All Books > buttons with the same functionally as explained above.

Generality Using Interfaces

- If a type exists only to implement an interface and will never have exported methods beyond that interface, there is no need to export the type itself.
- Exporting just the interface makes it clear the value has no interesting behavior beyond what is described in the interface. It also avoids the need to repeat the documentation on every instance of a common method:

```
type Block interface {
        BlockSize() int
        Encrypt(dst, src []byte)
        Decrypt(dst, src []byte)
}
type Stream interface {
        XORKeyStream(dst, src []byte)
}
// NewCTR returns a Stream that encrypts/decrypts using the given Block in
// counter mode. The length of iv must be the same as the Block's block size.
func NewCTR(block Block, iv []byte) Stream
```

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