Programming in Go

Matt Holiday
Christmas 2020

Homework

Interfaces in HTTP

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

type HandlerFunc func(ResponseWriter, *Request)

func (f HandlerFunc) ServeHTTP(w ResponseWriter, r *Request) {
    f(w, r)
}

// handler matches type HandlerFunc and so interface Handler
// so the HTTP framework can call ServeHTTP on it

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, world! from %s\n", r.URL.Path[1:])
}
```

Homework #4

Exercise 7.11 from *GOPL:* web front-end for a database

Add additional handlers [to the database example in §7.7, which is program `gopl.io/ch7/http4`] so that clients can create, read, update, and delete database entries. For example, a request of the form

```
/update?item=socks&price=6
```

will update the price of an item in the inventory and report an error if the item does not exist or if the price is invalid.

(We will *ignore* the race conditions for the purpose of this exercise.)

See my solution at: https://github.com/matt4biz/go-class-exer-7.11

Homework #4

```
package main

import (
    "fmt"
    "log"
    "net/http"
    "strconv"
)

type dollars float32  // DO NOT do this in real life!

func (d dollars) String() string {
    return fmt.Sprintf("$%.2f", d)
}

type database map[string]dollars
```

Homework #4

```
func (db database) list(w http.ResponseWriter, req *http.Request) {
    for item, price := range db {
        fmt.Fprintf(w, "%s: %s\n", item, price)
    }
}


func (db database) add(w http.ResponseWriter, req *http.Request) {
    item := req.URL.Query().Get("item")
    price := req.URL.Query().Get("price")

    if _, ok := db[item]; ok {
        msg := fmt.Sprintf("duplicate item: %q\n", item)

        http.Error(w, msg, http.StatusBadRequest) // 400
        return
    }
```

Homework #4

```
    if f64, err := strconv.ParseFloat(price, 32); err != nil {
        msg := fmt.Sprintf("invalid price: %q\n", price)

        http.Error(w, msg, http.StatusBadRequest) // 400
    } else {
        db[item] = dollars(f64)

        fmt.Fprintf(w, "added %s with price %s\n", item, dollars(f64))
    }
}



func (db database) update(w http.ResponseWriter, req *http.Request) {
    item := req.URL.Query().Get("item")
    price := req.URL.Query().Get("price")
```

Homework #4

```
    if _, ok := db[item]; !ok {
        fmt.Sprintf("no such item: %q\n", item)

        http.Error(w, msg, http.StatusNotFound) // 404
        return
    }

    if f64, err := strconv.ParseFloat(price, 32); err != nil {
        msg := fmt.Sprintf("invalid price: %q\n", price)

        http.Error(w, msg, http.StatusBadRequest) // 400
    } else {
        db[item] = dollars(f64)

        fmt.Fprintf(w, "new price %s for %s\n", dollars(f64), item)
    }
}
```

Homework #4

```
func (db database) fetch(w http.ResponseWriter, req *http.Request) {
    item := req.URL.Query().Get("item")

    if _, ok := db[item]; !ok {
        fmt.Sprintf("no such item: %q\n", item)

        http.Error(w, msg, http.StatusNotFound) // 404
        return
    }

    fmt.Fprintf(w, "item %s has price %s\n", item, db[item])
}
```

Homework #4

```
func (db database) drop(w http.ResponseWriter, req *http.Request) {
    item := req.URL.Query().Get("item")

    if _, ok := db[item]; !ok {
        fmt.Sprintf("no such item: %q\n", item)

        http.Error(w, msg, http.StatusNotFound) // 404
        return
    }

    delete(db, item)
    fmt.Fprintf(w, "dropped %s\n", item)
}
```

Homework #4

```
func main() {
    db := database{
        "shoes": 50,
        "socks": 5,
    }

    // all these handlers are method values closing over db
    // (each is cast to a HandlerFunc)

    http.HandleFunc("/list", db.list)     // func(ResponseWriter, *Request)
    http.HandleFunc("/create", db.add)
    http.HandleFunc("/update", db.update)
    http.HandleFunc("/delete", db.drop)
    http.HandleFunc("/read", db.fetch)

    log.Fatal(http.ListenAndServe("localhost:8000", nil))
}
```

```
$ curl localhost:8080/list
shoes: $50.00
socks: $5.00

$ curl localhost:8080/create?item=ties\&price=13
added ties with price $13.00

$ curl localhost:8080/list
shoes: $50.00
socks: $5.00
ties: $13.00

$ curl localhost:8080/read?item=ties
item ties has price $13.00

$ curl localhost:8080/update?item=r
no such item: "r"

$ curl localhost:8080/update?item=ties\&price=A
invalid price: "A"
```

```
$ curl localhost:8080/update?item=ties\&price=12
new price $12.00 for ties

$ curl localhost:8080/list
shoes: $50.00
socks: $5.00
ties: $12.00

$ curl localhost:8080/delete?item=ties
dropped ties

$ curl localhost:8080/list
shoes: $50.00
socks: $5.00
```