Go part 3, we’re goin’ there now

Admin

- Any questions?
  - Concerns?
  - read through chapter 7, in the go book
  - For next time listen to the cup-o-go Episode 48
  - https://cupogo.dev/
  - https://cupogo.dev/episodes/a-bunch-of-grape-things-are-happening
  - We can talk about it next time
    - In the mean time lets talk about the stuff in the news

Go Projects and Dependencies

- The original way that go did dependencies
  - go get <project git location>
  - Then the project would be downloaded, compiled and ready for inclusion in any of your go projects as a library.
  - Eg
    - go get -d -u gobot.io/x/gobot/…
- What could possibly go wrong?

Go Projects and Dependencies

- What could possibly go wrong?
  - Student list?

Go Projects and Dependencies

- What could possibly go wrong?
  - Student list?
  - My list: at least:
    - Since often pulling latest version of project, someone can slip malicious code into a project
    - Shared dependencies mean you can’t easily pin a project to a version of a dependency
      - This one uses v1 of the web framework that one uses v2
    - Shared dependencies makes it hard to tell exactly what dependencies there are for multiple large projects.

Current Approach: Go Modules

- Introduced incrementally in go 1.11 and 1.12, and made default in 1.13
- Go modules uses a go.mod file to determine just the dependencies for this project
  - Can pin a version.
- Goland will help keep these dependencies in sync.
- Old way not really easy any more

The Plan

- We want to get to interfaces,
  - But can’t really do that without understanding how Go does objects and methods
  - So a few slides on objects and methods

Methods

- In Object Oriented Programming
  - Methods are functions attached to objects
  - In python and java
    - Part of the class definition
  - In C++
    - Sort of – in the header file

Methods

- In Object Oriented Programming
  - Methods are functions attached to objects
  - In python and java
    - Part of the class definition
  - In C++
    - Sort of – in the header file
- In Go
  - Methods are called on types (usually structs)
  - But declared where ever (in same package)

A Struct to work with

- type Sprite struct {
- Pic BitMap
- Location Point
- }
- Let this be a simple 2D game struct
- Suppose it is in a game library and we are using it

Adding a method

- Methods can be added to a type anywhere
  - Look a lot like functions except you call them on an ‘object’
- Method:
  - func (<object called on>) <name>(<params>) <return type>{
- Example
  - func (spr *Sprite)move(dX, dY int){
  - //do stuff here
  - }

Methods: non-mutating

- Methods that don’t mutate their objects
  - const methods
  - Usually called on the object itself (objects also passed by value)
  - Eg:
  - func (spr Sprite) GetSize()(width int, height int){
  - //fill in here
  - }
  - character Sprite := LoadCharacter()
  - h, w := character.GetSize()

Methods Mutating

- If you want the method to mutate the object you call it on
  - Call it on a pointer to the object
  - Copy of pointer still points at same object.
  - From before:
    - `func (spr *Sprite)move(dX, dY int){`
    - `//do stuff here`
    - `}`
  - Call:
    - character Sprite := <get result here>
    - (&character).move(30, 20)

Syntactic sugar

- But ‘programmers are lazy’
  - And that was ugly
    - Sure there are ways to do it in multiple lines to make it slightly better but
  - But go helps out
  - **Iff** you are calling the method on an object held in a variable, then go will take the address of the variable
  - So
    - (&character).move(30, 20)
  - Becomes just
    - character.move(30,20)
  - And the pointer is passed
  - To
  - move automatically

Methods – finishing up

- You can call a method on nil
  - Calling on nil will not cause an error
  - Fixing the ‘million dollar mistake’?
- Still need to make sure your method doesn’t cause a panic if you do.
  - Eg:
    - Var badGuy Sprite //badGuy is nil
    - h, w, := badGuy.getSize()
    - Since nil is the zero value for sprites, h and w should be zero.

Any questions

- So methods in go
  - Kinda objective-c like
  - Neat: you can add a method to any struct in the package in any file in the package
  - Scary:
    - you can add a method to any struct in the package in any file in the package,
    - you better look through the whole folder
  - Keep your private data lower cased :-)

Interfaces

- Are contracts that types will provide some functionality
  - So far java-esque
  - Even the interface declaration looks right
  - From go by example:
    - type geometry interface {
    - area() float64
    - perim() float64
    - }
  -

Interfaces are implemented implicitly

- To implement an interface for a type T,
  - Just write all the methods that are called on T that are required by the interface
  - Continuing the example

```
type Rect struct {
    Width, Height float64
}
type circle struct {
    radius float64
}
```

Continuing the example (gobyexample.com)

- With these methods, both rect and circle implement the geometry interface
  - func (r rect) area() float64 {
  -     return r.width * r.height
  - }
  - func (r rect) perim() float64 {
  -     return 2*r.width + 2*r.height
  - }
  - func (c circle) area() float64 {
  -     return math.Pi * c.radius * c.radius
  - }
  - func (c circle) perim() float64 {
  -     return 2 * math.Pi * c.radius
  - }
  -

Interfaces can be variables

- var shape geometry = rect{width: 5, height 4}
- a:= shape.area()
- Real power
  - You make a function that takes an interface
    - Can pass anything that satisfies that interface.

Interface {}

- interface{} is empty interface
  - Matches any type that implements at least zero methods
  - What does this buy us?

Interface {}

- interface{} is empty interface
  - Matches any type that implements at least zero methods
  - What does this buy us?
    - It matches anything
    - Make this a param type and your function can handle any value of any type

Interface values

- Interface is actually a two item struct right?
  - Type (interface type)
  - Value (usually a pointer to something that implements the interface)

Dynamic dispatch

- Lets talk dynamic dispatch
  - Since we have some people with a non-cs background – what do I mean by “dynamic dispatch”
  - How does it differ from static dispatch (sometimes called early binding)?

Dynamic dispatch

- Lets talk dynamic dispatch
  - How does it work?
  - C++, when does it happen? When does it not?
  - How about Java?
  - (since python is interpreted not an issue here)

Dynamic dispatch

- Lets talk dynamic dispatch
  - How does it work?
  - C++, when does it happen? When does it not?
- In Go – interface methods use dynamic dispatch.

Sort.Interface

- How do you sort arbitrary collections
  - In python?
  - In java?
  - In C++
  - Anyone want to talk about something else?

Sort.Interface

- In Go:
  - Implement sort of Interface //below is official go code from sort.go
    - type SampleInterface interface {
    -
    - // Len is the number of elements in the collection.
    - Len() int
    -
    - // Less reports whether the element with
    - // index i should sort before the element with index j.
    - Less(i, j int) bool
    -
    - // Swap swaps the elements with indexes i and j.
    - Swap(i, j int)
    -
    - }

So how about that sort.Interface

- What do you think?

So how about that sort.Interface

- What do I think?
  - I like python better for starter stuff
    - Just defining one sort key function (the less function equivalent) is nicer to start
    - Can see that this works for any data structure
      - Nice, but not sure Go is ready for CS1, CS2 or Data Structures.

Type Assertions

- Type Assertions
  - Used to check/cast interface type to
    - Actual concrete type
    - Another interface type
  - Interface var x and Type T
  - x.(T) will assert the type and return x as a T or panic
    - What you just wanted to check?

Type Assertions

- Type Assertions
  - Used to check/cast interface type to
    - Actual concrete type
    - Another interface type
  - Interface var x and Type T
  - x.(T) will assert the type and return x as a T or panic
    - What you just wanted to check?
    - realVal, ok := x.(realType)
    -

Common look for type assertions

- If file, ok := w.(*os.File); ok{
  - //Use file here
  - }
- Recall that first statement does assignment, second statement (just checking Boolean value of ‘ok’) is used for the if statement.

Comparing interfaces

- What about comparing interfaces
  - Can be compared if
    - Both are nil
    - The underlying concrete type for both is same
    - The concrete objects are comparable and == returns true

co-routines

- If you are ‘of a certain age” you’ve heard of coroutines.
  - Functions that hold state between calls, used in non-preemptive multitasking

Goroutines

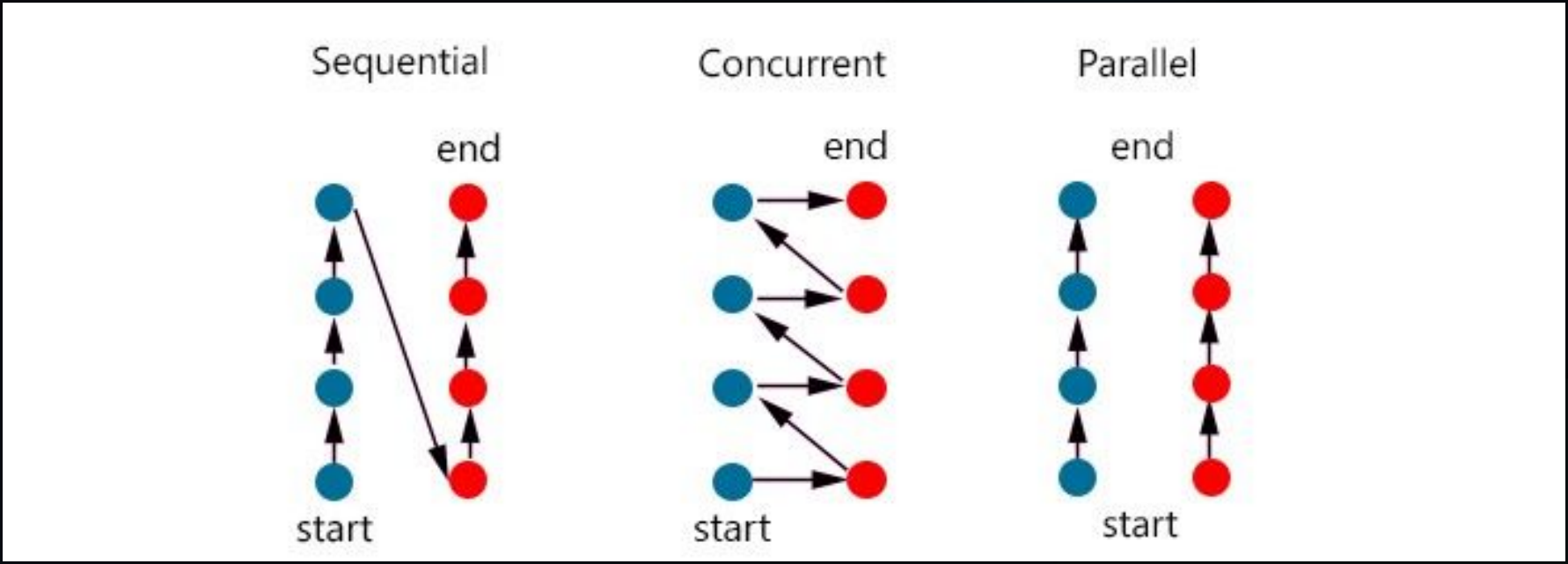
- So Go, being Punny (but not puny) has
- Goroutines
  - book: threads that work right
    - Go official: go routines are “Green threads” managed by go runtime rather than OS
  - Any function can be go routine
  - Keyword go before function call

Goroutines live forever

- Well sorta,
  - Will end when main.main ends
  - Or you can ask them to end

Sequential vs Concurrent

- Image credit:
  - https://medium.com/@k.wahome/concurrency-is-not-parallelism-a5451d1cde8d



Channels

- Used to communicate between go routines
- Have you done the producer/consumer lab yet in OS?
  - Lets have one of the CS background folks give us the nutshell summary since we probably have a couple of students who haven’t seen it yet
- Channels
  - communication between go routines
  - One goroutine writes and the other reads.

Make a Channel

- Make a channel with make
  - make(chan <type>)
  - Makes a channel of type <type> with no buffering
  - Sending data on unbuffered channel like this will block sender till receiver reads data
  -

Deadlock

- What is deadlock?

Deadlock

- What is deadlock?
  - All processes are blocked waiting for others
- Golang solves some deadlocks
  - If all goroutines associated with a particular channel are blocked
    - Go panics.

So how do you do it?

- data := make(chan int)
- In one go routine
  - data <- 3 //writes 3 to the data channel and sleeps till other go routine reads
- In The other Go routine
  - val := <- data //read a value from data channel and stuff it into val
  -

Producer consumer in Go

- func producer(link chan<- string) { //specify the link channel as send only
  - for _, m := range messages { //messages is a slice of strings, I committed it for space considerations.
  - link <- m
  - }
  - close(link)
- }
-
- func consumer(link <-chan string, done chan<- bool) {//specify link as receive only
  - for b := range link {
  - fmt.Println(b)
  - }
  - done <- true
- }
-
- func main() {
  - link := make(chan string)
  - done := make(chan bool)
  - go producer(link)
  - go consumer(link, done)
  - <-done //block till something gets sent to done
- }

You can find the original code without my comments (and additional examples) at

https://medium.com/better-programming/hands-on-go-concurrency-the-producer-consumer-pattern-c42aab4e3bd2

For next time

- Lets do the seminar approach in the
  - Lets talk about the podcast cup-o-go
  - What stood out to you?

JSON

- What do you all know about JSON?
  - Acknowledge pronunciation issues
  - Student ‘volunteers’?

For those not familiar

Data on the Internet

- Once upon a time
  - Data on the web (http/https) was all web pages intended to be viewed by people.
    - If we wanted to have a program read the data – need to ‘scrape’ the page.
- Back in 2000, Roy Fielding proposes REST framework (Ph.D thesis)
  - REpresentational State Transfer
  - Provide a way for web server to give data directly to program clients.
  - In last 5-10 years really used a lot

json

- json: JavaScript Object Notation
  - pronunciation note
  - json notation used by many RESTful interfaces to provide data
  - Says javascript but not really
  - Java vs javascript?
  - Java is to javascript as?

json

- json: JavaScript Object Notation
  - pronunciation note
  - json notation used by many RESTful interfaces to provide data
  - Says javascript but not really
  - Java vs javascript?
  - Java is to javascript as?
    - Car is to Carpet
- Official json spec
  - http://www.json.org/

Sample json

From https://openlibrary.org/dev/docs/api/lists


```
{
  "links": {
    "self": "/people/george08/lists.json",
    "next": "/people/george08/lists.json?limit=5&offset=5"
  },
  "size": 12,
  "entries": [
    {
      "url": "/people/george08/lists/OL13L",
      "full_url": "/people/george08/lists/OL13L/Various_Seeds_for_Testing",
      "name": "Various Seeds for Testing",
      "last_update": "2010-12-21T00:46:17.712513",
      "seed_count": 13,
      "edition_count": 13181
    },
```

```
{
      "url": "/people/george08/lists/OL97L",
      "full_url": "/people/george08/lists/OL97L/Time_Travel",
      "name": "Time Travel",
      "last_update": "2010-12-17T18:27:14.781336",
      "seed_count": 5,
      "edition_count": 838
    },
    ...
  ]}
```

From the web

- To get data from the web we use what protocol?

From the web

- To get data from the web we use what protocol?
- http(s) right?
- Go has you covered in the standard library
  - net/http package
  - See `http.Get(URLLoc)`
  - Returns a response struct and an error
    - https://golang.org/src/net/http/response.go
    - Mostly we care about StatusCode
    - And Body

Painless Json in go

- OK well ‘painless’ is optimistic, but far less painful than it was just 2 years ago.
- Use the encoding/json package.
  - After we use ioutil.ReadAll on the response.Body we have an array of bytes with the json in it.
  - Use json.Unmarshal to get a slice of structs
    - https://gobyexample.com/json
    - https://golang.org/pkg/encoding/json/#Unmarshal
    -

Lets try it

- If there is time lets try it on this free University API
- http://universities.hipolabs.com/search?name=young
- Example from a couple years ago
  - https://github.com/jsantore/APIgo2022
- Lets go through it and make sure you are comfortable with it

In class Exercise

- If there is time (and there should be)
  - Lets ask the user on the command line for the search term rather than hard coding ‘young”
  - Second, lets add a method to the UniversityData struct to print a University Data nicely and call it in line 29

Go:embed

- Amazing new feature added a few years ago.
  - You can embed files directly into the go program (the final executable) so all you have to give someone is a single executable file
  - Text files, images etc
  - Makes executable bigger, but no missing files possible

Go:embed II

- Usage
- You need the go:embed directive in a comment immediately over a global variable, which will hold the embedded asset
- Eg:

```
//go:embed assets/*
var EmbeddedAssets embed.FS
```

- This will take everything in the assets subfolder of the project and treat it as a file system

Go:Embed III

- Example loading image from embedded file system – even easier for a text file

```
func loadPNGImageFromEmbedded(name string) *ebiten.Image {
    pictNames, err := EmbeddedAssets.ReadDir("assets")
    if err != nil {
    log.Fatal("failed to read embedded dir ", pictNames, " ", err)
    }
    embeddedFile, err := EmbeddedAssets.Open("assets/" + name)
    if err != nil {
    log.Fatal("failed to load embedded image ", embeddedFile, err)
    }
    rawImage, err := png.Decode(embeddedFile)
    if err != nil {
    log.Fatal("failed to load embedded image ", name, err)
    }
    gameImage := ebiten.NewImageFromImage(rawImage)
    return gameImage
}
```