# Practical Go

@nikolay.dubina
2021-09-03

# Who am I?

Hello 👋, I am github.com/nikolaydubina

| | | | |
|---|---|---|---|
| ● Foodpanda | BE | SG | 2020~ |
| ● Founding Societies | BE | SG | 2020 |
| ● Balancehero | DE/ML/DS | KR | 2018~2020 |
| ● Facebook | BE/DE/ML | UK | 2016~2018 |
| ● Facebook | BE/FE | US | 2015 |
| ● Enswers | BE | KR | 2014 |
| ● KAIST | BSc. | KR | 2012~2016 |

# Outline

- OOP
- Logical Design
- Physical Design
- Naming Notation
- Should I?
  - enums
  - constructors
  - reflection
  - anonymous interfaces
  - pointers for performance
- Must have
- Databases
- Unit-Tests
- Integration-Tests
- Tracing
- Tools

# First class OOP

- Interfaces are powerful
- Don’t over-use
- Don’t afraid to use
- Good for testing
- Use Gang of Four patterns
- Can implement your favorite patterns from other languages
  - Java
  - Hack
  - Typescript/React
  - C++

# Logical Design

- Hexagonal is good
  - Repository + Business Domain + Front Ends
- SOLID is good
  - However do not afraid of big interfaces (Interface Segregation)
  - Break it down and compose into Facade if you want to
- Abstract away Repositories
- Abstract away Transport
- Keep core in Business Domain (entites, types)

# Physical Design

- Keep physical boundaries aligned with logical boundaries
  - Do not leak Business Domain packages into infrastructure/repositories/transport/etc.
- Do not nest packages excessively
  - Because on import only package name (leaf folder name) is used
- Keep name of package same as leaf folder name
  - To avoid renaming of packages on import
- Prefer large packages
  - Basic unit in Go is myclass.go + myclass_test.go
  - Basic unit is not package
- Keep everything in v0 or v1
  - Else make sure Go can recognize your v2+ versioning
- Use /internal to hide packages
- Use vocabulary types
- Avoid renaming on import
- Keep generate files in separate folder/package

# Naming Notation

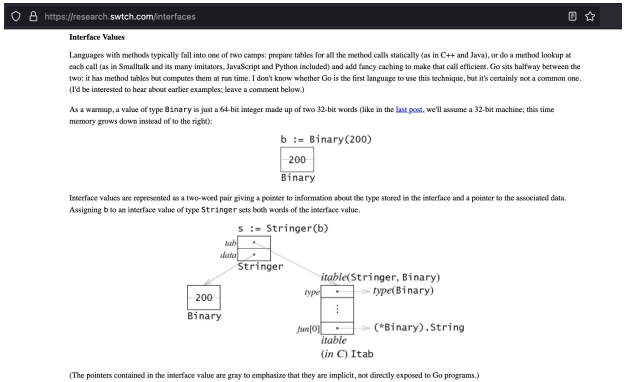
- Keep pair “var mypkg.MySymbol mypkg.MyType” non-repetitive
- MyEnt for entities
- Do not use I prefix/suffix for interfaces
- Do not use Enum prefix/suffix

# Should I do enums?

- Yes
- Keep as string
- Define new type
  - type Status string
- Define const strings for values
  - const Complete Status = “Complete”
- Define JSON unmarshaller that does validation inside
  - func (s *Status) Unmarshal(b []byte) error {var s string; json.Unmarshal(b, &s); switch s: …
- Don’t bother with iota
  - Your service is not as important to save few bits of DRAM
  - You spent more time trying doing it right
- If you confused, just add Enum suffix to your type
- And yes, people still can cast whatever to your type

# Should I use pointer receivers for my service?

- No
- Use structs + interfaces
- If you are worried it passes large, heavily nested struct on every call
  - … it is not true.
- If your struct contains interfaces to upstreams, it is very lightweight
  - … and you need to access those upstream anyways.
  - Thus, struct + interface is even more efficient
  - since you are saving random memory access via pointer.
- You will also save effort checking for nil



# Should I use constructor?

- Yes
- For compile time validation of required argos
  - GOOD NewMyClass(varA string) MyClass
  - BAD MyClass{VarA: varA}

# Should I use pointers for performance?

- No
- Don’t []*MyType
  - Unless you modify myType
- Don’t *[]MyType
  - Unless this is argument, such as receiver in Unmarshal method
- Don’t *[]*MyType
  - Wtf is this
- Don’t func (s *MyService) ForPerformance
  - Unless you struct is validly big
- Don’t func MyFunc (arg *MyStruct)
  - Just pass struct instead, or else: 1) implement Benchmark to show it is actually faster; 2) show that this is bottleneck in your service with tracing.

# Should I use reflection?

- No
- 99% of time you don’t need it
- It is slow
- It is common source of panics
- It is detected at runtime
- It requires a lot of tests
- It is hard to maintain with new types
- It is easy to make mistakes

# Should I use interface{}?

- No
- It has tricky cases with nil
- It is too weak type, it is a sign you can narrow down more
- Strong types is easy to read

# Must have

- Circuit-breaker
  - https://github.com/sony/gobreaker
- Rate-limiter
  - https://pkg.go.dev/golang.org/x/time/rate
- Env-config
  - https://github.com/kelseyhightower/envconfig
- Tracer/Monitoring
  - Datadog, NewRelic

# Databases

- Suggest avoiding ORMs, sqlx, squirrel, etc.
  - they are not good yet, add more trouble
  - standard go sql + database driver is good enough
- Keep DB specific structs private
- Export business domain types
  - to keep business domain same and easy switch to alternative repo

# Unit-Tests

- Avoid testing boundaries of service
  - Don’t test mysql in unit
  - Don’t test wire traffic in unit
- Use code generators for unit test doubles
  - good one https://github.com/vektra/mockery
- Write table tests
- Use constructors for mock dependencies
  - myUpstreamA: func() *mocks.UpstreamA
- Use asserts for calls
  - mua := tc.myUpstreamA()
  - defer mua.AssertExpectations(t)
- Assert number of calls called
  - s.On(“MyFunc”, mocks.Anything, 123).Once().Return(“success”, nil)

# Integration-Tests

- Use over-the-wire test doubles
  - good one https://www.mbtest.org/
  - more complex https://docs.pact.io/
- Set up databases/doubles/service/runner in Docker compose
- Don’t run e2e in parallel
- Reset environment on every test case
- Register doubles dynamically
- Write just as normal tests
- Good to test repositories
- Good to test transport
- Good to test whole service

# Tracing

- Don’t write logs all the time, they are expensive and need more work
- Do write metrics with tags, they are cheap and easy to work with
- Do define APM resources/services, as it is helps visualization
- Do mark APM traces as errors based on logs or whatever, not just return
- Don’t have too many tracing spans, just few important ones
- Don’t write warn logs too many times, prefer error
- Datadog metrics are hot

# Tools

- Avoid importing too many things, break dependencies, visualize
  - https://github.com/nikolaydubina/go-recipes
- Check if your v2+ version is recognized
- Benchmarks, Fuzzy testing
  - e.g. https://github.com/nikolaydubina/go-featureprocessing
- Linters
  - Suggests do not run aggregators
  - Run what you need and understand
  - https://github.com/nikolaydubina/go-featureprocessing/blob/main/Makefile

# Thanks!