Go Generics

Gwendal Leclerc(OVHcloud)

Meetup Golang Rennes – May 31st, 2022

OVHcloud

Who am I?

- Full Stack developer since 2013
- Go developer since 2015

Domain Team @OVHcloud since 2017

gwleclerc

@Skillo1989

gwendal-leclerc-02555b92

Why are we talking about this?

Because it has been released in Go 1.18!

Why so much time?

*“do you want slow **programmers**, slow **compilers** and bloated binaries, or slow **execution** times?”*
— Russ Cox, 2009

https://golang.org/doc/faq

> Go was intended as a language for writing server programs that would be easy to maintain over time. […] The design concentrated on things like scalability, readability, and concurrency. **Polymorphic programming did not seem essential** to the language's goals at the time, and so was left out for simplicity.
>
> Generics are convenient but **they come at a cost in complexity** in the type system and run-time. We haven't yet found a design that gives value proportionate to the complexity, although we continue to think about it. Meanwhile, Go's built-in maps and slices, plus the ability to use the empty interface to construct containers (with explicit unboxing) mean in many cases it is possible to **write code that does what generics would enable**, if less smoothly.

Use case: *func index(s ?, x ?) int*

- **Ad-hoc implementations**

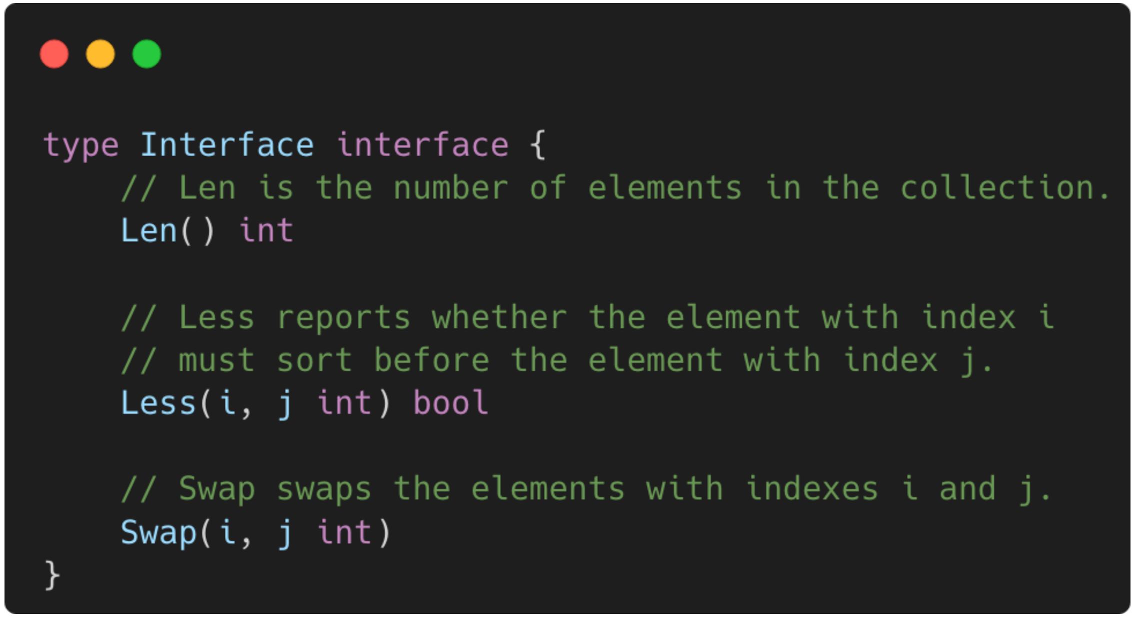
  func indexString(s []string, x string) int

  func indexInt(s []int, x int) int

- **Use []interface{}, interface{}**
  - the slice has to be converted manually beforehand…
  - do types really match ?

What about functional paradigms?
➔replaced by loops

How to code without generics?



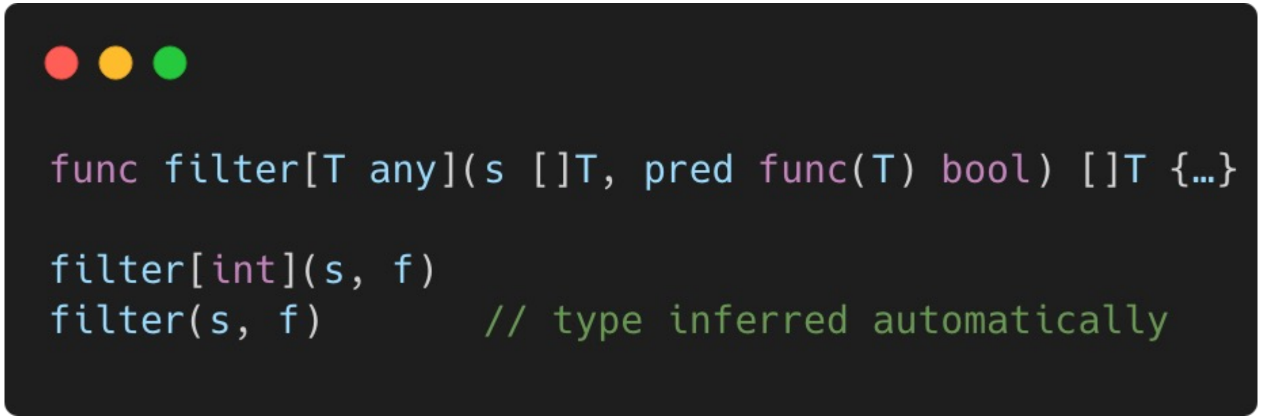
```go
type Interface interface {
    // Len is the number of elements in the collection.
    Len() int

    // Less reports whether the element with index i
    // must 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)
}
```

https://golang.org/pkg/sort

- Not always possible
- Code duplication to implement the interface…

Now, this is over!

Syntax



```
func filter[T any](s []T, pred func(T) bool) []T {…}

filter[int](s, f)
filter(s, f)        // type inferred automatically
```

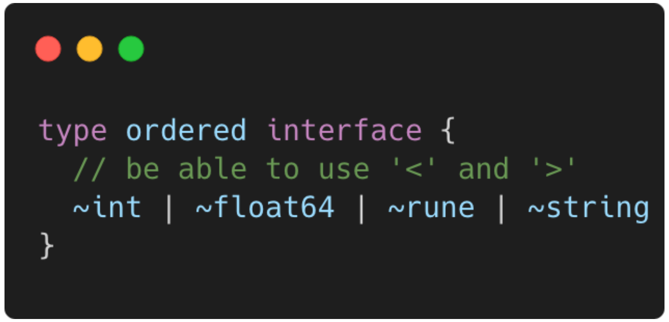
- backwards-compatible with Go < 1.18
- square brackets to avoid ambiguities
  - *a, b = w < x, y > (z)*
- official tutorial on go.dev

Demo

https://gotipplay.golang.org/

Constraints

- we often need to constrain the type with an **interface**
- **any** replaces **interface{}**
- **comparable** allows to use **==** and ***!=***
- constraints may include a set of types



```go
type ordered interface {
  // be able to use '<' and '>'
  ~int | ~float64 | ~rune | ~string
}
```

- many other options

Standard library

Probably in 3 steps, to evaluate usage and needs:

1. Add generics to the language (v1.18) DONE
   - **constraints** : Integer, Ordered, …
2. Add basic libraries (currently experimental)
   - **slices** : Index, Compare, …
   - **maps** : Keys, Equal, …
3. Add extra libraries (streams? Filter, Map, Reduce, …)
   - currently third-party like juniper

To go further : github proposals

Implementation

- **“slow programmers”**

  ↕

- **“slow compilers and bloated binaries”**

  design/generics-implementation-stenciling

  ↕

- **“slow execution times”**

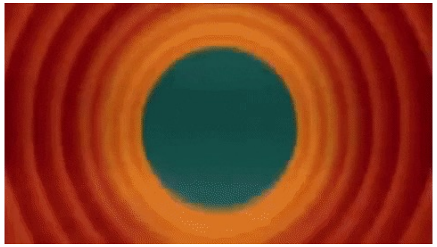
  design/generics-implementation-dictionaries

- **Using a mix of both:**

  design/generics-implementation-gcshape

- ⚠️ Performance issues in some cases!

Thank you!



Logo credits:
Renee French
DC Comics

OVHcloud