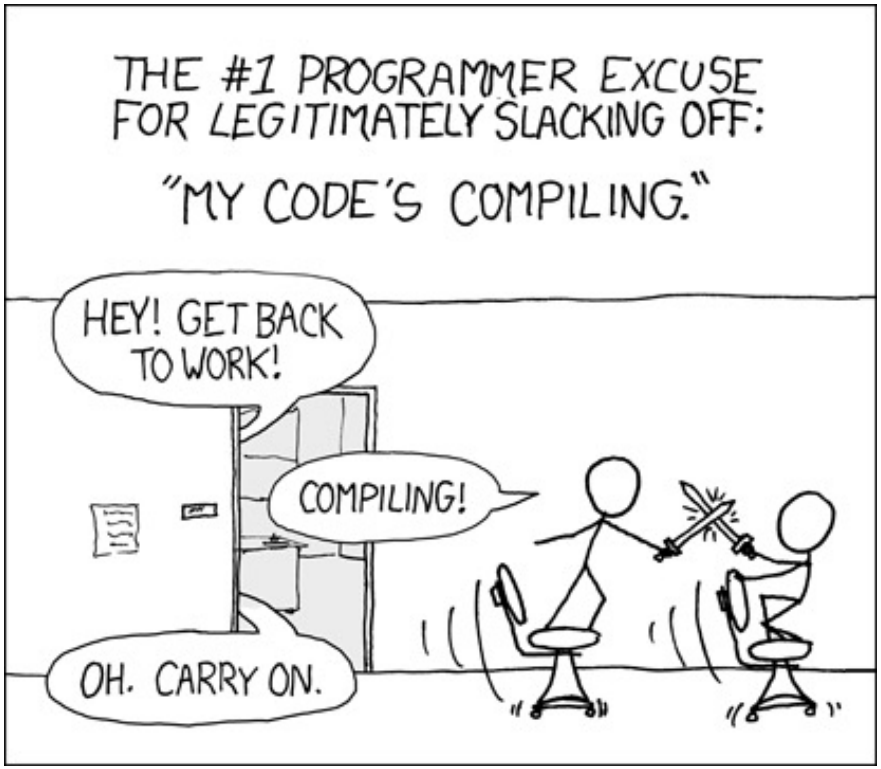
The Fundamentals and Design Patterns



Solving modern programming challenges with Go

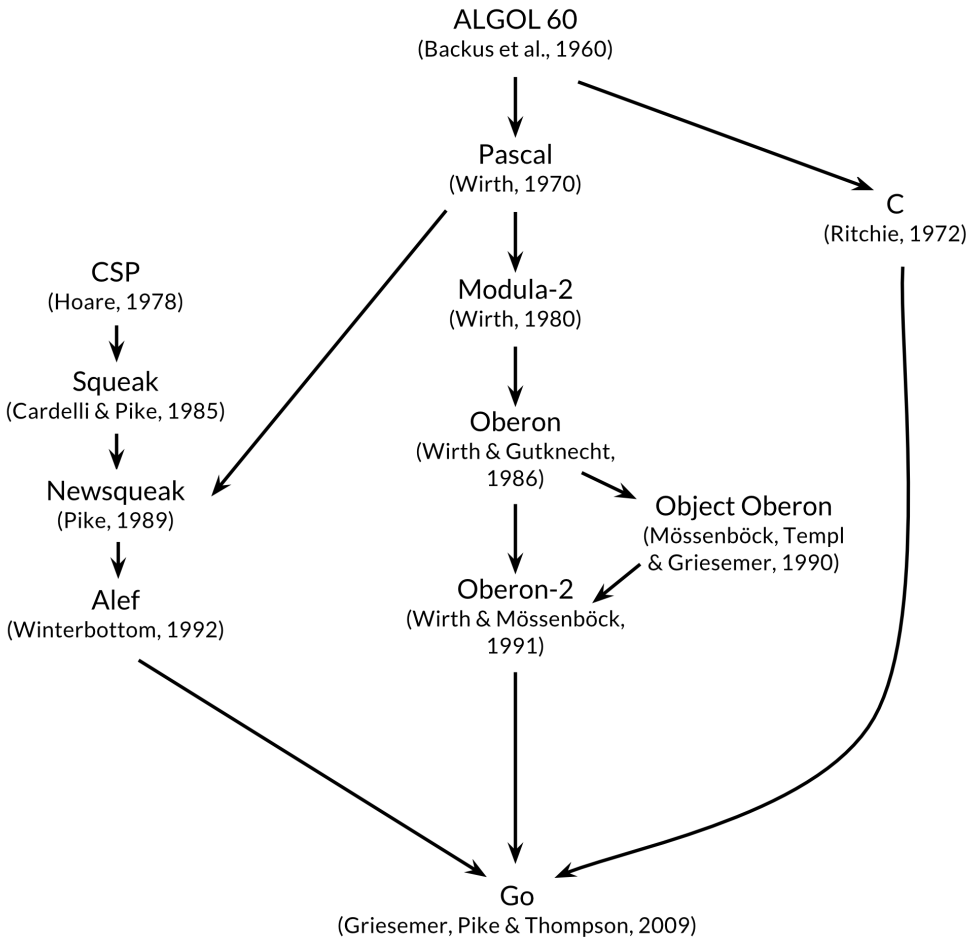
Developers must make an uncomfortable choice between rapid development and performance when choosing a language for their projects.

Languages like C and C++ offer fast execution, whereas languages Like Ruby and Python offer rapid development.

Go bridges these competing world and offers a high-performance language with features that make development fast.

Object-Oriented development Effective type system

Concurrency Garbage collector Fast compiler



The origins of Go

Like biological species, successful language
Beget offspring that incorporate the advantages
of their ancestors.

Go is sometimes described as a “C-like language”
Or as “C for the 21s century.”

From C, Go inherited its expression syntax,
control-flow statements, basic data types, call-by-value
parameter passing, pointers, and above all.

Requires Visual Studio Code Extension

All the extensions you should have to make app development in Go better.

Go

Contributed by Go Team at Google

Error Lens

Contributed by Alexander

Go Group Imports

Contributed by aleksandra

Go module initialization

Create a go module by using the command to initialize a module file

```
$ go mod init <module_name>
```

Give the module a meaningful and concise name. Such as ”Your project name”, ”Your service name”

For example: xver.cloud/internal/computes

Go Packages

Programs start running in package main

This program is using the packages with
Import paths “fmt” and “rsc.io/quote”

By convention, the package name is the same as
The last element of the import path.

For instance, the ”xver.cloud/internal/computes/nova” package
comprises files that begin with the statement package nova

```
package main

Import (
    “fmt”
    “rsc.io/quote”
)


func main() {
    fmt.Println(quote.Go())
}
```

Main package executation

Execute the main package with the go command to see the results.

```
$ go run . || $ go run main.go || $ go run <path of main package>
```

If the main package is not in the root directory But it's in a subdirectory.
You can put the path of the subdirectory to do this.

For example: $ go run ./cmd/api/main.go || $ go run ./cmd/api/.

Build an execution file with the main package

Build an execution file with the main package by go command and choose your architecture

```
$ GOOS=windows GOARCH=amd64 go build .
```

If you are using a different architecture, there are additional options for other hardware as well.

For example (linux,amd64): $ GOOS=linux GOARCH=amd64 go build .

For example (darwin,arm64): $ GOOS=darwin GOARCH=arm64 go build .

Live reload for Go apps

Tools that will help you work faster when testing logic or functions. that was created

https://github.com/cosmtrek/air

Data types, Variables and functions

Available data types and the actual format for declaring variables and functions within Go with Styles.

```
var foo int = 10    gx := 10.1    name := "yongyuth"    func (string) int
```

Pointer concepts

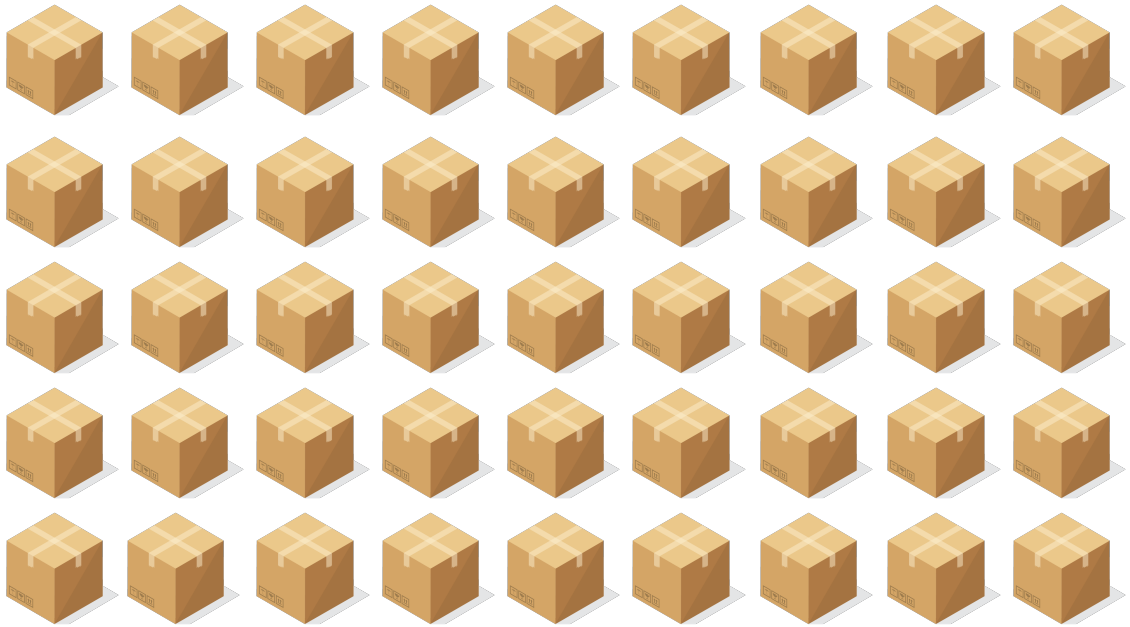
A pointer is a helper that allows you to access a variable though a pointer by address reference.

```
var foo int = 10
```

```
0x0006f7a156
```

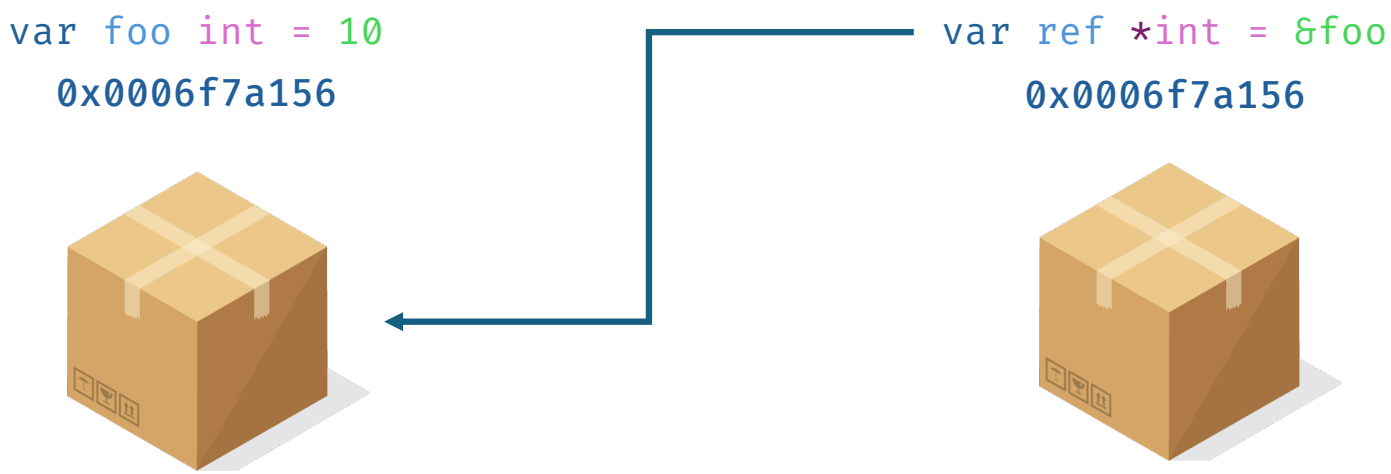
Pointer concepts

A pointer is a helper that allows you to access a variable
though a pointer by address reference.



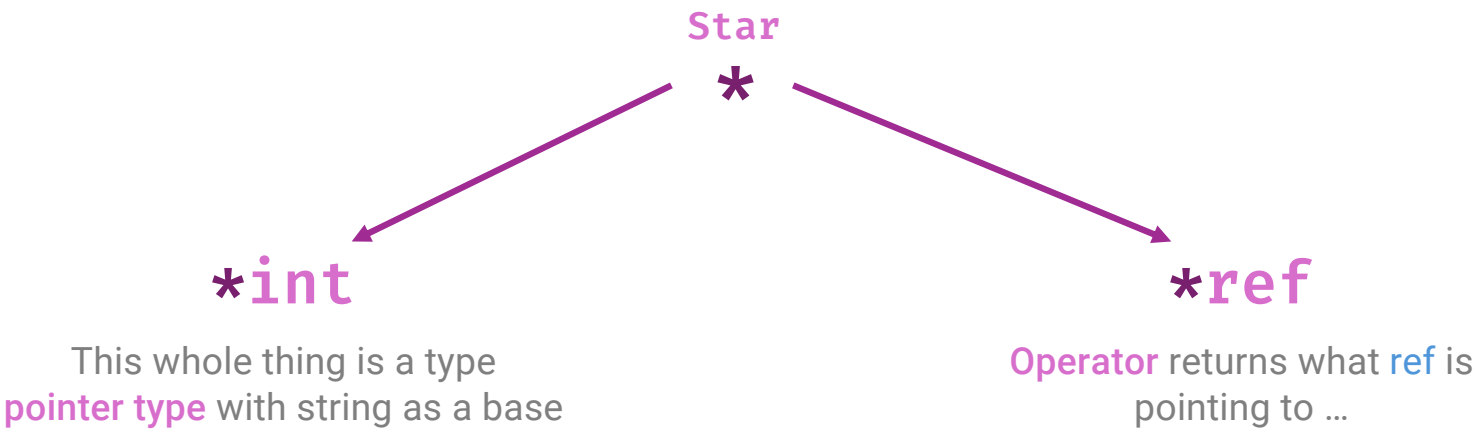
Pointer concepts

A pointer is a helper that allows you to access a variable
though a pointer by address reference.



Pointer concepts

A pointer is a helper that allows you to access a variable
though a pointer by address reference.



Flow control statements

This allows you to control the flow of your data and intercept what you need to benefit the program.

```
if {...}  for {...}  switch {...}  else {...}  defer
```

Exported names in Go

If you want to allow others to access
the functionality or interface and struct. Must use uppercase letters

For public: `func (s *Server) Run() error` || `func Contains[T comparable](v T) bool`

For private: `func (s *Server) listen() error` || `func health(http *http.Request) error`

Methods and interfaces

Specifies the behavior of the objects you create and what is available within any objects you create.

```
type Player struct {...}  type IPlayer interface {...}

func (p *Player) Walk() error {...}
```

Generics

Functions or structs that support multiple data types reduce code redundancy.

```
func PlayerClass[T any](t T) {...}
```

Concurrency and Channel

It allows you to run processes in parallel that makes the program more efficient.

```
go func (){...}()   go RoomMessageWebSocketHub()
```

Design Patterns in GO

Structured model to respond to businesses encountered in real life

Classification of patterns

Understand the meaning and classify different types of patterns.

Creational patterns

Builder

Structural patterns

Facade

Behavioral patterns

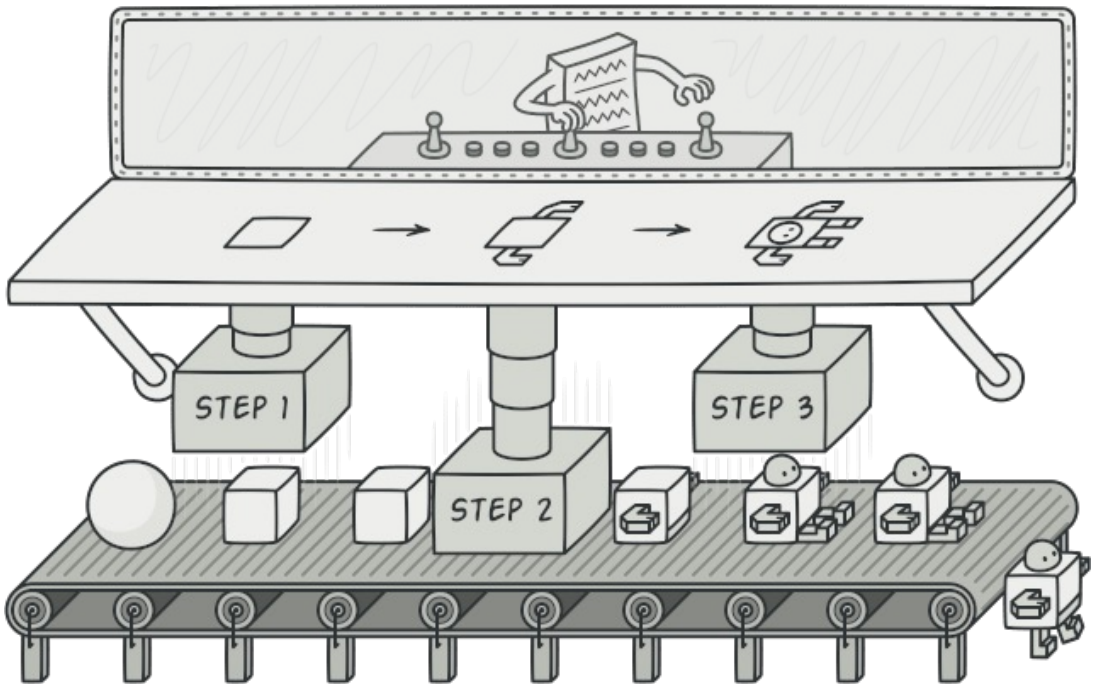
Iterator

Creational patterns (Builder)

These patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

Intent

Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.



Problem

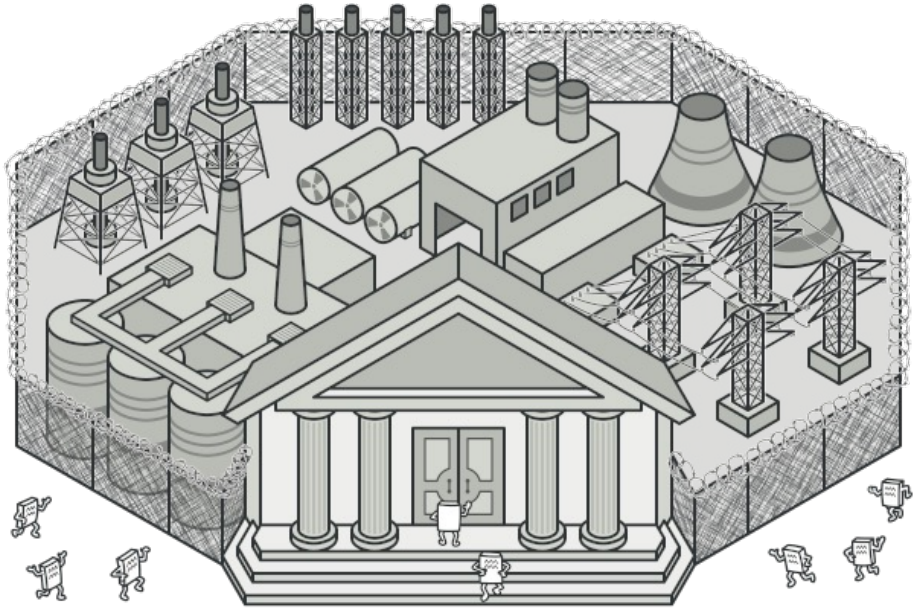
Imagine a complex object that requires laborious, step-by-step initialization of many fields and nested objects. Such initialization code is usually buried inside a monstrous constructor with lots of parameters. Or even worse: scattered all over the client code.

Structural patterns (Facade)

Structural design patterns explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient.

Intent

Facade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes.



Problem

Imagine that you must make your code work with a broad set of objects that belong to a sophisticated library or framework. Ordinarily, you'd need to initialize all of those objects, keep track of dependencies, execute methods in the correct order, and so on.

As a result, the business logic of your classes would become tightly coupled to the implementation details of 3rd-party classes, making it hard to comprehend and maintain.

Behavioral patterns (Strategy)

Behavioral design patterns are concerned with algorithms and the assignment of responsibilities between objects.

Intent

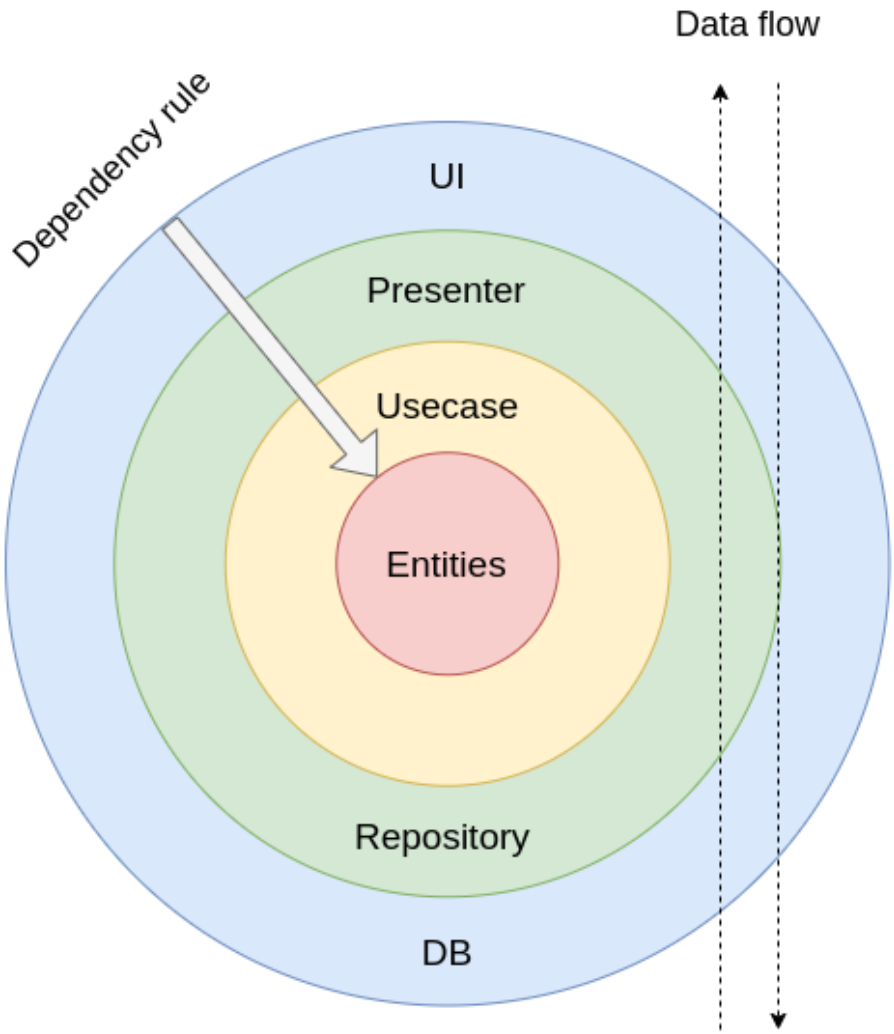
Strategy is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.

Problem

Imagine that customers at different levels come to pay and each level has a different discount rate. Of course, at each level The algorithm of actions is not the same. That creates a problem.

Workshop in GO

Creating an API to interact with the Database and as a bonus, a Clean Architecture



Clean Architecture

Architecture that helps you structure it better

framework, drivers, DB, and external interfaces

presenters, controllers, and repositories

interactors, input interfaces, and output interfaces

root entities, aggregate entities, and value objects