How to start writing Go code. Code structure and modules introduction.

Session 02

Golang course by Exadel

06 Oct 2022

Sergio Kovtunenko
Lead backend developer, Exadel

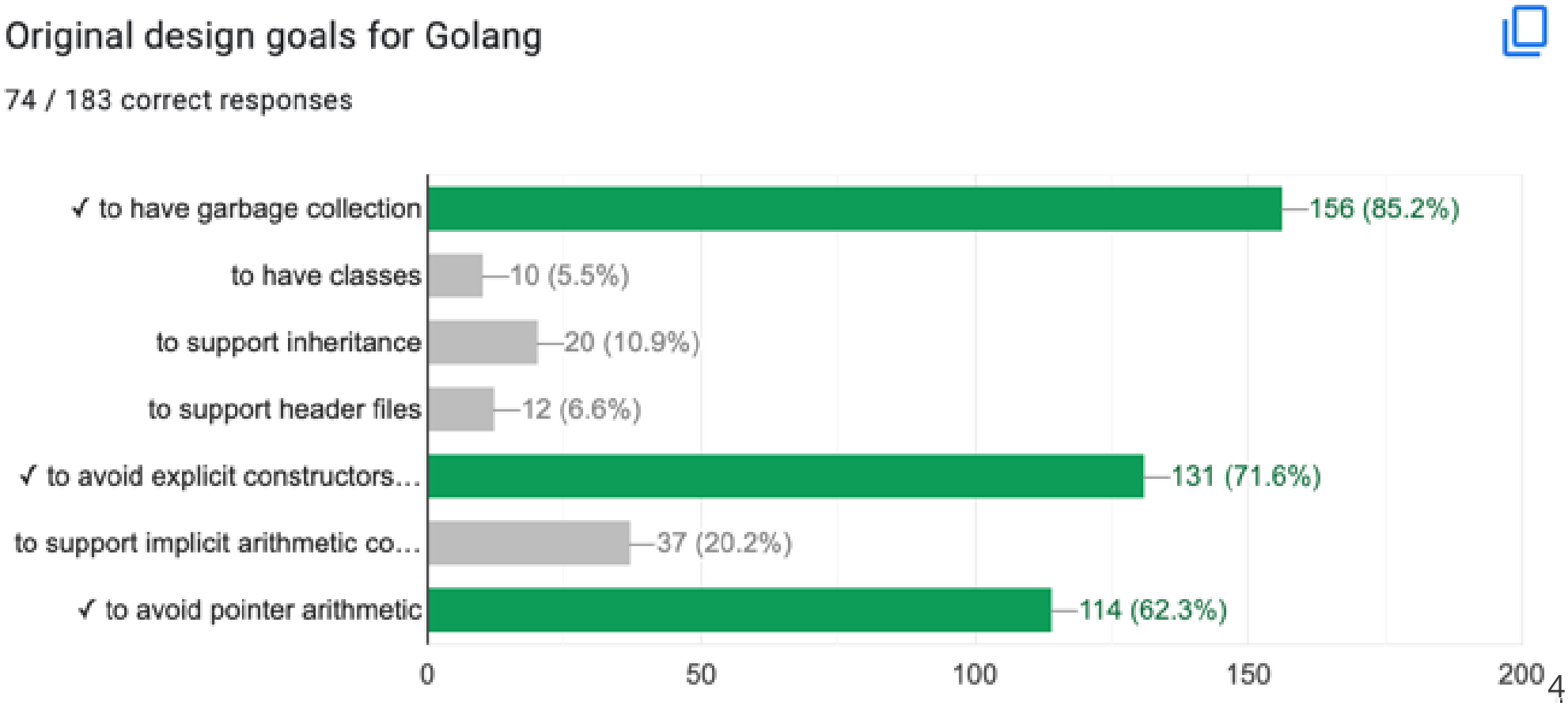
Agenda

- ▶️ Revisit assessment results from the past session
- ▶️ Different options to install Go toolchain
- ▶️ Code editor / IDE
- ▶️ Go toolchain environment variables
- ▶️ Introduction to Go Source files / Packages
- ▶️ GOPATH or no GOPATH ?
- ▶️ Introduction to Go Modules
- ▶️ Go toolchain most common commands
- ▶️ How to structure the code?
- ▶️ Next time...

Revisit assessment results from the past session

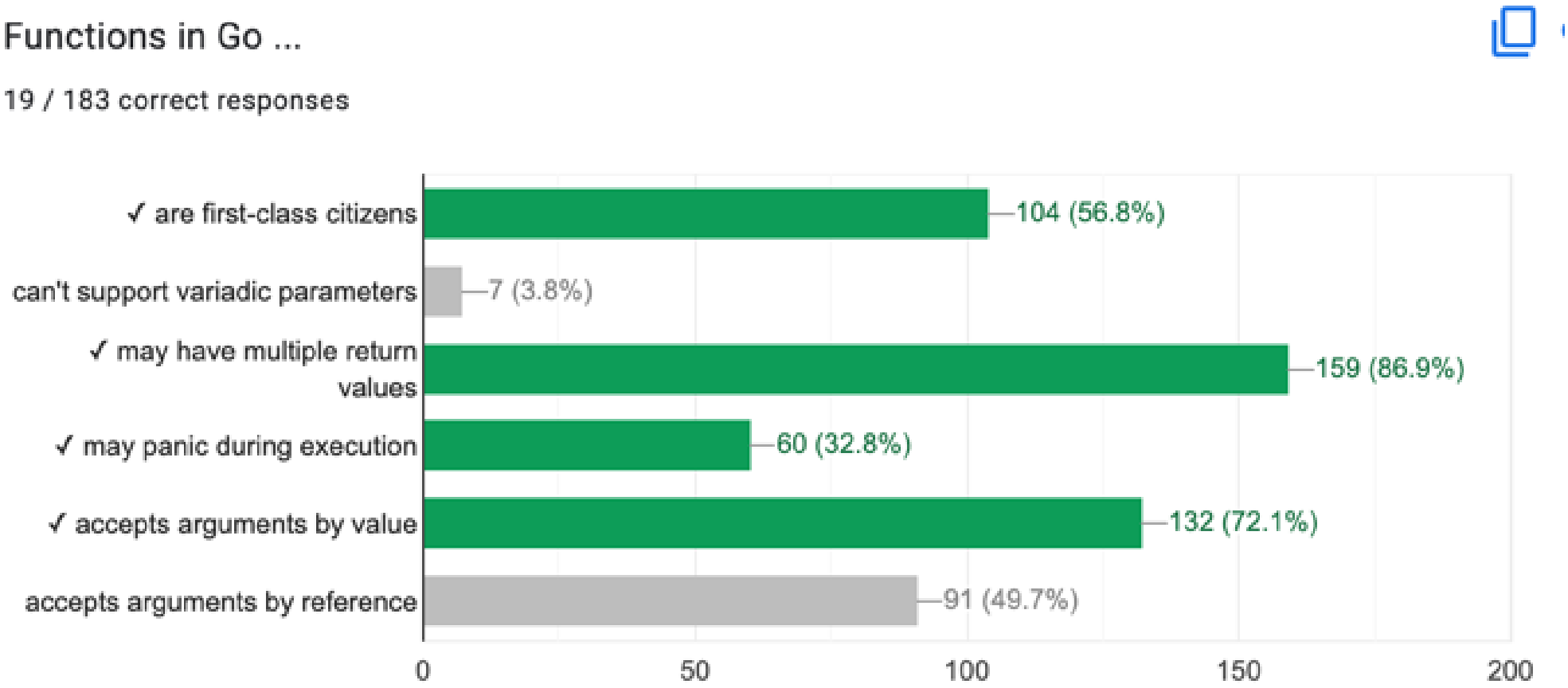
Frequently missed questions (1/3)

Question: Original design goals for Golang?



Frequently missed questions (2/3)

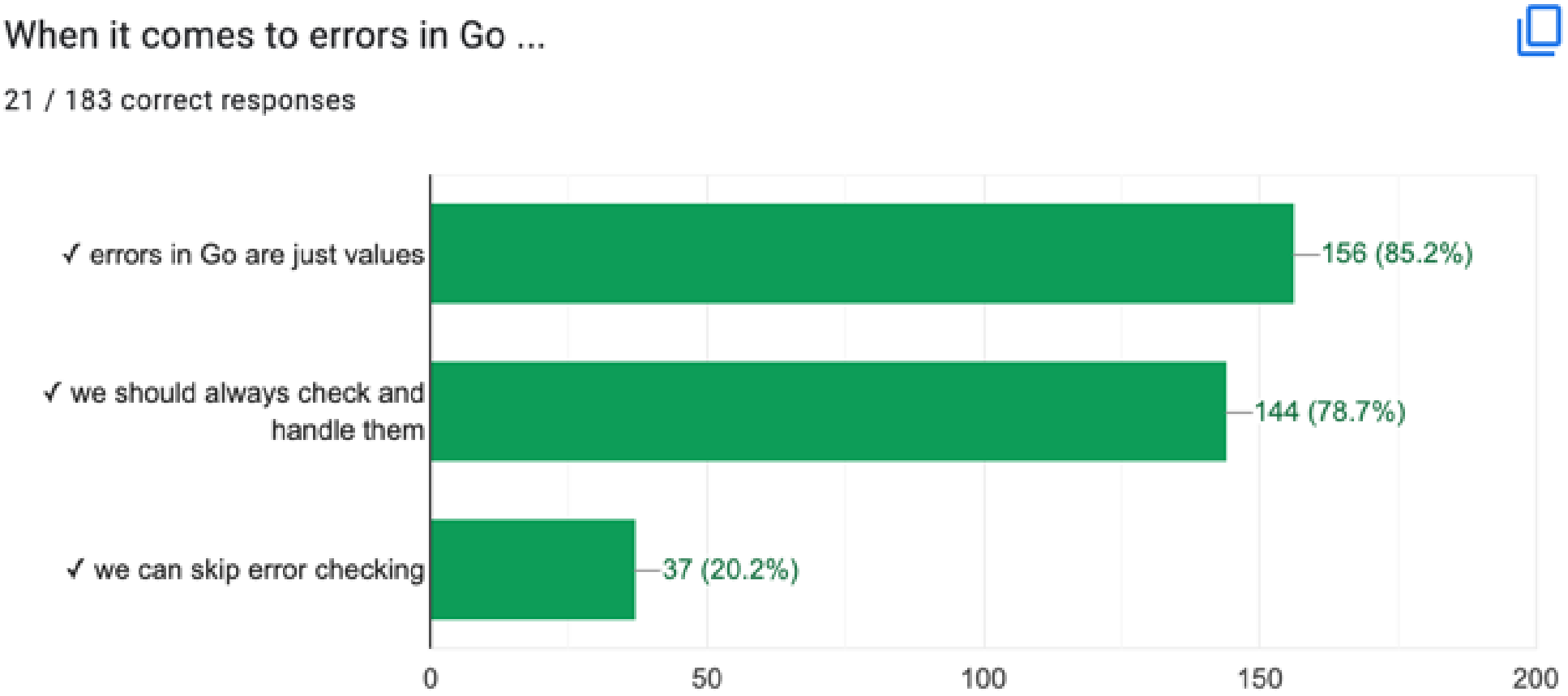
Question: Functions in Go ...



To learn more: "There is no pass-by-reference in Go" by Dave Cheney (https://dave.cheney.net/2017/04/29/there-is-no-pass-by-reference-in-go)

Frequently missed questions (3/3)

Question: When it comes to errors in Go ...



Different options to install Go toolchain

Installation options

- Official website (https://go.dev/dl/). More info: here (https://github.com/golang/go/wiki#working-with-go).
- Package managers (like `brew` on MacOS).
  - maybe the go toolchain already available in your Linux distribution => update it
- GVM (https://github.com/moovweb/gvm) to manage different versions of Go toolchain on the same machine.

Installation path:

- `$ cd ~/.gvm/`

Commands:

- `gvm list`
- `gvm listall`
- `gvm install go1.19.2 -B`
- `gvm use go1.19.2 [--default]`

Code editor / IDE

Editors and IDEs for Go

- Official recommendations located here (https://github.com/golang/go/wiki/IDEsAndTextEditorPlugins)
- Web based solutions: Go Play Space (https://goplay.space/) or Better Go Playground (https://goplay.tools/)
- Personal recommendations:
  - IntelliJ IDEA with Go plugin (https://www.jetbrains.com/idea/) / GoLand (https://www.jetbrains.com/go/)
  - VSCode (https://code.visualstudio.com/) with GoPLS
  - NeoVim (https://neovim.io/)
- VSCode DEMO
  - plugins
- IntelliJ IDEA DEMO
  - plugins

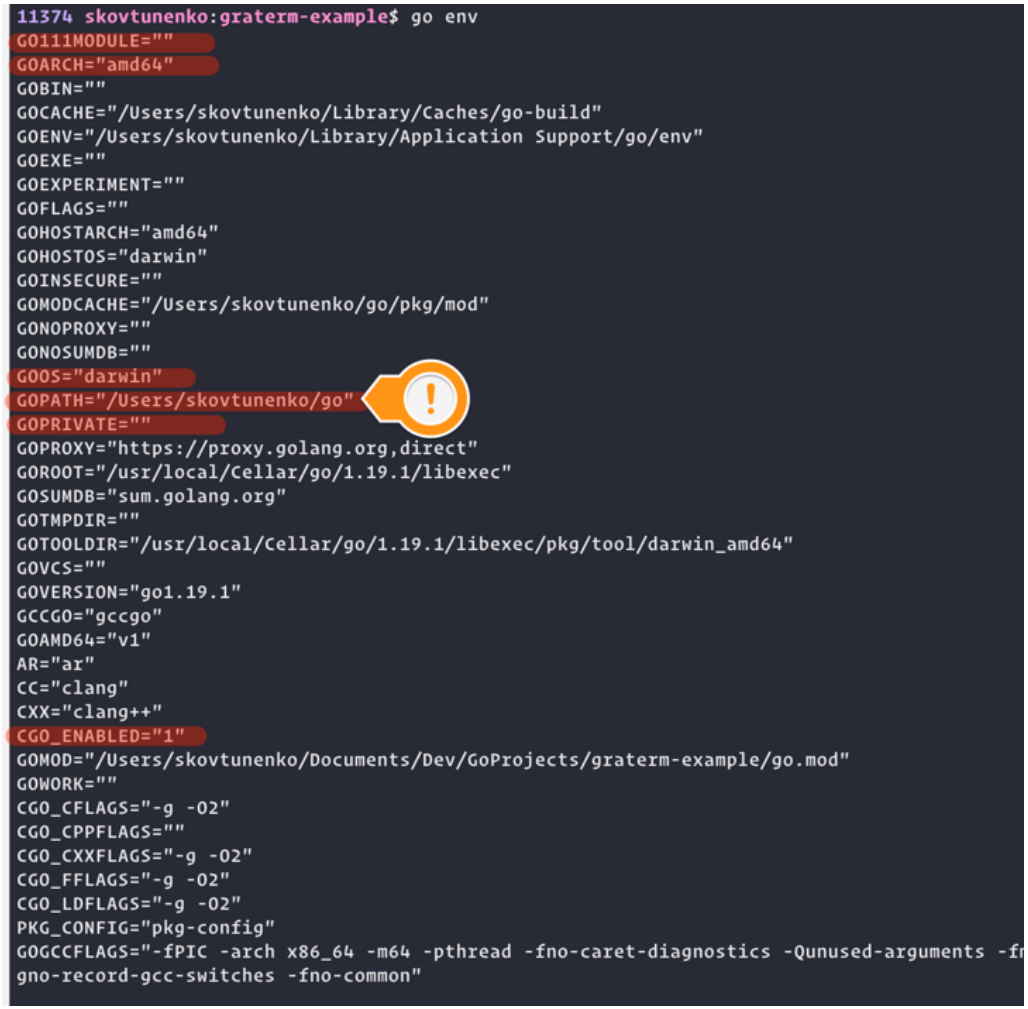
What to expect from IDE/Editor?

- ▶️ Search everywhere
- ▶️ Find all occurrences of "thing"
- ▶️ Show call hierarchy of "thing"
- ▶️ Go to definition of "thing"
- ▶️ Refactorings
- ▶️ Intelligent autocompletion

Go toolchain environment variables

Useful environment variables

▶️ To list them, run: $ go env



Go toolchain most common commands

Go toolchain most common commands

- The Go tool's has many commands:

```
$ go help
Go is a tool for managing Go source code.

Usage:

    go command [arguments]
```

- Worth exploring! Some highlights:

```
mod         module maintenance
build       compile packages and dependencies
get         download and install packages and dependencies
install     compile and install packages and dependencies
test        test packages
```

There are more useful subcommands. Check out `vet` and `fmt`.

Source: "Organizing Go code" by David Crawshaw (https://go.dev/talks/2014/organizeio.slide)

Introduction to Go Source files / Go Packages

Warning!

- ▶️ We will have a separate sessions for "Go Packages" and "Go Modules"
- ▶️ Please focus on high-level concepts.

Go programs are made up of packages

- All Go source is part of a package.
- Every file begins with a package statement.
- Programs start in package main.

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
}
```

Run

- For very small programs, `main` is the only package you need to write.
- The hello world program *imports* package `fmt`.
- The function `Println` is defined in the fmt package.
- The **source directory name** can be different from the **package name** 🤓.

Source: "Organizing Go code" by David Crawshaw (https://go.dev/talks/2014/organizeio.slide)

An example package: fmt

```
// Package fmt implements formatted I/O.
package fmt

// Println formats using the default formats for its
// operands and writes to standard output.
func Println(a ...interface{}) (n int, err error) {
    ...
}

func newPrinter() *pp {
    ...
}
```

▶️ The `Println` function is **exported**. It starts with an upper case letter, which means other packages are allowed to call it.

▶️ The `newPrinter` function is **unexported**. It starts with a lower case letter, so it can only be used inside the fmt package.

Source: "Organizing Go code" by David Crawshaw (https://go.dev/talks/2014/organizeio.slide)

The shape of a package

▶️ Packages collect **related code**.

▶️ They can be **big** or **small**, and may be spread across **multiple files**.

▶️ All the files in a package live in a **single directory**.

▶️ For example:

- the `net/http` package exports more than a hundred Public names.
- the `errors` package exports just a few.

Source: "Organizing Go code" by David Crawshaw (https://go.dev/talks/2014/organizeio.slide)

GOPATH or no GOPATH ?

GOPATH environment variable (legacy approach)

- `go get` always fetches the **latest code**, even if your build breaks.
- Your Go code is kept in a *workspace*.
  - A workspace contains *many* source repositories (git, hg).
  - The Go tool understands the layout of a workspace.
  - You don't need a `Makefile`. The file layout is everything.
- Legacy approach:

```
$GOPATH/
    src/
        github.com/user/repo/
            mypkg/
                mysrc1.go
                mysrc2.go
            cmd/mycmd/
                main.go
    bin/
        mycmd
```

Source: "Organizing Go code" by David Crawshaw (https://go.dev/talks/2014/organizeio.slide)

Adding new dependency in GOPATH mode

▶️ The `GOPATH` environment variable tells the Go tool where your workspace is located.

- `go get github.com/dsymonds/fixhub`

▶️ The `go get` command fetches source repositories from the internet and places them in your workspace.

▶️ Package paths matter to the Go tool. Using "github.com/..." means the tool **knows** how to fetch your repository.

▶️ The go install command builds a binary and places it in `$GOPATH/bin/fixhub`

- Running this command: `go install github.com/dsymonds/fixhub/cmd/fixhub`

▶️ `go get` fetched many repositories.

▶️ `go install` built a binary out of them.

Source: "Organizing Go code" by David Crawshaw (https://go.dev/talks/2014/organizeio.slide)

Legacy code layout in GOPATH mode

- Our workspace:

```
$GOPATH/
    bin/fixhub                              # installed binary
    pkg/darwin_amd64/                       # compiled archives
        code.google.com/p/goauth2/oauth.a
        github.com/...
    src/                                    # source repositories
        code.google.com/p/goauth2/
            .hg
            oauth                           # used by package go-github
            ...
        github.com/
            golang/lint/...                 # used by package fixhub
                .git
            google/go-github/...            # used by package fixhub
                .git
            dsymonds/fixhub/
                .git
                client.go
                cmd/fixhub/fixhub.go        # package main
```

Source: "Organizing Go code" by David Crawshaw (https://go.dev/talks/2014/organizeio.slide)

Introduction to Go Modules

What is a Go Module

▶️ **Go 1.11** introduced a new concept of Modules which brings first class support for managing dependency versions and enabling reproducible builds.

▶️ Go previously had no notion of dependency versions.

▶️ A **Module** is a way to **group together a set of packages** and give it a **version number** to mark it's existence (state) at a specific point in time.

- **Modules** have versions and the **version number is meaningful**.
- Go Modules use Semantic Versioning for their numbering scheme.

▶️ Modules record **precise dependency requirements** and create reproducible builds.

▶️ No more `GOPATH`

Source: "A gentle introduction to Golang Modules" by Ukiah Smith (https://ukiahsmith.com/blog/a-gentle-introduction-to-golang-modules/)

No more GOPATH

▶️ Modules allow for the deprecation of the `GOPATH`.

▶️ There is no longer a need to set it explicitly as a `go.mod` files defines the root of a Module, and allows the Go toolchain to know where everything is that it needs to work with.

- This was the purpose of GOPATH.

Useful module-related commands:

```
11376 skovtunenko:graterm-example$ go mod help
Go mod provides access to operations on modules.

Note that support for modules is built into all the go commands,
not just 'go mod'. For example, day-to-day adding, removing, upgrading,
and downgrading of dependencies should be done using 'go get'.
See 'go help modules' for an overview of module functionality.

Usage:

        go mod <command> [arguments]

The commands are:

        download    download modules to local cache
        edit        edit go.mod from tools or scripts
        graph       print module requirement graph
        init        initialize new module in current directory
        tidy        add missing and remove unused modules
        vendor      make vendored copy of dependencies
        verify      verify dependencies have expected content
        why         explain why packages or modules are needed

Use "go help mod <command>" for more information about a command.
```

How to structure the code?

How to structure the code?

▶️ Not official recommendations golang-standards/project-layout (https://github.com/golang-standards/project-layout)

- Clarification: this is not a standard Go project layout #117 (https://github.com/golang-standards/project-layout/issues/117)

▶️ Another take from Ben Johnson "Standard Package Layout" (https://medium.com/@benbjohnson/standard-package-layout-7cdbc8391fc1)

▶️ One more from Peter Bourgon "Go best practices, six years in" (https://peter.bourgon.org/go-best-practices-2016/#repository-structure)

▶️ Great talk from GopherCon 2018 - How Do You Structure Your Go Apps? (https://about.sourcegraph.com/blog/go/gophercon-2018-how-do-you-structure-your-go-apps)

▶️ Common recommended approach:

- start small, start with **flat project structure**, extract packages when necessary
- **no predefined structure**, everything is based on the domain
- there are *almost no frameworks* in Go to dictate project layout (this is not true)

▶️ **My approach**: steal great ideas from other languages: "Package by Feature" by Philipp Hauer's (https://phauer.com/2020/package-by-feature/)

Next time...

▶️ Session03:

Lexical elements, literals, primitives, variables, constants, declarations, and their scope; program initialization flow

- Comments
- Keywords, operators, and Identifiers
- Literals
- Typed and Untyped constants
- Lack of enums
- Variables
- Conversions
- Variable type inference

Thank you

Golang course by Exadel

06 Oct 2022

Sergio Kovtunenko
Lead backend developer, Exadel
skovtunenko@exadel.com (mailto:skovtunenko@exadel.com)
https://github.com/skovtunenko (https://github.com/skovtunenko)
@realSKovtunenko (http://twitter.com/realSKovtunenko)