Level 5–2

Gophers & Friends

Creating Packages

ON TRACK with GOLANG

When Single Files Grow Too Long

As we add **more code to the main file**, keeping logic for our program inside a single file gets complicated.

src/hello/main.go

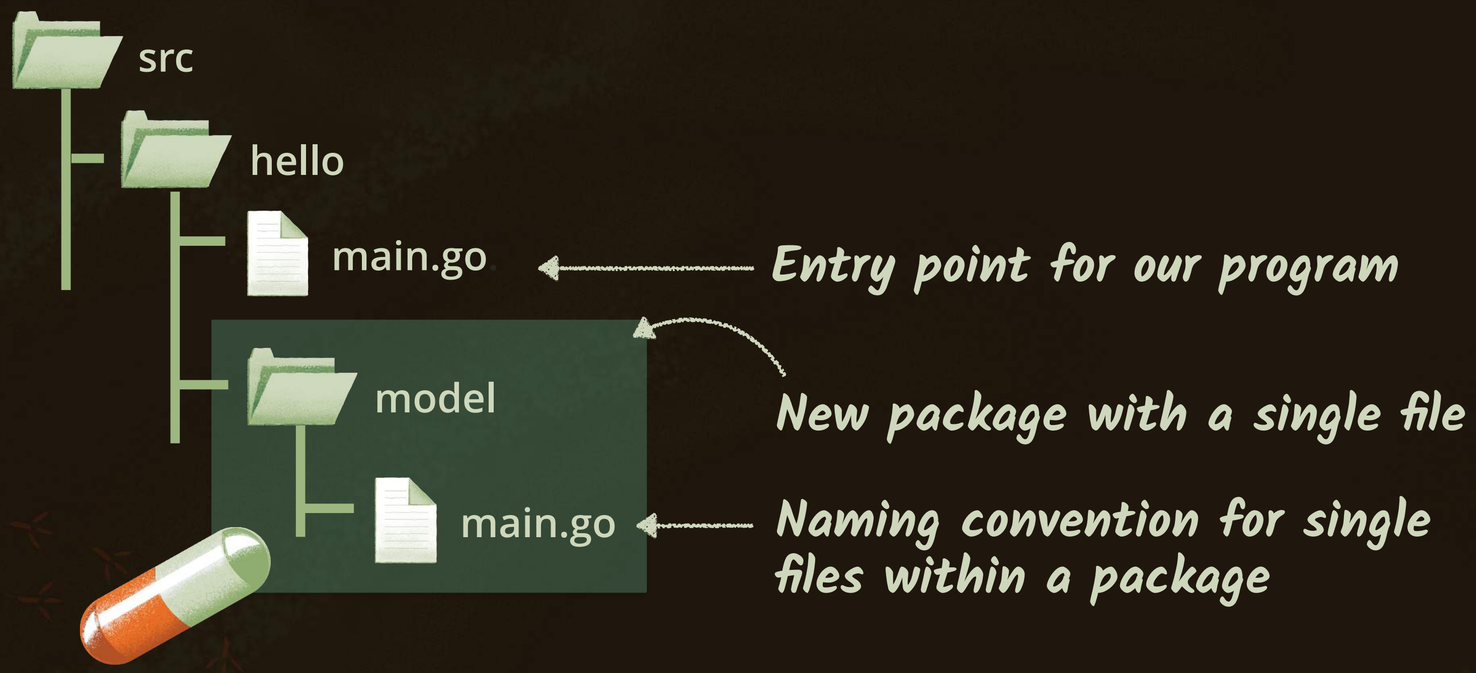
```
package main
import ...

type gopher struct { ... }
func (g gopher) jump() string { ... }
type horse struct { ... }
func (h horse) jump() string { ... }
type jumper interface { ... }

func getList() []jumper { ... }

func main() {
   ...
}
```

Too much code to look at!

Code inside this function is what really matters.

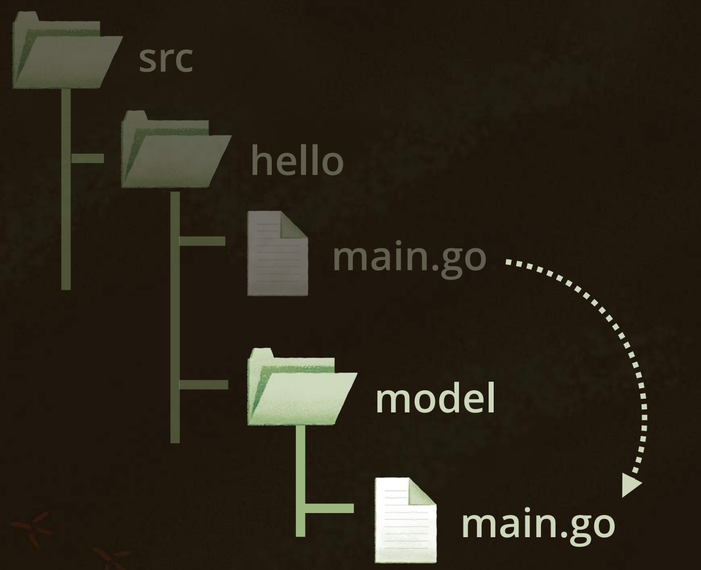
Creating a New Package Inside a Project

One way to refactor files growing too long is to create **project packages.** A new **package** is a **folder** within the project that holds logic for a specific part of the program.



Moving Code to New Package

In order to be accessed from outside packages, identifiers must be **explicitly exported** by adopting an **uppercase naming convention for the first letter.**



Move struct definitions and methods to new package.

src/hello/model/main.go

```
package model
```

New package definition

```
type gopher struct { ... }
func (g gopher) jump() string { ... }
type horse struct { ... }
func (h horse) jump() string { ... }
type jumper interface { ... }
```

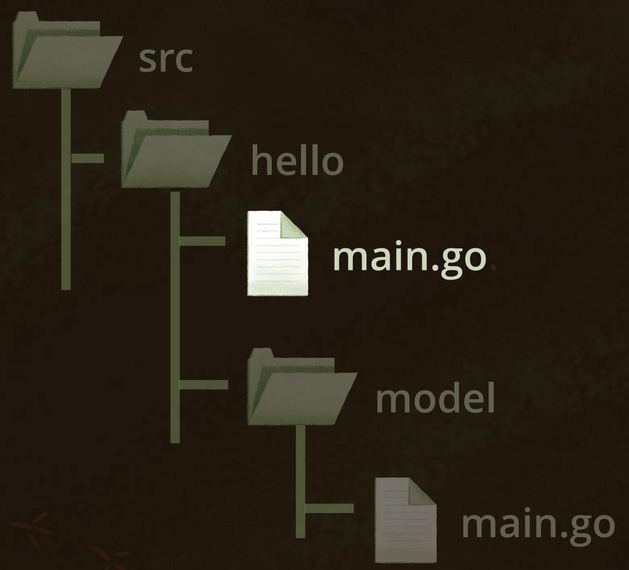
*Moved here from hello/main.go **with no changes***

```
func GetList() []jumper { ... }
```

Capitalized name means this function can now be accessed from outside packages.

Importing Package and Calling Functions

From the main source code file, we can import our new package by using the import statement followed by the project name (**hello**) and the new package name (**model**).



src/hello/main.go

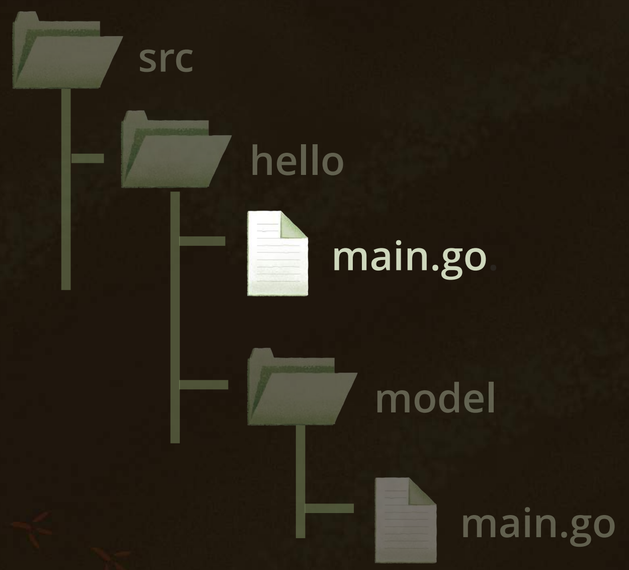
```
package main

import (
  "fmt"
  "hello/model"
)

func main() {
  jumperList := model.GetList()
  for _, jumper := range jumperList {

  }
}
```

Import new package.

Function namespaced by package name

Understanding Export Errors

References to unexported identifiers will cause the Go compiler to **raise errors.**



src/hello/main.go

```
1  package main
2
3  import (
4    "fmt"
5    "hello/model"
6  )
7
8  func main() {
9    jumperList := model.GetList()
10   for _, jumper := range jumperList {
11     fmt.Println(jumper.jump())
12   }
13 }
```

```
$ go run main.go
./main.go:11: jumper.jump undefined
  (cannot refer to unexported field
   or method jump)
```

Whoops! Looks like we forgot to export this method.

Exporting Methods

Interface methods and their corresponding implementations must also be capitalized in order to be invoked from other packages.

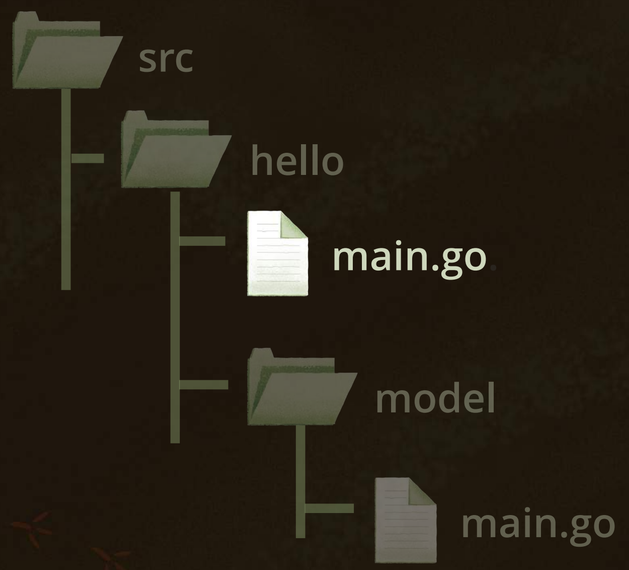
src/hello/model/main.go

```
package model

type gopher struct { ... }
func (g gopher) Jump() string { ... }
type horse struct { ... }
func (h horse) Jump() string { ... }

type jumper interface {
  Jump()
}

func GetList() []jumper {
  ...
}
```

Only the method names need to be exported — NOT the structs or interface.

Running With Correct Package Imports

Our program can now run without errors, and the main file looks a lot cleaner!



src/hello/main.go

```go
package main

import (
  "fmt"
  "hello/model"
)

func main() {
  jumperList := model.GetList()
  for _, jumper := range jumperList {
    fmt.Println(jumper.Jump())
  }
}
```

```
$ go run main.go
```

```
Phil can jump HIGH
Noodles can jump ok.
I will jump, Neigh!!
```