Concurrent Programming: Languages and Techniques

Channel-based Concurrency Module
Lab 1: Introduction to Go

22 September 2022

MIEI - Integrated Masters in Comp. Science and Informatics
Specialization Block

Bernardo Toninho
(with António Ravara and Carla Ferreira)

NOVALINCS

Course Infrastructure

- We are going to use GitHub Classroom to handle labs, mini-project and project submissions.

http://ctp.di.fct.unl.pt/~btoninho/teaching/lpc-22/

- Today's (ungraded) assignment is available here:

https://classroom.github.com/a/AgJ6bVhd

- Sign up, get a git repo and hack away. Don't forget to push.

- Mini-project coming up next week, deadline enforced by GitHub Classroom.

Setup

- Install (a recent version of) Go
- Use whatever IDE you want. Some suggestions:
  - VIM + vim-go
  - Emacs + go-mode / lsp-mode / …
  - VSCode + Go plugin
  - GoLand (JetBrains — Free for students)

Go Packages and Modules

- Go relies on **modules** to manage (external) dependencies and build projects.
- Go relies on **packages** to manage compilation units and namespaces.
- A **module** can contain many **packages**.
- A package can be made up of multiple files, all contained in the same folder.
- Folder names need not match package names, but it is helpful if the names match.
- **Can’t** have different package declarations in the same folder.

Go Packages

```
package main

import (
  "fmt"
)

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

Go Packages

```
package solver

import (
   …
)

type internalT struct {…}
func f1() {…}
func f2() {…}
…
type Solver = …
func (x Solver) Solve() {…}
```

Go Packages

```
package solver

import (
   …
)

type internalT struct {…}
func f1() {…}
func f2() {…}
…
type Solver = …
func (x Solver) Solve() {…}
```

Non-capitalized symbols are **not exported**.

Go Packages

```
package solver

import (
   …
)

type internalT struct {…}
func f1() {…}
func f2() {…}
…
type Solver = …
func (x Solver) Solve() {…}
```

Visible in packages that import this one.

Go Packages

Assuming a **`go.mod`** file defining a **`MyApp`** module:

```
package solver

import (
   …
)

type internalT struct {…}
func f1() {…}
func f2() {…}
…
type Solver = …
func (x Solver) Solve() {…}
```

```
package main

import (
   “fmt"
   “MyApp/solver”
)

func main() {
   solver.New(…).solve()
   fmt.Println("Hello, world!")
}
```

Go Packages

Assuming a `go.mod` file defining a `MyApp` module:

```
package solver

import (
   …
)

type internalT struct {…}
func f1() {…}
func f2() {…}
…
type Solver = …
func (x Solver) Solve() {…}
```

```
package main

import (
   “fmt"
   “MyApp/solver”
)

func main() {
   solver.New(…).solve()
   fmt.Println("Hello, world!")
}
```

Go Packages

Assuming a **`go.mod`** file defining a **`MyApp`** module:

```
package solver

import (
   …
)

type internalT struct {…}
func f1() {…}
func f2() {…}
…
type Solver = …
func (x Solver) Solve() {…}
```

```
package main

import (
   “fmt"
   s “MyApp/solver”
)

func main() {
   s.New(…).solve()
   fmt.Println("Hello, world!")
}
```

Lab 1

- Just to get a feel for some simple Go programming.
- Some sprinkles of concurrency with goroutines and channels.
- Enjoy… :)