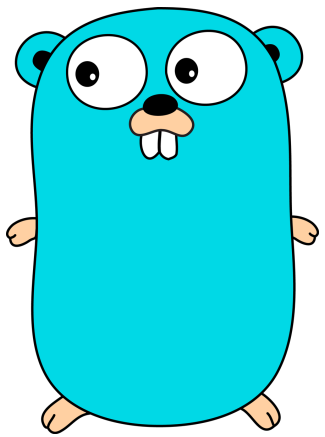
Go in a nutshell

Gianguido Sorà

Università degli Studi di Salerno

23/10/2015



Cos'è Go

- Un linguaggio di programmazione
- Made in **Google**, sviluppato da **Robert Griesemer, Rob Pike, Ken Thompson**
- *expressive, concise, clean and efficient*

Cos'è Go

- Un linguaggio di programmazione
- Made in **Google**, sviluppato da **Robert Griesemer, Rob Pike, Ken Thompson**
- *expressive, concise, clean and efficient*

Cos'è Go

- Un linguaggio di programmazione
- Made in **Google**, sviluppato da **Robert Griesemer, Rob Pike, Ken Thompson**
- *expressive, concise, clean and efficient*

Perché è "cool"?

- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go

Perché è "cool"?

- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go

Perché è "cool"?

- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go

Perché è "cool"?

- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go

Perché è "cool"?

- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go

Perché è "cool"?

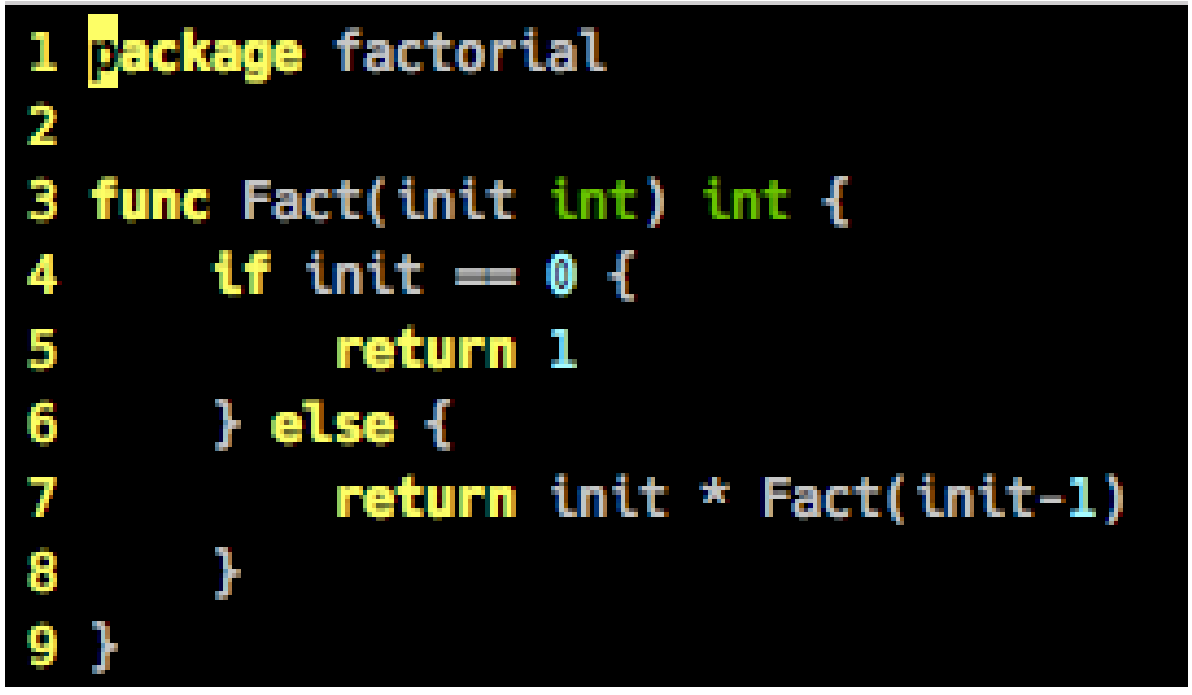
- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go

Perché è "cool"?

- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go

Perché è "cool"?

- compila in codice macchina
- è garbage-collected
- statically-typed, con reflection in run-time
- la sintassi è simile a quella di un linguaggio dinamico
- il modello di concorrenza integrato è facile da applicare
- è disponibile per tutti i maggiori OS...
- ...persino OpenBSD
- a partire dalla versione 1.4.1, il toolkit ed il compilatore sono completamente scritti in Go



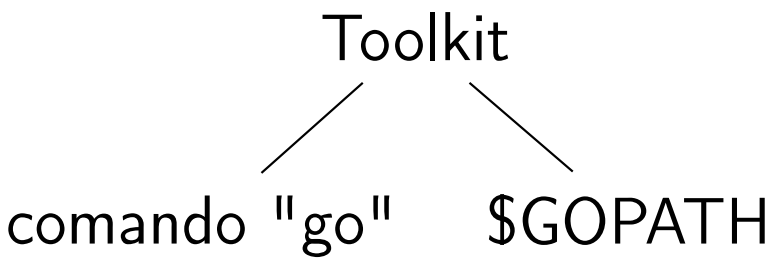
```
1 package factorial
2
3 func Fact(init int) int {
4     if init == 0 {
5         return 1
6     } else {
7         return init * Fact(init-1)
8     }
9 }
```

Figure: Semplice libreria per il calcolo del fattoriale di un numero.

```go
package main

import (
    "factorial"
    "fmt"
)

func main() {
    number := 10

    fmt.Println("fact(10): ", factorial.Fact(number))
}
```

Figure: Implementazione della libreria mostrata prima.

Il toolkit di Go



Il comando "go"

- gestisce la compilazione e l'installazione di codice Go
- gestisce anche il testing...
- ...la documentazione dei pacchetti...
- ...le loro dipendenze (!)
- è simile ad un *Makefile*, ma senza "make" e senza "configure"
- *$ go*

Il comando "go"

- gestisce la compilazione e l'installazione di codice Go
- gestisce anche il testing...
- ...la documentazione dei pacchetti...
- ...le loro dipendenze (!)
- è simile ad un *Makefile*, ma senza "make" e senza "configure"
- *$ go*

Il comando "go"

- gestisce la compilazione e l'installazione di codice Go
- gestisce anche il testing...
- ...la documentazione dei pacchetti...
- ...le loro dipendenze (!)
- è simile ad un *Makefile*, ma senza "make" e senza "configure"
- *$ go*

Il comando "go"

- gestisce la compilazione e l'installazione di codice Go
- gestisce anche il testing...
- ...la documentazione dei pacchetti...
- ...le loro dipendenze (!)
- è simile ad un *Makefile*, ma senza "make" e senza "configure"
- *$ go*

Il comando "go"

- gestisce la compilazione e l'installazione di codice Go
- gestisce anche il testing...
- ...la documentazione dei pacchetti...
- ...le loro dipendenze (!)
- è simile ad un *Makefile*, ma senza "make" e senza "configure"
- *$ go*

Il comando "go"

- gestisce la compilazione e l'installazione di codice Go
- gestisce anche il testing...
- ...la documentazione dei pacchetti...
- ...le loro dipendenze (!)
- è simile ad un *Makefile*, ma senza "make" e senza "configure"
- *$ go*

Il toolkit

- Go è stato progettato per essere **developer-friendly**
- *godoc* genera automaticamente la documentazione per pacchetto in base ai commenti scritti prima di ogni funzione, senza bisogno di particolare sintassi
- *gofmt* formatta un sorgente Go dato in input secondo le guidelines di formattazione standard
- `https://golang.org/doc/cmd`

Il toolkit

- Go è stato progettato per essere **developer-friendly**
- *godoc* genera automaticamente la documentazione per pacchetto in base ai commenti scritti prima di ogni funzione, senza bisogno di particolare sintassi
- *gofmt* formatta un sorgente Go dato in input secondo le guidelines di formattazione standard
- `https://golang.org/doc/cmd`

Il toolkit

- Go è stato progettato per essere **developer-friendly**
- *godoc* genera automaticamente la documentazione per pacchetto in base ai commenti scritti prima di ogni funzione, senza bisogno di particolare sintassi
- *gofmt* formatta un sorgente Go dato in input secondo le guidelines di formattazione standard
- `https://golang.org/doc/cmd`

Il toolkit

- Go è stato progettato per essere **developer-friendly**
- *godoc* genera automaticamente la documentazione per pacchetto in base ai commenti scritti prima di ogni funzione, senza bisogno di particolare sintassi
- *gofmt* formatta un sorgente Go dato in input secondo le guidelines di formattazione standard
- `https://golang.org/doc/cmd`

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

$GOPATH

- il codice Go deve essere tenuto in un **workspace**
- il workspace deve contenere le cartelle:
  - *src*: contiene i sorgenti organizzati in **pacchetti**
  - *pkg*: contiene i file oggetto generati, per ogni pacchetto
  - *bin*: eseguibili risultanti dalla compilazione
- *go build* compila un pacchetto contenuto in **src** e ne rende disponibile l'uso come libreria posando i file oggetto in **pkg**
- *go install* compila un pacchetto che contiene un **main** e ne trasferisce il binario risultante in **bin**
- *$GOPATH* è il workspace su cui lavorare
- dev'essere settato a mano dallo sviluppatore
- è preferibile settare sia *$GOPATH* che *$GOPATH/bin* nella propria *$PATH*

Il sistema dei pacchetti

- Go organizza il codice in unità chiamate **pacchetti**
- il pacchetto **"main"** è quello che contiene il metodo omonimo
- un pacchetto può avere un solo **main**
- un pacchetto può rappresentare una libreria
- i nomi di funzione che iniziano per lettera minuscola non sono visibili all'esterno del pacchetto
- l'opposto per quelli che iniziano per lettera maiuscola
- i pacchetti all'interno di *$GOPATH/src* possono essere inclusi dopo essere stati compilati tramite *go build*

Il sistema dei pacchetti

- Go organizza il codice in unità chiamate **pacchetti**
- il pacchetto **"main"** è quello che contiene il metodo omonimo
- un pacchetto può avere un solo **main**
- un pacchetto può rappresentare una libreria
- i nomi di funzione che iniziano per lettera minuscola non sono visibili all'esterno del pacchetto
- l'opposto per quelli che iniziano per lettera maiuscola
- i pacchetti all'interno di *$GOPATH/src* possono essere inclusi dopo essere stati compilati tramite *go build*

Il sistema dei pacchetti

- Go organizza il codice in unità chiamate **pacchetti**
- il pacchetto **"main"** è quello che contiene il metodo omonimo
- un pacchetto può avere un solo **main**
- un pacchetto può rappresentare una libreria
- i nomi di funzione che iniziano per lettera minuscola non sono visibili all'esterno del pacchetto
- l'opposto per quelli che iniziano per lettera maiuscola
- i pacchetti all'interno di *$GOPATH/src* possono essere inclusi dopo essere stati compilati tramite *go build*

Il sistema dei pacchetti

- Go organizza il codice in unità chiamate **pacchetti**
- il pacchetto **"main"** è quello che contiene il metodo omonimo
- un pacchetto può avere un solo **main**
- un pacchetto può rappresentare una libreria
- i nomi di funzione che iniziano per lettera minuscola non sono visibili all'esterno del pacchetto
- l'opposto per quelli che iniziano per lettera maiuscola
- i pacchetti all'interno di *$GOPATH/src* possono essere inclusi dopo essere stati compilati tramite *go build*

Il sistema dei pacchetti

- Go organizza il codice in unità chiamate **pacchetti**
- il pacchetto **"main"** è quello che contiene il metodo omonimo
- un pacchetto può avere un solo **main**
- un pacchetto può rappresentare una libreria
- i nomi di funzione che iniziano per lettera minuscola non sono visibili all'esterno del pacchetto
- l'opposto per quelli che iniziano per lettera maiuscola
- i pacchetti all'interno di *$GOPATH/src* possono essere inclusi dopo essere stati compilati tramite *go build*

Il sistema dei pacchetti

- Go organizza il codice in unità chiamate **pacchetti**
- il pacchetto **"main"** è quello che contiene il metodo omonimo
- un pacchetto può avere un solo **main**
- un pacchetto può rappresentare una libreria
- i nomi di funzione che iniziano per lettera minuscola non sono visibili all'esterno del pacchetto
- l'opposto per quelli che iniziano per lettera maiuscola
- i pacchetti all'interno di *$GOPATH/src* possono essere inclusi dopo essere stati compilati tramite *go build*

Il sistema dei pacchetti

- Go organizza il codice in unità chiamate **pacchetti**
- il pacchetto **"main"** è quello che contiene il metodo omonimo
- un pacchetto può avere un solo **main**
- un pacchetto può rappresentare una libreria
- i nomi di funzione che iniziano per lettera minuscola non sono visibili all'esterno del pacchetto
- l'opposto per quelli che iniziano per lettera maiuscola
- i pacchetti all'interno di *$GOPATH/src* possono essere inclusi dopo essere stati compilati tramite *go build*

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

E gli oggetti?

- ***esistono, ma non esistono***
- Go **non** implementa un vero e proprio paradigma OOP
- niente ereditarietà, niente polimorfismo
- è stato implementato un meccanismo simile adottando
  - **tipi integrati** (relazione has-a)
  - **pseudo-subtyping** (relazione is-a)
  - **true subtyping** (relazione is-a)
- in questo bar non si serve caffè

Tipi integrati: structs e object composition

- funzionano come se fosse C
- la dichiarazione è sintatticamente più semplice
- ogni struttura implica la definizione di un ADT
- è possibile definire strutture che contengono strutture, realizzando una relazione "has-a"
- è possibile definire *metodi*, funzioni che agiscono e sono legate ad una struttura di un determinato tipo

Tipi integrati: structs e object composition

- funzionano come se fosse C
- la dichiarazione è sintatticamente più semplice
- ogni struttura implica la definizione di un ADT
- è possibile definire strutture che contengono strutture, realizzando una relazione "has-a"
- è possibile definire *metodi*, funzioni che agiscono e sono legate ad una struttura di un determinato tipo

Tipi integrati: structs e object composition

- funzionano come se fosse C
- la dichiarazione è sintatticamente più semplice
- ogni struttura implica la definizione di un ADT
- è possibile definire strutture che contengono strutture, realizzando una relazione "has-a"
- è possibile definire *metodi*, funzioni che agiscono e sono legate ad una struttura di un determinato tipo

Tipi integrati: structs e object composition

- funzionano come se fosse C
- la dichiarazione è sintatticamente più semplice
- ogni struttura implica la definizione di un ADT
- è possibile definire strutture che contengono strutture, realizzando una relazione "has-a"
- è possibile definire *metodi*, funzioni che agiscono e sono legate ad una struttura di un determinato tipo

Tipi integrati: structs e object composition

- funzionano come se fosse C
- la dichiarazione è sintatticamente più semplice
- ogni struttura implica la definizione di un ADT
- è possibile definire strutture che contengono strutture, realizzando una relazione "has-a"
- è possibile definire *metodi*, funzioni che agiscono e sono legate ad una struttura di un determinato tipo

Pseudo-subtyping

- si applica sempre tramite structs
- la differenza è sintattica, logica e funzionale
- realizza una relazione "is-a"
- ...ma non è vero e proprio subtyping

Pseudo-subtyping

- si applica sempre tramite structs
- la differenza è sintattica, logica e funzionale
- realizza una relazione "is-a"
- ...ma non è vero e proprio subtyping

Pseudo-subtyping

- si applica sempre tramite structs
- la differenza è sintattica, logica e funzionale
- realizza una relazione "is-a"
- ...ma non è vero e proprio subtyping

Pseudo-subtyping

- si applica sempre tramite structs
- la differenza è sintattica, logica e funzionale
- realizza una relazione "is-a"
- ...ma non è vero e proprio subtyping

True subtyping: interfaces

- realizza una vera relazione "is-a"
- dichiarazione simile a quella delle strutture
- per implementare l'interfaccia, basta implementare i suoi metodi.

True subtyping: interfaces

- realizza una vera relazione "is-a"
- dichiarazione simile a quella delle strutture
- per implementare l'interfaccia, basta implementare i suoi metodi.

True subtyping: interfaces

- realizza una vera relazione "is-a"
- dichiarazione simile a quella delle strutture
- per implementare l'interfaccia, basta implementare i suoi metodi.

True subtyping: interfaces

```go
package main

import "fmt"

type Human interface {
        Talk()
}

type Person struct {
        Name    string
        Address Address
}

type Address struct {
        Number string
        Street string
        City   string
        State  string
        Zip    string
}

func (p *Person) Talk() {
        fmt.Println("Hi, my name is", p.Name)
}

func (p *Person) Location() {
        fmt.Println("I'm at", p.Address.Number, p.Address.Street, p.Address.City, p.Address.State, p.Address.Zip)
}

type Citizen struct {
        Country string
        Person
}

func (c *Citizen) Nationality() {
        fmt.Println(c.Name, "is a citizen of", c.Country)
}

func SpeakTo(h Human) {
        h.Talk()
}

func main() {
        p := Person{Name: "Dave"}
        c := Citizen{Person: Person{Name: "Steve"}, Country: "America"}

        SpeakTo(&p)
        SpeakTo(&c)
}
```

Figure: Esempio di implementazione del true subtyping tramite interfaccia.

True subtyping: interfaces

```
Hi, my name is Dave
Hi, my name is Steve
```

Figure: Risultato dell'esecuzione del codice precedente.

Gestione degli errori?

- ...no!
- Go non implementa *direttamente* meccanismi per gestire errori
- ogni funzione restituisce sempre un valore di errore insieme al risultato della sua computazione
- in base al valore di errore, il programmatore deve comportarsi di conseguenza

Gestione degli errori?

- ...no!
- Go non implementa *direttamente* meccanismi per gestire errori
- ogni funzione restituisce sempre un valore di errore insieme al risultato della sua computazione
- in base al valore di errore, il programmatore deve comportarsi di conseguenza

Gestione degli errori?

- ...no!
- Go non implementa *direttamente* meccanismi per gestire errori
- ogni funzione restituisce sempre un valore di errore insieme al risultato della sua computazione
- in base al valore di errore, il programmatore deve comportarsi di conseguenza

Gestione degli errori?

- ...no!
- Go non implementa *direttamente* meccanismi per gestire errori
- ogni funzione restituisce sempre un valore di errore insieme al risultato della sua computazione
- in base al valore di errore, il programmatore deve comportarsi di conseguenza

Gestione degli errori?

```
 1 func errorPlease() (dummy int, err error) {
 2     // funzione dummy, restituisce sempre 0 ed un errore
 3     return 0, errors.New("errorPlease: here's your error!")
 4 }
 5
 6 ...
 7
 8
 9 func main() {
10     value, err = errorPlease()
11
12     if err != nil {
13         fmt.Println("Oh well, error!")
14     }
15 }
```

Figure: Esempio di dichiarazione e gestione di funzione con valore di errore

Testing!

- Go include un sistema per testare il codice
- **Esempio:** il sorgente chiamato "Fle.go", viene testato da "Fle_test.go"
- ogni pacchetto può essere testato dal comando *go test*

Testing!

- Go include un sistema per testare il codice
- **Esempio:** il sorgente chiamato "Fle.go", viene testato da "Fle_test.go"
- ogni pacchetto può essere testato dal comando *go test*

Testing!

- Go include un sistema per testare il codice
- **Esempio:** il sorgente chiamato "Fle.go", viene testato da "Fle_test.go"
- ogni pacchetto può essere testato dal comando *go test*

Testing!

```go
package talkTests

import (
    "fmt"
)

func Fle(val string) (int, error) {
    // se val == fle, dummy = 1, altrimenti dummy = 0, errore

    if val == "fle" {
        return 1, nil
    } else {
        return 0, fmt.Errorf("Fle: noooo")
    }
}
```

Figure: Funzione da testare

Testing!

```go
package talkTests

import (
    "testing"
)

// per ogni funzione da testare, si crea una funzione
// con la firma TestNomeFunzione (t *testing.T)

func TestFle(t *testing.T) {

    tString := "fle"

    res, _ := Fle(tString)

    if res != 1 {
        t.Error("se la funzione avesse funzionato, avrei dovuto restituire 1!")
    }
}
```

Figure: Funzione di testing

Testing!

```
% go test
PASS
ok      talkTests       0.007s
```

Figure: Risultato del testing

Links

- http://play.golang.org/
- https://golang.org/
- https://tour.golang.org
- https://golang.org/doc/faq