objects & behaviour

object orientation in go

no classes

no inheritance (no type hierarchy)

methods are just functions

encapsulation

composition

subtyping

structs

```
type Person struct {
    Name string
    Age int
}

var p1 Person
p2 := Person{}

p3 := Person{"Gopher", 4}
p4 := Person{
    Name: "Igor",
    Age: 33, //comma is mandatory
}

fmt.Printf("%s is %d years old", p3.Name, p3.Age)
```

pass by value

function parameters are passed by value

```
func update(p Person) {
    p.Name = "Igor"
    p.Age = 33
}

gopher := Person{"Gopher", 4}
update(gopher)

fmt.Printf("%s is %d years old", gopher.Name, gopher.Age)
```

> Gopher is 4 years old

pointers

```
func update(p *Person) {
    p.Name = "Igor"
    p.Age = 33
}

gopher := &Person{"Gopher", 4}
update(gopher)

fmt.Printf("%s is %d years old", gopher.Name, gopher.Age)
> Igor is 33 years old


var p Person = *gopher
```

methods

Methods can be assigned to any named type

```
func (p Person) Speak() {
    fmt.Printf("Hello! I am %v.", p.Name)
}
gopher := Person{"Gopher", 4}
gopher.Speak()
```

> Hello! I am Gopher.

Methods can be called on pointer and non-pointer

```
gopher := &Person{"Gopher", 4}
gopher.Speak()
```

pointer receivers

receivers are just normal parameters

only syntactic sugar

```
func (p Person) HappyBirthday() {
    p.Age += 1
}
gopher.HappyBirthday()  // no change!

func (p *Person) HappyBirthday() {
    p.Age += 1
}
gopher.HappyBirthday()  // +1
```

encapsulation

encapsulation on package level

accessibility defined by capitalization

```
type Person struct { //public
    Name string   //public
    age int       //private
}
func (p Person) Speak() { //public
    fmt.Println("Hello!")
}
func (p *Person) happyBirthday() { //private
    p.age += 1
}
```

subtyping / embedding

```
type User struct {
    Person //embedding
    Id int
}

user := User{
    Person: Person{"Igor", 33},
    Id: 42,
}
user.Speak()
user.HappyBirthday()

fmt.Printf("%s is %d years old", user.Name, user.Age)
```

interfaces

interfaces are types

```
type Speaker interface {
    Speak()
}
```

no declaration required

a type implements an interface if it implements all methods

```
func (p Person) Speak() {
    fmt.Println("Hello!")
}
```

empty interface

by definition every type implements

```
interface{} // the empty interface
```

compatible with any type

```
func printStuff(anything interface{} ) {
    fmt.Printf("%v" , anything)
}
printStuff(5)
printStuff(Person{})
```

packages

used for decomposition & encapsulation

referenced via import

looked up from GOPATH

or relative paths: import "../addressbook"

executables always in main

first function to be executed: func init() {}

session 3 – objects & behaviour

https://github.com/iigorr/go-workshop
3-data/README.md + CheatSheet.md

3.1 Address Book

- object orientation
- maps
- interfaces
- packages

+ Variants