COMPARISON OBJECT ORIENTED PROGRAMMING IN GO VS OBJECTIVE-C

SLAWOMIR DANZL

SLAWOMIR.OLSZOWKA@STUD.FH-ROSENHEIM.DE

CONCEPTS OF PROGRAMMING LANGUAGE

17.01.2019

Technische Hochschule Rosenheim
Technical University of Applied Sciences

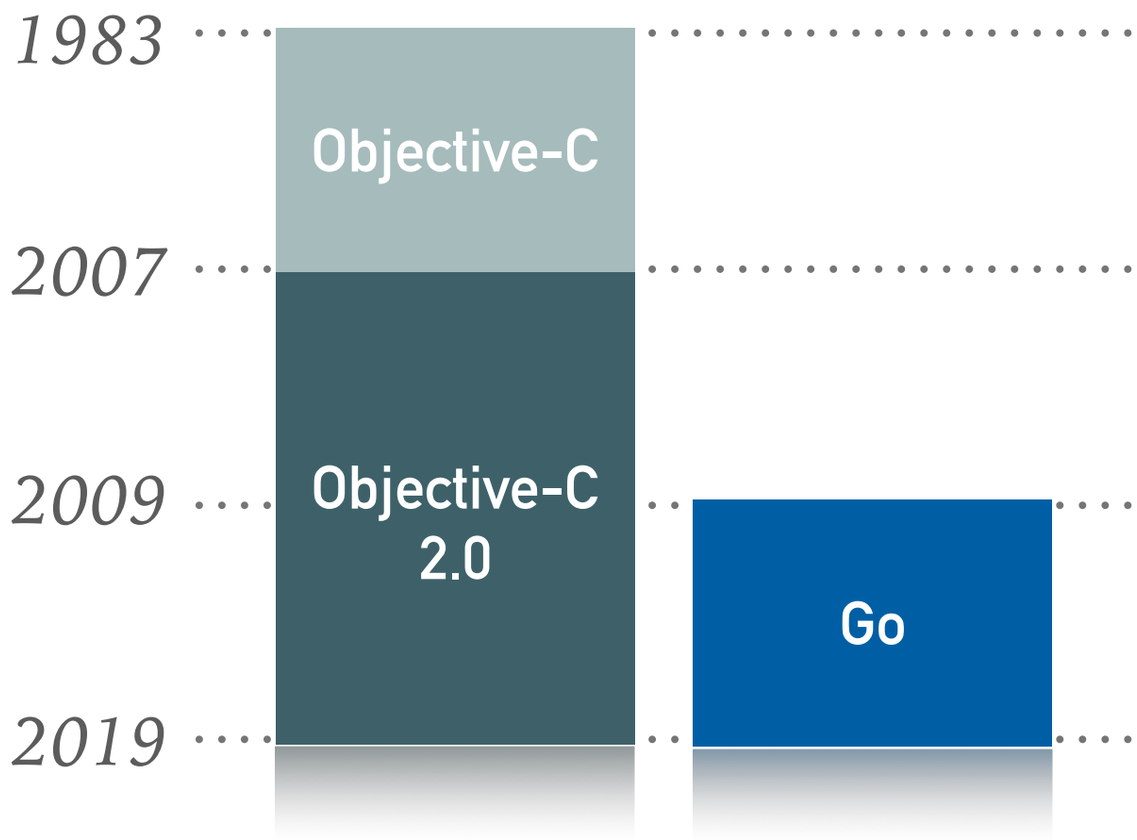
“Actually I made up the term “object-oriented", and I can tell you I did not have C++ in mind.”

Alan Key (1997)

AGENDA

1. History of Go and Objective-C
2. Object-Oriented Programming Principles
3. Comparison OOP in Go vs. Objective-C
4. Original Conception

Origins of Objective-C and Go



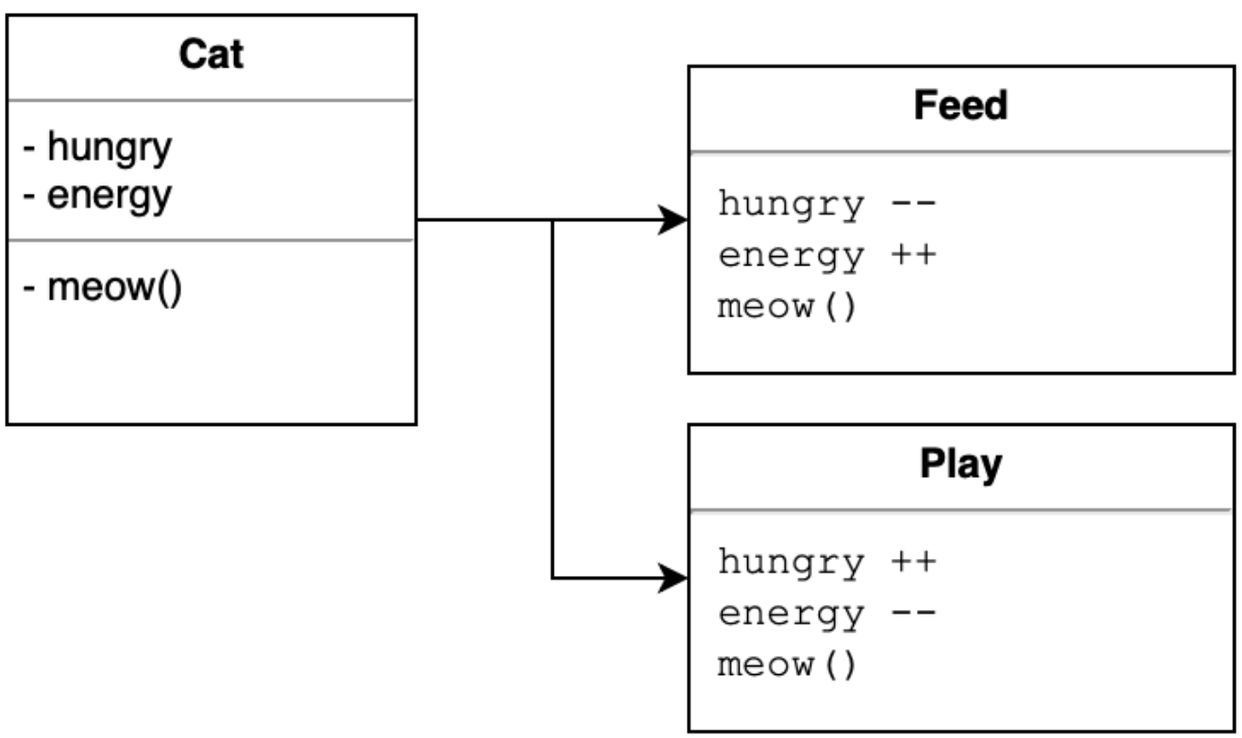
OBJECT-ORIENTED PROGRAMMING
PRINCIPLES

The four principles of object-oriented programming are:

- **encapsulation**
- **inheritance**
- **polymorphism**
- **dynamic method binding**

OBJECT-ORIENTED PROGRAMMING
ENCAPSULATION

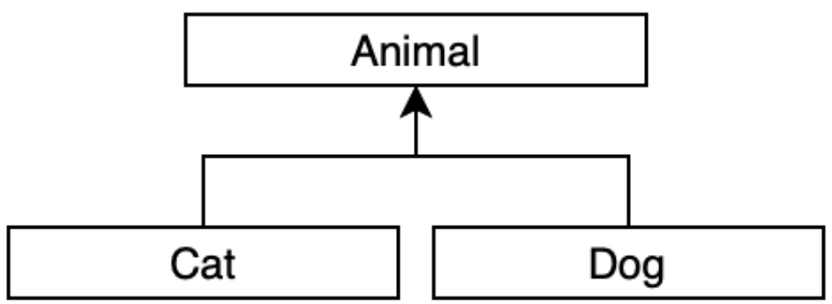
Encapsulation is achieved when each object keeps its state **private** (Information hiding).



Source: My own cat

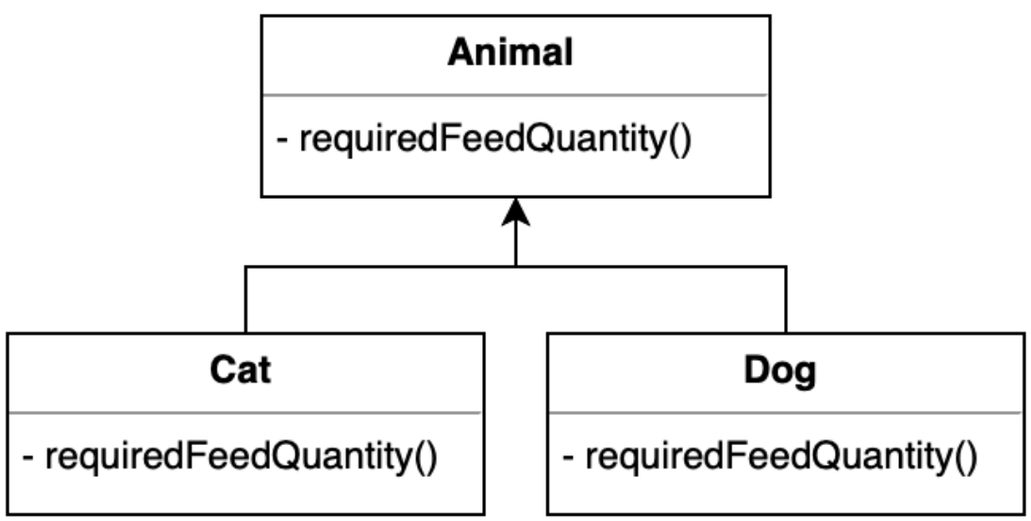
OBJECT-ORIENTED PROGRAMMING
INHERITANCE

Inheritance from a Hierarchy:
Deriving a class (**child**) from another (**parent**) class.



OBJECT-ORIENTED PROGRAMMING
POLYMORPHISM

Polymorphism ("many forms") gives a way to use a class exactly like its parent.
But each child class keeps its own methods as they are.



OBJECT-ORIENTED PROGRAMMING
DYNAMIC METHOD BINDING

Dynamic binding (late binding) is a mechanism by which a computer program waits **until runtime** to bind the name of a method.

OBJECT-ORIENTED COMPARISON

ENCAPSULATION

GO

```go
package cat

import "fmt"

// Cat struct can be exported outside of this package
type Cat struct{}

// Expose method can be exported outside of this package
func (cat *Cat) Expose() {
    fmt.Println("Meow! I'm exposed!")
}

// hide method can only be used within this package
func (cat *Cat) hide() {
    fmt.Println("Meow... this is super secret")
}

// Unhide uses the unexported hide function
func (cat *Cat) Unhide() {
    cat.hide()
    fmt.Println("...")
}
```

Public elements:
capitalizing the first letter

Private elements:
lowercase the first letter

[Objective C]

```objc
// Cat.h

#import "Animal.h"


NS_ASSUME_NONNULL_BEGIN

@interface Cat : Animal

// protected
@property(nonatomic, strong) NSString* sound;

// public
- (void) makeNoise;

- (void) calculateWeightAfterFeeding;

@end

NS_ASSUME_NONNULL_END
```

Public elements:
in the definition file (.h)

Private elements:
in the implementation file (.m)

OBJECT-ORIENTED COMPARISON

INHERITANCE

GO

No inheritance

Composition over inheritance principle.

Accomplish through both subtyping (is-a) and object composition (has-a) relationships between structs and interfaces.

[Objective C]

```
@interface Cat : Animal
```

Only multilevel inheritance

All Classes have only one base class.

All classes in Objective-C are derived from the superclass NSObject.

OBJECT-ORIENTED COMPARISON

POLYMORPHISM

GO

```go
type Weights interface {
    calculate() int
    source() string
}

type Cat struct {
    catName  string
    weight int
}

func calculateWeight(ic [] Weights) {
    // calculate weight for all cats
}

func main() {
    bigCat := Cat{catName: "Kitty", weight: 6}
    biggerCat := Cat{catName: "Tomcat", weight: 11}
    catWeights := []Weights{bigCat, biggerCat}
    calculateWeight(catWeights)
}
```

Polymorphism over Interfaces

[Objective C]

```objc
Cat *cat = [[Cat alloc] init];
[cat calculateWeightAfterFeeding:10];

Dog *dog = [[Dog alloc]init];
[dog calculateWeightAfterFeeding:10 andMoreFeeding:5];

NSArray *pets = [[NSArray alloc] initWithObjects: cat, dog, nil];

id object1 = [pets objectAtIndex:0];
[object1 weight];

id object2 = [pets objectAtIndex:1];
[object2 weight];
```

Polymorphism over Inheritance

OBJECT-ORIENTED COMPARISON

DYNAMIC METHOD BINDING

No dynamic method binding

The only way to have dynamically dispatched methods is through an interface.

Methods on a struct or any other concrete type are always resolved statically.

[Objective C]

```
Cat *cat = [[Cat alloc] init];
[cat calculateWeightAfterFeeding:10];

Dog *dog = [[Dog alloc]init];
[dog calculateWeightAfterFeeding:10 andMoreFeeding:5];

NSArray *pets = [[NSArray alloc] initWithObjects: cat, dog, nil];

id object1 = [pets objectAtIndex:0];
[object1 weight];

id object2 = [pets objectAtIndex:1];
[object2 weight];
```

Dynamic method binding available

In Objective-C, all methods are resolved dynamically at runtime.

The exact code executed is determined by both the method name (the selector) and the receiving object.

ORIGINAL CONCEPTION
DEFINITON OBJECT ORIENTED PROGRAMMING

Alan Kay's original conception (1993) of "object oriented" based on:

- **messaging**
- **local retention, protection and hiding of state-process**
- **extreme late-binding of all things**

ORIGINAL CONCEPTION
OBJECT ORIENTED PARADIGMS

	GO	[Objective C]
MESSAGING	Via Channels	Methods dynamically bound to messages
LOCAL RETENTION AND PROTECTING	Same as encapsulation	
LATE BINDING	Same as dynamic method binding	

ADDITIONAL OOP PARADIGMS

ABSTRACTION

GO

```go
type Animal interface {
    Sound() string
    MakeSound()
}

type abstractAnimal struct {Animal}

func (a abstractAnimal) MakeSound() {
    fmt.Printf("%v\n", a.Sound())
}
```

Abstraction over Interfaces

[Objective C]

```oc
@interface Animal : NSObject

- (void) makeSound;

@end


@implementation Animal

- (void) makeSound { [self doesNotRecognizeSelector:_cmd]; }

@end
```

No abstraction

Workaround (mock an abstract class): Making the methods or selectors call “doesNotRecognizeSelector” and therefore raise an exception making the class unusable.

OBJECT-ORIENTED COMPARISON

OVERVIEW

	GO	[Objective C]
ENCAPSULATION	✓	✓
INHERITANCE	⊘	✓
POLYMORPHISM	✓	✓
DYNAMIC METHOD BINDING	⚠	✓
MESSAGING	✓	✓
ABSTRACTION	✓	⚠

SUMMARY
COMPARISON OOP IN GO VS OBJECTIVE-C

Apple and Objective C stress on OOP, no wonder Objective-C has all the properties of an object-oriented language.

Go took the best parts of OOP, left out the rest and gave a better way to write polymorphic code.

SUMMARY
EVALUATION

Comparison depends on the point of view, definition of OOP and the application purpose.

Perhaps the evaluation according to OOP is also an unsuitable criterion.

SOURCES

Alan Kay, The Early History of Smalltalk, 1993, https://dl.acm.org/citation.cfm?doid=155360.155364

Go Language Website, golang.org

Apple Website, apple.com

Aaron Hillegass, Objective-C Programming, 2011, Addison Wesley

Amit Singh, A Brief History of Mac OS X, 2003, Mac OS X Internals

Matt Galloway, Effective Objective-C 2.0, 2013, Addison-Wesley Professional