# Compare object oriented programming in Go with TypeScript

Ali Piriyaie


1. Type system in Go
   Type system in TypeScript
2. Class model in Go
   Class model in TypeScript
3. Inheritance in Go
   Inheritance in TypeScript
4. Polymorphism in Go
   Polymorphism in TypeScript

# Type system in Go

## Basic types

- string
- bool
- int8, uint8, byte, int16, uint16, int32, rune, uint32, int64, unit64, int, uint
- float32, float64
- complex64, complex128

## Composite types

- pointer
- struct
- function
- array, slice, map
- channel
- interface

# Type system in TypeScript

## Basic types

- boolean
- number
- string
- null
- undefined

## Composite types

- array, tuple, enum
- any
- object
- function
- interface

# Class model in Go

```
type dog struct {
    name string
    breed string
    weight float32
    age uint8
}

func (d dog) walk(){
    fmt.Printf("Dog %s walks. \n", d.name)
}

func (d *dog) addAge(){
    d.age++
    fmt.Printf(
    "Hurray! It`s %s's birthday. He/She is %v years
    old now.",
    d.name,
    d.age)
}
```

# Class model in Go

```
func NewDog(
    name string,
    breed string,
    weight float32,
    age uint8) *dog {
            dog := new(dog)
            dog.name = name
            dog.breed = breed
            dog.weight = weight
            dog.age = age
            return dog
}


// Create instance
digga := NewDog("Digga", "German Shepherd", 44.6, 4)
```

# Class model in TypeScript

```
class Dog {
    private name: string;
    private breed: string;
    private weight: number;
    private age: number;

    public walk(){
        console.log("Dog " + this.name + " walks.");
    }

    public addAge(){
          this.age++;
          console.log(
          "Hurray! It`s " + this.name +
          "'s birthday. He/She is " + this.age +
          " years old now.");
    }

}
```

# Class model in TypeScript

```
constructor(gender: string,
            name: string,
            breed: string,
            weight: number,
            age: number) {
                this.name = name;
                this.breed = breed;
                this.weight = weight;
                this.age = age;
}



// Create instance
var jax = new Dog("Jax", "Australian Shepherd", 24.6, 1);
```

# Inheritance in Go

```
type animal struct {
    kind string
    gender string
}

type dog struct {
    animal
    name string
    breed string
    weight float32
    age uint8
}

// Create instance
digga := dog{
    animal: animal{kind: "Dog", gender: "male"},
    name: "Digga",
    breed: "German Shepherd",
    weight: 44.3,
    age: 4}
```

# Inheritance in TypeScript

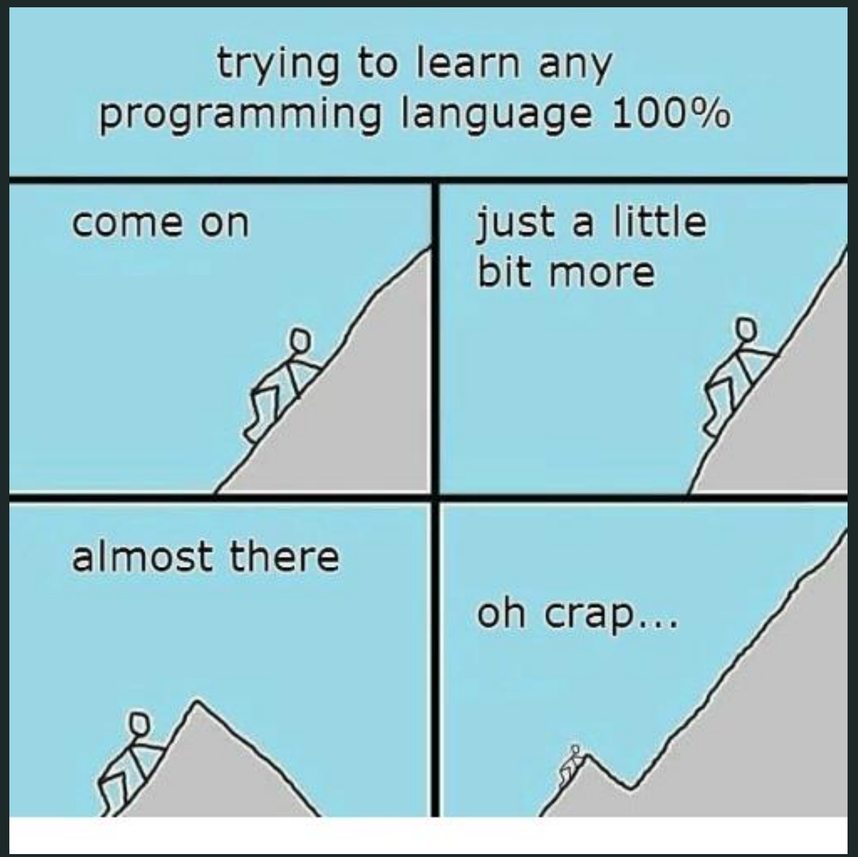
```
class Animal {
     private kind: string;
     private gender: string;

     constructor(kind: string, gender: string) {
          this.kind = kind;
          this.gender = gender;
     }
}

class Dog extends Animal {
     private name: string;
     private breed: string;
     private weight: number;
     private age: number;

     constructor(gender: string,
                 name: string,
                 breed: string,
                 weight: number,
                 age: number) {
                      super("Dog", gender);
                      this.name = name;
                      this.breed = breed;
                      this.weight = weight;
                      this.age = age;
     }
}
```

# Polymorphism in Go

```
type movable interface {
    walk()
}

type animal struct {
    kind string
    gender string
}

func (a animal) walk(){
    fmt.Println("Animal walks.");
}

type dog struct {
    animal
    name string
    breed string
    weight float32
    age uint8
}

func (d dog) walk(){
    fmt.Printf("Dog %s walks. \n", d.name)
}

func foo(m movable){
    m.walk()
}
```

# Polymorphism in TypeScript

```
class Animal implements moveable {
     private kind: string;
     private gender: string;

     public walk(): any{
          console.log("Animal walks.");
     }

     // constructor
}

class Dog implements moveable {
     private name: string;
     private breed: string;
     private weight: number;
     private age: number;

     public walk(): any{
          console.log("Dog " + this.name + " walks.");
     }

     // constructor
}

interface moveable {
     walk(): any;
}

function foo(m: moveable){
     m.walk();
}
```


trying to learn any programming language 100%
come on
just a little bit more
almost there
oh crap...