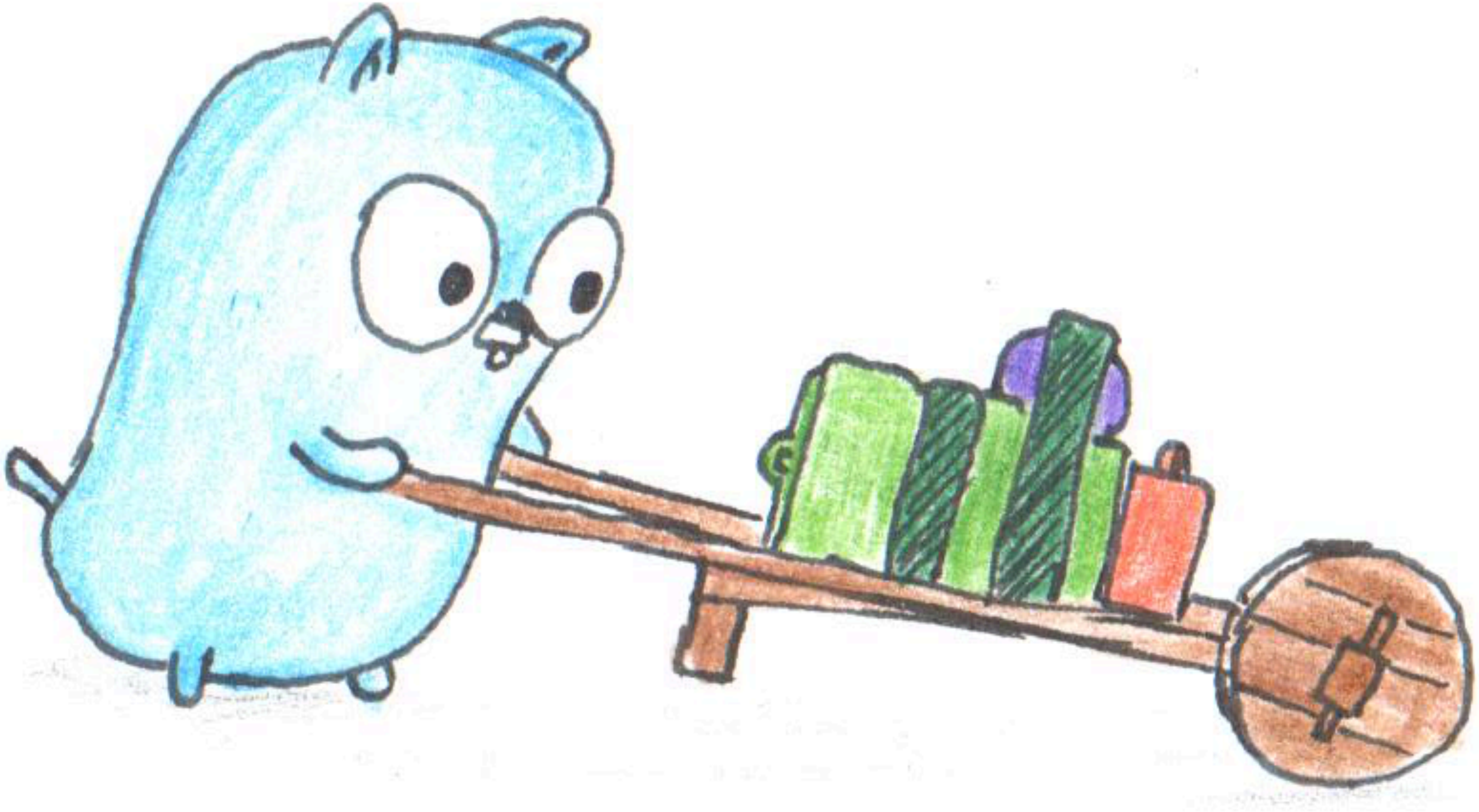


Go Part 2
Object Oriented Programing

Jin Hyun Kim

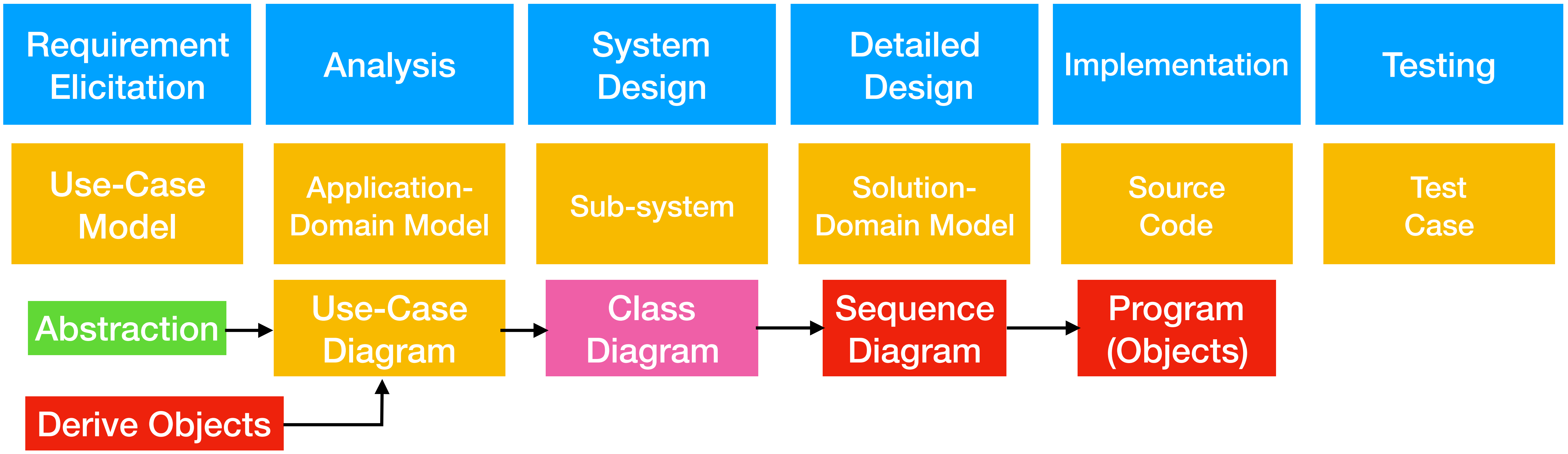
Fall, 2019

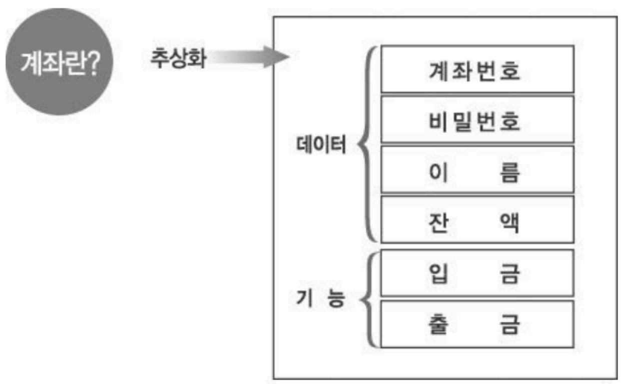
Contents

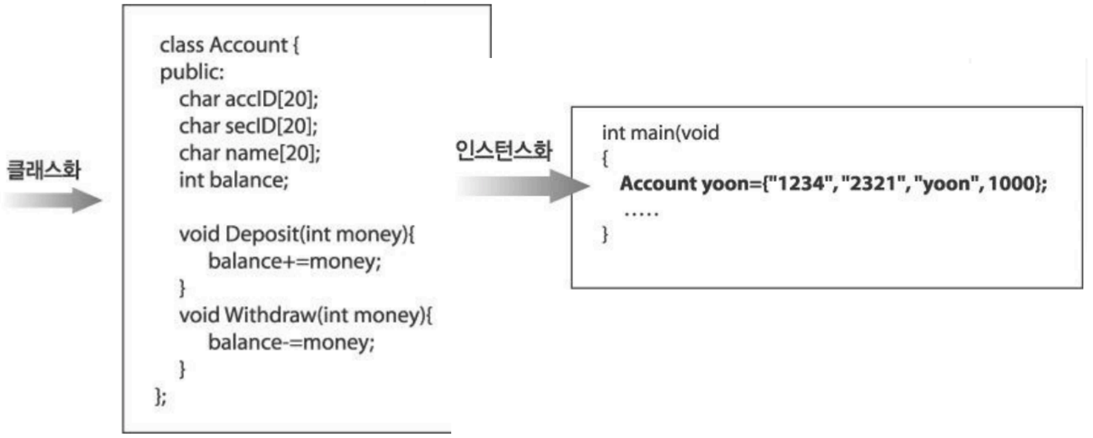
References

- https://code.egym.de/introduction-to-oop-in-golang-e4841a9c4e3e
- https://golangkorea.github.io/post/go-start/object-oriented/
- https://mingrammer.com/translation-go-and-oop/
- Alan A. A. Donovan, Brian W. Kernighan. “The Go Programming Language”.
- Effective Go. https://golang.org/doc/effective_go.html
- https://golang.org/

Review







```
class Account {
public:
   char accID[20];
   char secID[20];
   char name[20];
   int balance;

   void Deposit(int money){
       balance+=money;
   }
   void Withdraw(int money){
       balance-=money;
   }
};
```

```
int main(void
{
  Account yoon={"1234","2321","yoon",1000};
   .....
}
```

Review

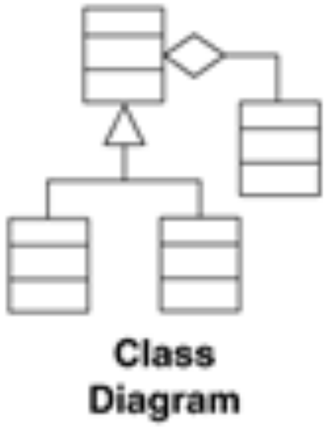
Analysis

Application-Domain Model

Use Case Model

Requirement Elicitation

Use-Case Model

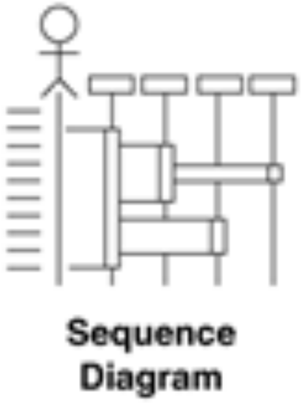


System Design

Sub-system

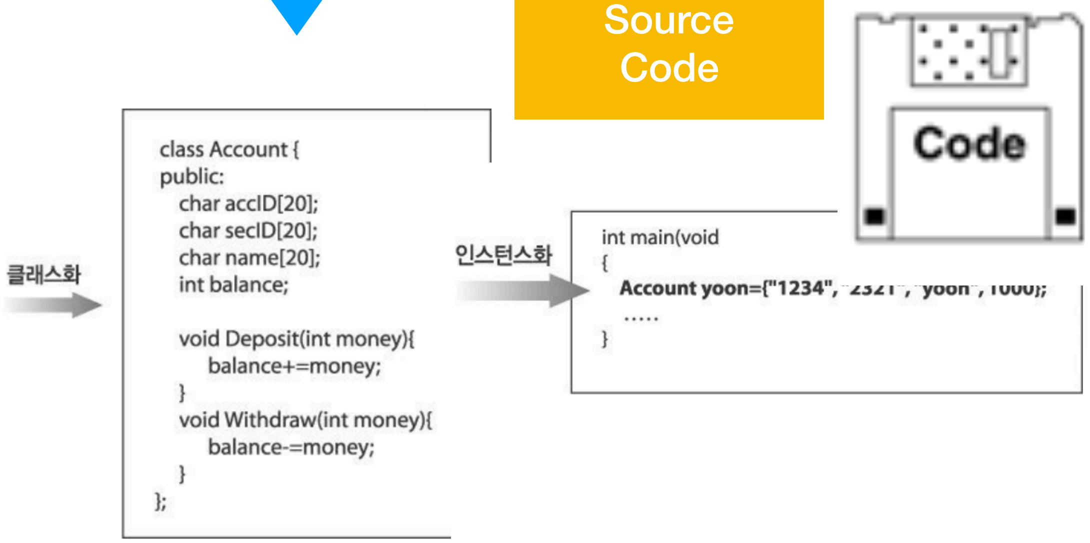
Detailed Design

Solution-Domain Model



Implementation

Source Code

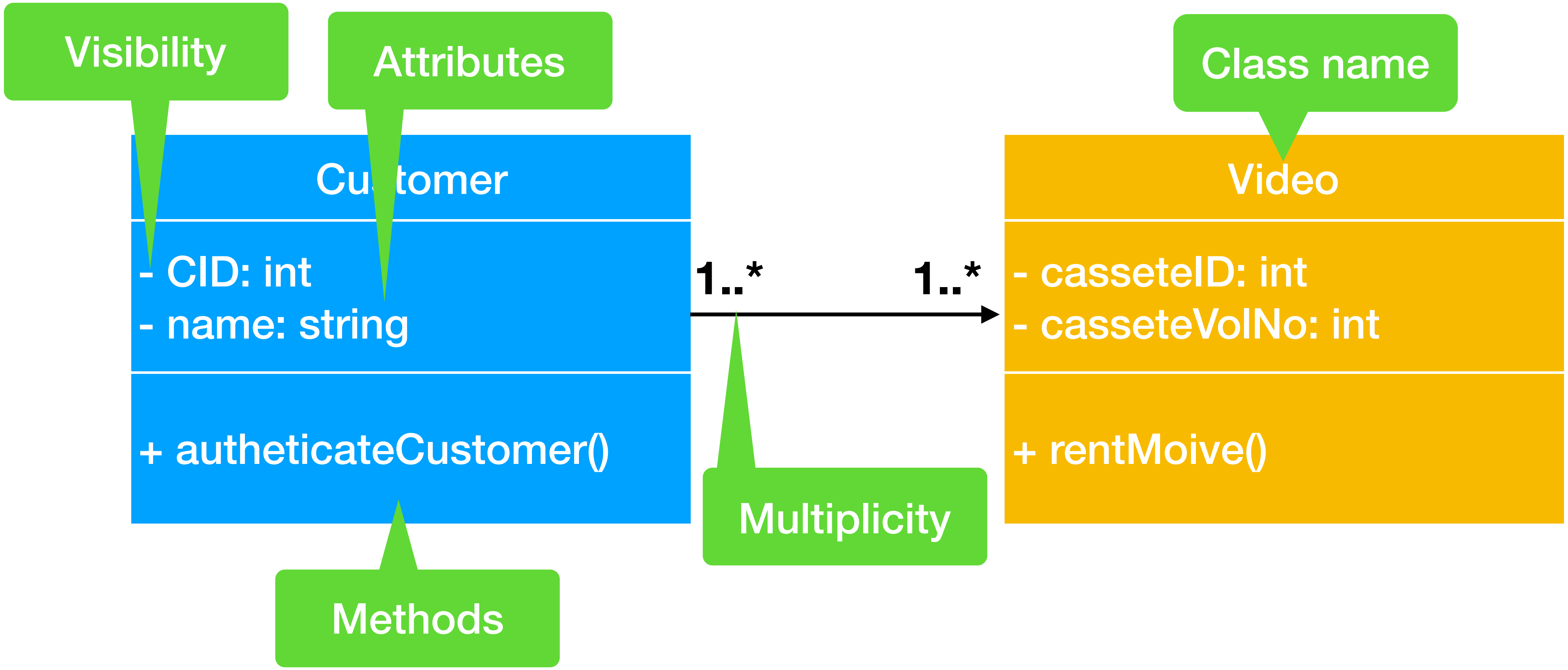


```
class Account {
public:
    char accID[20];
    char secID[20];
    char name[20];
    int balance;

    void Deposit(int money){
        balance+=money;
    }
    void Withdraw(int money){
        balance-=money;
    }
};
```

```
int main(void
{
   Account yoon={"1234", "2321", "yoon", 1000};
   .....
}
```

Review

- Class



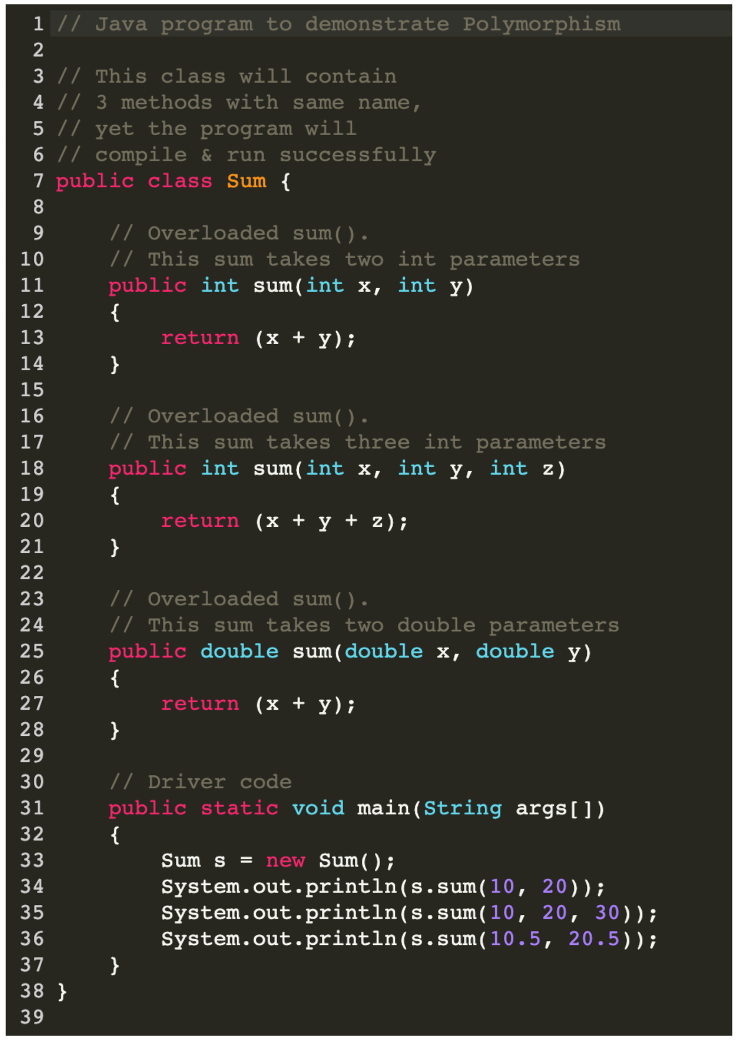
Review

- Encapsulation

Visibility	Symbol	Accesible to
Public	+	All objects within your system
Protected	#	Instances of the implementing class and its subclasses.
Private	-	Instances of the implementing class.

- Inheritance (vs Generalization)

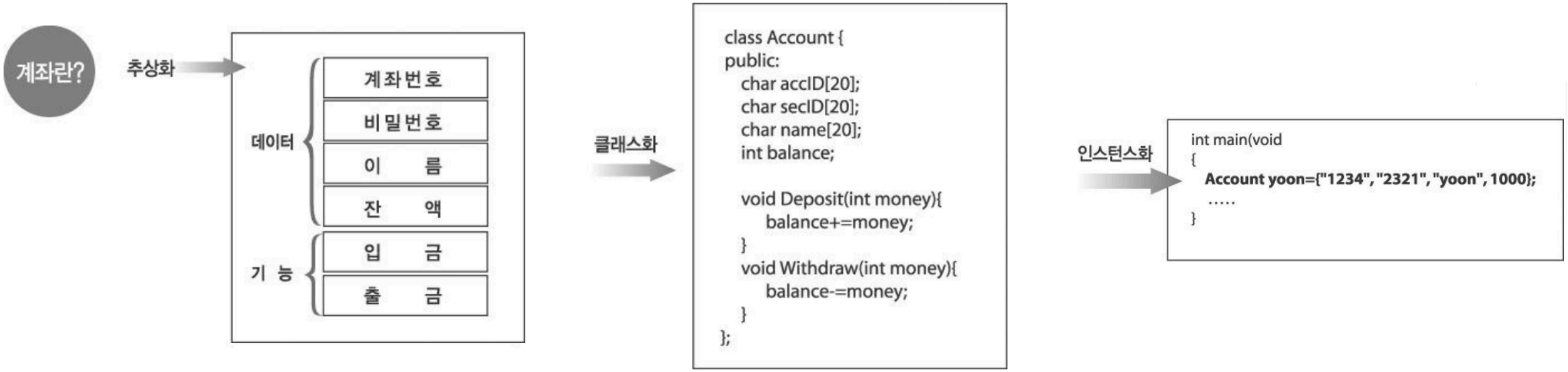
- Polymorphism



```java
// Java program to demonstrate Polymorphism

// This class will contain
// 3 methods with same name,
// yet the program will
// compile & run successfully
public class Sum {

    // Overloaded sum().
    // This sum takes two int parameters
    public int sum(int x, int y)
    {
        return (x + y);
    }

    // Overloaded sum().
    // This sum takes three int parameters
    public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }

    // Overloaded sum().
    // This sum takes two double parameters
    public double sum(double x, double y)
    {
        return (x + y);
    }

    // Driver code
    public static void main(String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}

```

Review

- Abstraction to Object

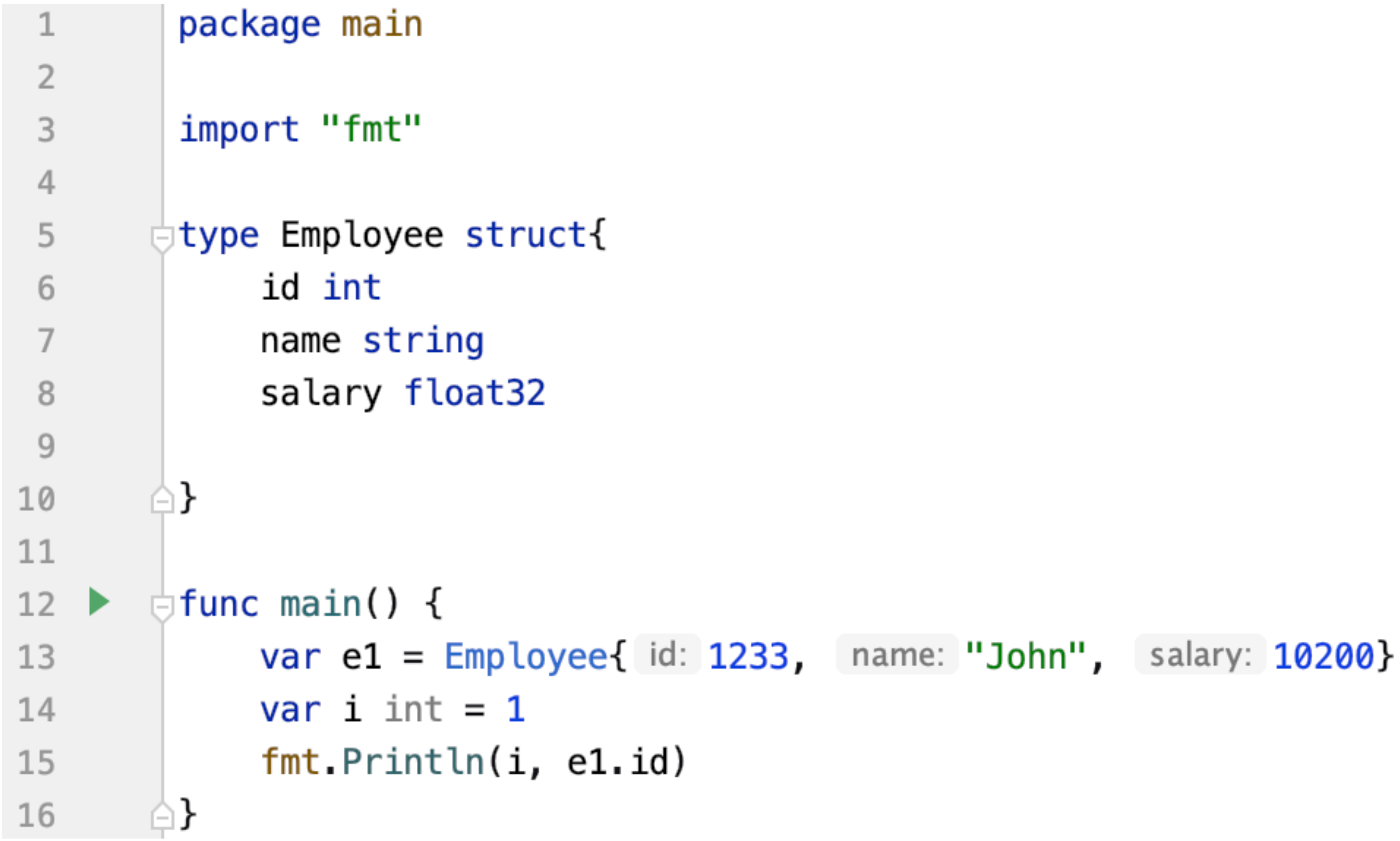


```
class Account {
public:
   char accID[20];
   char secID[20];
   char name[20];
   int balance;

   void Deposit(int money){
        balance+=money;
   }
   void Withdraw(int money){
        balance-=money;
   }
};
```

```
int main(void
{
   Account yoon={"1234", "2321", "yoon", 1000};
   .....
}
```

Review

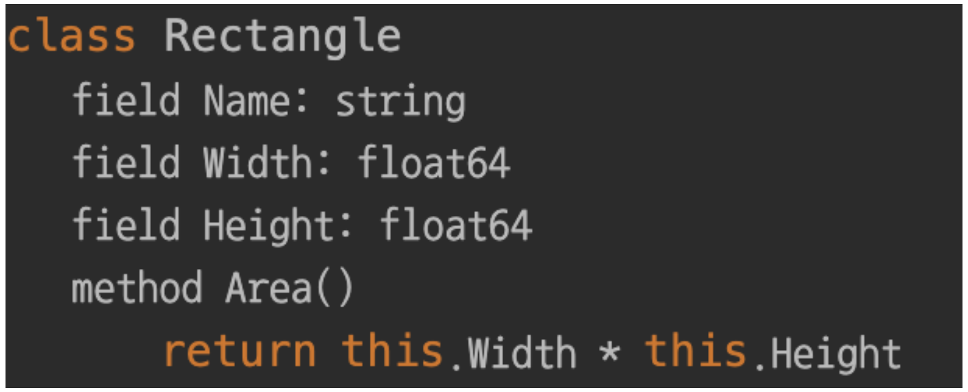
- Go’s Struct



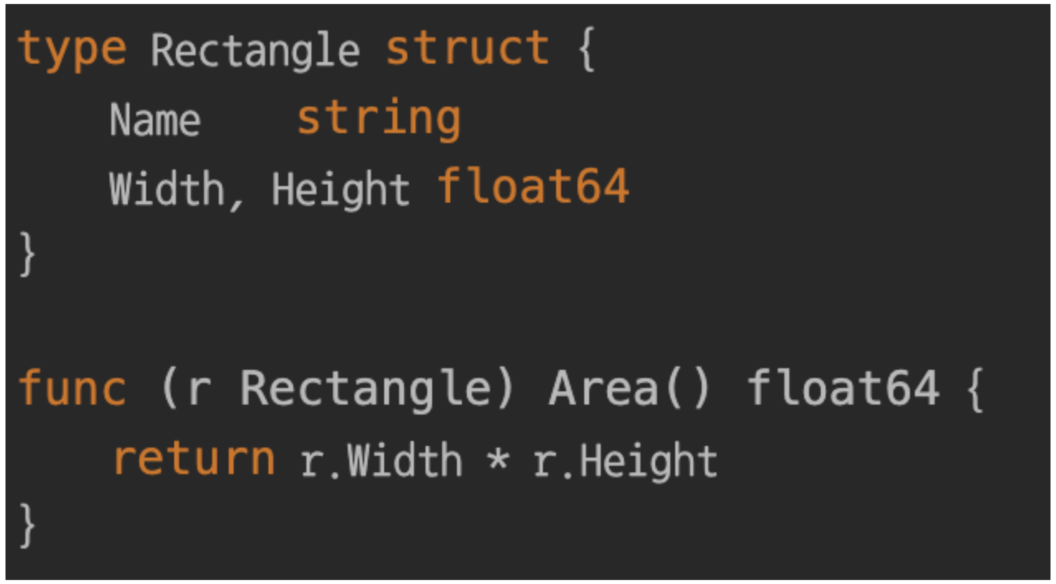
```go
package main

import "fmt"

type Employee struct{
    id int
    name string
    salary float32

}

func main() {
    var e1 = Employee{ id: 1233,  name: "John",  salary: 10200}
    var i int = 1
    fmt.Println(i, e1.id)
}
```

Methods

- Class vs Method



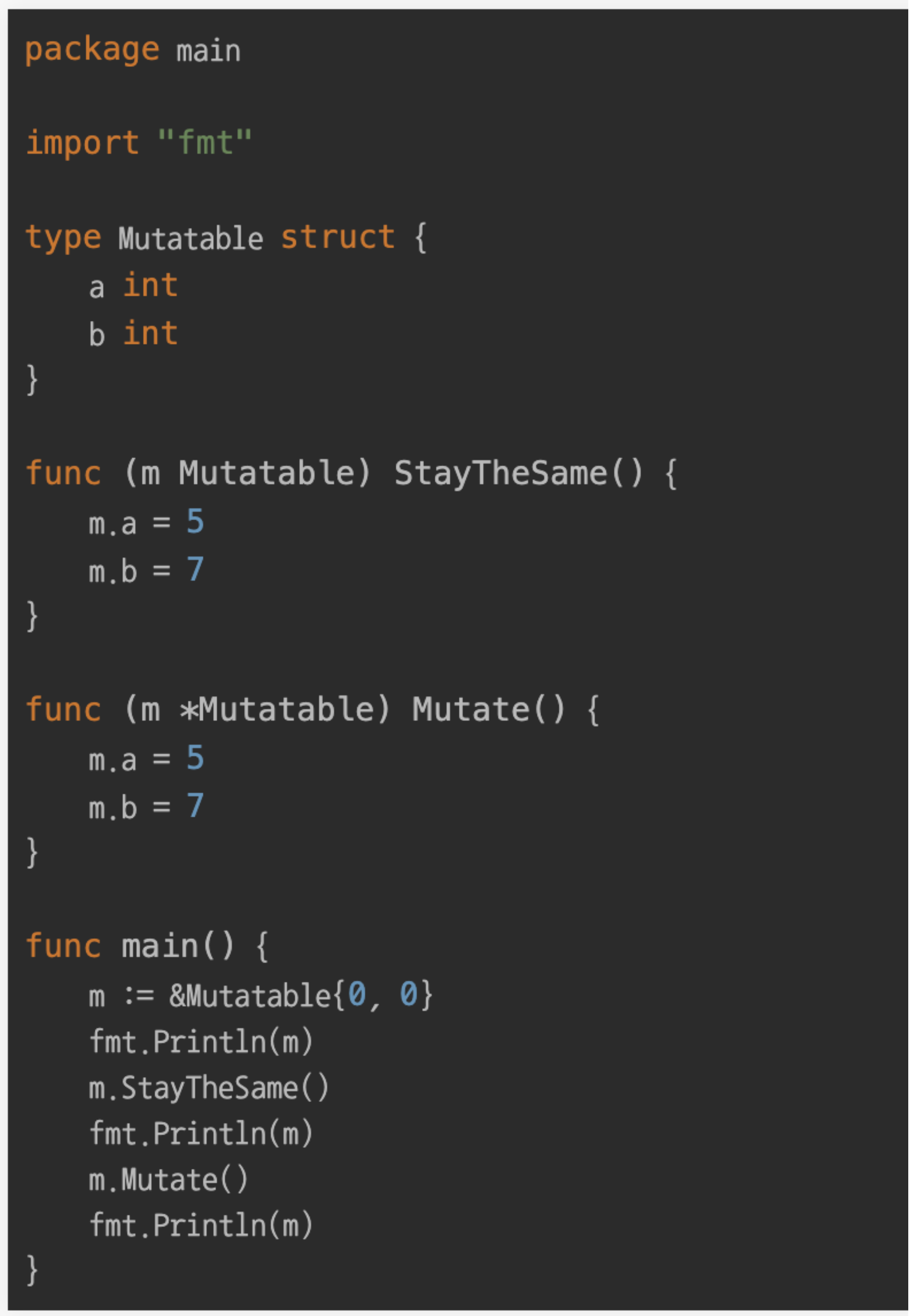
```
class Rectangle
   field Name: string
   field Width: float64
   field Height: float64
   method Area()
       return this.Width * this.Height
```



```
type Rectangle struct {
    Name    string
    Width, Height float64
}

func (r Rectangle) Area() float64 {
    return r.Width * r.Height
}
```

Methods

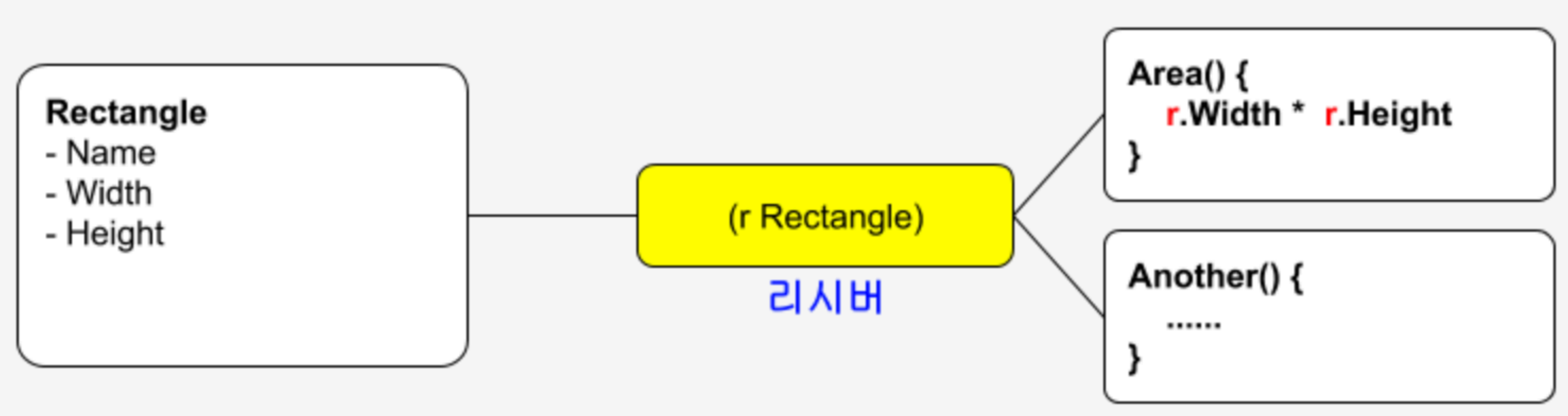
- Value vs Pointer receiver



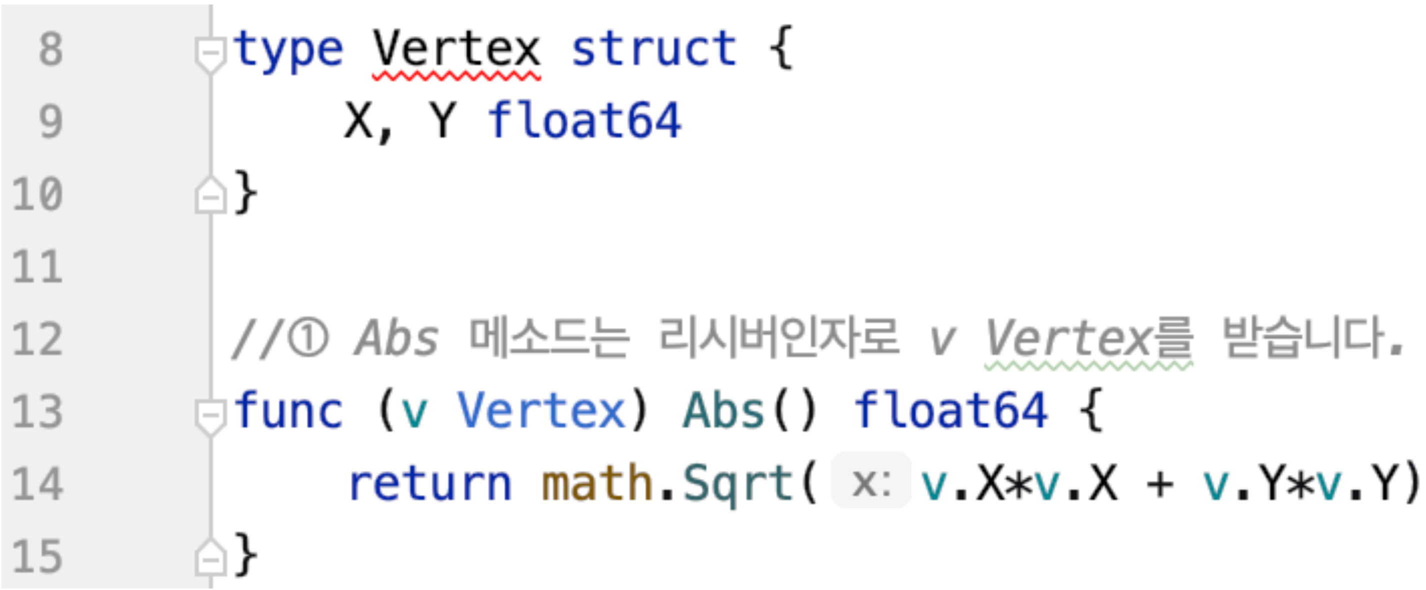
```go
package main

import "fmt"

type Mutatable struct {
    a int
    b int
}

func (m Mutatable) StayTheSame() {
    m.a = 5
    m.b = 7
}

func (m *Mutatable) Mutate() {
    m.a = 5
    m.b = 7
}

func main() {
    m := &Mutatable{0, 0}
    fmt.Println(m)
    m.StayTheSame()
    fmt.Println(m)
    m.Mutate()
    fmt.Println(m)
}
```

Methods

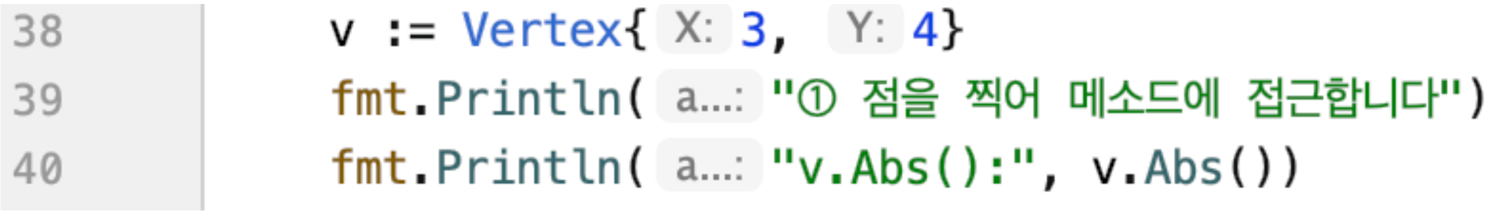
- The method of classical classes is implemented with a function with **receiver** in Go



Methods



```
type Vertex struct {
    X, Y float64
}

//① Abs 메소드는 리시버인자로 v Vertex를 받습니다.
func (v Vertex) Abs() float64 {
    return math.Sqrt( x: v.X*v.X + v.Y*v.Y)
}
```



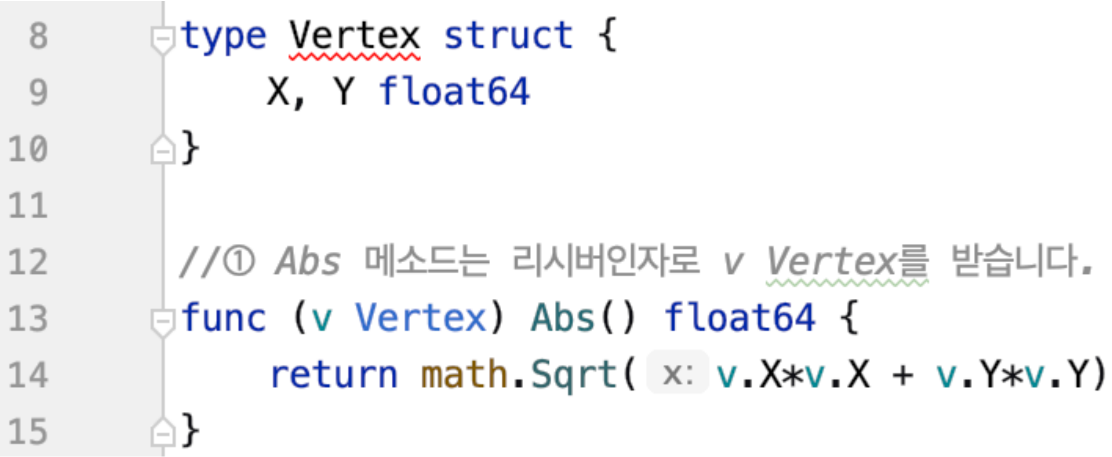
```
    v := Vertex{ X: 3,  Y: 4}
    fmt.Println( a...: "① 점을 찍어 메소드에 접근합니다")
    fmt.Println( a...: "v.Abs():", v.Abs())
```

method-lab-02.go 참조

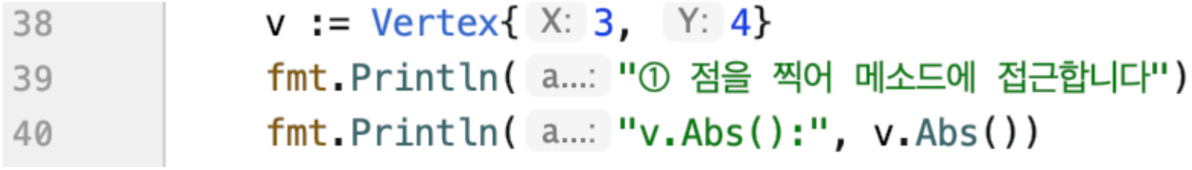
Methods

- How to modify variables through methods?

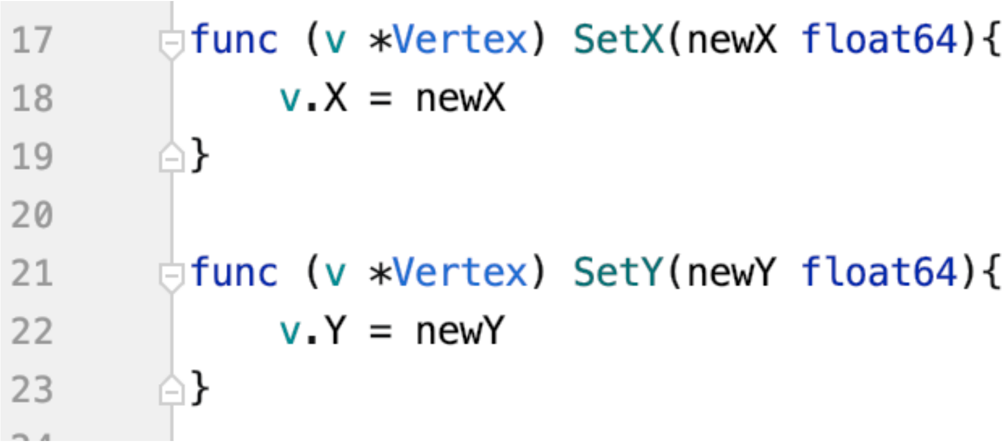
- Value receiver
- Pointer receiver



```
type Vertex struct {
	X, Y float64
}

//① Abs 메소드는 리시버인자로 v Vertex를 받습니다.
func (v Vertex) Abs() float64 {
	return math.Sqrt( x: v.X*v.X + v.Y*v.Y)
}
```



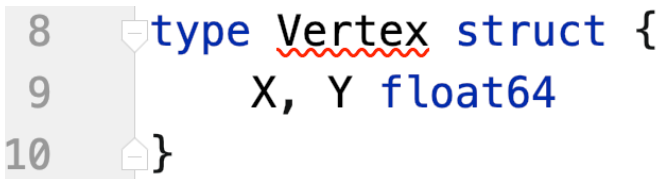
```
	v := Vertex{ X: 3,  Y: 4}
	fmt.Println( a...: "① 점을 찍어 메소드에 접근합니다")
	fmt.Println( a...: "v.Abs():", v.Abs())
```



```
func (v *Vertex) SetX(newX float64){
	v.X = newX
}

func (v *Vertex) SetY(newY float64){
	v.Y = newY
}
```

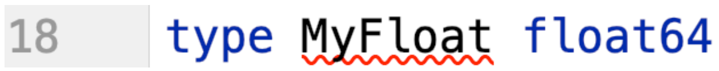
Methods

- Value receiver in struct



```go
type Vertex struct {
	X, Y float64
}
```

- Value receiver in type

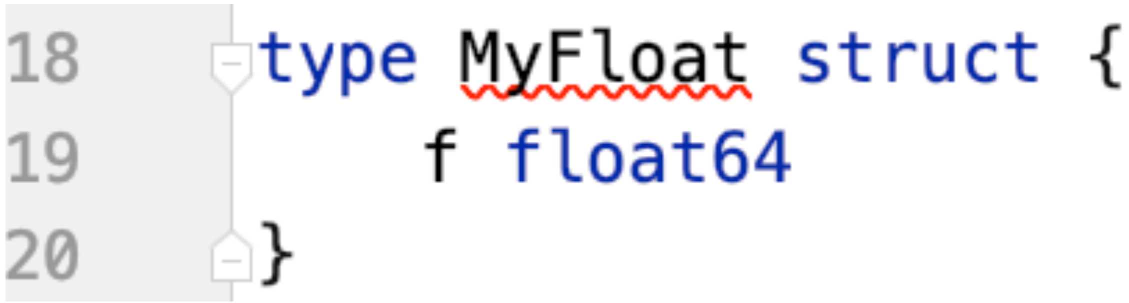


```go
type MyFloat float64
```

method-lab-02.go 참조

Self-Exercise

- 다음과 같은 struct receiver를 이용하여 method-lab-02.go와 동일한 행위를 하도록 수정하라 (주어진 시간 10분)



```go
type MyFloat struct {
	f float64
}
```

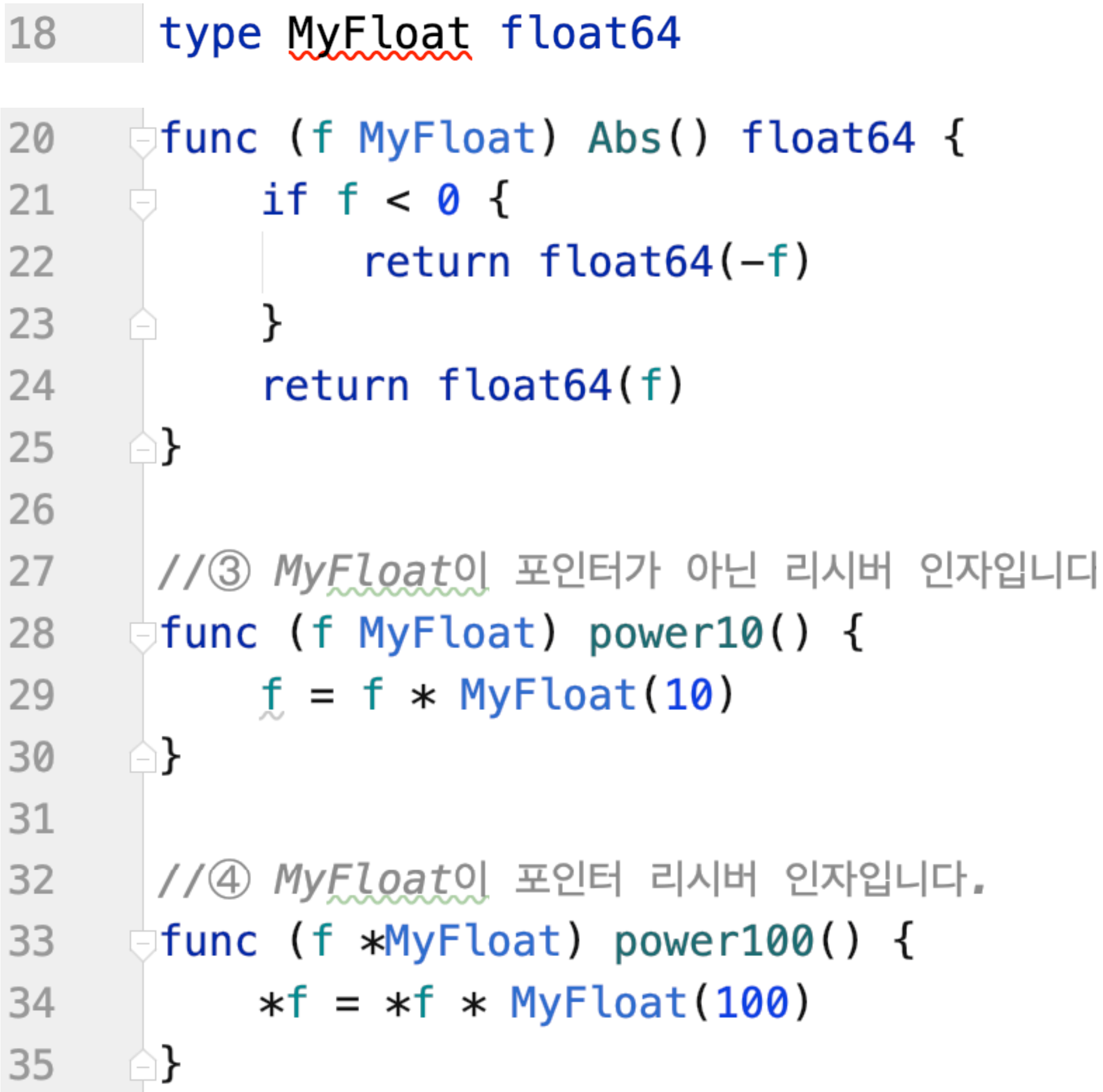
- 10분 뒤에 모든 작업을 중지하고 자신의 파트너와 자리를 바꾸어서 파트너의 코드를 이어서 작성하라. 이때 파트너와 상의하여 각자의 코드를 완성시키라.

Go OOP

- Lab - ABC (class-lab-02-ABC.go)
    - Create ABC struct
    - Create methods for ABC receiver
    - Let ABCD inherit ABC
    - Create a method for ABCD

Methods

- Struct receiver



```go
type MyFloat float64

func (f MyFloat) Abs() float64 {
	if f < 0 {
		return float64(-f)
	}
	return float64(f)
}

//③ MyFloat이 포인터가 아닌 리시버 인자입니다
func (f MyFloat) power10() {
	f = f * MyFloat(10)
}

//④ MyFloat이 포인터 리시버 인자입니다.
func (f *MyFloat) power100() {
	*f = *f * MyFloat(100)
}
```

Question:
f와 MyFloat(10) 의 의미의 차이는?

OOP in Go

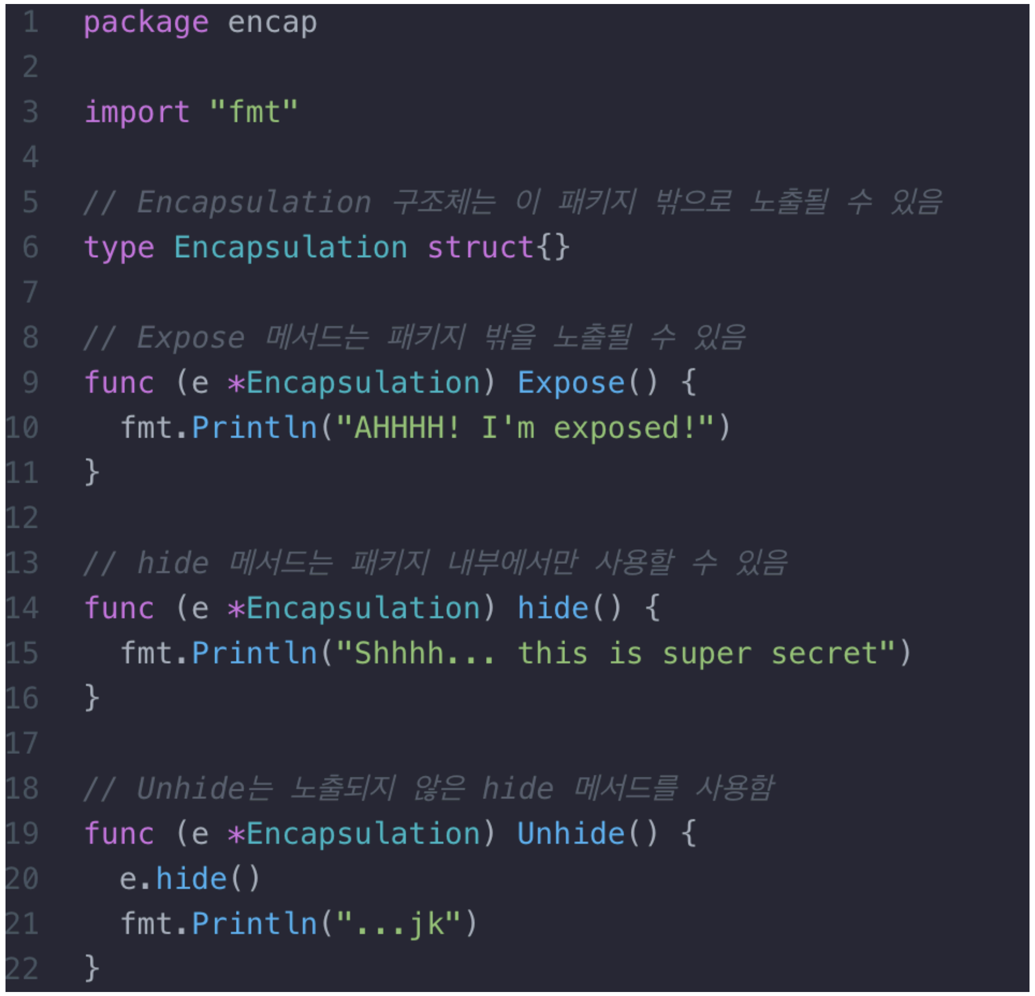
Class OOP	GO OOP
Encapsulation	Packages
Inheritance	Composition
Polymophism	Interface
Abstraction	Embeding

Encapsulation

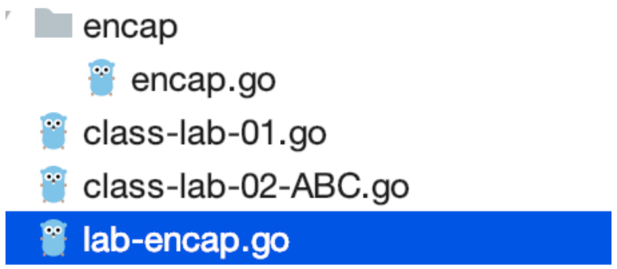
- In Go lang,
  - In terms of package, encapsulation is implemented by **package**
  - Package scope:
    - Public (Exported), Private (Unexported)

Encapsulation

- Encapsulated by package

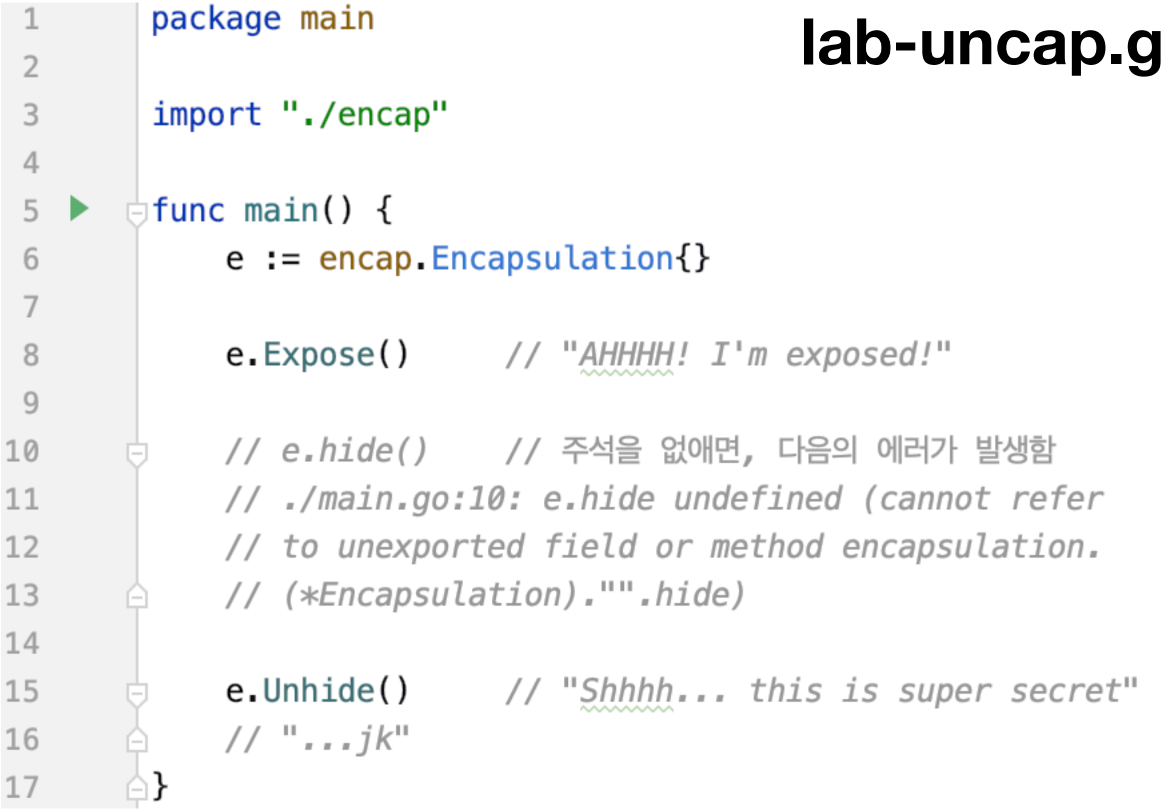


```go
package encap

import "fmt"

// Encapsulation 구조체는 이 패키지 밖으로 노출될 수 있음
type Encapsulation struct{}

// Expose 메서드는 패키지 밖을 노출될 수 있음
func (e *Encapsulation) Expose() {
  fmt.Println("AHHHH! I'm exposed!")
}

// hide 메서드는 패키지 내부에서만 사용할 수 있음
func (e *Encapsulation) hide() {
  fmt.Println("Shhhh... this is super secret")
}

// Unhide는 노출되지 않은 hide 메서드를 사용함
func (e *Encapsulation) Unhide() {
  e.hide()
  fmt.Println("...jk")
}
```



```
encap
    encap.go
class-lab-01.go
class-lab-02-ABC.go
lab-encap.go
```

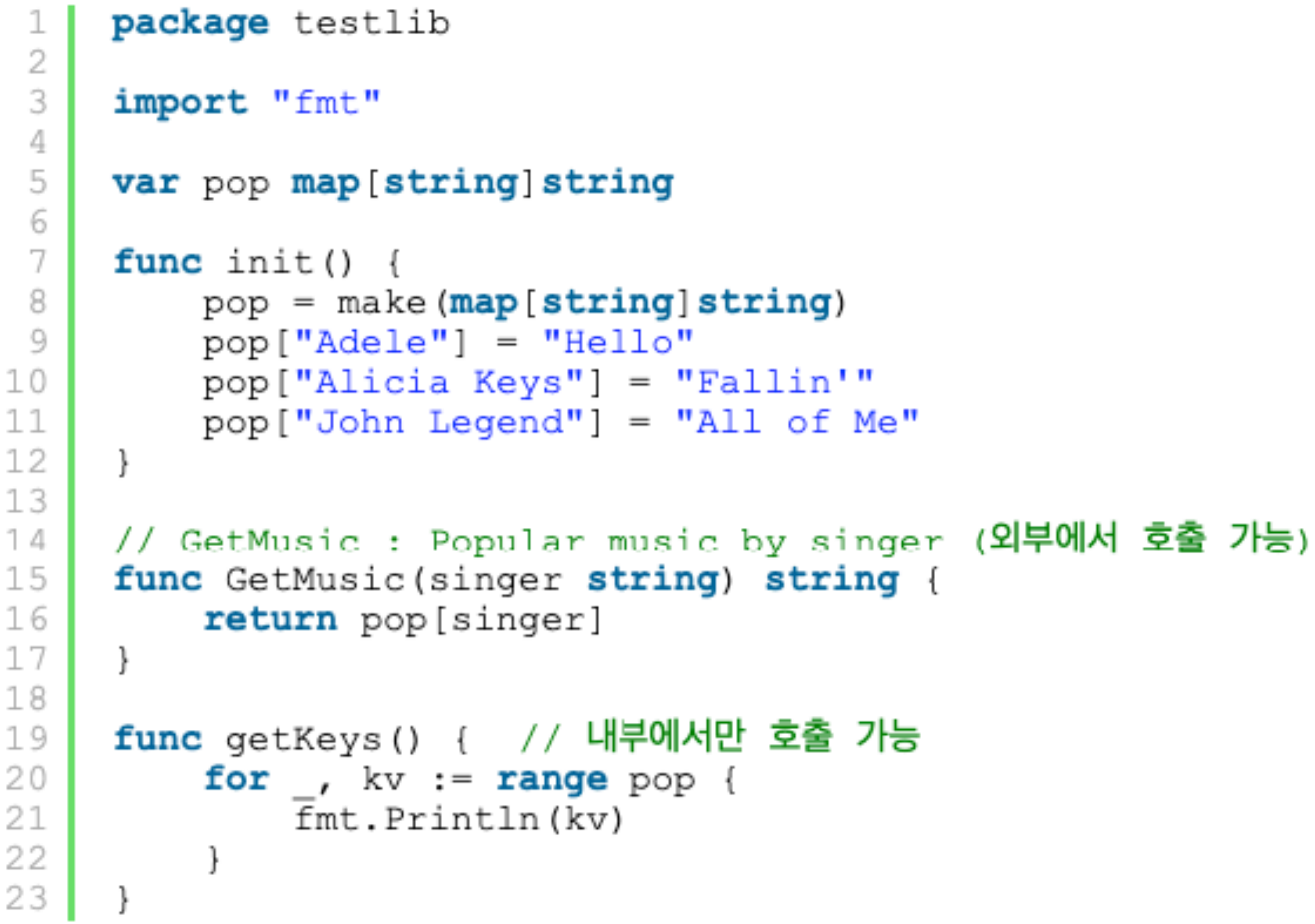
lab-uncap.go



```go
package main

import "./encap"

func main() {
    e := encap.Encapsulation{}

    e.Expose()     // "AHHHH! I'm exposed!"

    // e.hide()    // 주석을 없애면, 다음의 에러가 발생함
    // ./main.go:10: e.hide undefined (cannot refer
    // to unexported field or method encapsulation.
    // (*Encapsulation)."".hide)

    e.Unhide()     // "Shhhh... this is super secret"
    // "...jk"
}
```

Package

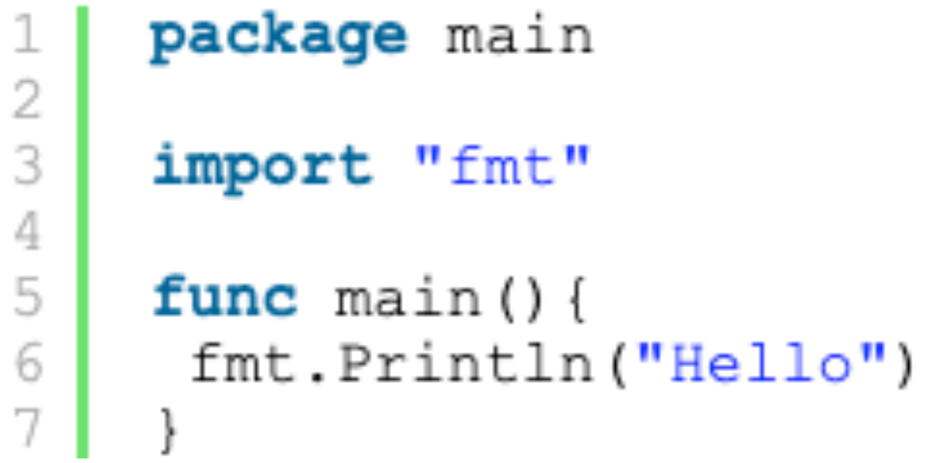
- Type



```go
package testlib

import "fmt"

var pop map[string]string

func init() {
	pop = make(map[string]string)
	pop["Adele"] = "Hello"
	pop["Alicia Keys"] = "Fallin'"
	pop["John Legend"] = "All of Me"
}

// GetMusic : Popular music by singer (외부에서 호출 가능)
func GetMusic(singer string) string {
	return pop[singer]
}

func getKeys() {  // 내부에서만 호출 가능
	for _, kv := range pop {
		fmt.Println(kv)
	}
}
```

Package

- Main package



```go
package main

import "fmt"

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

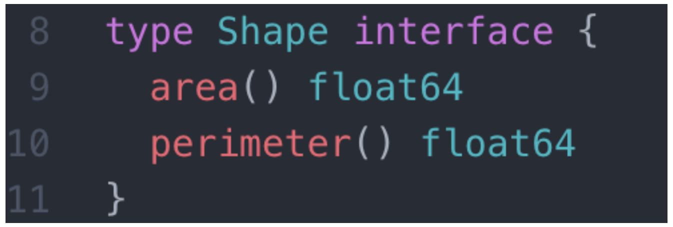
Package

- Locations
  - Standard library (package) in
    - GOROOT/pkg
  - User-defined package
    - GOPATH/src
    - GOPATH/pkg
    - GOPATH/userpackage

Polymorphism

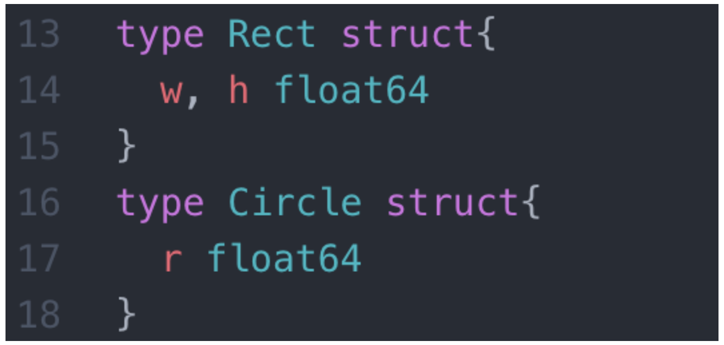
- In Golang, polymorphism is implemented by interface.

1. **Declare interface**



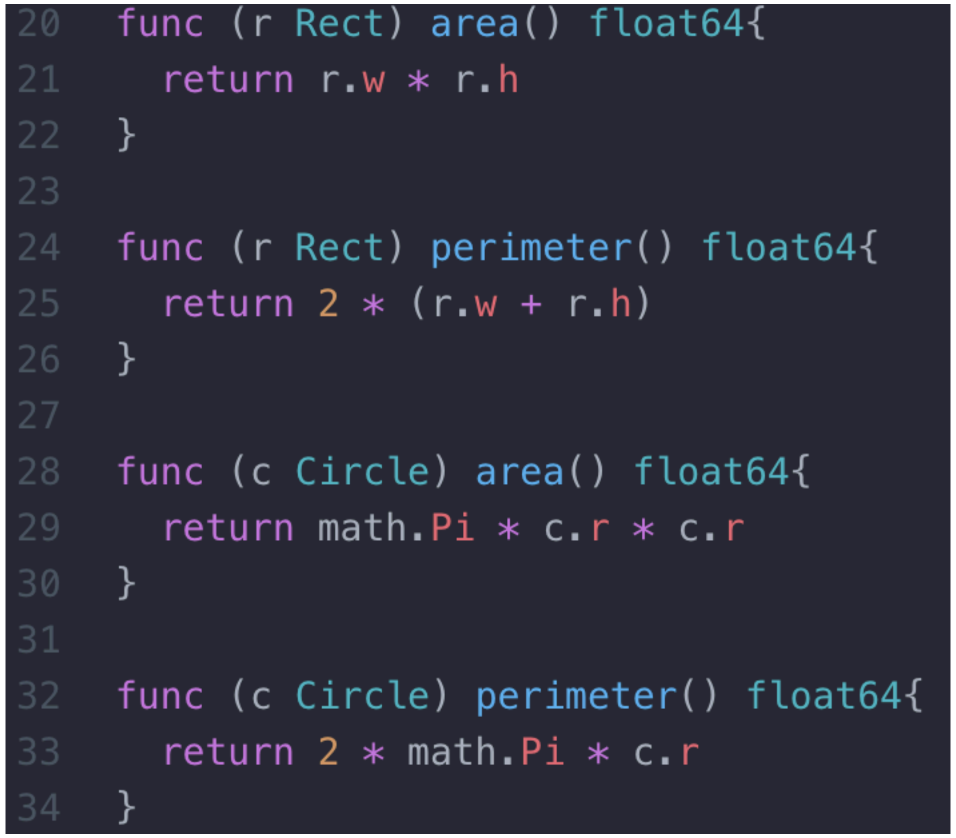
```go
type Shape interface {
  area() float64
  perimeter() float64
}
```

2. **Declare struct**



```go
type Rect struct{
  w, h float64
}
type Circle struct{
  r float64
}
```

3. **Implement interfaces**

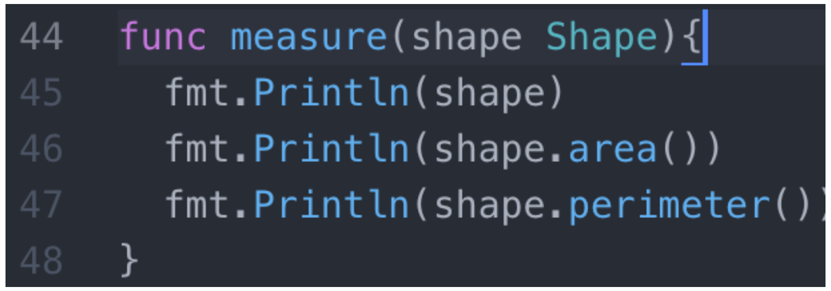


```go
func (r Rect) area() float64{
  return r.w * r.h
}

func (r Rect) perimeter() float64{
  return 2 * (r.w + r.h)
}

func (c Circle) area() float64{
  return math.Pi * c.r * c.r
}

func (c Circle) perimeter() float64{
  return 2 * math.Pi * c.r
}
```

Polymorphism

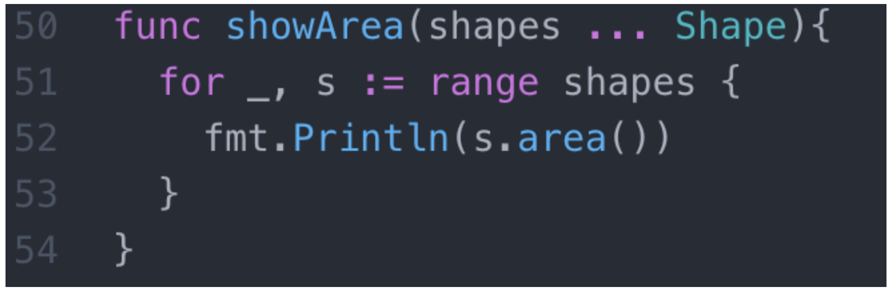
- In Golang, polymorphism is implemented by interface.

4. A. Use interface

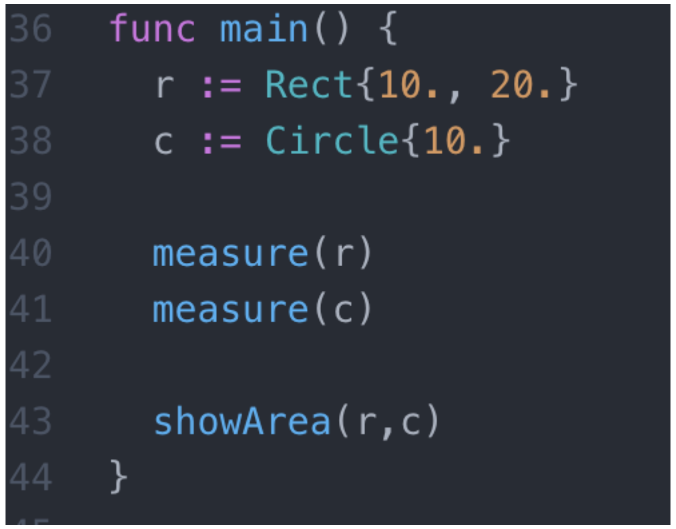


```go
func measure(shape Shape){
  fmt.Println(shape)
  fmt.Println(shape.area())
  fmt.Println(shape.perimeter())
}
```

4. B. Use interface



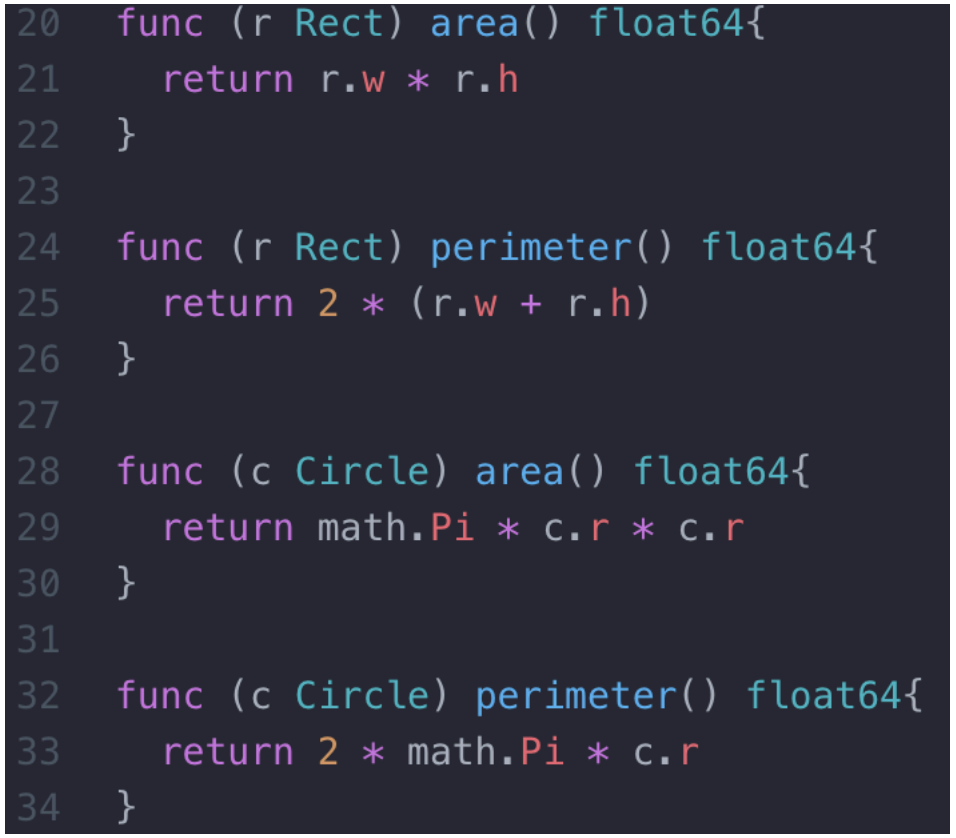
```go
func showArea(shapes ... Shape){
  for _, s := range shapes {
    fmt.Println(s.area())
  }
}
```



```go
func main() {
  r := Rect{10., 20.}
  c := Circle{10.}

  measure(r)
  measure(c)

  showArea(r,c)
}
```

Pop Question

- What if some implementation of interfaces by structs is removed? (만약 어떤 인터페이스 구현이 없어지면 어떤 문제가 생길까?)



```
func (r Rect) area() float64{
  return r.w * r.h
}

func (r Rect) perimeter() float64{
  return 2 * (r.w + r.h)
}

func (c Circle) area() float64{
  return math.Pi * c.r * c.r
}

func (c Circle) perimeter() float64{
  return 2 * math.Pi * c.r
}
```

Wrap-up

Class OOP	GO OOP
Encapsulation	Packages
Inheritance	Composition
Polymophism	Interface
Abstraction	Embeding