

Go Part 2

Object Oriented Programing

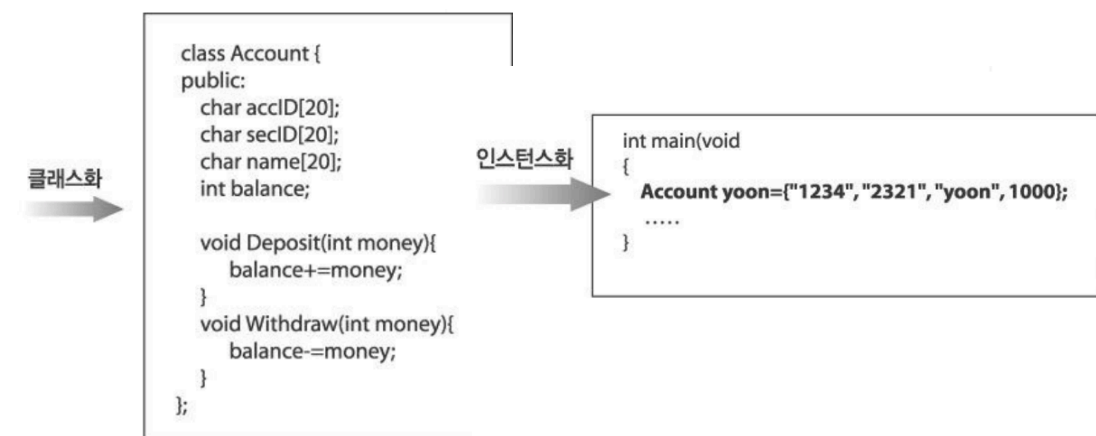
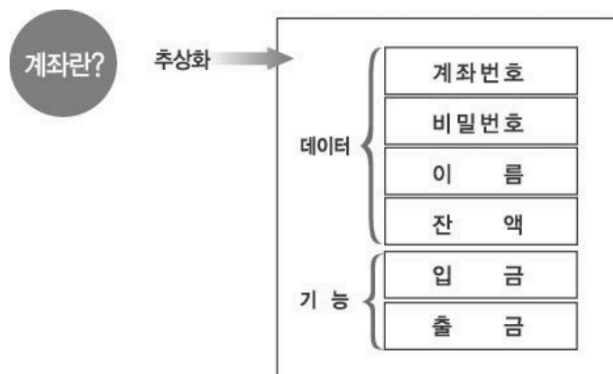
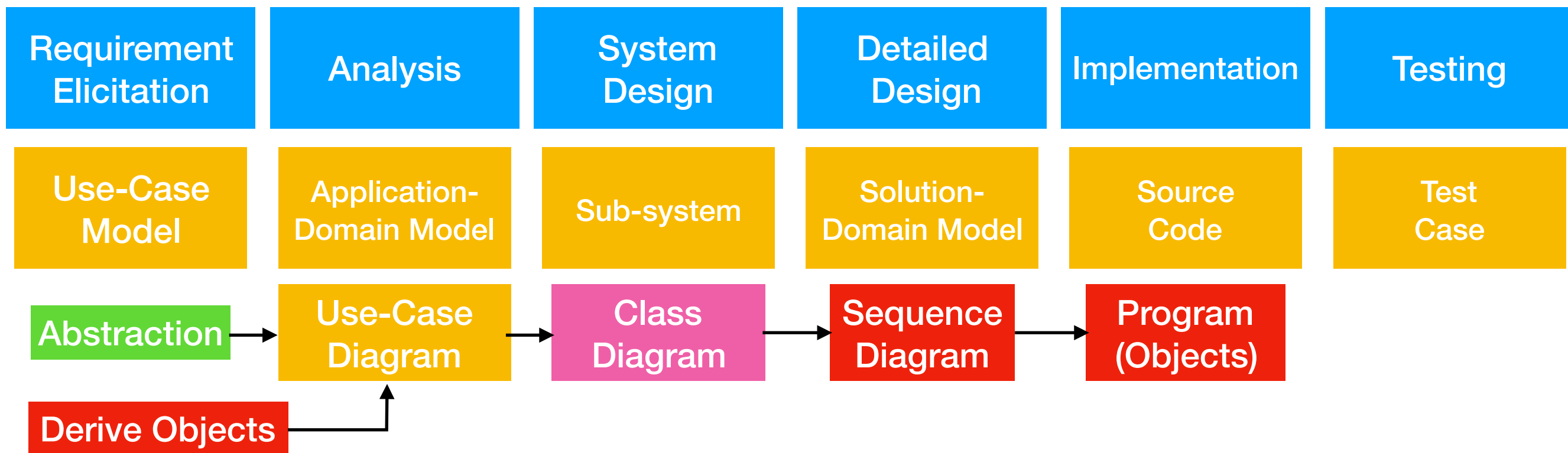
Jin Hyun Kim
Fall, 2019

Contents

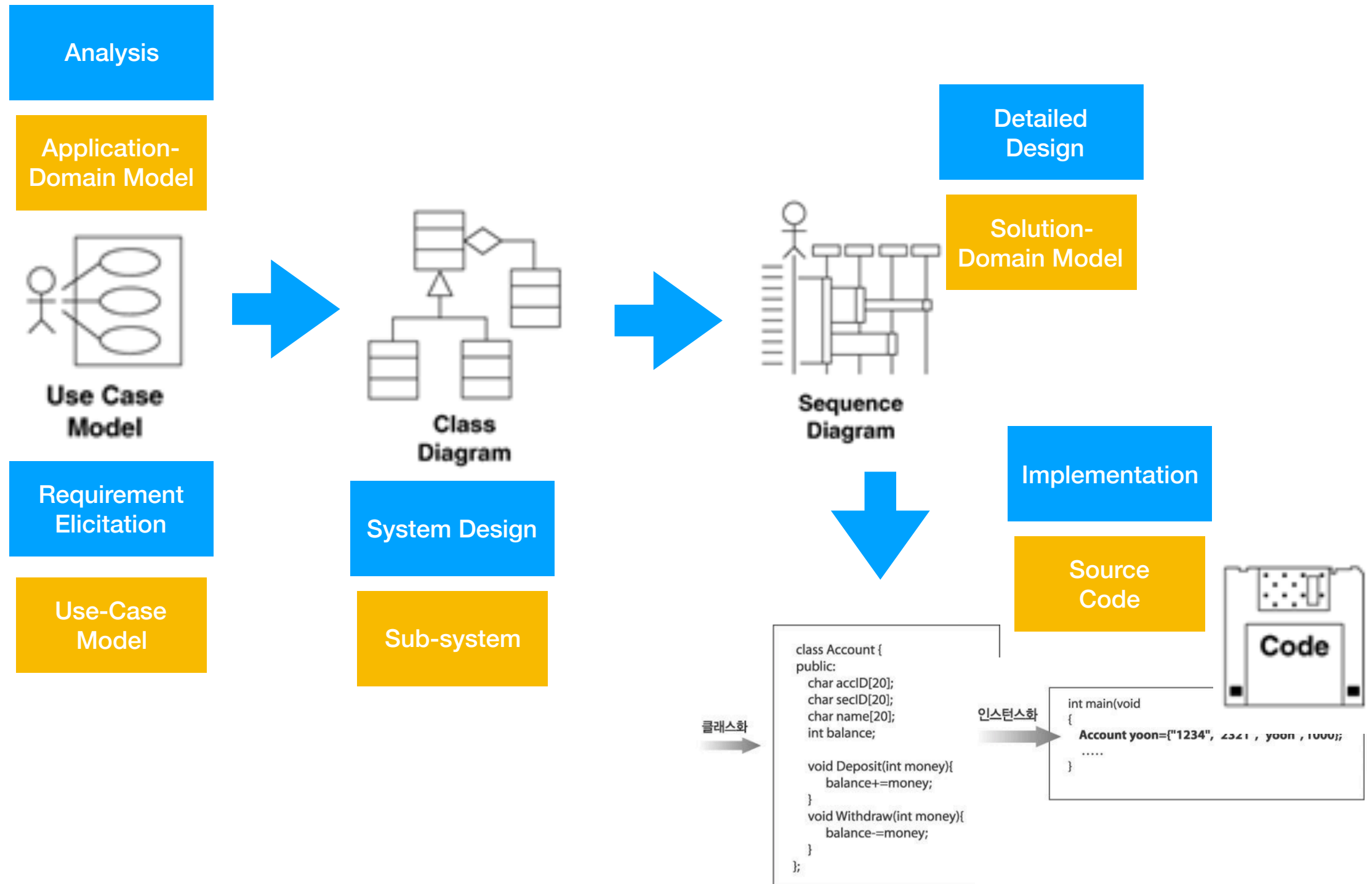
References

- <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

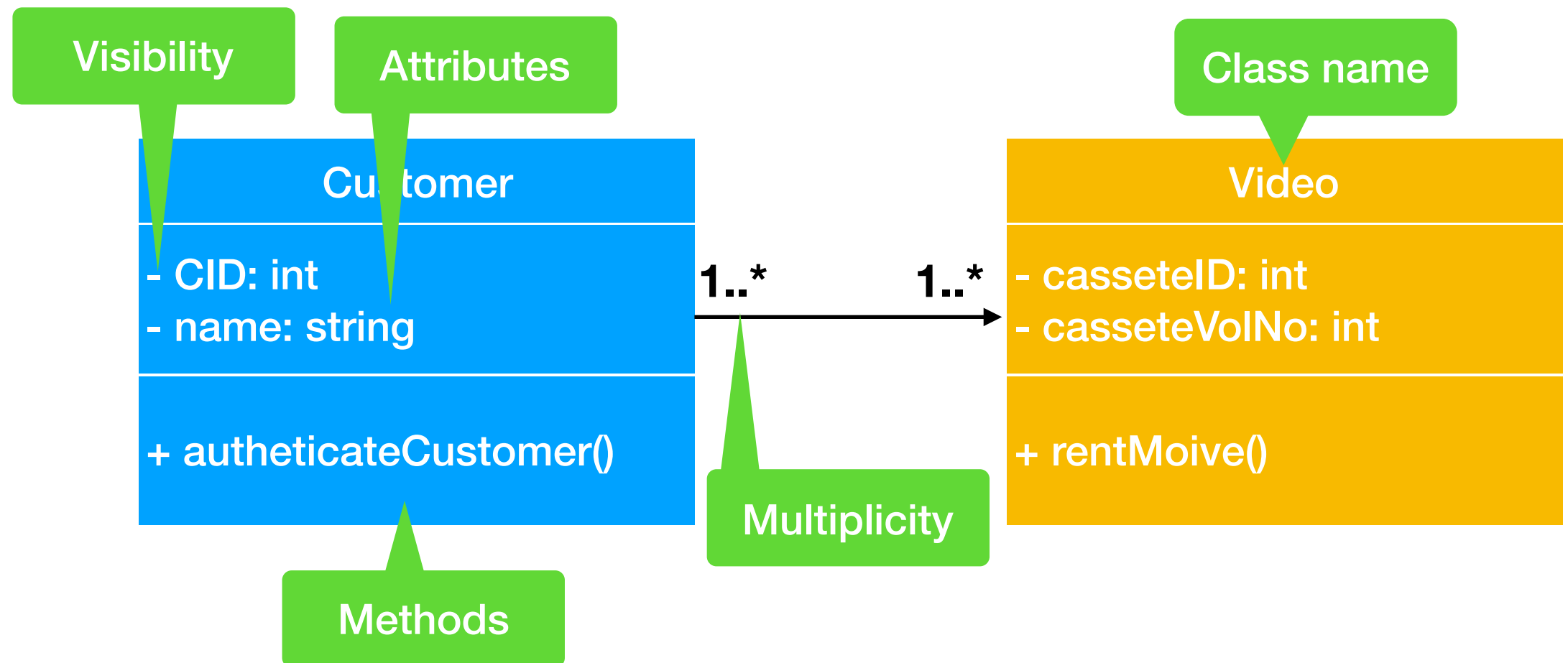


Review



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.

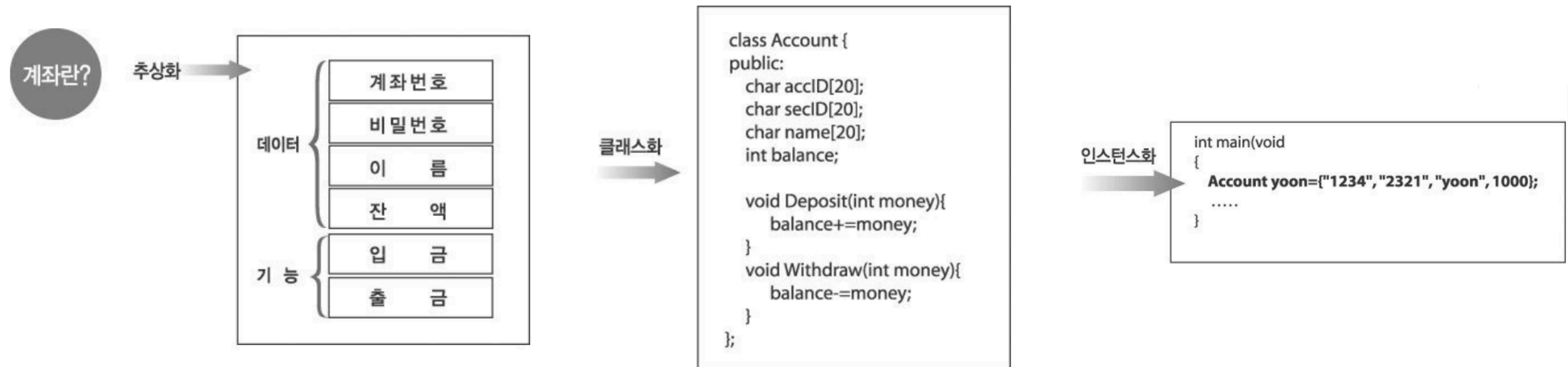
- Inheritance (vs Generalization)

- Polymorphism

```
1 // Java program to demonstrate Polymorphism
2
3 // This class will contain
4 // 3 methods with same name,
5 // yet the program will
6 // compile & run successfully
7 public class Sum {
8
9     // Overloaded sum().
10    // This sum takes two int parameters
11    public int sum(int x, int y)
12    {
13        return (x + y);
14    }
15
16    // Overloaded sum().
17    // This sum takes three int parameters
18    public int sum(int x, int y, int z)
19    {
20        return (x + y + z);
21    }
22
23    // Overloaded sum().
24    // This sum takes two double parameters
25    public double sum(double x, double y)
26    {
27        return (x + y);
28    }
29
30    // Driver code
31    public static void main(String args[])
32    {
33        Sum s = new Sum();
34        System.out.println(s.sum(10, 20));
35        System.out.println(s.sum(10, 20, 30));
36        System.out.println(s.sum(10.5, 20.5));
37    }
38 }
39
```

Review

- Abstraction to Object



Review

- Go's Struct

```
1 package main
2
3 import "fmt"
4
5 type Employee struct{
6     id int
7     name string
8     salary float32
9
10 }
11
12 func main() {
13     var e1 = Employee{ id: 1233, name: "John", salary: 10200}
14     var i int = 1
15     fmt.Println(i, e1.id)
16 }
```

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

```
package main

import "fmt"

type Mutable struct {
    a int
    b int
}

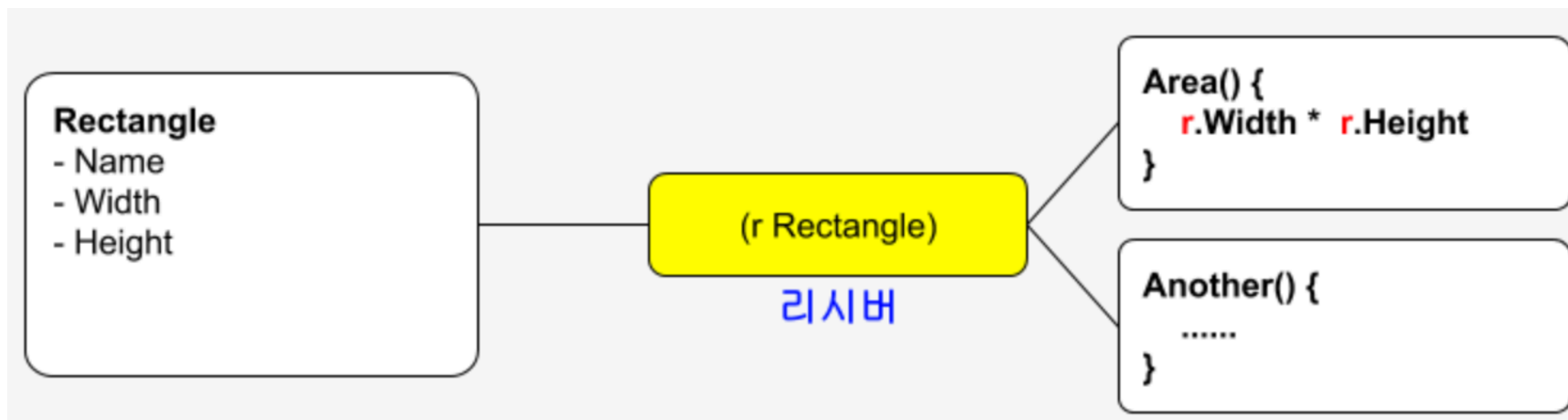
func (m Mutable) StayTheSame() {
    m.a = 5
    m.b = 7
}

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

func main() {
    m := &Mutable{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

```
8  type Vertex struct {  
9      X, Y float64  
10 }  
11  
12 //① Abs 메소드는 리시버인자로 v Vertex를 받습니다.  
13 func (v Vertex) Abs() float64 {  
14     return math.Sqrt(x: v.X*v.X + v.Y*v.Y)  
15 }
```

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

method-lab-02.go 참조

Methods

- How to modify variables through methods?
- Value receiver
- Pointer receiver

```
8  type Vertex struct {  
9      X, Y float64  
10 }  
11  
12 //① Abs 메소드는 리시버인자로 v Vertex를 받습니다.  
13 func (v Vertex) Abs() float64 {  
14     return math.Sqrt(x: v.X*v.X + v.Y*v.Y)  
15 }
```

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

```
17 func (v *Vertex) SetX(newX float64){  
18     v.X = newX  
19 }  
20  
21 func (v *Vertex) SetY(newY float64){  
22     v.Y = newY  
23 }
```

Methods

- Value receiver in struct

```
8 type Vertex struct {  
9     X, Y float64  
10 }
```

- Value receiver in type

```
18 type MyFloat float64
```

Methods

- Value receiver in type

```
18  type MyFloat float64
20  func (f MyFloat) Abs() float64 {
21      if f < 0 {
22          return float64(-f)
23      }
24      return float64(f)
25  }
26
27  //③ MyFloat이 포인터가 아닌 리시버 인자입니다
28  func (f MyFloat) power10() {
29      f = f * MyFloat(10)
30  }
31
32  //④ MyFloat이 포인터 리시버 인자입니다.
33  func (f *MyFloat) power100() {
34      *f = *f * MyFloat(100)
35  }
```

Question:

f와 MyFloat(10) 의 의미의 차이는?

Self-Exercise

- 다음과 같은 구조체 안의 value receiver 를 이용하여 method-lab-02.go와 동일한 행위를 하도록 수정하라 (주어진 시간 10분)

```
18 type MyFloat struct {  
19     f float64  
20 }
```

- 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

OOP in Go

Class OOP

GO OOP

Encapsulation

Packages

Inheritance

Composition

Polymorphism

Interface

Abstraction

Embedding

Encapsulation

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

Encapsulation

- Encapsulated by package

```
1 package encap
2
3 import "fmt"
4
5 // Encapsulation 구조체는 이 패키지 밖으로 노출될 수 있음
6 type Encapsulation struct{}
7
8 // Expose 메서드는 패키지 밖을 노출될 수 있음
9 func (e *Encapsulation) Expose() {
10     fmt.Println("AHHHH! I'm exposed!")
11 }
12
13 // hide 메서드는 패키지 내부에서만 사용할 수 있음
14 func (e *Encapsulation) hide() {
15     fmt.Println("Shhhh... this is super secret")
16 }
17
18 // Unhide는 노출되지 않은 hide 메서드를 사용함
19 func (e *Encapsulation) Unhide() {
20     e.hide()
21     fmt.Println("...jk")
22 }
```

```
encap
├── encap.go
├── class-lab-01.go
├── class-lab-02-ABC.go
└── lab-encap.go
```

```
1 package main
2
3 import "./encap"
4
5 func main() {
6     e := encap.Encapsulation{}
7
8     e.Expose()    // "AHHHH! I'm exposed!"
9
10    // e.hide()    // 주석을 없애면, 다음의 에러가 발생함
11    // ./main.go:10: e.hide undefined (cannot refer
12    // to unexported field or method encapsulation.
13    // (*Encapsulation)."".hide)
14
15    e.Unhide()    // "Shhhh... this is super secret"
16    // "...jk"
17 }
```

lab-uncap.go

Package

- Type

```
1 package testlib
2
3 import "fmt"
4
5 var pop map[string]string
6
7 func init() {
8     pop = make(map[string]string)
9     pop["Adele"] = "Hello"
10    pop["Alicia Keys"] = "Fallin'"
11    pop["John Legend"] = "All of Me"
12 }
13
14 // GetMusic : Popular music by singer (외부에서 호출 가능)
15 func GetMusic(singer string) string {
16     return pop[singer]
17 }
18
19 func getKeys() { // 내부에서만 호출 가능
20     for _, kv := range pop {
21         fmt.Println(kv)
22     }
23 }
```

Package

- Main package

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello")
7 }
```

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**

```
8  type Shape interface {  
9      area() float64  
10     perimeter() float64  
11 }
```

2. Declare **struct**

```
13 type Rect struct{  
14     w, h float64  
15 }  
16 type Circle struct{  
17     r float64  
18 }
```

3. Implement interfaces

```
20 func (r Rect) area() float64{  
21     return r.w * r.h  
22 }  
23  
24 func (r Rect) perimeter() float64{  
25     return 2 * (r.w + r.h)  
26 }  
27  
28 func (c Circle) area() float64{  
29     return math.Pi * c.r * c.r  
30 }  
31  
32 func (c Circle) perimeter() float64{  
33     return 2 * math.Pi * c.r  
34 }
```

Polymorphism

- In Golang, polymorphism is implemented by interface.

4. A. Use interface

```
44 func measure(shape Shape){  
45     fmt.Println(shape)  
46     fmt.Println(shape.area())  
47     fmt.Println(shape.perimeter())  
48 }
```

4. B. Use interface

```
50 func showArea(shapes ... Shape){  
51     for _, s := range shapes {  
52         fmt.Println(s.area())  
53     }  
54 }
```

```
36 func main() {  
37     r := Rect{10., 20.}  
38     c := Circle{10.}  
39  
40     measure(r)  
41     measure(c)  
42  
43     showArea(r,c)  
44 }
```

Pop Question

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

```
20 func (r Rect) area() float64{
21     return r.w * r.h
22 }
23
24 func (r Rect) perimeter() float64{
25     return 2 * (r.w + r.h)
26 }
27
28 func (c Circle) area() float64{
29     return math.Pi * c.r * c.r
30 }
31
32 func (c Circle) perimeter() float64{
33     return 2 * math.Pi * c.r
34 }
```

Wrap-up

Class OOP

GO OOP

Encapsulation

Packages

Inheritance

Composition

Polymorphism

Interface

Abstraction

Embedding