



# Piscine Golang - Day02

🕒 Created @October 24, 2021 11:28 PM

요약: 이 문서는 Struct, Method, Interface, Type Assertion, Type Switch를 다룹니다.

서브젝트 오류 문의 : bigpel66@icloud.com

---

## Contents

- III* Exercise 00 : Grand Opening!
- IV* Exercise 01 : Product has been shipped
- V* Exercise 02 : Competitor arrived.
- VI* Exercise 03 : I'm the boss...
- VII* Exercise 04 : Let's check where it belongs to...
- VIII* Exercise 05 : ft\_Queue

# Chapter I - General Rules

- 모든 Exercise들은 Golang 1.17로 진행하며 테스트 코드와 함께 작성되어야 합니다. 작성된 테스트 코드들은 Exercise와 함께 제출되어야 합니다.
- 제출된 테스트 코드 내에 테스트 케이스는 반드시 존재해야 합니다.
- 적극적인 테스트 케이스는 권장되며, 특히 예외 케이스에 신경써야 합니다.
- 평가자는 자신이 생각했던 테스트 케이스가 존재하지 않는 경우, 피평가자에게 테스트 케이스를 요청할 수 있습니다. 이 때 피평가자의 코드가 동작하지 않으면 평가는 종료됩니다.
- 테스트 진행은 "testing" 패키지를 이용해야 하며, "testing"과 "[github.com/stretchr/testify/assert](https://github.com/stretchr/testify/assert)" 둘 중 하나를 일관성 있게 사용해야 합니다.
- 테스트 진행 시 testing 객체를 이용한 벤치마킹 역시 적극 권장됩니다.
- Exercise의 제출은 동료 평가 및 Moulinette 채점으로 **PASS** 혹은 **FAIL**이 결정됩니다.
- Exercise들은 gofmt를 이용하여 포매팅이 되어 있어야 합니다. 동료 평가 진행 시, 제출된 Exercise가 포매팅이 되어 있지 않다면 **FAIL**입니다.
- Exercise들은 golanci-lint를 이용했을 때, 별다른 문제가 없다는 것이 검증되어야 합니다. golanci-lint에서 문제가 있는 부분이 발견되면 **FAIL**입니다.
- 동료 평가 진행을 위한 주석은 적극 허용됩니다. 각 함수 및 변수들에게 적용된 주석은 Golang에서 정식으로 지원하는 godoc을 이용하여 Document로 만들 수 있습니다.
- 컴파일은 **go build**를 이용하며, 이를 위해 **go mod**를 먼저 이해해야 합니다. **go mod**를 이용하지 않고 제출된 Exercise 역시 **FAIL**입니다.
- 제공된 서브젝트를 철저히 파악하여 Golang의 기초 지식을 모두 얻을 수 있도록 하십시오.
- 서브젝트에서 요구하는 내용들이 짧더라도, 이 내용들을 이해하고 예상한대로 작동되도록 시간을 쏟는 것은 굉장히 가치 있는 행동입니다. 다양한 시도를 권장합니다.
- *Go ? Ahead ! Write in Go.*



go mod를 이용하기 전에 GOPATH, GOROOT를 알아보십시오.

## Chapter II - Preamble

어떤 언어에서도 문서화를 중요시 하는 것은 동일하겠지만, Golang은 Golang을 기반으로 하는 프로젝트에서 문서화를 특히 중요시 여기는 마인드가 잘 녹아있는 언어이다. 문서화는 소프트웨어의 유지보수가 가능하도록 만들고, 접근성을 좋게 만드는 중요한 부분이다. 하지만 프로그래밍을 하면서 문서화를 이어가다보면, 코드에서 생긴 변경점을 문서에 늘 상 반영하는 것은 여간 쉬운 일이 아니다. Golang은 개발자가 문서를 매우 쉽게 구성할 수 있도록 godoc이라는 문서화 도구를 제공한다. 이는 코드 작성 시, 몇가지 규칙만 준수하면 된다. 굳이 문서화 때문이 아니더라도 문서화에 이용될 수 있는 주석들은 여전히 그 존재만으로 코드를 이해할 수 있는 좋은 매개체가 된다. 따라서 Golang을 이용하여 프로그래밍을 할 때, godoc에서 요구하는 간단한 규칙을 이용하여 주석을 함께 기재하는 습관은 굉장히 중요하다.

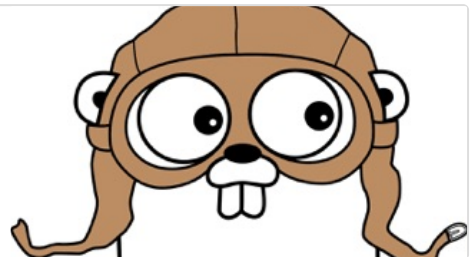
Golang에서 간단한 문서화를 제공하는 점과 더불어 Golang은 TDD (Test-Driven Development)를 학습하고 진행하기에 정말 편한 언어이다. Golang에는 테스트 진행을 위한 프레임워크를 내장하고 있으며, 이를 이용할 때 몇가지 규칙만 준수하면 편하게 테스트를 할 수 있다. 따라서 TDD를 고수하여 탄탄한 프로그램을 만들 수 있다. 이 때 해당 프레임워크에서는 단순히 테스트만 할 수 있는 것이 아니라, 벤치마킹도 지원하므로 해당 프레임워크를 잘 이용하여 프로그래밍을 할 수 있도록 습관을 들이는 것이 좋다.

Golang을 효율적으로 사용하는 방법과 테스트를 주도하여 프로그래밍을 하는 방법은 아래에서 학습해볼 수 있다.

### Effective Go

Go is a new language. Although it borrows ideas from existing languages, it has unusual properties that make effective Go programs different in character from programs written in its

 [https://golang.org/doc/effective\\_go](https://golang.org/doc/effective_go)




### Learn Go with Tests

 <https://quii.gitbook.io/learn-go-with-tests/>



## Chapter III

### Exercise 00 : Grand Opening!

<u>Aa</u> Name	 Tags
<u>Turn-in directory</u>	ex00
<u>Files to turn in</u>	product/product.go , product/product_test.go , main.go , go.mod
<u>Module name</u>	ex00
<u>Forbidden packages</u>	None

당신은 퇴직 자금으로 쇼핑몰을 창업했고, 내일 개업일을 맞이하게 됩니다.

쇼핑몰에서 이용할 결제 시스템 구축은 다행스럽게도 방금 마무리가 되었고, 판매할 물건의 정보를 기입하는 일만 남았습니다.

하지만 불행히도 정해진 금액을 기록해둔 종이가 보이지 않습니다. 따라서 상품의 이름과 해당 가격을 Product라는 구조체를 이용하여 직접 저장해야 합니다.

Product 구조체는 다음과 같이 선언해야 합니다.

```
type Product struct {  
    name string  
    price uint  
}
```

구조체 내부의 변수는 동일한 패키지에서는 그냥 접근할 수 있지만, 외부 패키지에서 직접 이용하는 것은 불가능하게 되어 있으므로 외부에서 접근할 수 있는 수단이 필요합니다.

이에 따라 외부 패키지에서 접근할 수 있는 수단으로 메서드를 정의하십시오. (외부 패키지에서는 값을 설정하고 값을 불러올 수 있어야 합니다.)

값을 설정하는 메서드를 작성할때 해당 메서드는 **error** 타입을 반환해야 합니다.

또한 Product를 한 번에 초기화하여 이용할 수 있도록, 아래와 같은 생성자 함수를 정의하십시오.

```
func NewProduct(price uint, name string) (*Product, error)
```

특정 상황에 대한 예외처리는 자유롭게 할 수 있으며, 테스트 코드 작성은 필수입니다.  
main 함수의 구성은 자유로우니 다양한 방법으로 Product 구조체를 사용해보십시오.

## Chapter IV

### Exercise 01 : Product has been shipped.

<u>Aa</u> Name	<u>≡</u> Tags
<u>Turn-in directory</u>	ex01
<u>Files to turn in</u>	product/product.go , product/product_test.go , dj/dj.go , dj/dj_test.go , main.go , go.mod
<u>Module name</u>	ex01
<u>Forbidden packages</u>	None

개장 준비를 철저히 한 덕분에 편안한 마음으로 개장일을 맞이했고, 드디어 첫 상품 판매에 성공했습니다. 입금을 확인했기 때문에, 상품을 고객에게 배송해야 합니다.

쇼핑몰 홍보가 제대로 되어서 그런지 첫 상품이 판매된 후, 상품 배송 준비를 하는 도중에도 상품 판매가 이뤄지고 있는 상황이 되었습니다. 따라서 이 모든 상황을 적절히 대처할 수 있도록 배송 업체와 제휴를 맺기로 결심했습니다. 우리는 Dj라는 이름의 적절한 배송 업체를 구조체로 만들 것입니다.

Dj의 구조체는 다음과 같이 선언해야 합니다.

```
type Dj struct {
    to    string
    from  string
}
```

Dj라는 구조체가 무언가 허전하다고 생각했다면, 정확하게 보았습니다. 배송 업체인데도 배송 물품이 없는 것을 확인할 수 있습니다! 따라서 Dj 구조체 내부에는 이전에 정의해뒀던 Product 구조체를 반드시 임베딩하고 있도록 해야 합니다. 그리고 Dj 구조체 역시 Product 구조체처럼 NewDj라는 생성자 함수를 갖고 있습니다.

위와 같은 정보들 외에도, 배송 업체는 배송 물품 정보를 확인할 수 있는 Info 메서드와 배송 시작을 알리는 Send 메서드를 갖고 있도록 정의해야 합니다. 각 함수의 원형은 아래와 같습니다.

```
func (dj *Dj) Info() bool
```

```
func (dj *Dj) Send()
```

Info 메서드는 임베딩되어 있는 Product의 내용을 출력해주는 역할을 수행할 수 있어야 하고, 만약 내용이 비어있다면 Product 구조체의 누락된 필드가 존재한다고 알려주어야 합니다.

누락된 필드는 name → price 순으로 확인하며, 누락된 필드가 존재할 시 출력되는 메시지는 다음과 같습니다.

- name 누락 경우

```
▶ ./ex01
상 품 명 을 먼 저 설 정 해 주 세 요
```

- price 누락 경우

```
▶ ./ex01
상 품 의 가 격 을 먼 저 설 정 해 주 세 요
```

Info 메서드가 요구하는 필드들이 모두 채워진 상태가 되면, 아래와 같은 형식으로 출력될 수 있어야 합니다.

```
▶ ./ex01
상 품 명   : candy
상 품 가 격 : 1000원
```

Send 메서드는 상품의 정보, 수신인 및 송신인의 존재 유무들을 파악하며, 이와 같은 정보들이 존재하지 않을 경우에는 누락된 필드를 설정하라는 메시지와 함께 메서드를 종료해야 합니다.

누락된 필드는 product.name → product.price → dj.to → dj.from 순으로 확인해야 합니다.

- product.name 누락 경우

```
▶ ./ex01
상 품 명 을 먼 저 설 정 해 주 세 요
```

- product.price 누락 경우

```
▶ ./ex01
상 품 의 가 격 을 먼 저 설 정 해 주 세 요
```

- dj.to 누락 경우

```
▶ ./ex01
상 품 명   : candy
상 품 가 격 : 1000원
택 배 발 송 전 발 송 자 를 먼 저 입 력 해 주 세 요
```

- dj.from 누락 경우

```
▶ ./ex01
상 품 명   : candy
상 품 가 격 : 1000원
택 배 발 송 전 발 송 자 를 먼 저 입 력 해 주 세 요
```

누락된 필드가 존재하지 않을 경우 다음과 같은 메시지를 출력해야 합니다.



```
▶ ./ex01
상 품 명   : candy
상 품 가 격 : 1000원
[DJ] yongckim 님 이 jseo 님 에 게 일 반 배 송 을 시 작 했 습 니 다 .
```

특정 상황에 대한 예외처리는 자유롭게 할 수 있으며, 테스트 코드 작성은 필수입니다.  
main 함수의 구성은 자유로우니 다양한 방법으로 Dj 구조체를 사용해보십시오.

## Chapter V

### Exercise 02 : Competitor arrived.

<u>Aa</u> Name	<u>≡</u> Tags
<u>Turn-in directory.</u>	ex02
<u>Files to turn in</u>	product/product.go , product/product_test.go , dj/dj.go , dj/dj_test.go , coubang/coubang.go , coubang/coubang_test.go , main.go , go.mod
<u>Module name</u>	ex02
<u>Forbidden packages</u>	None

쇼핑몰을 개장하고 시간이 조금 흘렀고, 사업은 아주 순조로운 것처럼 보였지만 기존의 배송 업체가 가격 인상을 요구하고 있습니다. 우연찮게도 쇼핑몰 이메일로 새로운 배송 업체가 좋은 가격의 제휴 제안을 하면서, 새로운 배송 업체와 계약을 하게 되었습니다. 안타깝게도 새로운 배송 업체가 지원하지 않는 지역이 몇 군데 있어서, 기존의 배송 업체와의 악연을 끊지는 못한 상황이 되었습니다. 이와 같은 상황에 따라 쇼핑몰에서 새롭게 이용할 배송 업체를 Coubang이라는 이름의 구조체를 만들 것입니다.

Coubang 구조체는 다음과 같이 선언해야 합니다. Coubang은 기존 배송 업체와 달리 멤버쉽 유무에 따라 로켓 배송을 지원하게 됩니다. 그리고 Dj 구조체와 마찬가지로 상품에 대한 정보를 갖고 있어야 하므로, Product 구조체를 임베딩 해줘야 합니다. 잊지 않았겠지만 Dj 구조체에서 Product 구조체처럼 NewDj라는 생성자 함수를 갖고 있었던 것처럼, Coubang도 NewCoubang이라는 생성자 함수를 이용하여 생성할 수 있어야 합니다.

```
type Coubang struct {
    to      string
    from    string
    membership bool
}
```

Coubang 역시 메서드로 Dj 구조체가 가졌던 Info와 Send 메서드를 동일하게 가지며, 구현 조건은 동일하지만 누락된 필드가 모두 존재하게 되었을 때는 아래와 같은 형식으로 출력될

수 있어야 합니다.

- Info

```
▶ ./ex02
상 품 명   : candy
상 품 가 격 : 1000원
```

- Send

```
▶ ./ex02
상 품 명   : candy
상 품 가 격 : 1000원
[COUBANG] yongckim 님이 min-jo 님에게 일반 배송을 시작했습니다.
```


Coubang이라는 배송 업체는 특이하게도 로켓 배송을 지원하기도 합니다. 따라서 RocketSend라는 메서드를 추가로 정의해야 합니다. RocketSend의 정의 방법은 Send 메서드와 동일한 조건으로 이뤄져야 합니다. 다만 출력 형식이 아래와 같이 나타난다는 점에서 Send 메서드와 차이가 있습니다.

```
▶ ./ex02
상 품 명   : candy
상 품 가 격 : 1000원
[COUBANG] yongckim 님이 min-jo 님에게 로켓 배송을 시작했습니다.
```

이전 챕터들과 마찬가지로 특정 상황에 대한 예외처리는 자유롭게 할 수 있으며, 테스트 코드 작성은 필수입니다. 또한 main 함수에서는 각 함수를 이용하여 다양한 예시 상황이 정의 되어 있어야 합니다.

## Chapter VI

### Exercise 03 : I'm the boss...

<u>Aa</u> Name	 Tags
<u>Turn-in directory.</u>	ex03
<u>Files to turn in</u>	product/product.go , product/product_test.go , dj/dj.go , dj/dj_test.go , coubang/coubang.go , coubang/coubang_test.go , shipping/shipping.go , shipping/shipping_test.go , main.go , go.mod
<u>Module name</u>	ex03
<u>Forbidden packages</u>	None

Dj 배송 업체가 경쟁사의 등장으로 위기 의식을 느낀 것 같습니다. 나날이 성장하는 쇼핑몰이 Dj와의 관계를 끊고 Coubang과 독점 계약을 하려는 소식을 듣자, Dj로부터 굉장히 괜찮은 제안 받아서 앞으로는 두 배송 업체를 모두 이용하기로 했습니다. 지금보다 쇼핑몰이 더 커져서 국내 배송 업체 뿐만 아니라 해외 배송 업체도 이용해야 하는 경우에는 지금과 같은 방식으로 안 되겠다고 느꼈습니다. 이에 따라 배송 업체들을 하나로 묶어서 관리할 수 있도록 Sender라는 인터페이스를 정의하십시오. Sender라는 인터페이스는 곧 Send 메서드를 갖고 있는 모든 타입을 지칭한다는 점을 고려하십시오.

Sender 인터페이스를 생성하여 임의의 배송 업체를 지칭할 수 있게 되었다면, 각 배송 업체의 Send를 신경쓰지 않아도 되는 아래와 같은 함수의 원형으로 된 Shipping이라는 함수를 정의하십시오. Shipping 함수는 내부적으로 Sender의 타입에 따라 각자에게 해당되는 올바른 Send 메서드를 호출해주는 역할을 수행합니다.

```
func Shipping(Sender)
```


특히 Shipping 함수 호출 시에 Sender의 타입이 Coubang 구조체를 지칭하는 상황이라면, 멤버십 가입 여부를 판별하여 멤버십 가입이 되어 있을 때는 RocketSend 메서드를 이용할 수 있도록 만드십시오.

이전 챕터들과 마찬가지로 특정 상황에 대한 예외처리는 자유롭게 할 수 있으며, 테스트 코드 작성은 필수입니다. 또한 main 함수에서는 각 함수를 이용하여 다양한 예시 상황이 정의되어 있어야 합니다.

인터페이스로 받은 매개변수를 다른 타입으로 바꿔줄 수 있을지 고민해보십시오.

## Chapter VII

### Exercise 04 : Let's check where it belongs to...

<u>Aa</u> Name	 Tags
<u>Turn-in directory.</u>	ex04
<u>Files to turn in</u>	product/product.go , product/product_test.go , dj/dj.go , dj/dj_test.go , coubang/coubang.go , coubang/coubang_test.go , shipping/shipping.go , shipping/shipping_test.go , check/check.go , check/check_test.go , main.go , go.mod
<u>Module name</u>	ex04
<u>Forbidden packages</u>	None

쇼핑몰은 꽤나 커져서 드디어 별도의 물류 창고를 두게 되었습니다. 모든 일에는 처음이 있듯이, 이번에도 문제가 발생했습니다. 물류 창고 운영을 위한 직원들을 고용하여, 물류 창고 내에서 이뤄지는 프로세스를 직접 설계시키고 운영하도록 팀을 구성했습니다. 하지만 안타깝게도 이들의 미숙함으로 출고품과 입고품이 난잡하게 섞이면서 송신으로 이용하지 않는 배송 업체의 판별이 요구되는 상황이 되었습니다.

다행히 새롭게 계약한 배송 업체는 아직까지 이용하지 않는 상황이므로, 난잡하게 섞인 물품들은 Dj와 Coubang의 배송 물품으로만 나누면 되는 상황입니다. 따라서 우리는 3가지 경우로 나누어 처리할 것입니다.

- Dj
- Coubang
- 그 외

Sender 인터페이스를 매개변수로 이용하도록 되어 있으며, 해당 물품의 정보를 출력하고 어떤 업체가 배송하는지 판별해주는 Check라는 함수를 위의 케이스를 고려하여 정의하십시오. 함수의 원형은 아래와 같습니다.

```
func Check(sender)
```

---

Check 함수의 호출은 아래와 같은 형식으로 출력이 되어야 한다는 점을 참고하십시오.

```
▶ ./ex04
상 품 명   : ps4
상 품 가 격 : 200000원
COUBANG 물 품 입 니 다 .
```

```
▶ ./ex04
상 품 명   : ps4
상 품 가 격 : 200000원
DJ 배 송 물 품 입 니 다 .
```

이 때 만일 판별된 배송 업체가 Dj와 Coubang이 아닌 경우엔 다음과 같이 출력되어야 합니다.

```
▶ ./ex04
알 수 없는 배송업체입니다 .
```

타입으로 케이스를 나누는 **스위치**가 있다면 정말 쉬울 것입니다.

## Chapter VIII

### Exercise 05 : ft\_Queue

<u>Aa</u> Name	<u>≡</u> Tags
<u>Turn-in directory</u>	ex05
<u>Files to turn in</u>	queue/queue.go , queue/queue_test.go , main.go go.mod , and whatever else you need
<u>Module name</u>	ex05
<u>Forbidden packages</u>	None

Queue는 선입 선출(First-In First-Out) 구조를 가지는 자료구조입니다. 타입에 구애받지 않고 자유롭게 사용할 수 있는 ft\_Queue를 만들어보세요. ft\_Queue는 어떤 타입이든 저장할 수 있어야 하며, Queue안의 타입들이 서로 달라도 문제가 없어야 합니다. ft\_Queue는 다음과 같이 사용할 수 있습니다.

```
var q Queue  
  
q := Queue{}
```

ft\_Queue는 다음과 같은 메서드를 가지고 있어야 합니다.

- Push : ft\_Queue의 가장 마지막 인덱스에 데이터를 추가합니다.
- Pop : ft\_Queue의 가장 첫번째 인덱스의 데이터를 반환하고, 해당 데이터를 큐에서 제거합니다.
- Empty : ft\_Queue가 현재 비어있는지 확인하고 비어있으면 true, 데이터가 존재하면 false를 반환합니다.
- Size : ft\_Queue에 현재 들어있는 데이터의 수를 출력합니다.

main문은 ft\_Queue를 사용하는 예시를 보여야 합니다.