

Table of  
Contents

WHY

ABOUT

APPLY

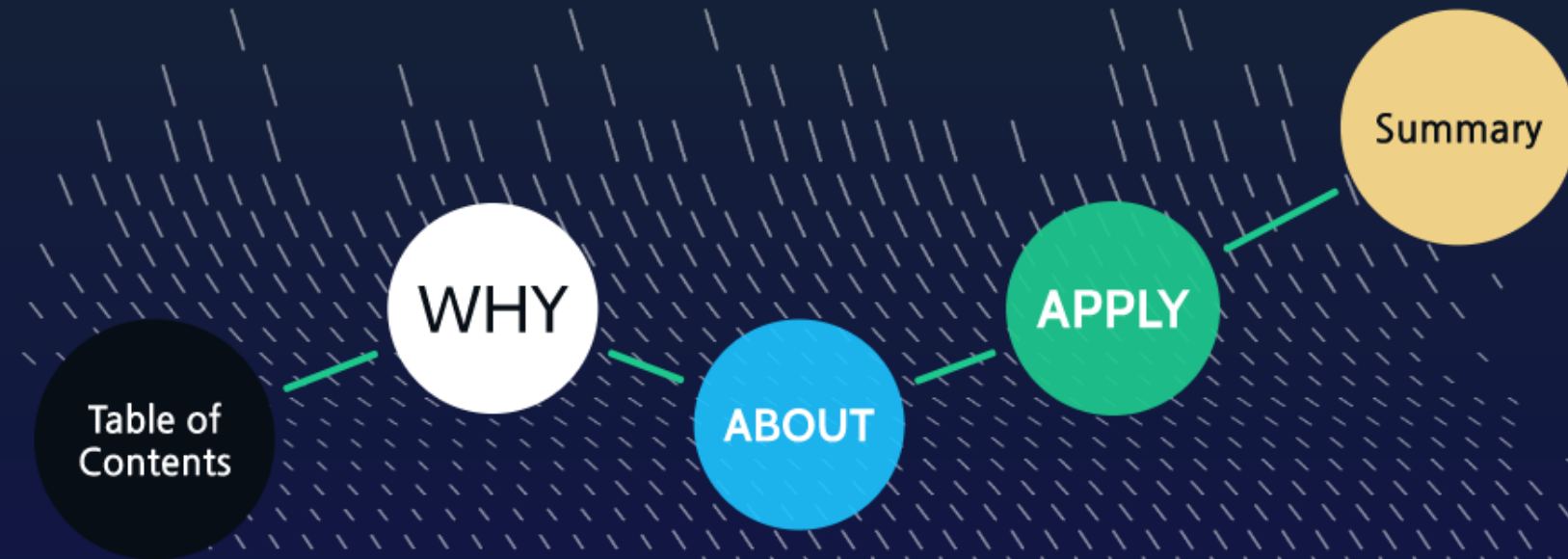
Summary

# Design Pattern

네버리스트

# Table of Contents

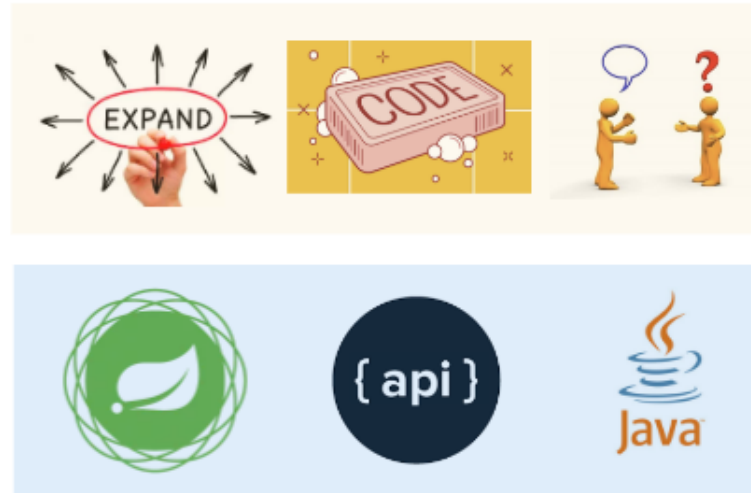
- **WHY** design pattern
- **ABOUT** design pattern
- **APPLY** design pattern

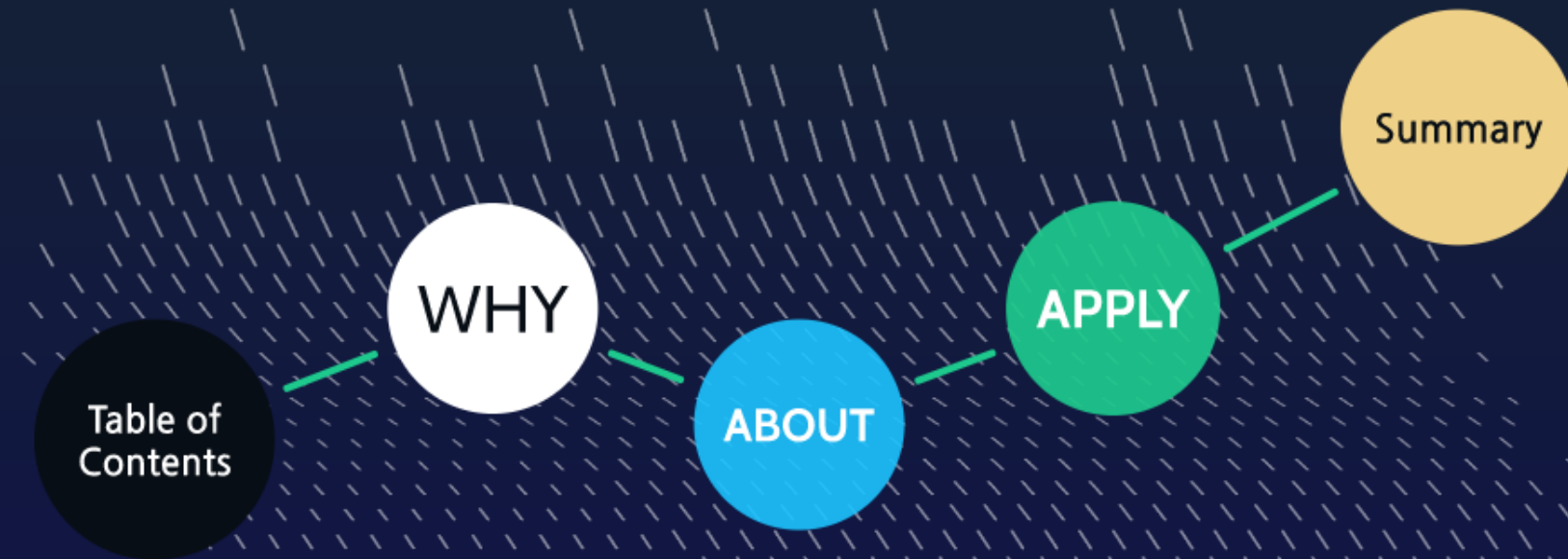


# Design Pattern

네버리스트

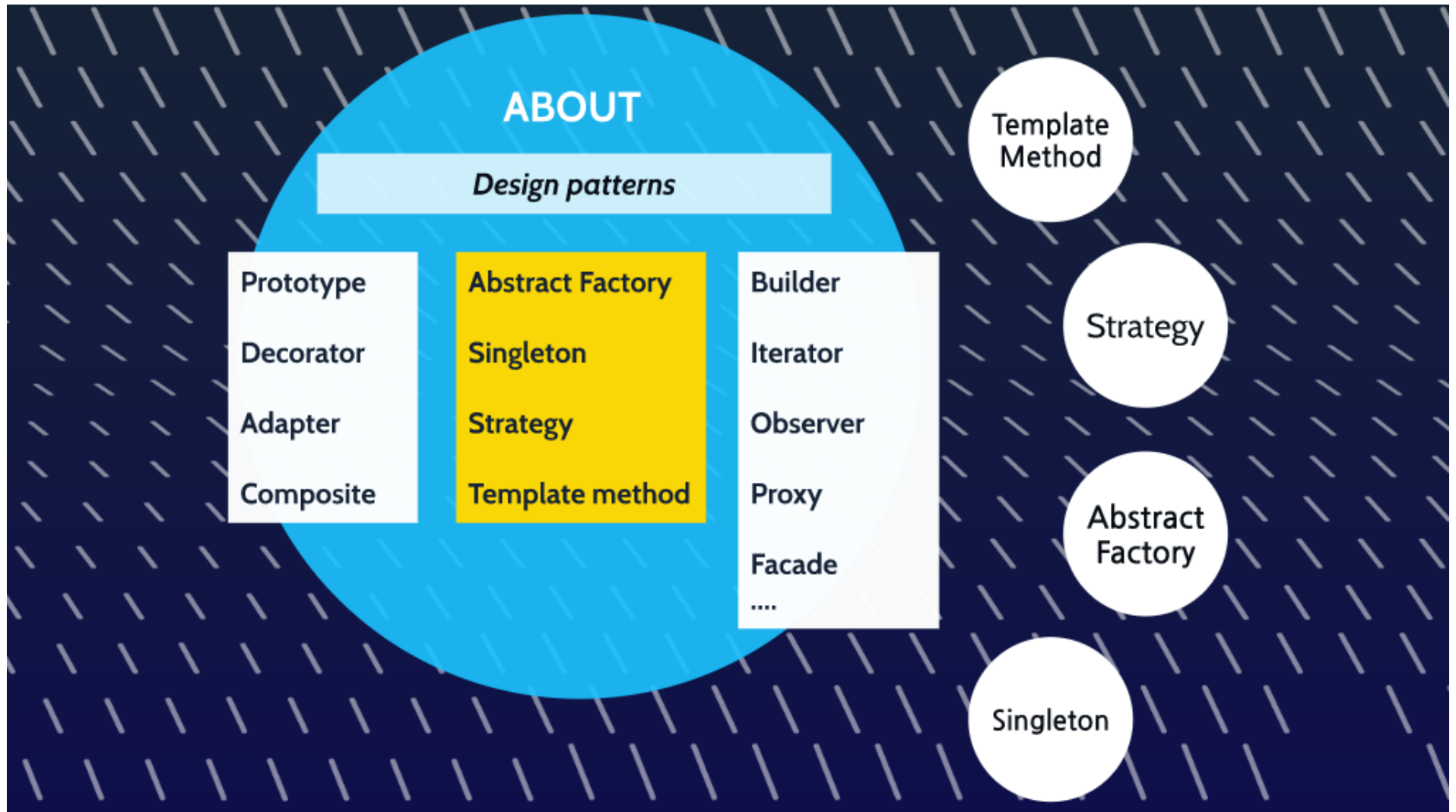
# WHY





# Design Pattern

네버리스트



# ABOUT

## *Design patterns*

Abstract Factory

Singleton

Strategy

Template method

Builder

Iterator

Observer

Proxy

Facade

....

Template  
Method

Strategy

Abstract  
Factory

Singleton

# ABOUT

## *Design patterns*

Abstract Factory

Singleton

Strategy

Template method

Template  
Method

Strategy

Abstract  
Factory

Singleton



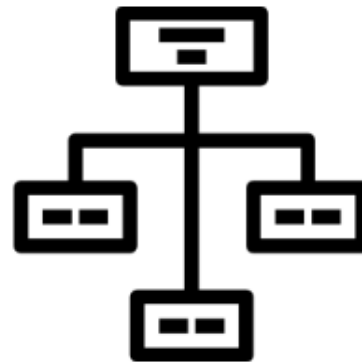
# Template Method

알고리즘의 구조를 유지하면서  
서브클래스에서 특정 단계를 재정의하는 패턴



# Strategy

알고리즘 군을 정의하고  
각각을 캡슐화하여 교환할 수 있도록 하는 패턴



# Abstract Factory

인터페이스를 이용하여 서로 연관되거나 의존하는 객체를 구상 클래스를 지정하지 않고 생성하게 하는 패턴

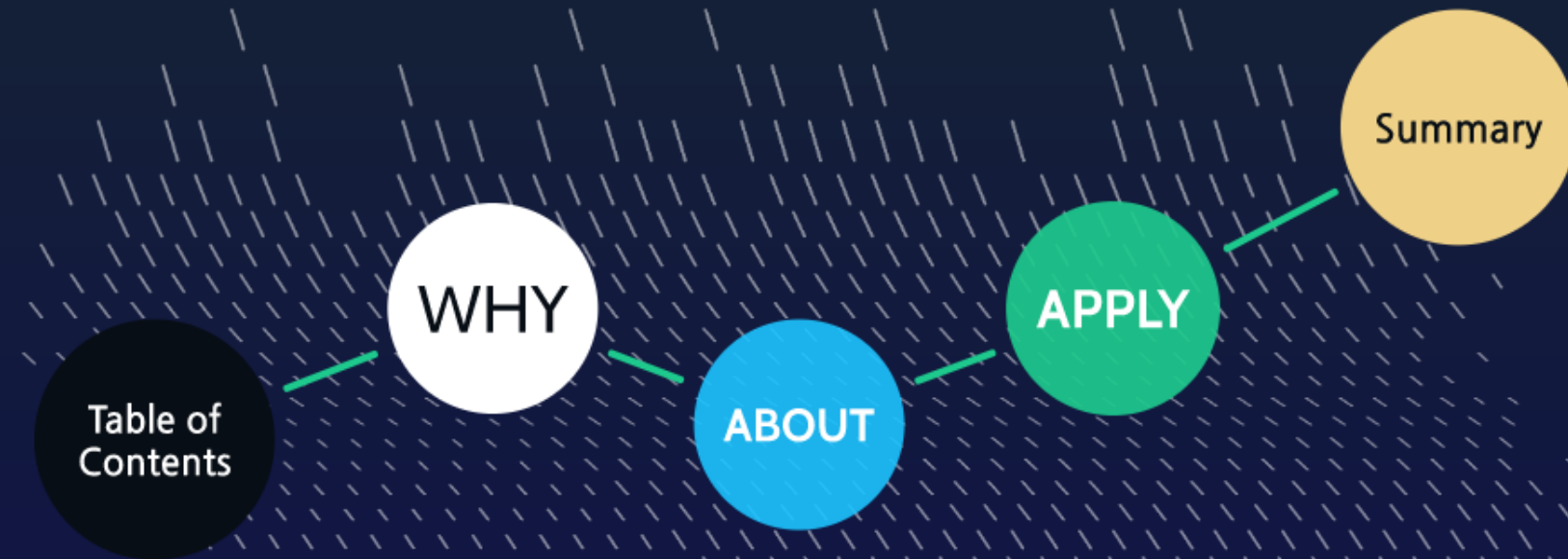


{abstract}

# Singleton

인스턴스를 하나만 생성하게 하는 패턴



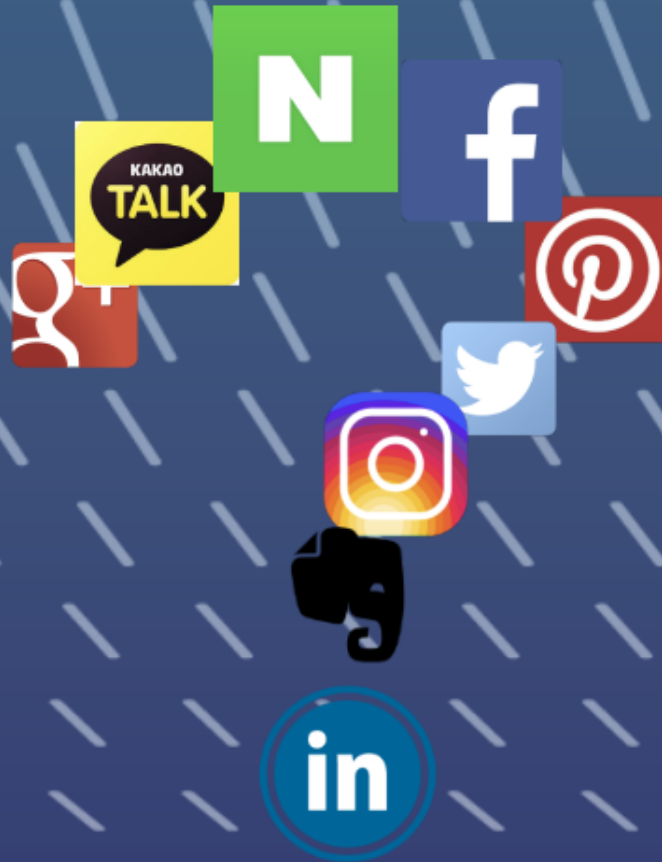


# Design Pattern

네버리스트



# OAuth 2.0 Provider 확장시..



# APPLY 1st Trial

## LoginController

- NaverLoginService
- GoogleLoginService
- .....
- N-th LoginService

## NaverLoginService

## GoogleLoginService

## FacebookLoginService

## KakaoLoginService

- N-th LoginService
- private fields (apiKey, apiSecret...)
- login/getProfile 관련 메소드

*more  
Detail..*

컨트롤러가 갈수록  
지저분해진다.



*more Detail..*

```
if ("naver".equals(sns)){  
    naverLoginService.login();  
    //etc method  
}
```

```
else if ("google".equals(sns)){  
    googleLoginService.login();  
    //etc method  
}
```

```
else if ("facebook".equals(sns)){  
    facebookLoginService.login();  
    //etc method  
}
```

```
else if ("kakao".equals(sns)) {  
    kakaoLoginService.login();  
    //etc method  
}
```

```
else if ("n-thSNS".equals(sns)) {  
    n-thSNSLoginService.login();  
    //etc method  
}
```

## 새로 로그인 서비스가 추가될 때마다...

1. 컨트롤러에 각 로그인 서비스가 인스턴스 변수로 추가됨.
2. @Autowired 생성자에 그 로그인 서비스 인스턴스를 주입받는 로직 추가
3. if-else문에 \*LoginService.login() 로직에 추가.

## 새로 로그인 서비스가 추가될 때마다...

1. 컨트롤러에 각 로그인 서비스가 인스턴스 변수로 추가됨.
2. @Autowired 생성자에 그 로그인 서비스 인스턴스를 주입받는 로직 추가
3. if-else문에 \*LoginService.login() 로직에 추가.

**컨트롤러가 갈수록  
지저분해진다.**

# APPLY 2nd Trial

```
@Service("naver")
public class NaverLoginService
    implements LoginService
```

```
{
...
}
@Service("kakao")
public class KakaoLoginService
    implements LoginService
```

```
{
...
}
@Service("facebook")
public class FaceBookLoginService
    implements LoginService
{
.....
}
```

Controller 변경점

그런데 말입니다...

## Controller 변경점

```
@GetMapping("/{sns:[google|naver|facebook]+}")  
public String login(@PathVariable String sns ...) {  
  
    LoginService loginService;  
    loginService = applicationContext.getBean(sns);  
    loginService.login();  
    ....  
}
```

## 그런데 말입니다...

### 1. 컨트롤러의 책임은 과연 어디까지인가?

: 현재, 컨트롤러가 실질적으로 팩토리 역할을 함.

### 2. DL의 사용에서 우려되는 점은 무엇인가?

: DL을 쓰기 때문에, 빈 컨테이너로부터 빈을 주입받는  
IoC 원칙에 부합하지 않습니다.

# APPLY Template Method

Request Profile

공통 Logic

Send request

차별 Logic

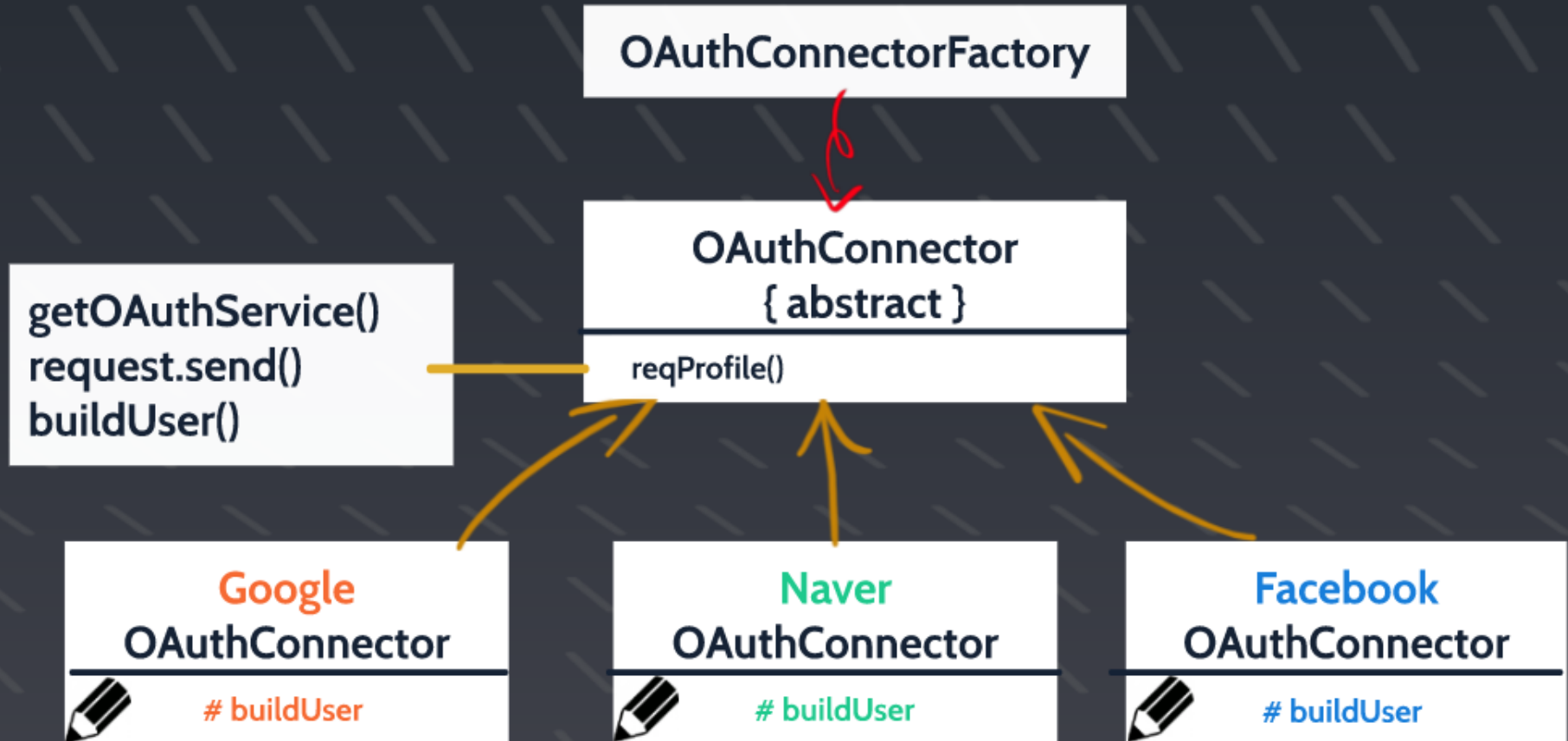
Get OAuthService

Parsing DataFormat

Build UserDomain

*Template  
method  
applied*

# Template method applied





## *scribejava Library*

*create OAuthService Instance  
using **API** key & **SECRET** key*

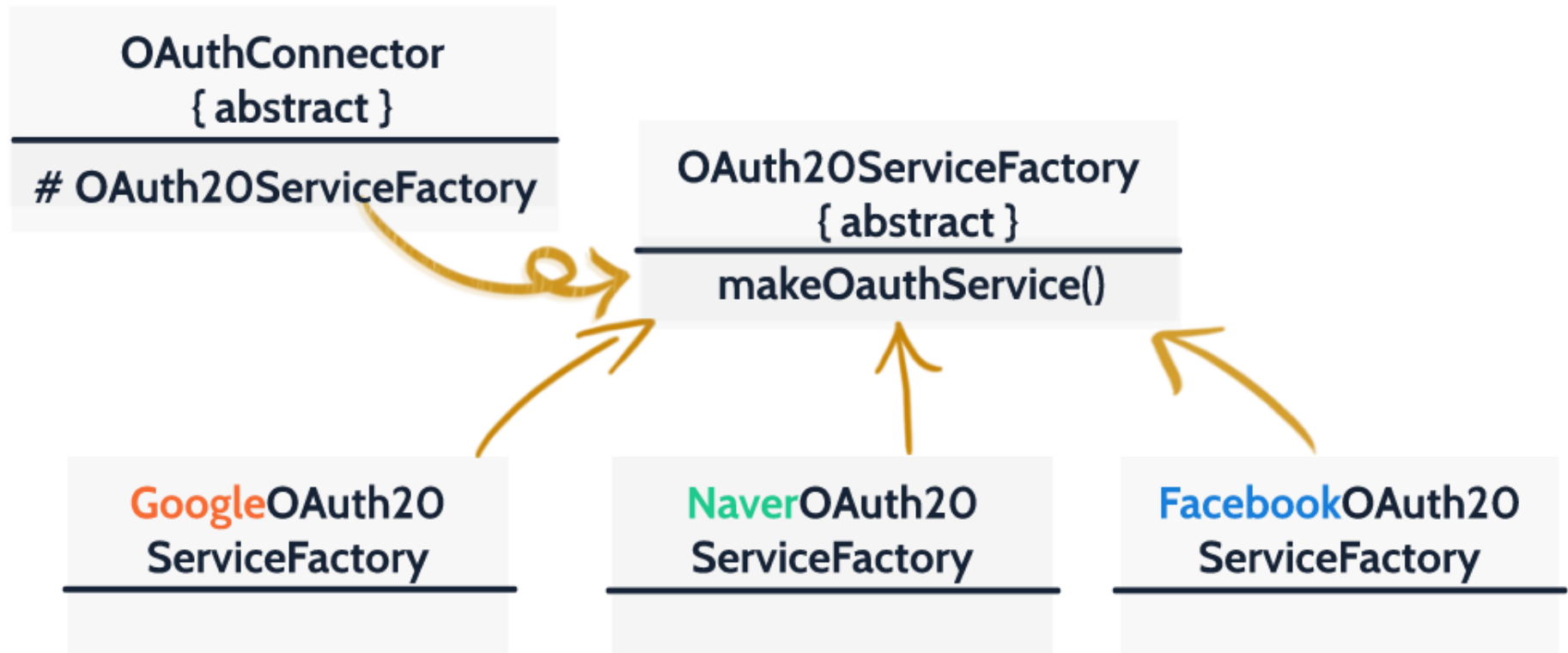
*Get  
OAuthService  
in ReqProfile*

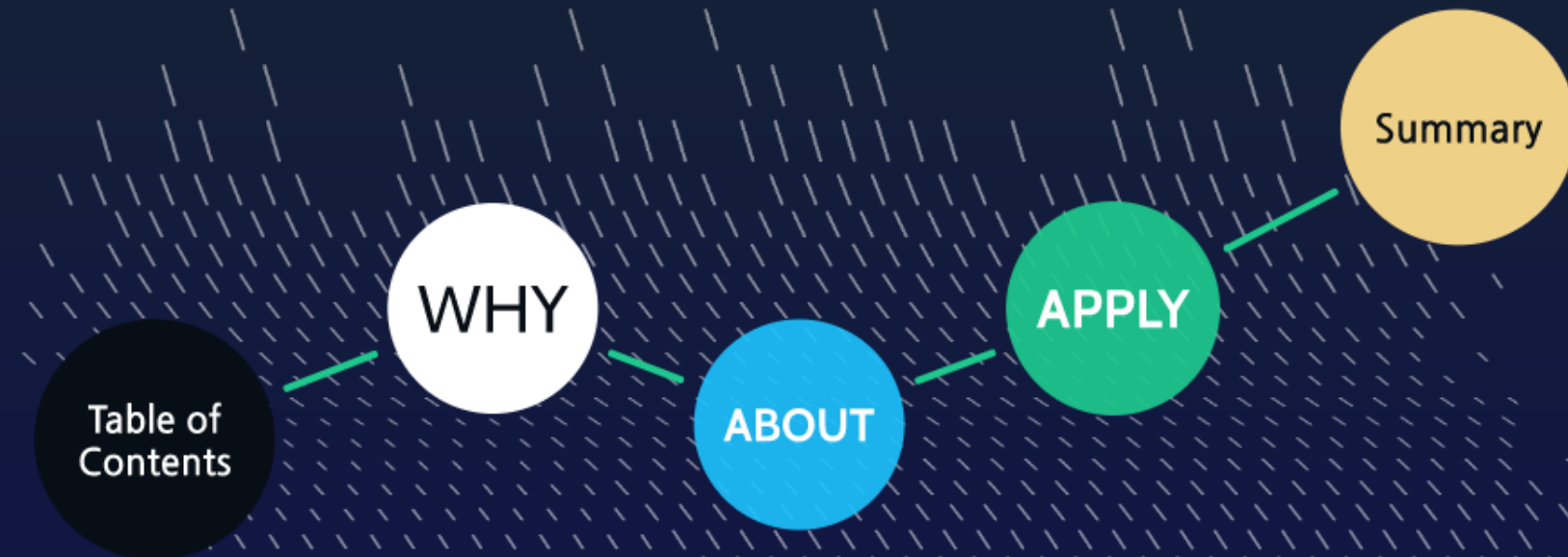
*Abstract  
Factory  
applied*

# Get OAuthService in ReqProfile

```
public OAuth20Service getOAuthService() {  
    return new ServiceBuilder ()  
        .apiKey ( API_KEY )  
        .apiSecret ( API_SECRET )  
        .callback ( REDIRECT_URI )  
        .scope ( "openid email profile" )  
        .build ( GoogleLoginApi.instance ( ) );  
}
```

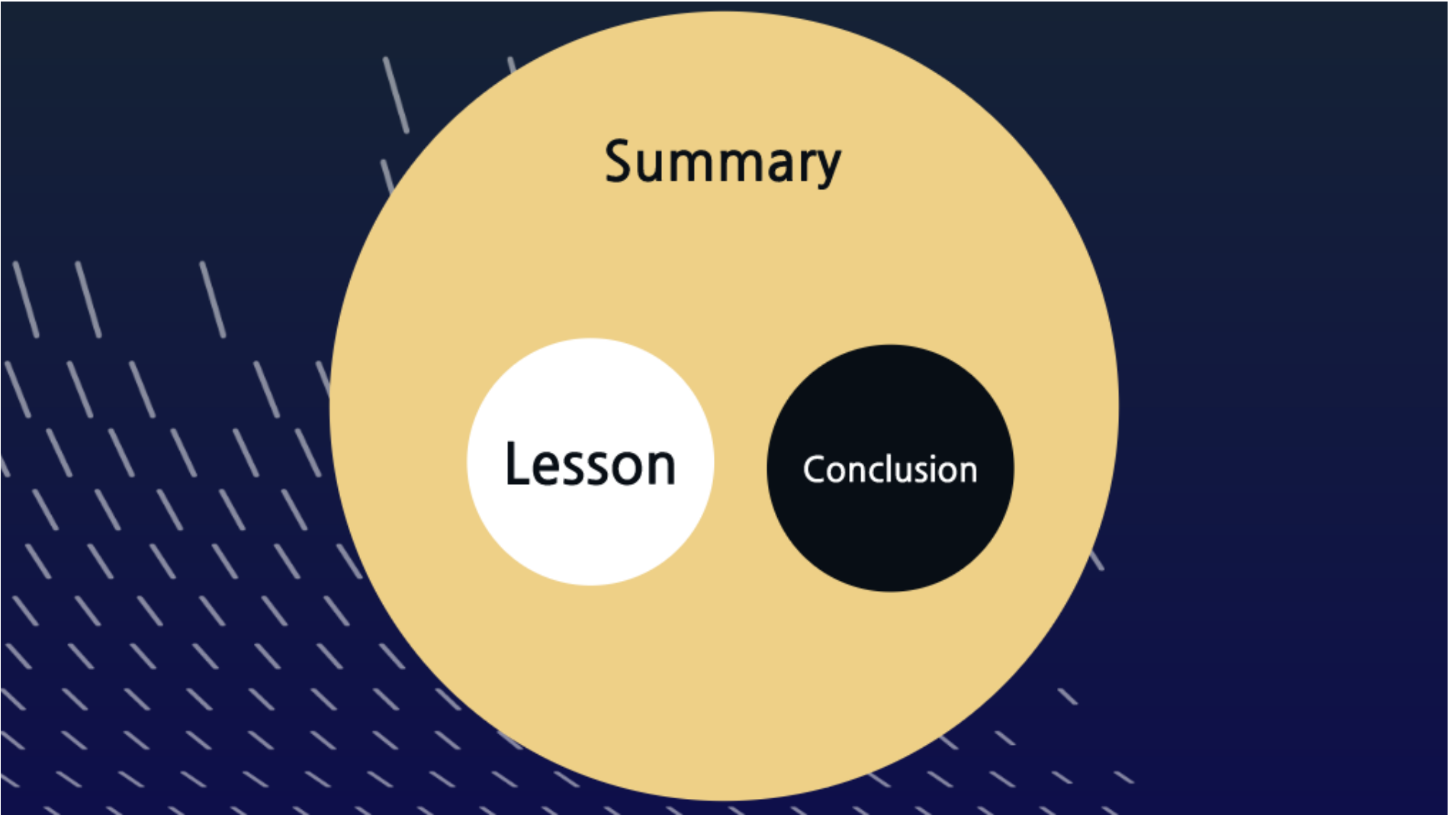
# Abstract Factory applied





# Design Pattern

네버리스트



Summary

Lesson

Conclusion

# Lesson

코드가 한층 **간결**해짐

구조적으로 **공고**해짐

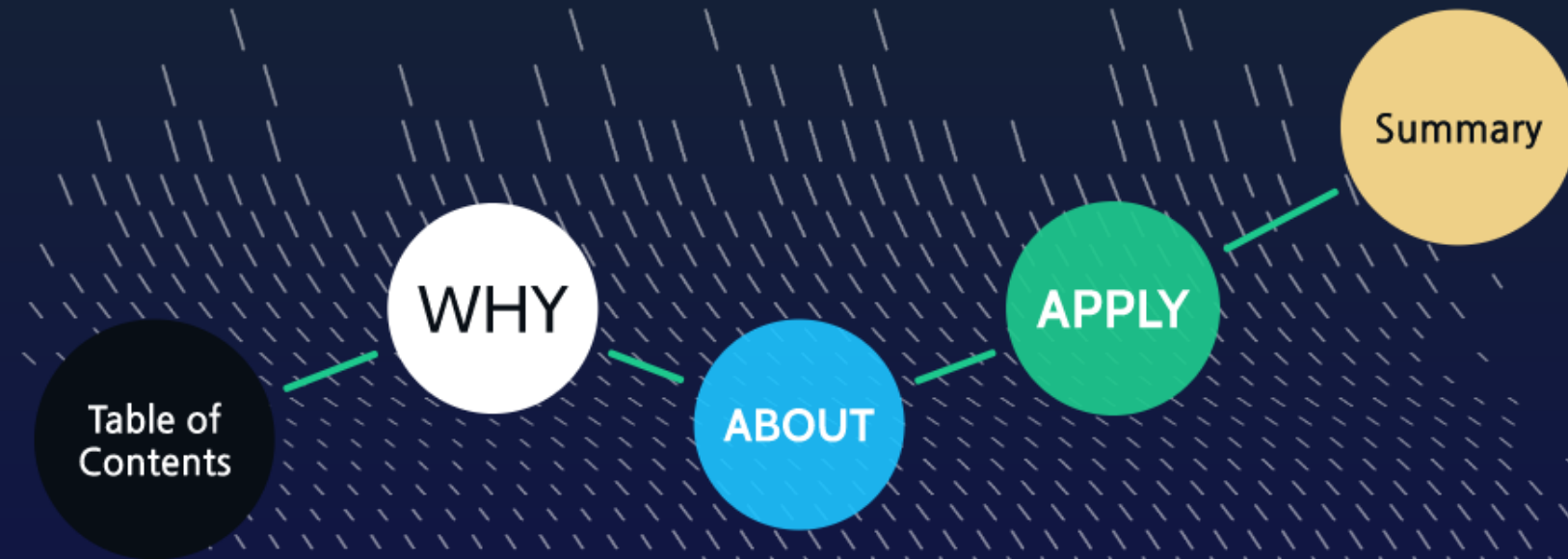
지속적인 리팩토링의 중요성을 느낌

# Conclusion

반복되는 문제 해결을 위한 리팩토링에서 출발

변화의 시기가 다른 코드의 분리

정적인 구조보다는 패턴의 의도가 중요



# Design Pattern

네버리스트