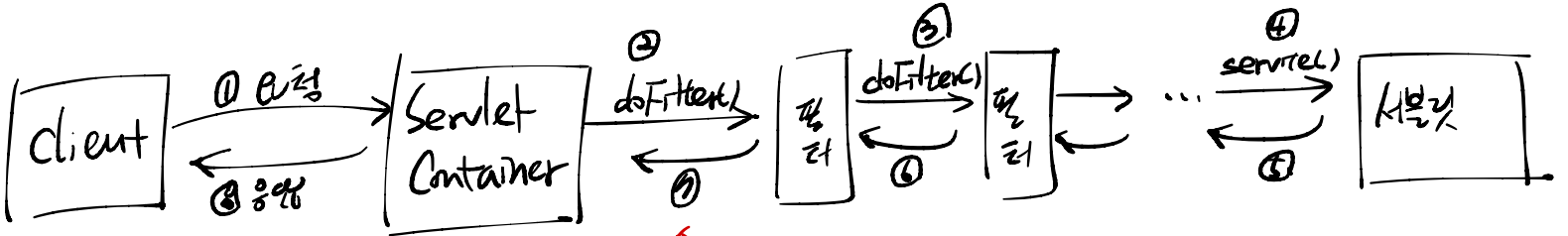
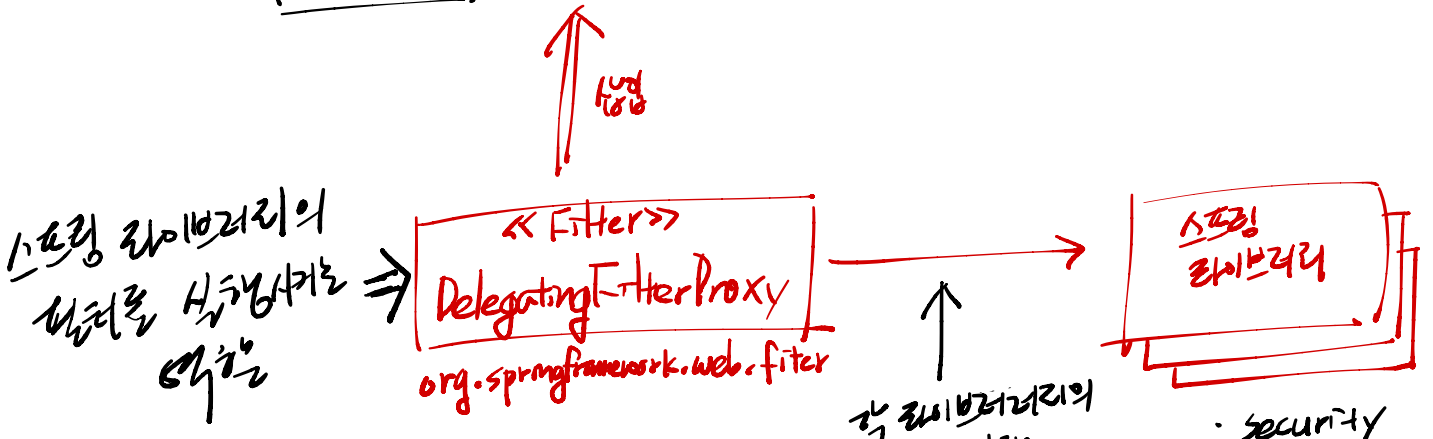
11. Spring Security 개념

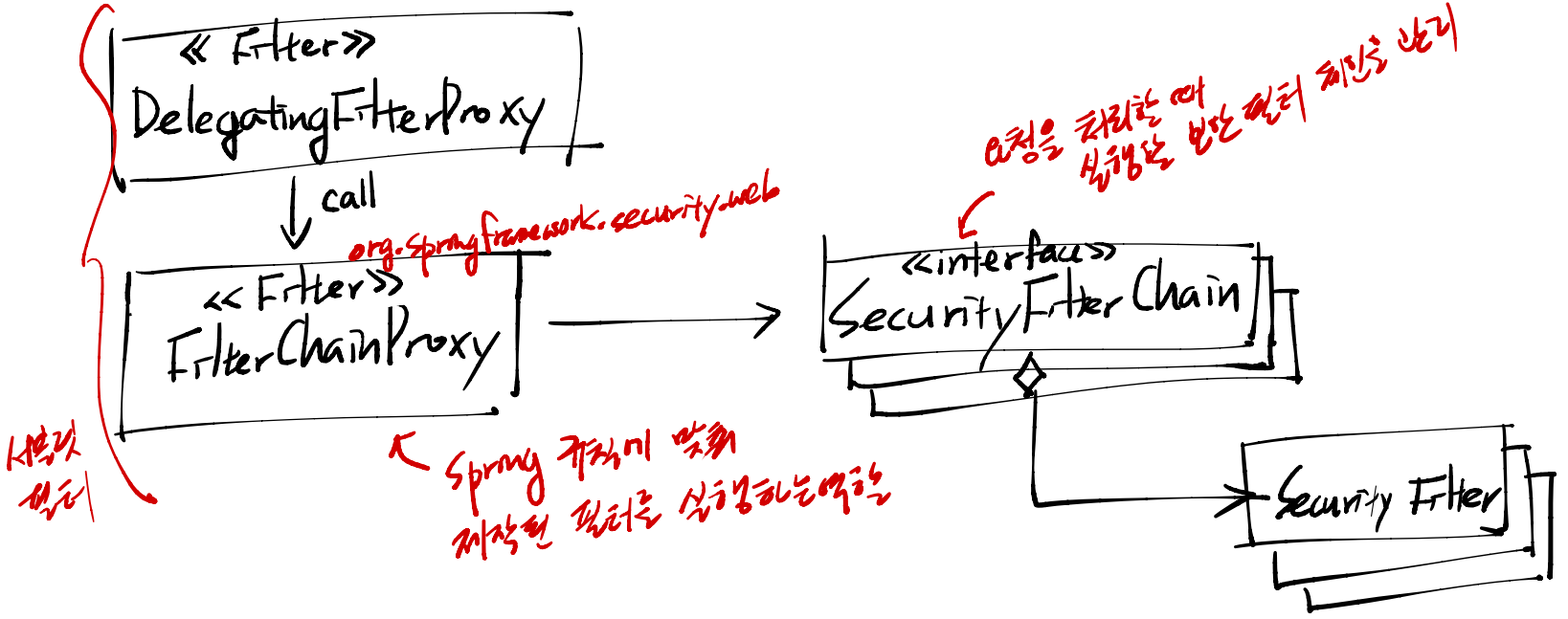
* 구조





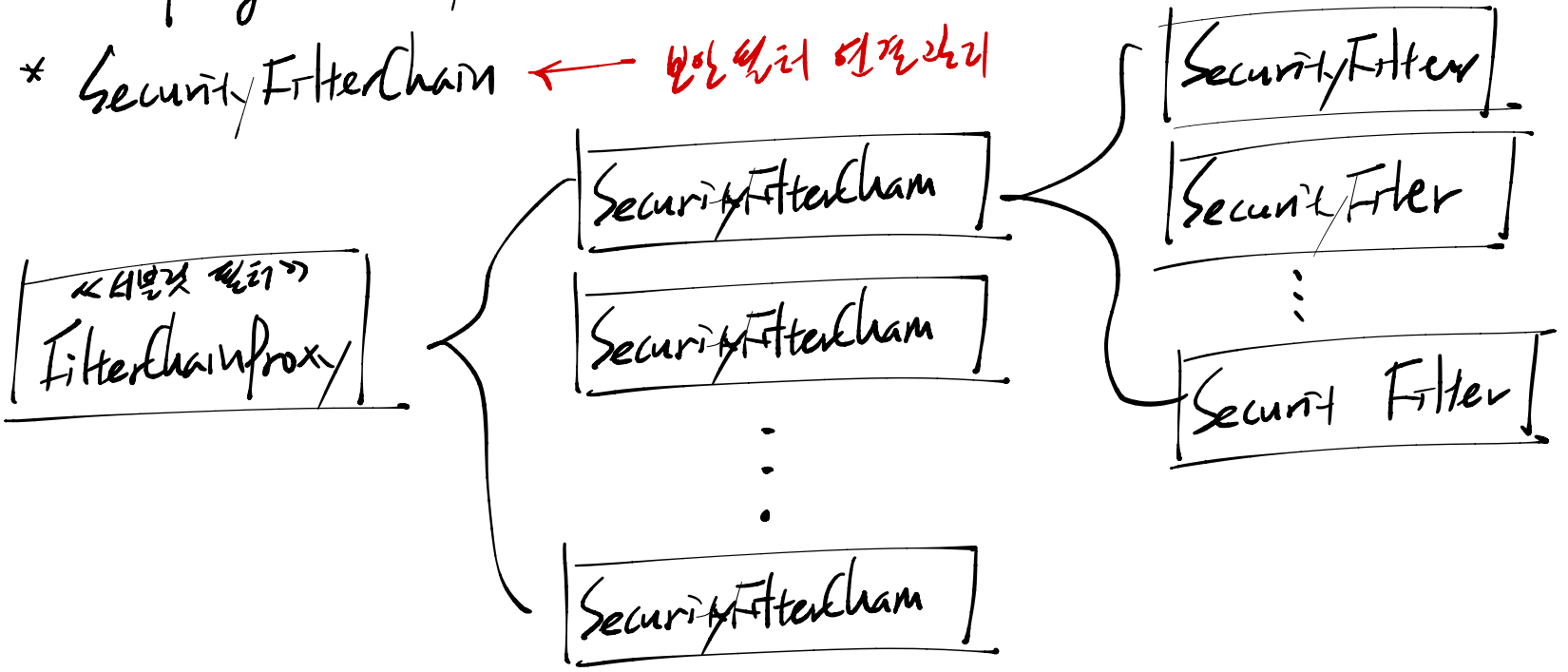
11. Spring Security 구조

* FilterChainProxy ← Spring Security 라이브러리의 서블릿 필터



n/ Spring Security 개념

* SecurityFilterChain ← 보안필터 연결관리



* HttpSecurity ← Security Filter 설정 오퍼레

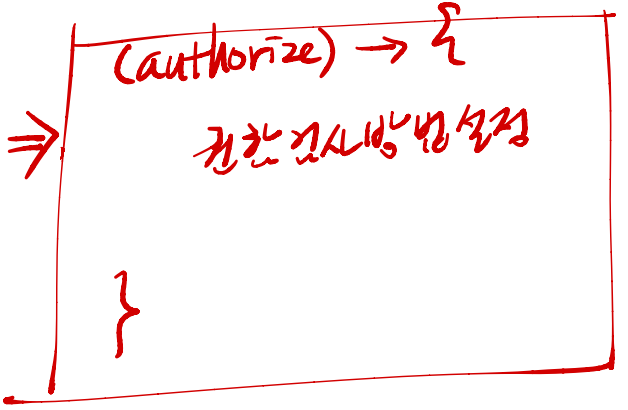
```
httpSecurity.authorizeHttpRequest( )
```

요청에 대해 실행 권한을 검사할 객체 등록

보안권한설정

요청이 들어오면 설정된 검사방법에 따라 권한검사를 수행

```
(authorize) → {
    권한검사방법설정
}
```



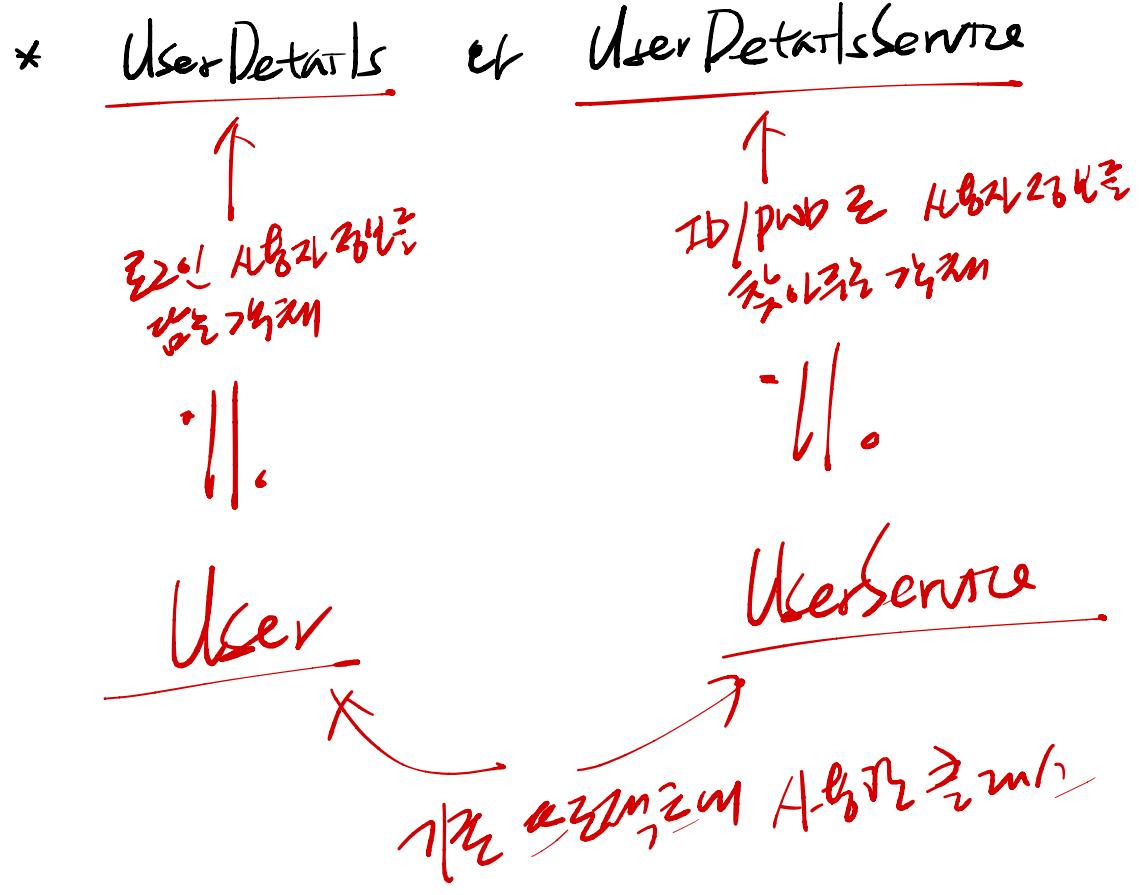
* 실행권한설정

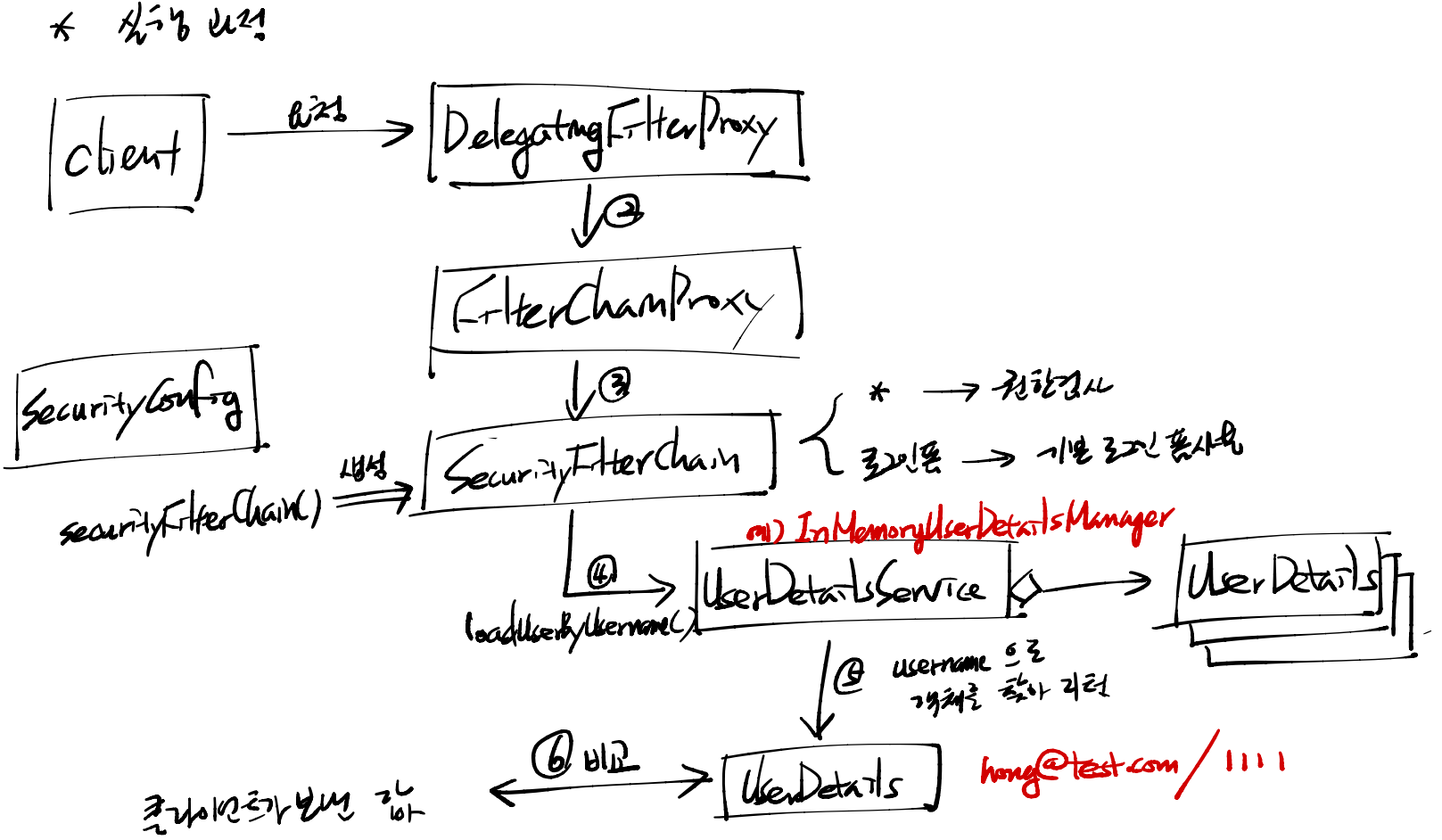
```
authorize.anyRequest().authenticated()
```

authorize → 권한설정

anyRequest() → 모든 요청

authenticated() → 로그인 된 사용자만 실행할 수 있다


* UserDetails 와 UserDetailsService
UserService


* 실행 순서
Client
요청
DelegatingFilterProxy
①
FilterChainProxy
②
SecurityConfig
생성
SecurityFilterChain
securityFilterChain()
* → 전체경로
로그인폼 → 기본 로그인 폼사용
③
④
loadUserByUsername()
UserDetailsService
예) InMemoryUserDetailsManager
UserDetails
⑤ username 으로 객체를 찾아 리턴
⑥ 비교
UserDetails
hong@test.com / 1111
클라이언트가보낸 값

* 객체 생성

① classic 방식 → ② modern 방식

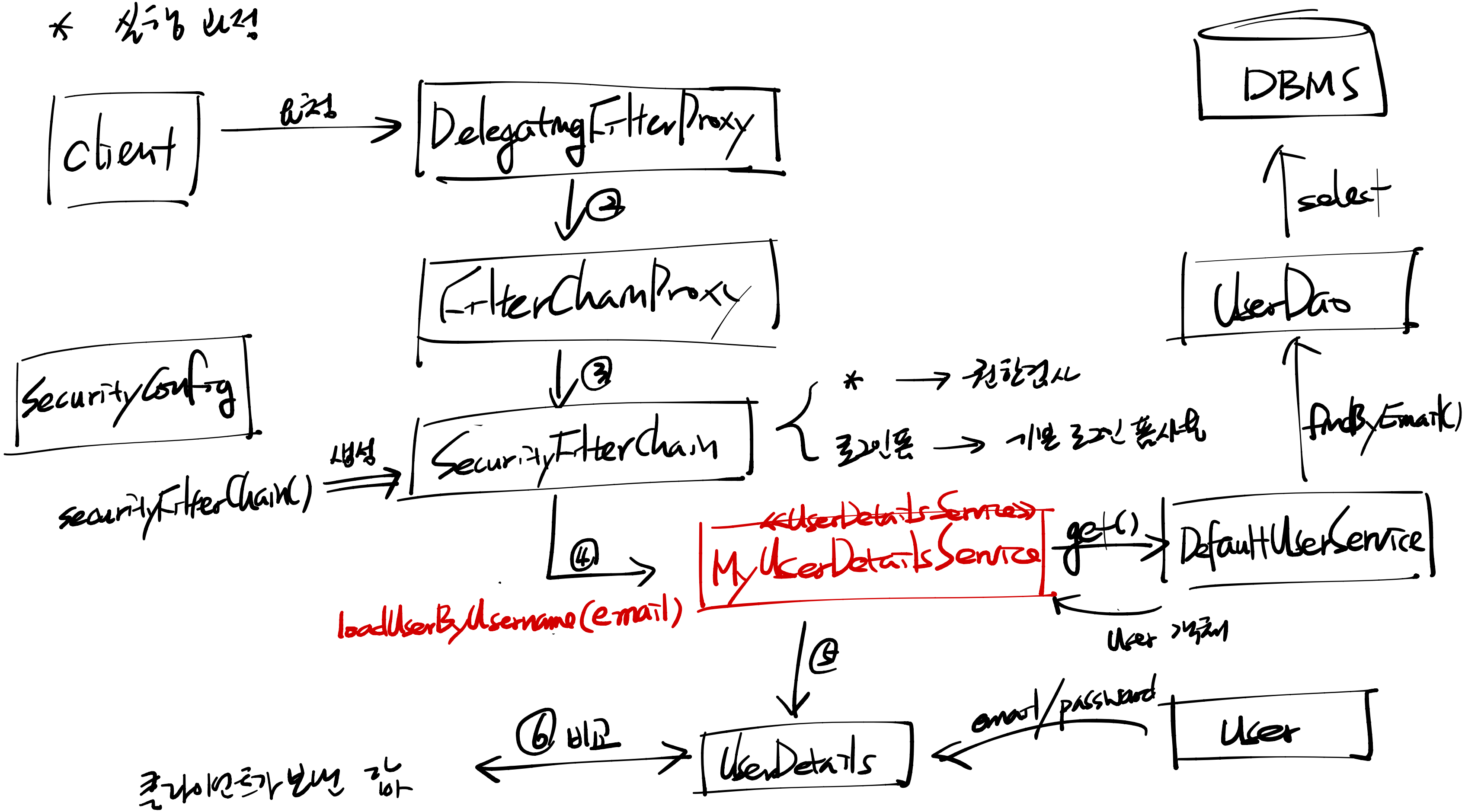
```
자동차 car = new 자동차();
car.setSunRoof();
car.setNavigation();
car.setBlackBox();
car.cotng();
car.prntLabel();
:
```

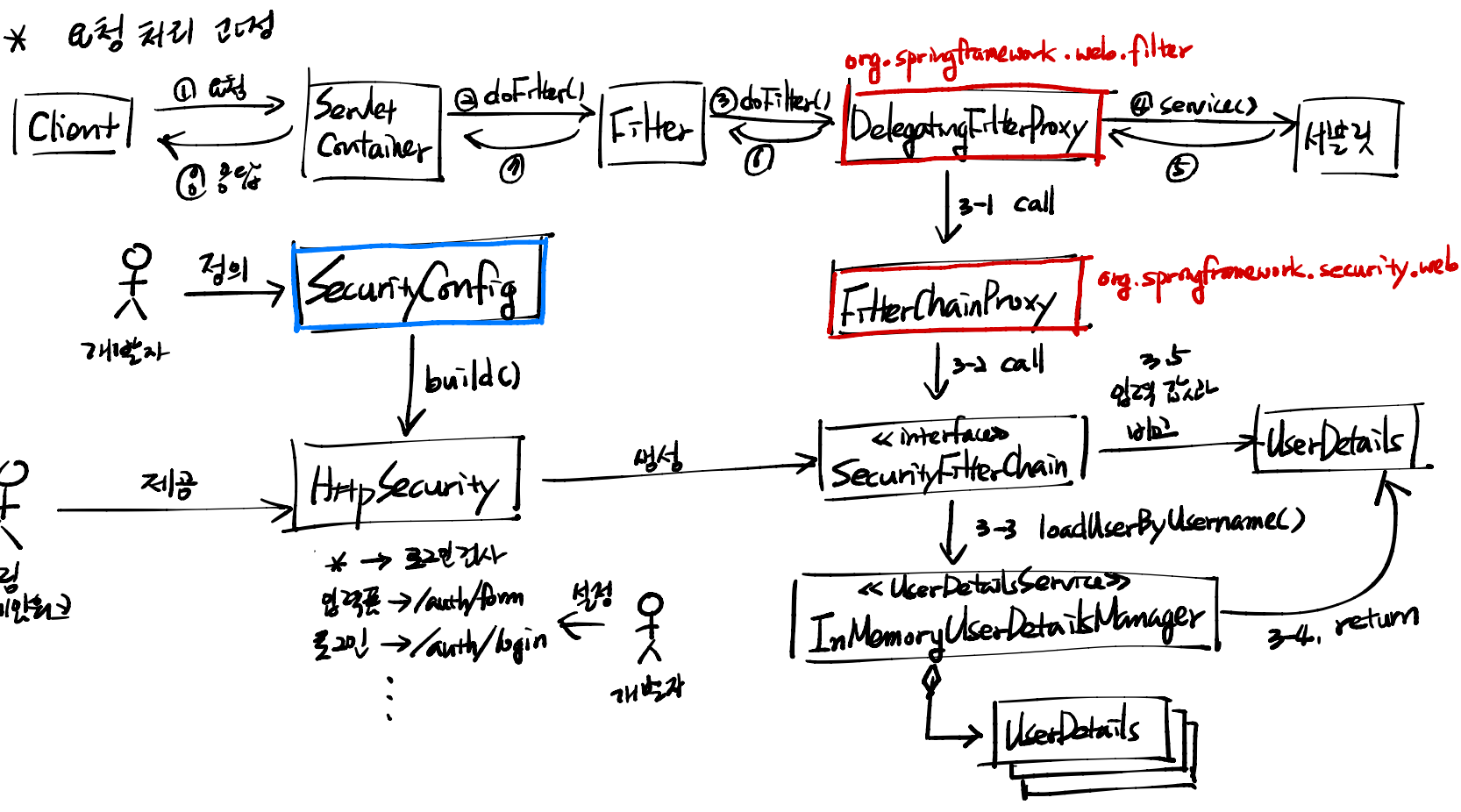
```
자동차 car = new 자동차().setSunRoof()
                          .setNavigation()
                          .setBlackBox()
                          .cotng()
                          .prntLabel();
```

return this

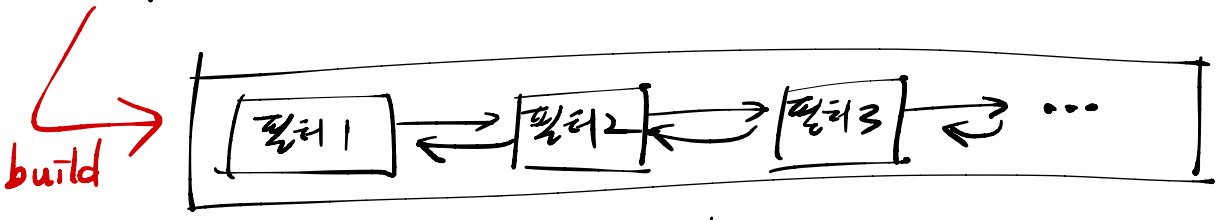
"Method Chaining"

* 실행 과정

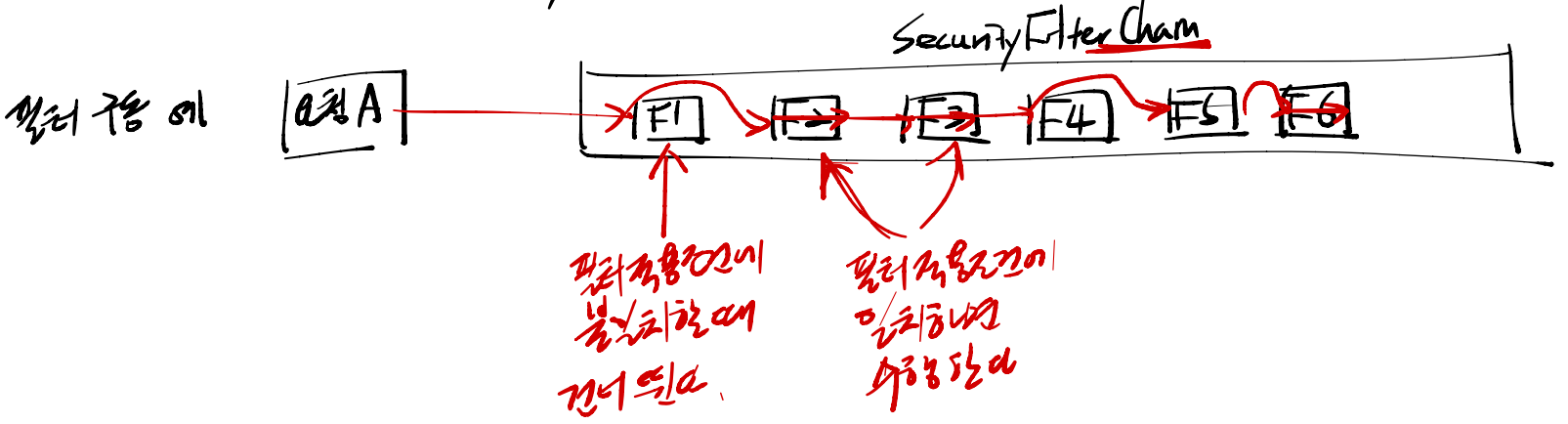

client
요청
DelegatingFilterProxy
②
FilterChainProxy
③
SecurityConfig
securityFilterChain()
생성
SecurityFilterChain
* → 권한검사
로그인폼 → 기본 로그인 폼사용
④
loadUserByUsername(email)
<UserDetailsService>
MyUserDetailsService
get()
DefaultUserService
User 가져옴
findByEmail()
UserDao
select
DBMS
⑤
User
email/password
UserDetails
⑥ 비교
클라이언트가 보낸 값


* 요청 처리 과정
org.springframework.web.filter
Client
① 요청
⑥ 응답
Servlet Container
② doFilter()
①
Filter
③ doFilter()
①
DelegatingFilterProxy
④ service()
⑤
서블릿
3-1 call
org.springframework.security.web
FilterChainProxy
3-2 call
3-5 인증 검사는 위임
<<interface>> SecurityFilterChain
UserDetails
3-3 loadUserByUsername()
<<UserDetailsService>> InMemoryUserDetailsManager
3-4. return
UserDetails
개발자
정의
SecurityConfig
build()
스프링 프레임워크
제공
HttpSecurity
생성
* → 로그인 검사
인증폼 → /auth/form
로그인 → /auth/login
설정
개발자

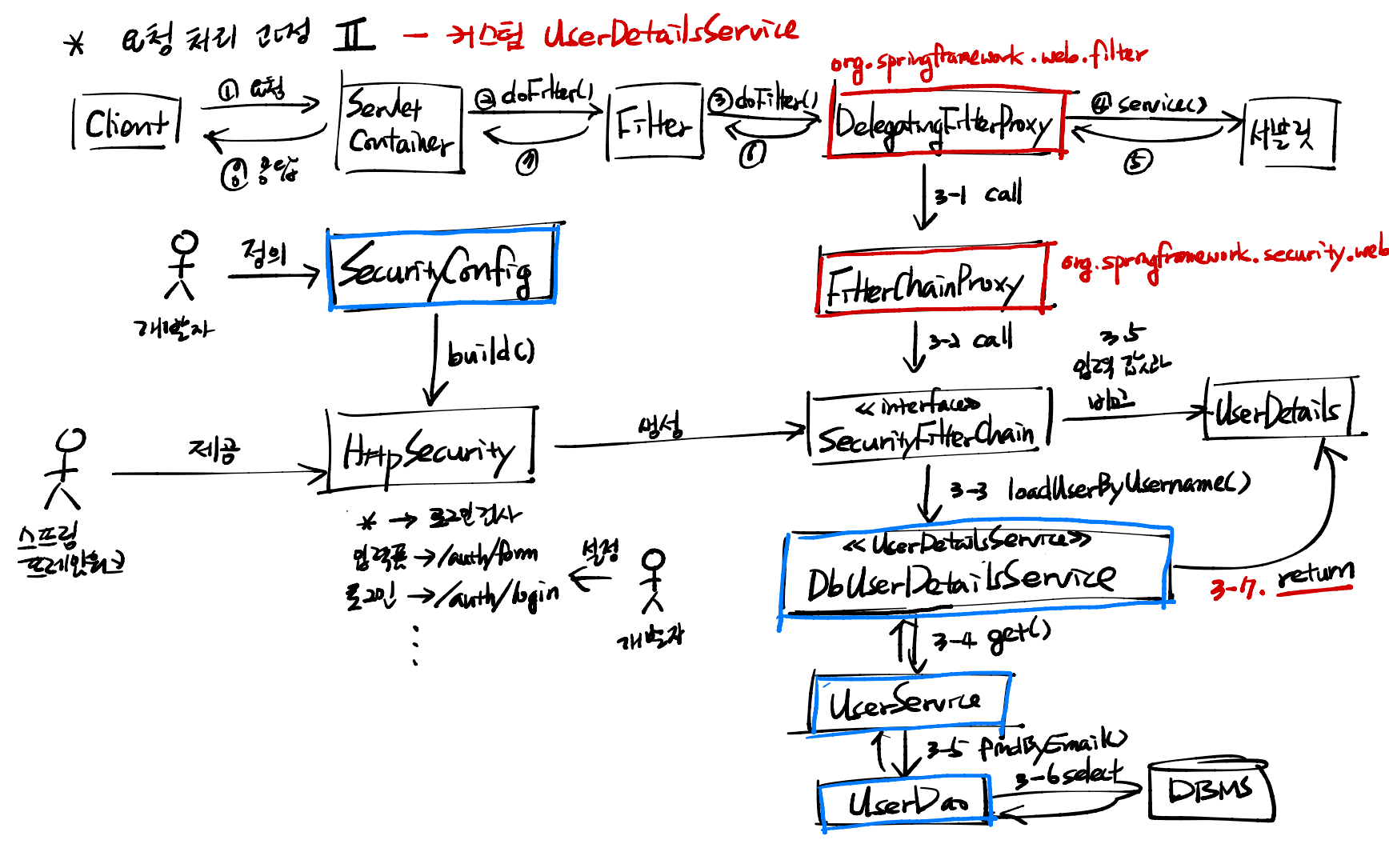
* HttpSecurity ← 개발자가 설정한대로 필터체인을 구성

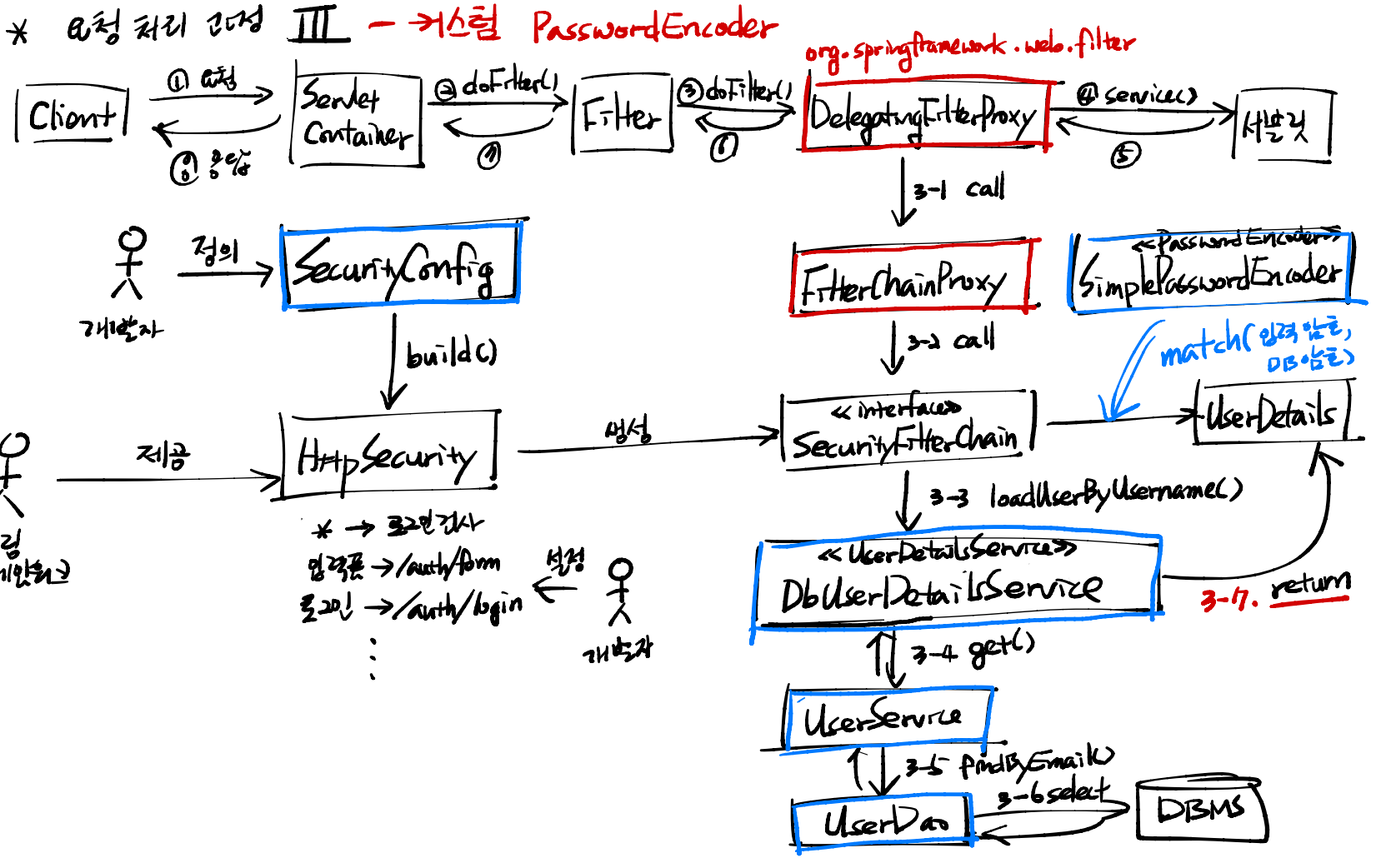


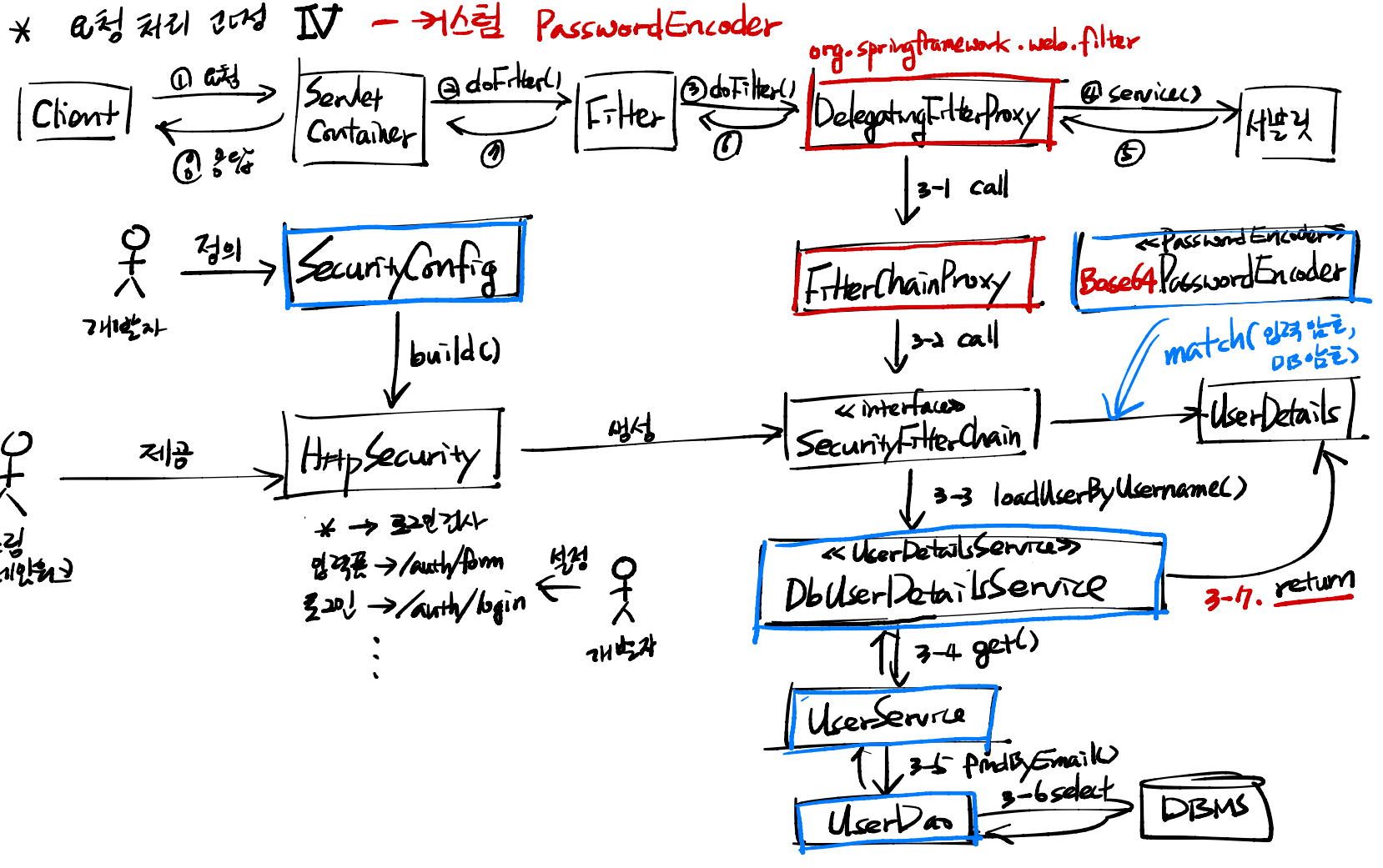
SecurityFilterChain

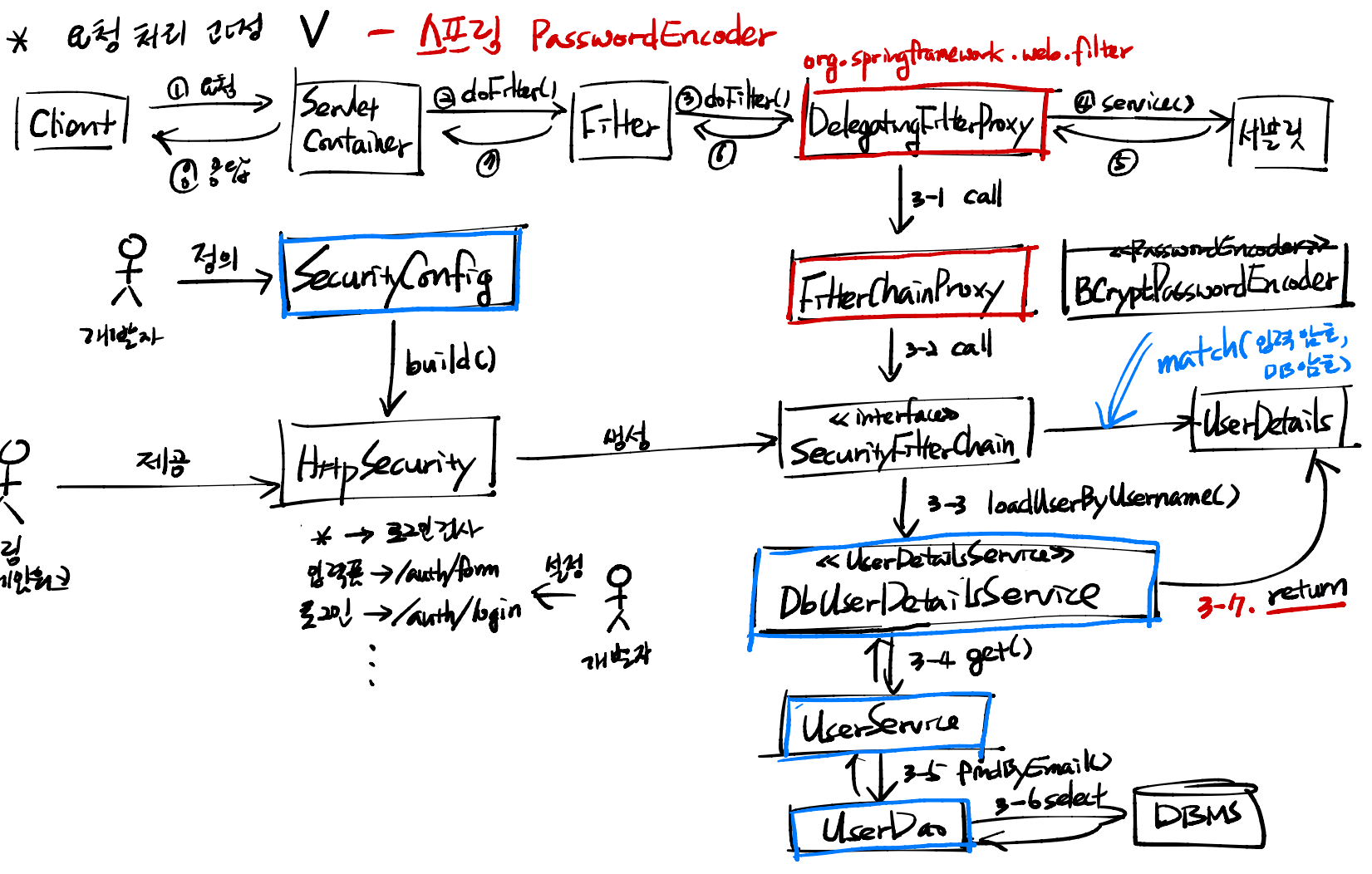


* 인증 처리 과정 II – 커스텀 UserDetailsService




* 요청 처리 과정 III - 커스텀 PasswordEncoder
org.springframework.web.filter
Client
① 요청
⑥ 응답
Servlet Container
② doFilter()
⑤
Filter
③ doFilter()
④
DelegatingFilterProxy
⑦ service()
⑧
서블릿
3-1 call
개발자
정의
SecurityConfig
build()
FilterChainProxy
3-2 call
<<PasswordEncoder>>
SimpleBCryptPasswordEncoder
match(입력 암호, DB 암호)
스프링 프레임워크
제공
HttpSecurity
생성
<<interface>>
SecurityFilterChain
UserDetails
* → 로그인 검사
입력폼 → /auth/form
로그인 → /auth/login
설정
개발자
3-3 loadUserByUsername()
<<UserDetailsService>>
DbUserDetailsService
3-7. return
3-4 get()
UserService
3-5 findByEmail()
UserDao
3-6 select
DBMS


* 요청 처리 과정 IV - 커스텀 PasswordEncoder
org.springframework.web.filter
Client
① 요청
⑥ 응답
Servlet Container
② doFilter()
⑤
Filter
③ doFilter()
④
DelegatingFilterProxy
④ service()
⑤
서블릿
3-1 call
FilterChainProxy
3-2 call
<<PasswordEncoder>>
Base64PasswordEncoder
match(입력 암호, DB 암호)
UserDetails
개발자
정의
SecurityConfig
build()
HttpSecurity
제공
스프링 프레임워크
* → 로그인 검사
인증폼 → /auth/form
로그인 → /auth/login
설정
개발자
생성
<<interface>>
SecurityFilterChain
3-3 loadUserByUsername()
<<UserDetailsService>>
DbUserDetailsService
3-7. return
3-4 get()
UserService
3-5 findByEmail()
UserDao
3-6 select
DBMS


* 요청 처리 과정 V - 스프링 PasswordEncoder
org.springframework.web.filter
Client
① 요청
⑥ 응답
Servlet Container
② doFilter()
④
Filter
③ doFilter()
⑤
DelegatingFilterProxy
⑦ service()
⑧
서블릿
3-1 call
개발자
정의
SecurityConfig
build()
FilterChainProxy
<<PasswordEncoder>>
BCryptPasswordEncoder
3-2 call
match(입력 암호, DB 암호)
제공
HttpSecurity
생성
<<interface>>
SecurityFilterChain
UserDetails
스프링 프레임워크
* → 로그인 검사
입력폼 → /auth/form
로그인 → /auth/login
설정
개발자
3-3 loadUserByUsername()
<<UserDetailsService>>
DbUserDetailsService
3-7. return
3-4 get()
UserService
3-5 findByEmail()
UserDao
3-6 select
DBMS

* Log Level

FATAL – 시스템을 더이상 작동하게 할 수 없는 오류가 발생할 때.
ex) log.fatal("오류 메시지");
↳ DB 연결 실패, 메모리 부족

ERROR – 특정 기능 오류 발생, 그러나 시스템 실행에는 문제 없을 경우.
ex) log.error("오류 메시지");
↳ 요청 파라미터 에러와 같은 오류. 날짜형식오류

WARN – 잠재적으로 문제를 발생시킬 수 있는 경우,
↳ deprecated 된 라이브러리나 API 사용

INFO – 시스템 관리자가 실행 상태를 확인할 수 있도록 정보를 출력하는 것
↳ 클라이언트의 IP 주소, 요청 URL 등

DEBUG – 개발과 디버깅에 필요한 정보를 출력할 때
↳ 변수 값, 호출 메서드 명 등

TRACE – 상세 모니터링이나 실행 경로를 추적할 때 사용할 레벨 출력