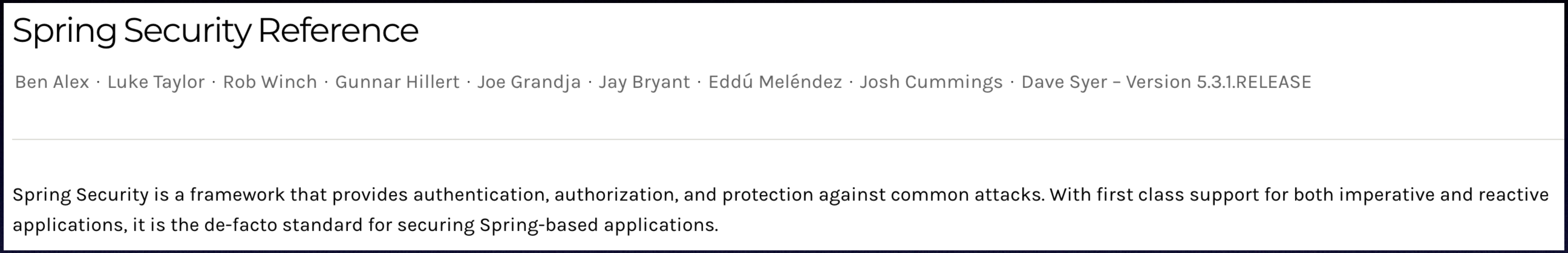
Spring Boot 5장

양시영

Spring Security


Spring Security Reference

Ben Alex · Luke Taylor · Rob Winch · Gunnar Hillert · Joe Grandja · Jay Bryant · Eddú Meléndez · Josh Cummings · Dave Syer – Version 5.3.1.RELEASE

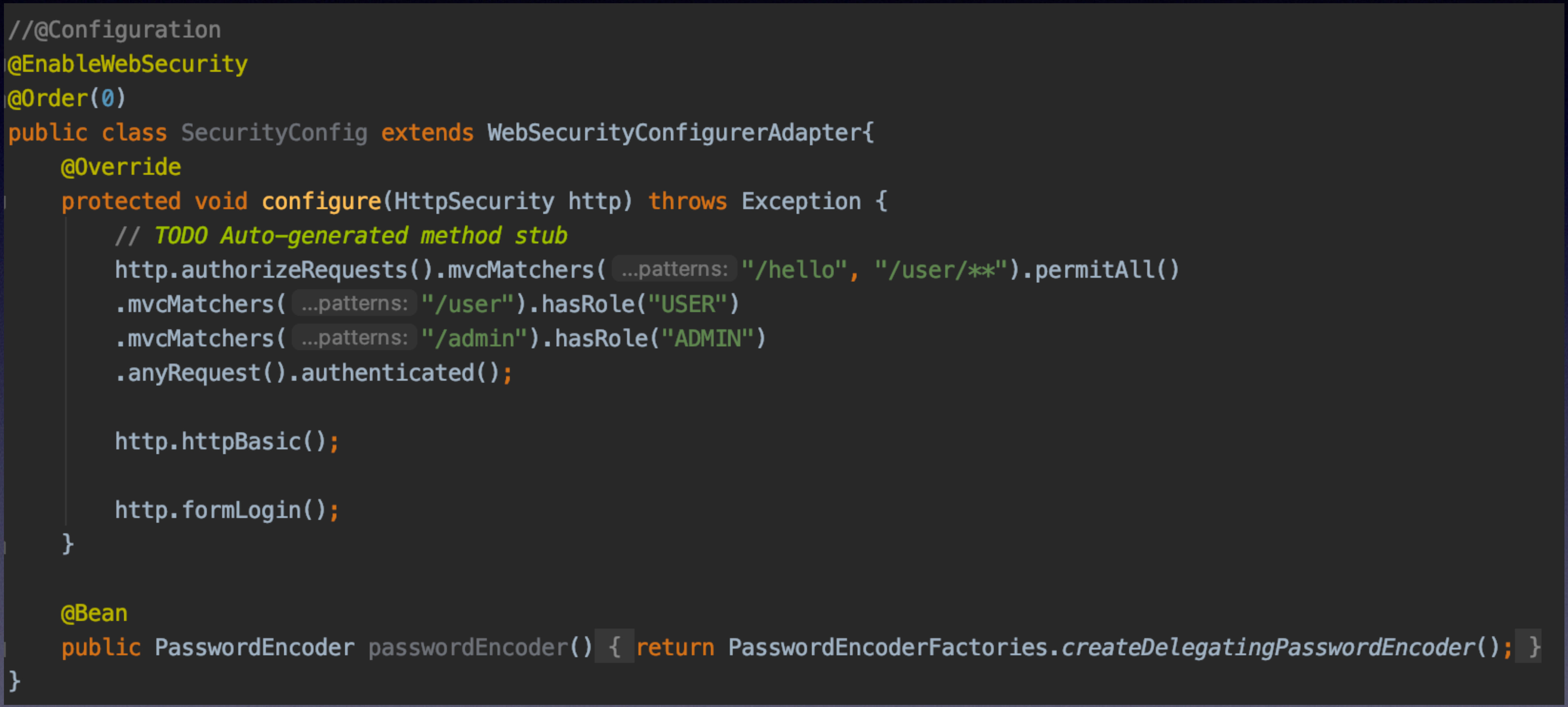
Spring Security is a framework that provides authentication, authorization, and protection against common attacks. With first class support for both imperative and reactive applications, it is the de-facto standard for securing Spring-based applications.

슾흐링 시큐이티는 인증과 인가 그리고
대부분의 공격에 대한 방어책을 제공하며…
슾흐링 기반 서비스의 보안 표준이라고 한다

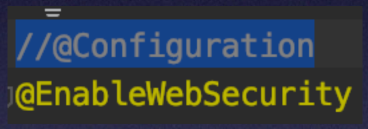
인증? 인가?

는 내 세션 주제이기 때문에 빔일…

Spring Security 기본 설정

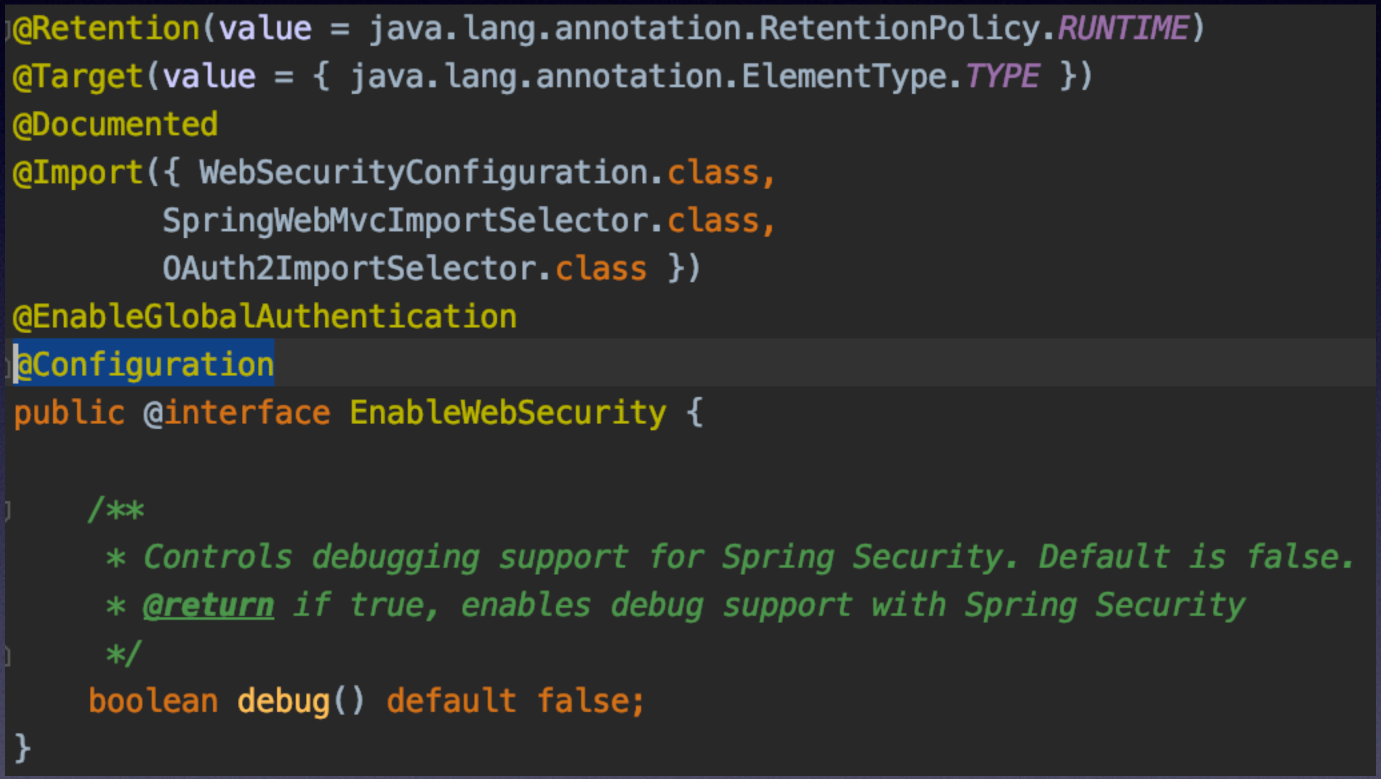


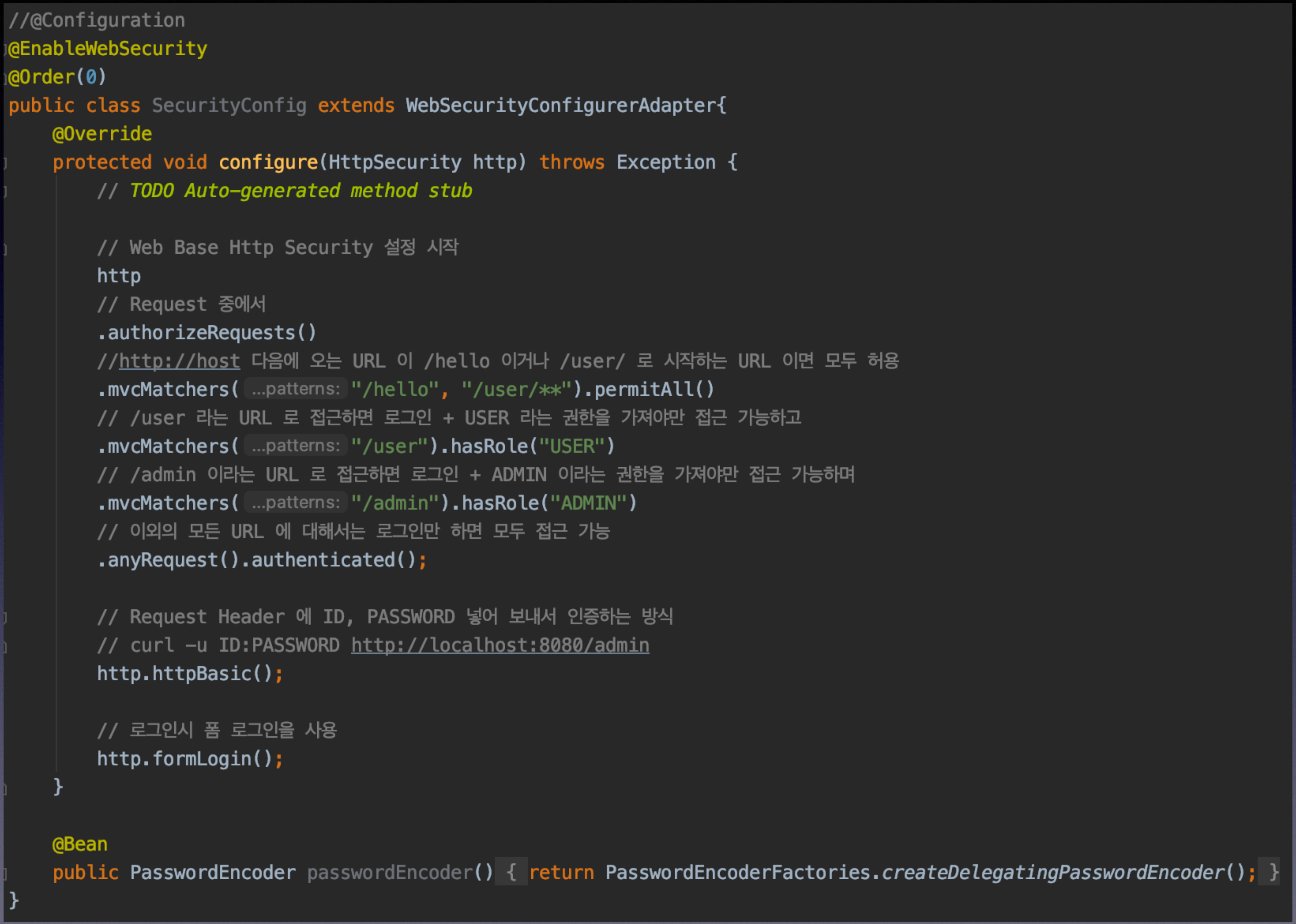
```
//@Configuration
@EnableWebSecurity
@Order(0)
public class SecurityConfig extends WebSecurityConfigurerAdapter{
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // TODO Auto-generated method stub
        http.authorizeRequests().mvcMatchers( ...patterns: "/hello", "/user/**").permitAll()
        .mvcMatchers( ...patterns: "/user").hasRole("USER")
        .mvcMatchers( ...patterns: "/admin").hasRole("ADMIN")
        .anyRequest().authenticated();

        http.httpBasic();

        http.formLogin();
    }

    @Bean
    public PasswordEncoder passwordEncoder() { return PasswordEncoderFactories.createDelegatingPasswordEncoder(); }
}
```



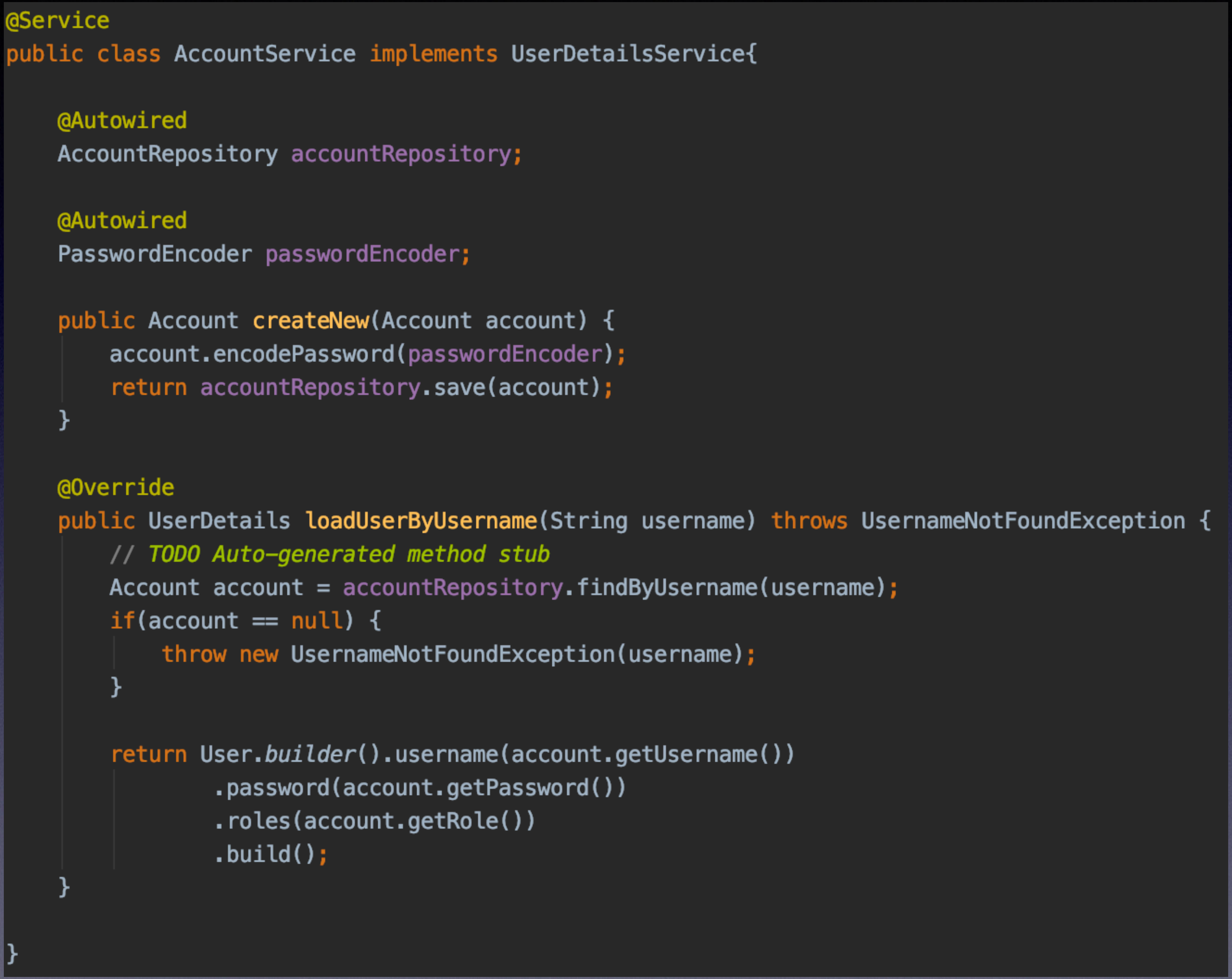
```
//@Configuration
@EnableWebSecurity
```

@Configuration 주석 한 이유

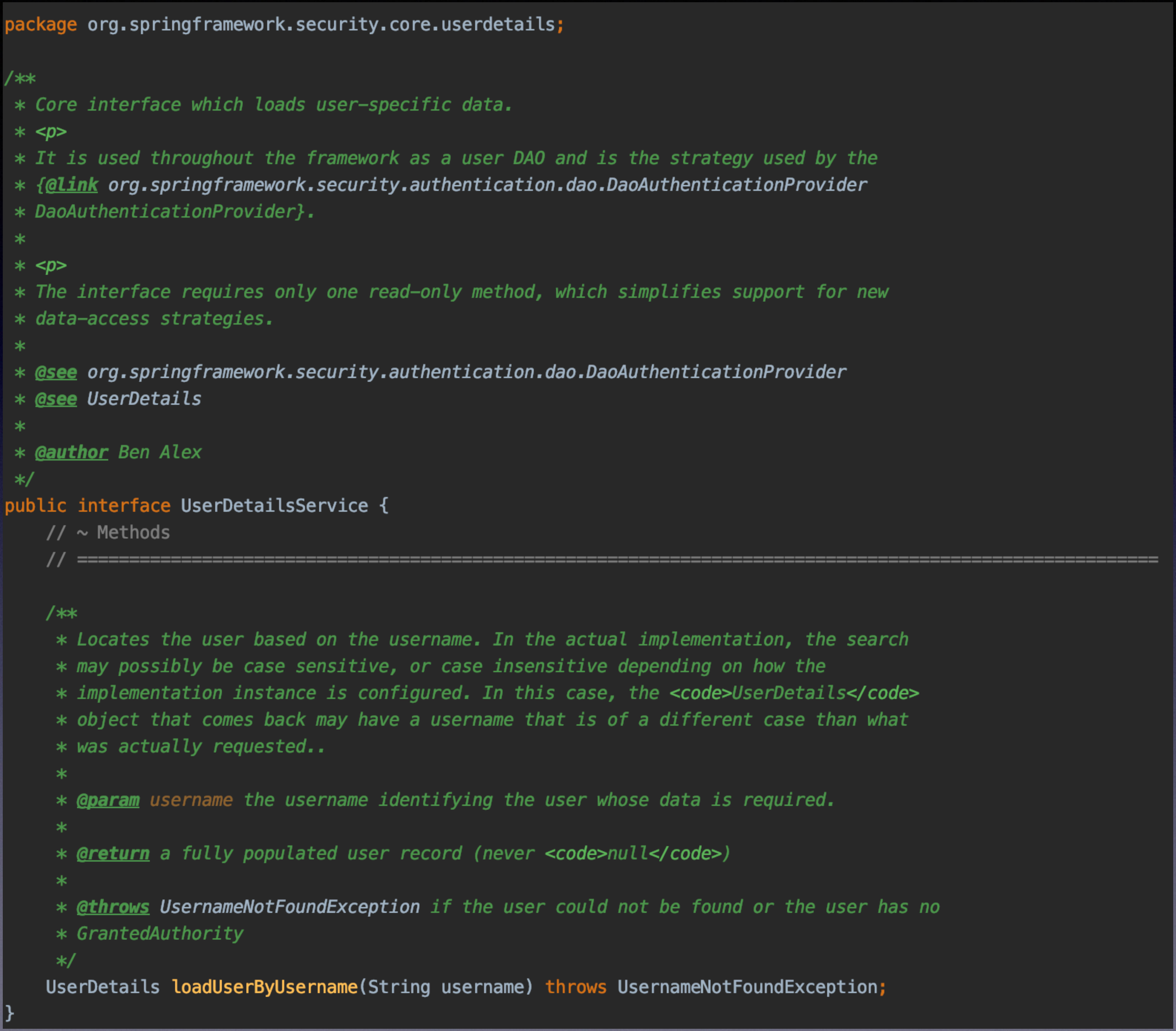

```java
@Retention(value = java.lang.annotation.RetentionPolicy.RUNTIME)
@Target(value = { java.lang.annotation.ElementType.TYPE })
@Documented
@Import({ WebSecurityConfiguration.class,
        SpringWebMvcImportSelector.class,
        OAuth2ImportSelector.class })
@EnableGlobalAuthentication
@Configuration
public @interface EnableWebSecurity {

    /**
     * Controls debugging support for Spring Security. Default is false.
     * @return if true, enables debug support with Spring Security
     */
    boolean debug() default false;
}
```



```
//@Configuration
@EnableWebSecurity
@Order(0)
public class SecurityConfig extends WebSecurityConfigurerAdapter{
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // TODO Auto-generated method stub

        // Web Base Http Security 설정 시작
        http
        // Request 중에서
        .authorizeRequests()
        //http://host 다음에 오는 URL 이 /hello 이거나 /user/ 로 시작하는 URL 이면 모두 허용
        .mvcMatchers( ...patterns: "/hello", "/user/**").permitAll()
        // /user 라는 URL 로 접근하면 로그인 + USER 라는 권한을 가져야만 접근 가능하고
        .mvcMatchers( ...patterns: "/user").hasRole("USER")
        // /admin 이라는 URL 로 접근하면 로그인 + ADMIN 이라는 권한을 가져야만 접근 가능하며
        .mvcMatchers( ...patterns: "/admin").hasRole("ADMIN")
        // 이외의 모든 URL 에 대해서는 로그인만 하면 모두 접근 가능
        .anyRequest().authenticated();

        // Request Header 에 ID, PASSWORD 넣어 보내서 인증하는 방식
        // curl -u ID:PASSWORD http://localhost:8080/admin
        http.httpBasic();

        // 로그인시 폼 로그인을 사용
        http.formLogin();
    }

    @Bean
    public PasswordEncoder passwordEncoder() { return PasswordEncoderFactories.createDelegatingPasswordEncoder(); }
}
```



```
@Service
public class AccountService implements UserDetailsService{

    @Autowired
    AccountRepository accountRepository;

    @Autowired
    PasswordEncoder passwordEncoder;

    public Account createNew(Account account) {
        account.encodePassword(passwordEncoder);
        return accountRepository.save(account);
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // TODO Auto-generated method stub
        Account account = accountRepository.findByUsername(username);
        if(account == null) {
            throw new UsernameNotFoundException(username);
        }

        return User.builder().username(account.getUsername())
                .password(account.getPassword())
                .roles(account.getRole())
                .build();
    }

}
```

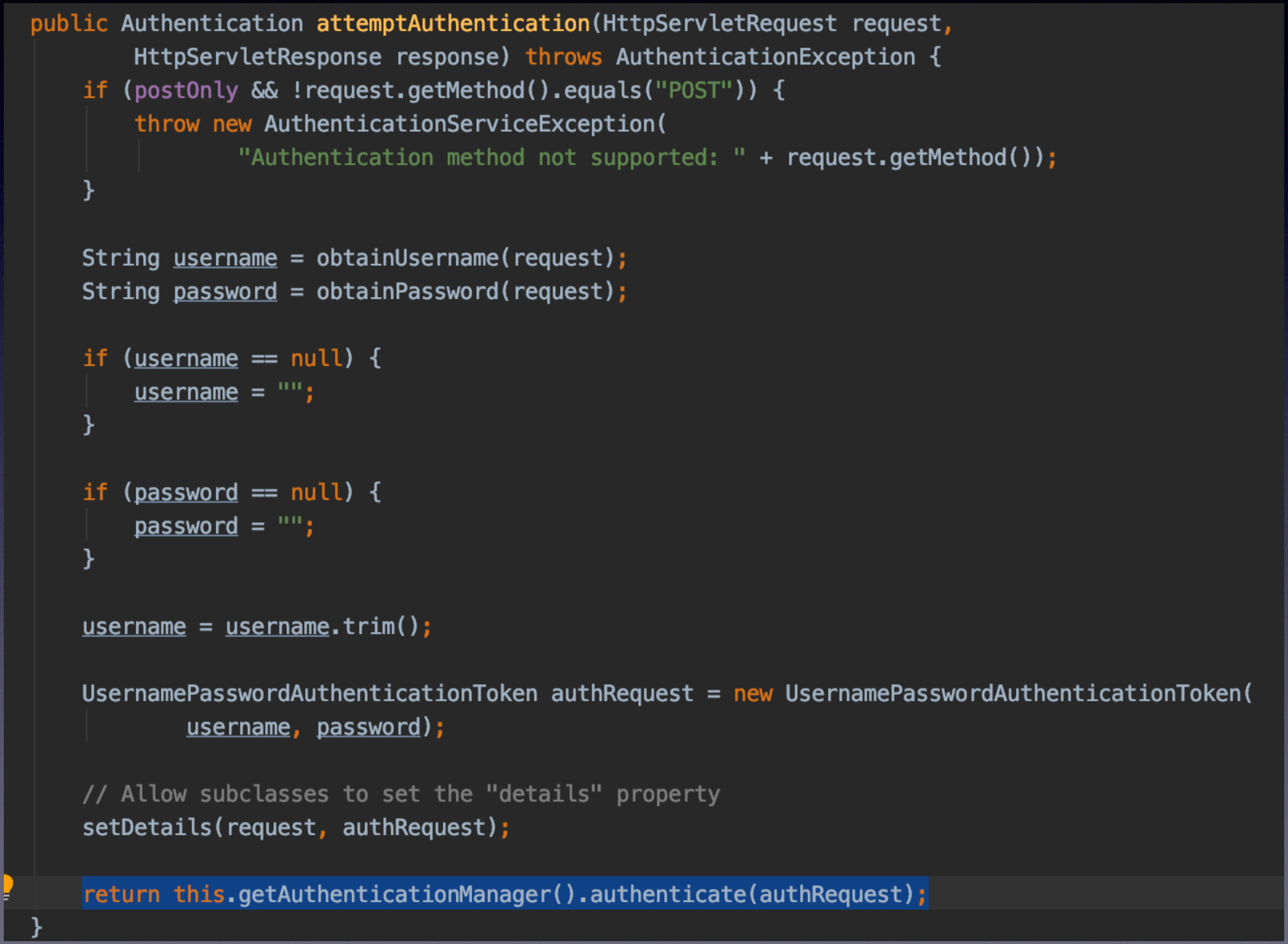


```java
package org.springframework.security.core.userdetails;

/**
 * Core interface which loads user-specific data.
 * <p>
 * It is used throughout the framework as a user DAO and is the strategy used by the
 * {@link org.springframework.security.authentication.dao.DaoAuthenticationProvider
 * DaoAuthenticationProvider}.
 *
 * <p>
 * The interface requires only one read-only method, which simplifies support for new
 * data-access strategies.
 *
 * @see org.springframework.security.authentication.dao.DaoAuthenticationProvider
 * @see UserDetails
 *
 * @author Ben Alex
 */
public interface UserDetailsService {
    // ~ Methods
    // ========================================================================================================

    /**
     * Locates the user based on the username. In the actual implementation, the search
     * may possibly be case sensitive, or case insensitive depending on how the
     * implementation instance is configured. In this case, the <code>UserDetails</code>
     * object that comes back may have a username that is of a different case than what
     * was actually requested..
     *
     * @param username the username identifying the user whose data is required.
     *
     * @return a fully populated user record (never <code>null</code>)
     *
     * @throws UsernameNotFoundException if the user could not be found or the user has no
     * GrantedAuthority
     */
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}
```

1. WebAsyncManagerIntergrationFilter
2. SecurityContextPersistenceFilter
3. HeaderWriterFilter
4. CsrfFilter
5. LogoutFilter
6. UsernamePasswordAuthenticationFilter
7. DefaultLoginPageGeneratingFilter
8. DefaultLogoutPageGeneratingFilter
9. BasicAuthenticationFilter
10. RequestCacheAwareFtiler
11. SecurityContextHolderAwareReqeustFilter
12. AnonymousAuthenticationFilter
13. SessionManagementFilter
14. ExeptionTranslationFilter
15. FilterSecurityInterceptor

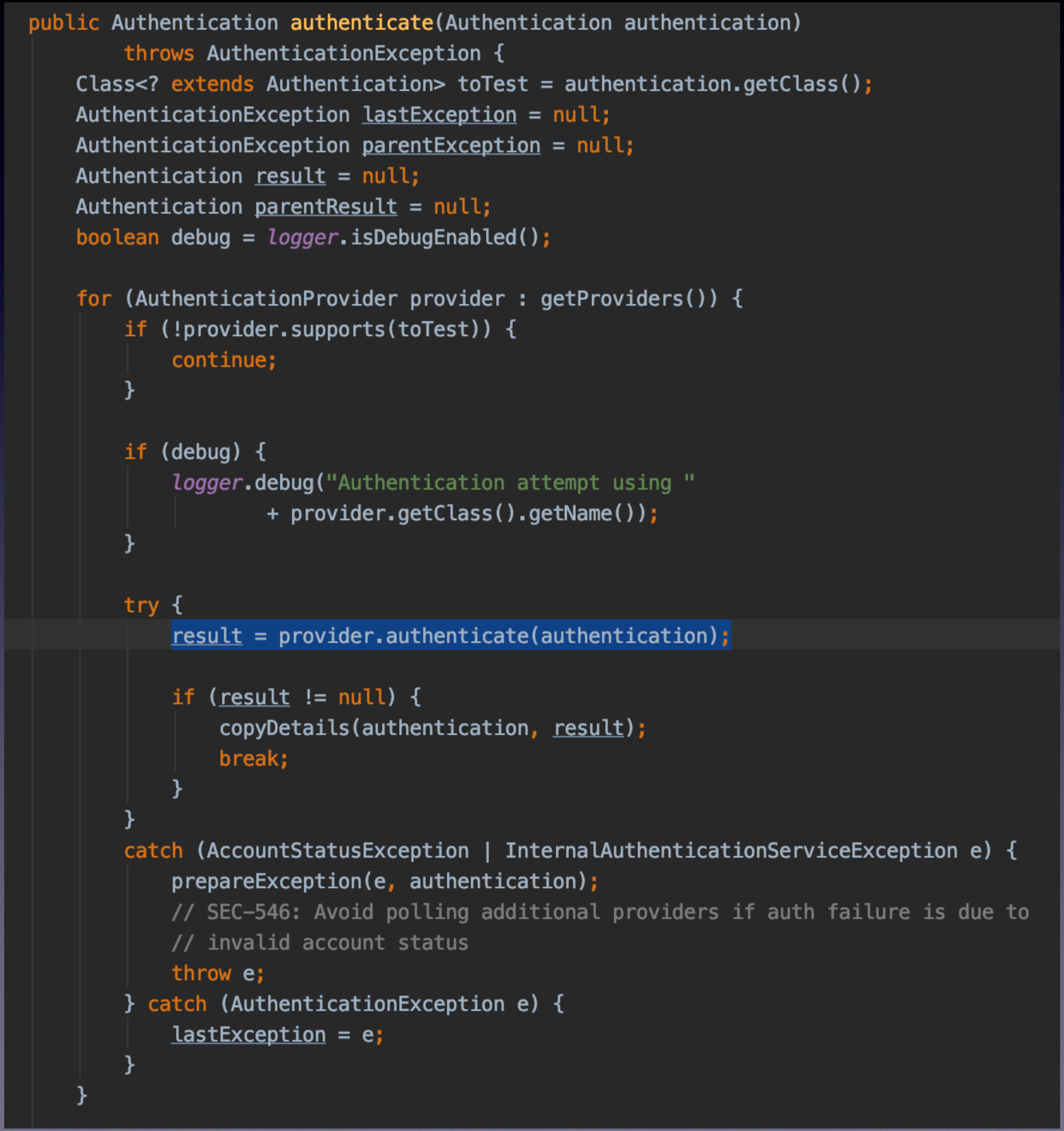
UsernamePasswordAuthenticationFilter

UsernamePasswordAuthenticationFilter



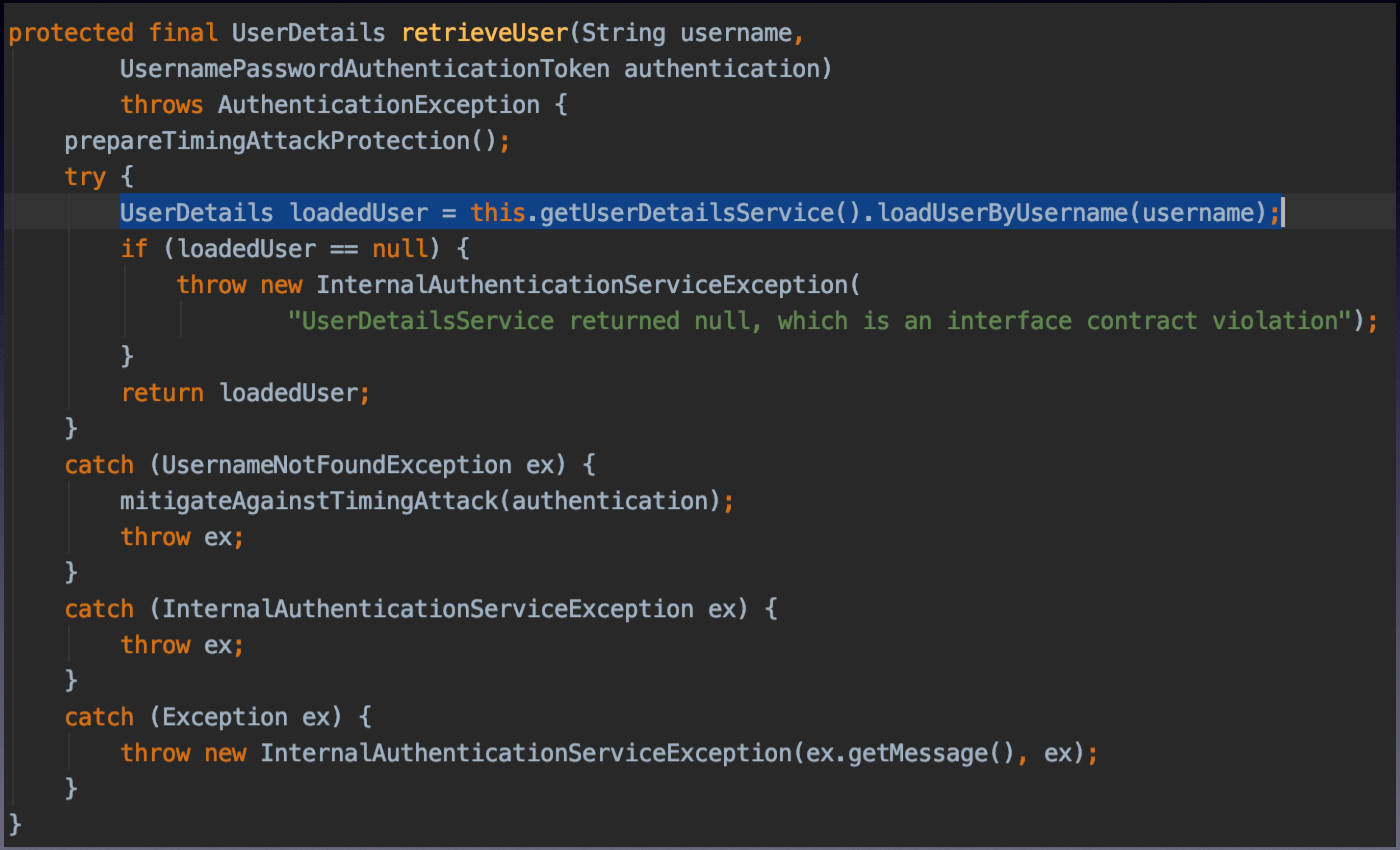
```java
public Authentication attemptAuthentication(HttpServletRequest request,
        HttpServletResponse response) throws AuthenticationException {
    if (postOnly && !request.getMethod().equals("POST")) {
        throw new AuthenticationServiceException(
                "Authentication method not supported: " + request.getMethod());
    }

    String username = obtainUsername(request);
    String password = obtainPassword(request);

    if (username == null) {
        username = "";
    }

    if (password == null) {
        password = "";
    }

    username = username.trim();

    UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(
            username, password);

    // Allow subclasses to set the "details" property
    setDetails(request, authRequest);

    return this.getAuthenticationManager().authenticate(authRequest);
}
```

UsernamePasswordAuthenticationFilter
-> ProviderManager(implements AuthenticationManager)



```java
public Authentication authenticate(Authentication authentication)
        throws AuthenticationException {
    Class<? extends Authentication> toTest = authentication.getClass();
    AuthenticationException lastException = null;
    AuthenticationException parentException = null;
    Authentication result = null;
    Authentication parentResult = null;
    boolean debug = logger.isDebugEnabled();

    for (AuthenticationProvider provider : getProviders()) {
        if (!provider.supports(toTest)) {
            continue;
        }

        if (debug) {
            logger.debug("Authentication attempt using "
                    + provider.getClass().getName());
        }

        try {
            result = provider.authenticate(authentication);

            if (result != null) {
                copyDetails(authentication, result);
                break;
            }
        }
        catch (AccountStatusException | InternalAuthenticationServiceException e) {
            prepareException(e, authentication);
            // SEC-546: Avoid polling additional providers if auth failure is due to
            // invalid account status
            throw e;
        } catch (AuthenticationException e) {
            lastException = e;
        }
    }
```

UsernamePasswordAuthenticationFilter
-> ProviderManager(implements AuthenticationManager)
-> DaoAuthenticationProvider



```java
protected final UserDetails retrieveUser(String username,
        UsernamePasswordAuthenticationToken authentication)
        throws AuthenticationException {
    prepareTimingAttackProtection();
    try {
        UserDetails loadedUser = this.getUserDetailsService().loadUserByUsername(username);
        if (loadedUser == null) {
            throw new InternalAuthenticationServiceException(
                    "UserDetailsService returned null, which is an interface contract violation");
        }
        return loadedUser;
    }
    catch (UsernameNotFoundException ex) {
        mitigateAgainstTimingAttack(authentication);
        throw ex;
    }
    catch (InternalAuthenticationServiceException ex) {
        throw ex;
    }
    catch (Exception ex) {
        throw new InternalAuthenticationServiceException(ex.getMessage(), ex);
    }
}
```

결론

책 내용도 너무 좋지만

그냥.. 왜 쓰고 어떻게 돌아가는지 알고 쓰자는..
말입니다..