Chpater4_4

1. 사용자 인지하기

OrderController에서 (즉, 주문할 때) 주문 폼에 바인딩 되는 Order 객체를 최초 생성할 때 해당 주문을 하는 사용자의 이름과 주소를 주문 폼에 미리 넣을 수 있다면 효과적

사용자 주문 데이터를 데이터베이스에 저장할 때 주문이 생성되는 User와 Order를 연관시킬 수 있어야 함

<Order Class>

```
@Data
@Entity
@Table(name="Taco_Order")
public class Order implements Serializable {
        private static final long serialVersionUID = 1L;
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        private Long id;
        private Date placedAt;
        @ManyToOne
        private User user;

    ....
```

- @ManyToOne
  - 한 건의 주문이 한 명의 사용자에게 속한다는 것을 나타냄
  - Order : User = 多 : 1

<OrderController Class>

```
@Slf4j
@Controller
@RequestMapping("/orders")
@SessionAttributes("order")
public class OrderController {
        private OrderRepository orderRepo;
        public OrderController(OrderRepository orderRepo) {
                this.orderRepo = orderRepo;
        }
        @GetMapping("/current")
        public String orderForm(@AuthenticationPrincipal User user,
                        @ModelAttribute Order order) {
                if (order.getDeliveryName() == null) {
                        order.setDeliveryName(user.getFullname());
```

```
                }
                if (order.getDeliveryStreet() == null) {
                        order.setDeliveryStreet(user.getStreet());
                }
                if (order.getDeliveryCity() == null) {
                        order.setDeliveryCity(user.getCity());
                }
                if (order.getDeliveryState() == null) {
                        order.setDeliveryState(user.getState());
                }
                if (order.getDeliveryZip() == null) {
                        order.setDeliveryZip(user.getZip());
                }
                return "orderForm";
        }
        @PostMapping
        public String processOrder(@Valid Order order, Errors errors, SessionStatus
sessionStatus
                                                                , @AuthenticationPrincipal User
user) {
                if (errors.hasErrors()) {
                        return "orderForm";
                }
                order.setUser(user);
                orderRepo.save(order);
                sessionStatus.setComplete();
                return "redirect:/";
        }
}
```

- processOrder()
  - 주문을 저장하는 일을 수행
  - 인증된 사용자가 누군지 결정한 후 Order 객체의 setUser()를 호출해서 해당 유저와 연결해야함
- orderForm()
  - @AuthenticationPrincipal User user를 통해 user를 통해서 사용자 정보를 가져올 수 있음
  - get 함수를 통해 받을수 있는 값은 받아옴(이름, 주소, 도시, State...)

사용자(User) 결정 법

로그인 한 사용자의 정보를 받고 싶을때는 기본적으로 Principal 객체로 받아서 사용

- Principal 객체를 컨트롤러 메서드에 주입

```
@PostMapping
```

```
        public String processOrder(@Valid Order order, Errors errors, SessionStatus
sessionStatus
                                                        ,Principal principal) {
                if (errors.hasErrors()) {
                        return "orderForm";
                }
                User user = userRepository.findByUsername(principal.getName());
                order.setUser(user);

                orderRepo.save(order);
                sessionStatus.setComplete();
                return "redirect:/";
        }
```

- Authentication 객체를 컨트롤러 메서드에 주입

```
@PostMapping
        public String processOrder(@Valid Order order, Errors errors, SessionStatus
sessionStatus
                                                        , Authentication
authentication) {
                if (errors.hasErrors()) {
                        return "orderForm";
                }
                User user = (User) authentication.getPrincipal():

                order.setUser(user);
                orderRepo.save(order);
                sessionStatus.setComplete();
                return "redirect:/";
        }
```

↑위의 방법들은 잘 동작은 하지만 보안과 관련없는 코드도 존재하게 됨

- SecurityContextHolder를 사용해서 보안 컨텍스트를 얻음
  - 보안 특정 코드가 길지만 컨트롤러 처리 메서드는 물론이고 애플리케이션의 어디서든 사용가능

```
@PostMapping
        public String processOrder(@Valid Order order, Errors errors, SessionStatus
sessionStatus
                                                        , Authentication
authentication) {
                if (errors.hasErrors()) {
                        return "orderForm";
                }
```

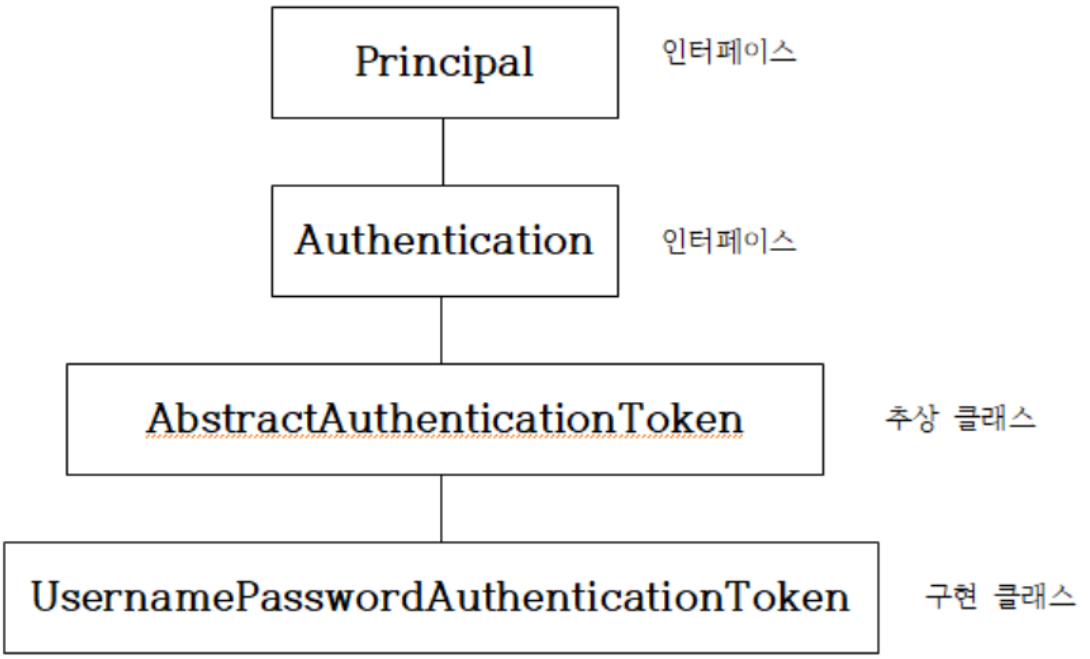
```
                Authentication authentication = SecurityContextHolder.getContext()
////
                        .getAuthentication();
////
                User user = (User) authentication.getPrincipal():
////

                order.setUser(user);
                orderRepo.save(order);
                sessionStatus.setComplete();
                return "redirect:/";
        }
```

- @AuthenticationPrincipal 어노테이션을 메서드에 지정
  - User객체를 인자로 전달 가능
  - @AuthenticationPrincipal
    - 타입 변환이 필요 없고 Authentication과 동일하게 보안 특정 코드만 가짐
    - **UserDetailsService에서 Return한 객체** 를 파라메터로 직접 받아 사용

```
@PostMapping
        public String processOrder(@Valid Order order, Errors errors, SessionStatus
sessionStatus
                                                                , @AuthenticationPrincipal User
user) {
                if (errors.hasErrors()) {
                        return "orderForm";
                }
                order.setUser(user);
                orderRepo.save(order);
                sessionStatus.setComplete();
                return "redirect:/";
        }
```

<추가 정보>



※ Principal 객체

컨트롤러의 처리기 메소드에서 자동 파라미터로 주입받을 수 있는 타입 중 하나 다만 가장 구현체의 최상위 인터페이스이기 때문에 이 타입으로 받으면 사용할 만한 메소드가 getName() 정도(ID 정보만 가져다 사용할 수 있다고 보면 됨) UserDetailsService에 의해 반환된 UserDetails의 instance.

```
@RequestMapping("/")
        public String main(Principal principal) {

                if (principal != null) {
                        System.out.println("타입정보 : " + principal.getClass());
                        System.out.println("ID정보 : " + principal.getName());
                }
                return "main";
        }
```

※ Authentication 객체

UsernamePasswordAuthenticationToken 구현체 및 세션 정보를 보관하는 객체에서 필요한 정보를 뽑아내는 메소드를 가지고 있음따라서 실제로 인증 정보를 사용하기 위해 사용되는 객체 타입이 바로 Authentication

AuthenticationManager.authenticate(Authentication)에 의해 인증된 principal 또는 token
Principal, credentials, authorities을 가지고 있으며 이 3가지를 통해 확인이 가능

- **Principal**
  user를 식별 하며 '누구?'에 대한 정보
  UserDetailsService에 의해 반환된 UserDetails의 instance

- **authorities**
  user에게 부여된 권한 (GrantedAuthority 참고) ex) ROLE_ADMINISTRATOR, ROLE_HR_SUPERVISOR와, ROLE_USER 등등..
- **credentials**
  주체가 올바르다는 것을 증명하는 자격 증명
  일반적으로 암호이지만 인증 관리자와 관련된 암호일 수 있다

메서드

- Object getPrincipal() : 첫 번째 생성자로 주입한 객체 반환
- Object getCredentials() : 두 번째 생성자로 주입한 객체 반환
- Collection<? extends GrantedAuthority> getAuthorities() : 세 번째 생성자인 권한 리스트 객체 반환
- Object getDetails() : 세션정보를 가진 WebAuthenticationDetails 객체 반환

```
@RequestMapping("/")
        public String main(Authentication authentication) {

                if (authentication != null) {
                        System.out.println("타입정보 : " + authentication.getClass());
                        // 세션 정보 객체 반환
                        WebAuthenticationDetails web =
(WebAuthenticationDetails)authentication.getDetails();
                        System.out.println("세션ID : " + web.getSessionId());
                        System.out.println("접속IP : " + web.getRemoteAddress());

                        // UsernamePasswordAuthenticationToken에 넣었던 UserDetails 객체
반환
                        UserDetails userVO = (UserDetails)
authentication.getPrincipal();
                        System.out.println("ID정보 : " + userVO.getUsername());
                }
                return "main";
        }
```

※SecurityContextHolder

시큐리티가 인증한 내용들을 가지고 있으며, SecurityContext를 포함하고 있고 SecurityContext를 현재 스레드와 연결

Constructor Summary

Constructors

Constructor and Description
`SecurityContextHolder()`

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
static void	**clearContext**() Explicitly clears the context value from the current thread.
static SecurityContext	**createEmptyContext**() Delegates the creation of a new, empty context to the configured strategy.
static SecurityContext	**getContext**() Obtain the current SecurityContext.
static SecurityContextHolderStrategy	**getContextHolderStrategy**() Allows retrieval of the context strategy.
static int	**getInitializeCount**() Primarily for troubleshooting purposes, this method shows how many times the class has re-initialized its SecurityContextHolderStrategy.
static void	**setContext**(SecurityContext context) Associates a new SecurityContext with the current thread of execution.
static void	**setStrategyName**(String strategyName) Changes the preferred strategy.
String	**toString**()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait