12.10 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☞Check Your Progress 1

1) JCA has been described with the core classes and interface in section 12.2.1
2) Comparison of Message Digest and Message Authentication Code

Message Digest	Message Authentication Code
A message digest algorithm takes a single input a message and produces a "message digest" (aka hash)	A Message Authentication Code algorithm takes two inputs, a message and a secret key, and produces a MAC
It allows you to verify the integrity of the message	It allows you to verify the integrity and the authenticity of the message
Any change to the message will (ideally) result in a different hash being generated.	Any change to the message **or** the secret key will (ideally) result in a different MAC being generated.
An attacker that can replace the message and digest is fully capable of replacing the message and digest with a new valid pair.	Nobody without access to the secret should be able to generate a MAC calculation that verifies the integrity and authenticity of the message

3) Man-in-the-middle attack is possible. Chuck, a man in the middle, attacks and intercepts all the messages sent by Alice. Chuck performs below-
    - Instead of forwarding message m, he forwards message n. Instead of the signature on m with Alice's private key, he forwards a signature on n with his private key.
    - Instead of Alice's public key he sends his public key to Bob. Bob will never see the difference as he does not know Alice's public key beforehand.

   A flaw in the approach is that the public key should not be transferred along with the message. Public key should be available to the person prior to verification of the signature.

☞Check Your Progress 2

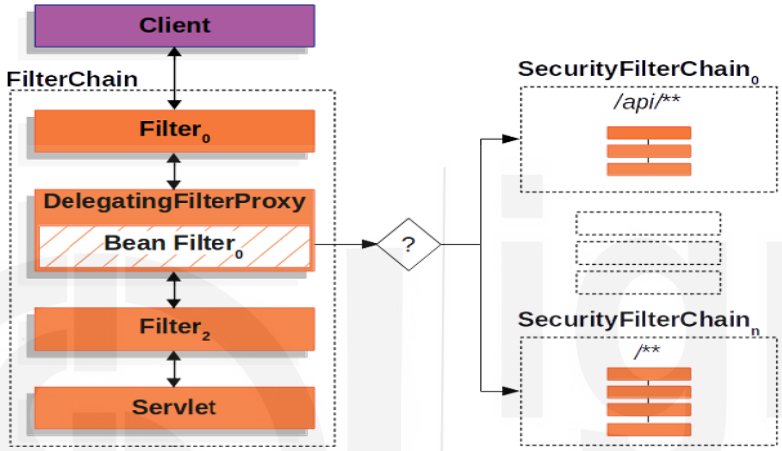
1) JSSE with the advantages has been written in section 12.2.2
2) java.net.URL class demonstrates the easiest way to add SSL to your applications. This approach is useful, but it is not flexible enough to let you create a secure application that uses generic sockets. Using JSSE API to implement SSL gives us full control over the protocols that need to be supported.
3) If javax.net.ssl.SSLHandshakeException occurs, it indicates that there is a handshake_failure. We haven't created or established the SSL certificate that the client and server should be referencing. Normally a certificate is obtained from a trusted certificate authority for a public site, but for testing and development purposes, it's common practice to create a self-signed certificate, which just means you are the issuer. These are typically issued to the localhost domain, so they'll function locally but not be trusted elsewhere.
   Refer section 12.2.2 for keytool to generate keypair and how to use this to solve javax.net.ssl.SSLHandshakeException issue.

☞Check Your Progress 3

1) Refer section 12.4, Spring Security has been explained in detail.
2) Refer Authentication in Section 12.4
3) Refer Authorization in Section 12.4

☞Check Your Progress 4

1) SecurityFilterChain is a FilterChain component which has zero or more Security Filters in an order.



Refer Section 12.5 for more details for SecurityFilterChain.

2) **Hint:** configure method definition in Spring Security Configuration will be as -

```
@Override
   protected void configure(HttpSecurity http) throws Exception
{
    http
    .authorizeRequests()
    .antMatchers("/static","/register").permitAll()
    .antMatchers("/user/**").hasAnyRole("USER", "ADMIN") // can pass
    multiple roles
    .antMatchers("/admin/**").hasRole("ADMIN")
    .anyRequest().authenticated()
    .and()
    .formLogin()
    .loginUrl("/login")
    .permitAll();
}
```

- /static and /register is allowed to everyone so permitAll() is used.
- /user/** must be accessible to either "User" or "Admin" roles. Thus hasAnyRole("USER", "ADMIN") is used.
- /admin/** must be accessible to the only Admin role. Thus hasRole("ADMIN") is used
- /login must be allowed for all so permitAll() is used.

3) We need to customize the search query. Spring security can adapt the customize search queries very easily. We simply have to provide our own

SQL statements when configuring the AuthenticationManagerBuilder. Two methods for customization has been provided as ***usersByUsernameQuery*** and ***authoritiesByUsernameQuery***. Refer to Section 12.5 for customizing the search query.

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth)
  throws Exception
{
     auth.jdbcAuthentication()
     .dataSource(dataSource)
     .usersByUsernameQuery("select email,password,enabled "
     + "from users "
     + "where email = ?")
     .authoritiesByUsernameQuery("select email,authority "
     + "from authorities "
     + "where email = ?");
}
```

4) Refer Securing the URLs in section 12.5

12.11 REFERENCES/FURTHER READING

- Craig Walls, "Spring Boot in action" Manning Publications, 2016. (https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf)
- Christian Bauer, Gavin King, and Gary Gregory, "**J**ava Persistence with Hibernate",Manning Publications, 2015.
- Ethan Marcotte, "Responsive Web Design", Jeffrey Zeldman Publication, 2011(http://nadin.miem.edu.ru/images_2015/responsive-web-design-2nd-edition.pdf)
- Tomcy John, "Hands-On Spring Security 5 for Reactive Applications",Packt Publishing,2018
- https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html
- https://docs.*oracle*.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html
- https://docs.spring.io/spring-security/site/docs/3.1.x/reference/springsecurity.html
- https://spring.io/guides/topicals/spring-security-architecture
- http://tutorials.jenkov.com/java-cryptography/index.html
- https://www.cloudflare.com/learning/security/what-is-web-application-security/
- https://www.baeldung.com/java-ssl
- https://www.baeldung.com/java-security-overview
- https://www.baeldung.com/spring-security-login
- https://dzone.com/articles/spring-security-authentication