Spring security (SS)

“Spring Security” used to be known as Acegi

a Spring-based declarative security framework for handling authentication and authorisation, at both the web request level and at the method invocation level

You can declare “who” invokes “what”!

SS uses dependency injection and AOP

Authentication, authorisation

Authentication: process of verifying that the users, also called principal (people, program, services) of our application are who they say they are. Successful authentication result in authenticated principal

For example when login or when prompted for password

Authorization: accessibility of the secured system. Two steps

1. Role assignment: mapping of authenticated principal to one or more roles (authorities)

For example, anonymous user, registers login and become customer

2. Assignment of roles to secured resources of the system. Payment panel should only be visible to Customers who log in

An overview of SS

Security Interceptor: intercept access to resources to enforce security AND applies security rules.

You don't directly use Security Interceptor as it delegates the task to one of these:

- Authentication Manger
- Access Decision Manager
- Run-as Manager
- After-invocation Manager
An overview of SS

After passing through the previous two managers, **Run-as Manager** is used to manage your authentication and access while accessing the resources.

For example, you may be allowed to view (read access) but not modify the contents (write access).

After-invocation Manager enforces security after the access.

For example, altering the returned values so that the user is only able to access certain properties of the returned object.

AuthenticationManager

requires implementing a single method `authenticate()`.

If successful, returns an `org.acegisecurity.Authentication` object.

If authentication fails, an `AuthenticationException` will be thrown.

We don't directly implement A/M, instead we use `ProviderManager`, an implementation of `AuthenticationManager`.

ProviderManager

```xml
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
    <property name="providers">
        <list>
            <ref local="daoAuthenticationProvider"/>
            <ref local="anonymousAuthenticationProvider"/>
        </list>
    </property>
</bean>
```

ProviderManager has a list of authentication providers through `providers` property—out of many, we use two (see code in previous slide):

- DaoAuthenticationProvider:
- AnonymousAuthenticationProvider: authenticates a user as an anonymous user.

Extension of the previous example to include security

- `ehcache.xml` This is a simple example that you see SS, we will move to SS3
- `Shop-spring-hib.xml` //setting for security
  - `shop/`
    - `dao/` RoleDAO added
    - `model/` Role...
    - `usecases/` Modified
    - `Main.java` Modified

daoAuthenticationProvider

1. Retrieves user info such as username and password from a database.
2. `daoAuthenticationProvider` performs authentication by comparing the username/password retrieved from the database with the principal and credentials passed in an Authentication object from the authentication manager.

Outcome: `Authentication` object or `AuthenticationException`. 
DaoAuthenticationProvider

so you must create DAO interfaces and their implementations
1. Observe DAO objects for Role
2. Look at lines 214-222
3. Look at line 212
Next we look at them in details.

crypted passwords

Suppose that we want to store the password encrypted
We (application) don't want to know the users passwd, so user-provided password must also be encrypted.
There are various options:
• encoding.PlaintextPasswordEncoder
  No encryption (default)
• encoding.ShaPasswordEncoder
  Does Secure Hash Algorithm (SHA) on the password
Similarly encoding.Md5PasswordEncoder

encrypted passwords

What is a salt?
Two salt sources available for encryption are:
• SystemWideSaltSource
  Provides the same salt for all users
• ReflectionSaltSource
  Encoded each user's data using a different salt value (better security)
Also to enable caching UserCache property must be declared.

Controlling access

After authentication Access Decision Manager decides if a user has required privileges to access secured resources.
(org.acegisecurity.AccessDecisionManager)

Behind the scene a voting mechanism is used:
AccessDecisionManager uses an AccessDecisionVoter to decide, on the basis of user's authorities and configuration required by the resource (Role assignments) to give access or to deny or to abstain vote.
See lines 275-291 for sample of Consfs

Controlling access

<table>
<thead>
<tr>
<th>AccessDecisionManager</th>
<th>AccessDecisionVoter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategies for tallying the votes:</td>
<td></td>
</tr>
<tr>
<td>1. <strong>AffirmativeBased</strong> Allows access if at least one voter votes to grant access</td>
<td></td>
</tr>
<tr>
<td>2. <strong>ConsensusBased</strong> Allows access if a consensus of voters vote to grant access</td>
<td></td>
</tr>
<tr>
<td>3. <strong>UnanimousBased</strong> Allows access if all voters vote to grant access</td>
<td></td>
</tr>
<tr>
<td>In the example AffirmativeBased (Line 246)</td>
<td></td>
</tr>
</tbody>
</table>

Enterprise Systems

Lecture 19
Spring security (RECAP)

a Spring-based declarative security framework for handling authentication and authorisation, at both the web request level and at the method invocation level.

Authentication, authorisation (RECAP)

Authentication: process of verifying that the users, also called principal (people, program, services) of our application are who they say they are. Successful authentication result in authenticated principal

Authorization: accessibility of the secured system.

Authentication, authorisation (RECAP)

Role-based access control

Authorization:
1. User Role assignment: mapping of authenticated principal to one or more roles (authorities)
   For example, anonymous user, registers login and become customer
2. Assignment of roles to secured resources of the system. Payment panel should only be visible to customers who log in

Preferences

How to keep values of attributes and use them.
Servlet sessions! You have seen:
Preferences preferences = Preferences.userNodeForPackage(Main.class);
What is a preference? java.util.prefs.preferences
A method of storing and retrieving user preferences that persist across application invocations.
API: the preferences are "remembered" from one run of an application to the next.

Preferences

... automatically maintains separate preference lists for multiple users,
transparently handles storing the preferences information
How? Uses registry and hidden files to "node in a hierarchical collection of preference data"
Obtain Preferences using static method
userNodeForPackage()

Preferences

Preferences preferences = Preferences.userNodeForPackage(Main.class);
or
Preferences.userRoot().node(this.getClass().getName());
Stored variable can be retrieved next time by get, getBoolean, getInt, ...
String username = preferences.get("principal", "");
Preferences

Preferences objects hold data as key/value:
To store a preferences item invoke put()
preferences.put("principal", "");
Why preferences are used in the FMC code?
To simulate the servlet session
What is that?

Authentication

what is the story?
Authentication token = null;
boolean userAuthenticated = false;
... //create a token and use it for authentication
token = new
UsernamePasswordAuthenticationToken(username, password);
userAuthenticated =
    fmcService.authenticate(token);

Authentication

Authentication an interface that contains:
• identity of the principal
• its credentials and
• GrantedAuthority (an array of them).
GrantedAuthority has various implementations,
we use GrantedAuthorityImpl that assign a
string that represents the authority of the
principal (for example “Role_anonymous”)
This is user-role assignment in access control.

Authentication

The are a number of authentication providers
for processing for example
DaoAuthenticationProvider (for hibernate)
provides AuthenticationDao

Authentication

We can do this programatically:
if (!userAuthenticated)
{
    token = new AnonymousAuthenticationToken(
        new GrantedAuthority[] { //an array
            new GrantedAuthorityImpl("Role_anonymous")
        });
}

Storing authentication object

How can we access an authentication?
Authentication auth =
    SecurityContextHolder.getContext().
        getAuthentication();
This is accessing the populated Authentication
object that applies to the current principal.
Access is via an implementation of
ContextHolder (here SecurityContextHolder)
Storing authentication object

You see the code
SecureContext provides a mutator and accessor for the Authentication object
ContextHolder is set with a correct Authentication for a necessary duration (for example HTTP servlet request)
What about our example?

Wiring and assigning role to methods

Within MethodSecurityInterceptor wire:
1. authenticationManager
2. accessDecisionManager and
3. objectDefinitionSource with the values of the form
FMC.usecases.FMCService.createDB=Role_anonymous
FMC.usecases.FMCService.getCustomerByEmail=Role_user,Role_admin
See FMCSpring.xml bean with id=“FMCServiceSecurity”

This wiring lets u authenticate

// if everything ok
}if (auth != null)
{
    SecurityContextHolder.getContext().setAuthentication(auth);

This wiring lets u authenticate

// if everything ok
}if (auth != null)
{
    SecurityContextHolder.getContext().setAuthentication(auth);

daoAuthenticationProvider

Makes use of four beans (see the class diag.):
<bean id="daoAuthenticationProvider" ..
<property name="userDetailsService"
ref="customerDAO"/>
<property name="userCache"
ref="userCache"/>
<property name="passwordEncoder"
ref="passwordEncoder"/>
<property name="saltSource"/>
userCache : cache parameters such as time-to-live..
For example in password encoder, `ShaPasswordEncoder` is used.

```xml
<bean id="passwordEncoder" class="org.acegisecurity.providers.encoding.ShaPasswordEncoder"/>
```

**How to decide to allow access?**

Acegi intercepts access to an object or a webpage, but what if there are multiple factors involved in decision?

Example:

```
Patient: Patient
AccessPatientsDetails(r:Role,e:emergencyLevel):Boolean
Complex logic: different principals, password... level of emergency
```

**Voting**

User can implement their own `AccessDecisionManager` to control all aspects of authorisation. But, spring provides a number of implementations:

```xml
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false"/>
    <property name="decisionVoters">
        <list>
            <ref local="roleVoter"/>
        </list>
    </property>
</bean>
```

**What does this mean?**

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false"/>
    <property name="decisionVoters">
        <list>
            <ref local="roleVoter"/>
        </list>
    </property>
</bean>
```

**Architecture**

`AccessDecisionVoter` provides 3 possible outcomes for accessing the secure resources:

- ACCESS_GRANTED
- ACCESS_DENIED
- ACCESS_ABSTAIN

`RoleVoter` is an implementation of `AccessDecisionManager` provided by Spring.

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false"/>
    <property name="decisionVoters">
        <list>
            <ref local="roleVoter"/>
        </list>
    </property>
</bean>
```

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false"/>
    <property name="decisionVoters">
        <list>
            <ref local="roleVoter"/>
        </list>
    </property>
</bean>
```

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false"/>
    <property name="decisionVoters">
        <list>
            <ref local="roleVoter"/>
        </list>
    </property>
</bean>
```

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false"/>
    <property name="decisionVoters">
        <list>
            <ref local="roleVoter"/>
        </list>
    </property>
</bean>
```

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="false"/>
    <property name="decisionVoters">
        <list>
            <ref local="roleVoter"/>
        </list>
    </property>
</bean>
```

See Lines 242-250.
Controlling access

RoleVoter decides on its vote by comparing all configuration attributes which are prefixed with Role_ with all of the authorities granted to authenticated user.
If RoleVoter finds a match, it will vote ACCESS_GRANTED, otherwise, will vote ACCESS_DENIED
RoleVoter will abstain from voting when the authorities required for access are not prefixed with Role_.

Securing method invocations

As expected AoP is used to enforce secure access to methods, See line 258.
MethodSecurityInterceptor determines if the user has been authenticated and if it can call the method.
If positive outcome, the method call will be invoked.
If not:
• AuthenticationException: user cannot be authenticated.
• AccessDeniedException: user hasn't been granted authority to make the call

Deciding on votes

three AccessDecisionManager:
1. ConsensusBased:
grant or deny access based on the consensus of non-abstain votes.
Properties are provided to control behavior in the event of an equality of votes or if all votes are abstain.
<property name="allowIfAllAbstainDecisions" value="false"/>

2. AffirmativeBased
grant access if one or more ACCESS_GRANTED were received (i.e. there was at least one grant vote).
Similarly there is a parameter that controls the behaviour if all voters abstain.

3. UnanimousBased
grants if provider expects unanimous ACCESS_GRANTED ignoring abstains deny access if there is any ACCESS_DENIED vote.)
private static void logout(FMCService fmcservice, Preferences preferences){
    preferences.put("principal", "");
    preferences.put("password", "");
    Authentication tok = new
    AnonymousAuthenticationToken("anonymousKey","anonymous", new
    GrantedAuthority[]{new
    GrantedAuthorityImpl("Role_anonymous")});
}

A few words on web access
carried out by setting filters (implementation of javax.servlet.Filter)
We specify patterns of access and the role that can access
Create an XML access file in WEB-INF and add pattern and role
<http auto-config="true">
    <intercept-url pattern="/*" access="ROLE_USER"/>
</http>

Implementation by
adding filters to intercept Request Response
Picture

Putting everything together
(Short review)

What we have learned in this course:
Starting point
Hibernate
ORM
Persistent domain model
POJO

Presentation tier
JSP, Wicket...
Business Services
Layer-indepedent of persistence layer
Security (acegi)
DAO interface implementation using ORM
DAO interface for persistent operations

Transactions
Briefly looked at JDBC
Key concept was Transitions:
A mechanism for grouping operations on a database so that either all of them complete together or none of them do.
Among other things:
Patterns for writing Transactions
Pitfalls of Transactions
Performance
Object Relational Mapping

- Two parallel world (DB and Application)
- Object Relational Mismatch
- Differences between OO and Relational model
- Can we stop repeating?
  - automatically produce artifacts from POJO (DB, Transactions...)

Object/relational mapping is the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database.

In ORM part we learnt

- Object lifecycle in Hibernate
- Equality
- Session and its use
- Querying (some HQL!)
- Cascading persistence
- Transactions
- Mapping
- Versioning support
- Advanced topics (hitting the limit!)
  - Unrepeatable read
  - Phantom read
  - Dirty read

Hibernate Object life cycle

Mid layer

How do I interact with the persistent layer:

Let us use components,

We learnt two powerful techniques:

- Dependency Injection
- Aspect oriented Programming

Solution:

1. Make Beans
2. Wire them to build application
3. Use aspects for cross cutting concerns (Transactions, logging, security, ...)

Mid layer (continue)

How do I interact with the persistent layer:

1. Templates (JDBC, Hibernate, ...)
2. Callbacks

Declarative Transactions:

(propagation, isolation)
**Spring security**
a Spring-based **declarative** security framework for handling **authentication** and **authorisation**, at **both** the web request level and at the **method invocation level**. 

Within **MethodSecurityInterceptor** wire:
1. `authenticationManager`
2. `accessDecisionManager` and
3. `objectDefinitionSource` with the values of the form: `FMCService.createDB=Role_anonymous`

**This wiring lets u authenticate**
1. Use `authenticationManager` by passing `token:Authentication`
2. Set authentication on the context

**Task for future & post ES**
Hibernate:
Performance tuning, UML tools for making POJO, DB,…
Spring:
Spring MVC (or GWT), Advance Spring security and Spring ROO