



Spring Security

Spring Security

- ▶ Security, establishing who a user is (authentication), and allowing or disallowing actions (authorization) are vital to any serious application.
- ▶ In this Spring Security Module we will look at:
 - ▶ Authentication in a web environment
 - ▶ Requiring Authorization for certain web pages
 - ▶ Requiring Authorization for method calls
 - ▶ Defending against common attacks



Basic Example

- We'll create a basic example to show the essentials of Spring Security
 - Configured with Java Config
 - Configured with XML
- Then we'll go into the details of the different parts

WebAppInitializer

```
public class MyWebAppInitializer implements WebApplicationInitializer {
```

```
    @Override
```

```
    public void onStartUp(ServletContext container) throws ServletException {
```

```
        // Create the 'root' Spring application context
```

```
        AnnotationConfigWebApplicationContext rootContext
```

```
            = new AnnotationConfigWebApplicationContext(),
```

```
        rootContext.register(WebConfig.class, SecurityConfig.class);
```

```
        container.addListener(new ContextLoaderListener(rootContext));
```

```
        // Create the dispatcher servlet
```

```
        ServletRegistration.Dynamic appServlet = container.addServlet("mvc",
```

```
            new DispatcherServlet(new GenericWebApplicationContext()));
```

```
        appServlet.setLoadOnStartup(1);
```

```
        appServlet.addMapping("/");
```

```
        container.addFilter("springSecurityFilterChain",
```

```
            new DelegatingFilterProxy("springSecurityFilterChain"))
```

```
            .addMappingForUrlPatterns(null, false, "/*");
```

```
    }
```

```
}
```

Load both WebConfig and SecurityConfig

Apply Security Filter to all incoming requests

WebConfig

Normal SpringMVC WebConfig

```
@Configuration
@EnableWebMvc
@ComponentScan("cs544")
public class WebConfig implements WebMvcConfigurer{
    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver bean = new InternalResourceViewResolver();

        bean.setViewClass(JstlView.class);
        bean.setPrefix("/WEB-INF/view/");
        bean.setSuffix(".jsp");

        return bean;
    }
}
```

SecurityConfig

You can think of the `.and()` as the closing of an XML tag it take you back to the root

@EnableWebSecurity and
Extends WebSecurityConfigAdapter

Showing
2 of 3
configure
methods
that can be
overridden

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("admin").password("{noop}123").roles("USER,ADMIN").and()
            .withUser("user").password("{noop}bla").roles("USER");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests().antMatchers("/important/**").hasRole("USER").and()
            .formLogin().and()
            .logout();
    }
}
```

Creates an inMemory use details
without encoded passwords
(just for demo, not good for production!)

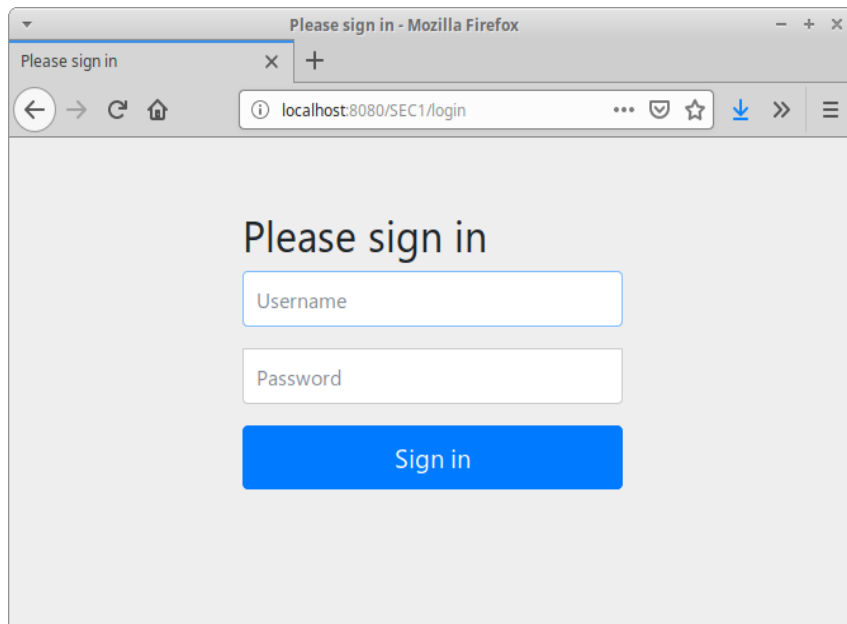
Make sure anyone wanting to access
anything under important
has the USER role

People can login with a form
and logout

Generated login.jsp

You can also write your own

- Spring Security generates a form-login
 - When not logged in and try to access `/important/**`



```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>security</display-name>
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- Needed when using Spring with Filter -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/springconfig.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>

```

Or you can use a web.xml instead of the
Initializer class

Automatically loads SpringMVC-servlet.xml

Loads springconfig.xml as root config

Filter applies security

SpringMVC-servlet.xml

Normal SpringMVC Config

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!-- scan for @RequestMapping annotations-->
    <mvc:annotation-driven />

    <!-- scan for @Controller (and other component) annotations in the following package -->
    <context:component-scan base-package="springmvc.helloworld" />

    <!-- Resolves views to .jsp resources in the /WEB-INF/views directory -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

Springconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:sec="http://www.springframework.org/schema/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd">

  <sec:http>
    <sec:intercept-url pattern="/important/**" access="ROLE_USER"/>
    <sec:form-login />
    <sec:logout />
  </sec:http>

  <sec:authentication-manager>
    <sec:authentication-provider>
      <sec:user-service>
        <sec:user name="test" password="{noop}123" authorities="ROLE_USER, ROLE_ADMIN" />
        <sec:user name="bob" password="{noop}bobiscool" authorities="ROLE_USER" />
      </sec:user-service>
    </sec:authentication-provider>
  </sec:authentication-manager>
</beans>
```

Security namespace

http elements specify
url patterns for security

Authentication manager
/ provider configuration

Applications



CS544 EA

Spring Security: Authorizing Web Requests

Turn off Filter for URL

- It is possible to turn off the security filter

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/js/**", "/css/**").and()
        .debug(true);
    }
}
```

The 3rd
config
method

Both /js/ and /css/
and everything below
those directories
has no security applied

We can also enable
debugging here.

Spring Security can be
very tricky to work with

```
<beans ...>
<sec:http pattern="/js/**" security="none" />
<sec:http pattern="/css/**" security="none" />
<sec:debug />
```

XML equivalent

Debugging

- 2 things needed to debug Spring Security:
 - Enable in settings (previous slide)
 - Enable debug output on logger (log4j2.xml shown)

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="debug">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="Console"/>
    </Root>
    <Logger name="org.springframework.security" level="debug">
      <AppenderRef ref="Console"/>
    </Logger>
  </Loggers>
</Configuration>
```

Permit All Access

- Requests everyone should be able to make

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/login", "/logout", "/index")
            .permitAll();
    }
}
```

Allows everyone to request these URLs

```
<sec:http>
  <sec:intercept-url pattern="/login" access="permitAll" />
  <sec:intercept-url pattern="/logout" access="permitAll" />
  <sec:intercept-url pattern="/index" access="permitAll" />
</sec:http>
```

Role Access

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers(HttpMethod.GET, "/addContact")
            .hasRole("ADMIN");
    }
}
```

`.antMatchers()` has optional
HttpMethod arg

Specify the required role

```
<sec:http>
  <sec:intercept-url method="GET" pattern="/addContact" access="hasRole('ADMIN')" />
</sec:http>
```

Optional method property

Chaining & Order

- Order specified is **important**
 - Spring Sec goes top to bottom until match (then stops)

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/login", "/logout", "/index").permitAll().and()
            .authorizeRequests()
                .antMatchers(HttpMethod.GET, "/addContact").hasRole("ADMIN").and()
            .authorizeRequests()
                .antMatchers(HttpMethod.POST).hasRole("ADMIN").and()
            .authorizeRequests()
                .antMatchers(HttpMethod.GET, "/contacts").hasRole("USER").and()
            .formLogin().and()
            .logout();
    }
}
```

Any POST needs ADMIN

XML Version

- Order in XML is important for the same reason

```
<sec:http>
  <sec:intercept-url pattern="/login.jsp" access="permitAll" />
  <sec:intercept-url pattern="/logout" access="permitAll" />
  <sec:intercept-url pattern="/index" access="permitAll" />
  <sec:intercept-url pattern="/addContact" method="GET" access="hasRole('ADMIN')" />
  <sec:intercept-url method="POST" access="hasRole('ADMIN')" />
  <sec:intercept-url pattern="/contacts" method="GET" access="hasRole('USER')" />
  <sec:form-login />
  <sec:logout />
</sec:http>
```

Expressions

- You can write expressions to specify multiple attributes that may be needed for authorization
 - Primarily XML, but also JavaConf

```
<sec:http>
  <sec:intercept-url pattern="/admin/**" access="hasRole('ADMIN') and hasIpAddress('192.168.1.0/24')" />
  ...
</sec:http>
```

http

```
.authorizeRequests()
.antMatchers("/admin/**").hasIpAddress("192.168.1.0/24")
.antMatchers("/admin/**").hasRole("ADMIN");
```

http

```
.authorizeRequests()
.antMatchers("/admin/**")
.access("hasRole('ADMIN') and hasIpAddress('192.168.1.0/24')");
```

Common Built-in Expressions

Expression	Description
hasRole([role])	Returns true if the principal has the role
hasAnyRole([role1,role2])	Returns true if the principal has any of the roles
principal	Gives direct access to the principal object
authentication	Gives direct access to the authentication object
permitAll	Always evaluates to true
denyAll	Always evaluates to false
isAnonymous()	Returns true if the principal is anonymous
isRememberMe()	Returns true if the principal is a remember-me user
isAuthenticated()	Returns true if the principal is not anonymous
isFullyAuthenticated()	Returns true if the principal is not anon or remember-me

Applications



CS544 EA

Spring Security: Login / Logout

Customize Login

- You can create a custom login page
 - Instead of using the generated one
 - Spring Sec lets you configure every detail

http

```
.authorizeRequests()
    .antMatchers(HttpMethod.GET, "/addContact").hasRole("ADMIN")
    .and()

...

.formLogin().loginPage("/login").loginProcessingUrl("/login")
    .usernameParameter("username").passwordParameter("password")
    .failureUrl("/index")
    .defaultSuccessUrl("/contacts")
    .permitAll().and()

.logout();
```

You can also provide a login-handler method (not shown)

.permitAll() makes sure users have access to all login related URLs

XML Config

```
<sec:http use-expressions="false">
  <sec:intercept-url pattern="/login" access="permitAll" />
  <sec:intercept-url pattern="/index" access="permitAll" />
  <sec:intercept-url pattern="/addContact" method="GET" access="hasRole('ADMIN')" />
  <sec:form-login login-page="/login" login-processing-url="/login"
    username-parameter="username" password-parameter="password"
    authentication-failure-url="/index"
    default-target-url="/contacts" />
  <sec:logout />
</sec:http>
```

No permitAll option on the tag

Custom Login Form

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Login Page!</h1>
    <c:if test="${error eq true}">
      <p>${sessionScope["SPRING_SECURITY_LAST_EXCEPTION"].message}</p>
    </c:if>
    <form method="post" action="<c:url value='/login' />">
      User: <input name="username"
value='<c:if test="${not empty param.login_error}"><c:out value="${SPRING_SECURITY_LAST_USERNAME}" /></c:if>' />
        <br />
      Pass: <input type="password" name='password' /> <br />
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
      <input type="submit" />
    </form>
  </body>
</html>
```

Customize Logout

- You can similarly configure the logout

http

```
.logout().logoutUrl("/logout").logoutSuccessUrl("/index")  
.invalidateHttpSession(true).deleteCookies("cookie-names");
```

```
<sec:logout logout-url="/logout" logout-success-url="/index"  
delete-cookies="cookie-names" invalidate-session="true" />
```

Options also exist to provide a
logoutHandler and a
logoutSuccessHandler

Applications



CS544 EA

Spring Security: Taglib

Taglib

Tag	Use
<code><sec:authorize access="expression"></code>	Show content if expression is true
<code><sec:authorize url="/location"></code>	Show content if allowed to access location
<code><sec:authentication></code>	Access properties of logged in person
<code><sec:csrfInput /></code>	Add csrf tags into a form
<code><sec:csrfMetaTags /></code>	Add csrf meta tags (to access with JS)

`<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>`



CSRF

- Cross Site Request Forgery
 - Tricks user into making a request
 - Accidentally submitting a form to a site they're (hopefully) already logged into
- Mitigated by having a token on the real form
 - If random token value is present, request is real

Spring Security CSRF

- Spring Security includes CSRF protection
 - Always on by default
 - Spring Tags `<form:form>` automatically includes token
- Any `<form>` submitted without `csrf_token` not accepted
 - Common mistake made by beginners
 - They forget to include token on their `<form>`
- Spring Security give 403 Forbidden on submit

Examples

```
<sec:authorize access="hasRole('ADMIN')">  
  <a href="addContact"> Add a Contact</a>  
</sec:authorize>
```

```
<sec:authorize url="/addContact">  
  <a href="addContact"> Add a Contact</a>  
</sec:authorize>
```

```
<sec:authorize access="!isAuthenticated()">  
  <p><a href="login">Login</a></p>  
</sec:authorize>  
<sec:authorize access="isAuthenticated()">  
  <p>Welcome Back, <sec:authentication property="name"/></p>  
  <p><a href="logout">Logout</a></p>  
</sec:authorize>
```

CSRF

```
<%@taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
    <sec:csrfMetaTags />
    <script type="text/javascript" language="javascript">
      var csrfParameter = $("meta[name='_csrf_parameter']").attr("content");
      var csrfHeader = $("meta[name='_csrf_header']").attr("content");
      var csrfToken = $("meta[name='_csrf']").attr("content");
    </script>
  </head>
  <body>
    <h1>Login Page!</h1>
    <c:if test="${error eq true}">
      <p>${sessionScope['SPRING_SECURITY_LAST_EXCEPTION'].message}</p>
    </c:if>
    <form method="post" action="<c:url value='/login' />">
      User: <input name="username"
value='<c:if test="${not empty param.login_error}"><c:out value="${SPRING_SECURITY_LAST_USERNAME}" /></c:if>' />
        <br />
      Pass: <input type="password" name='password' /> <br />
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
      <sec:csrfInput />
      <input type="submit" />
    </form>
  </body>
</html>
```

If CSRF needed in JS

Clean replacement

Applications



CS544 EA

Spring Security: Authentication Providers

Plain Text

- So far we've used **plain text: bad for security**

@Override

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
    auth.inMemoryAuthentication()  
        .withUser("admin").password("{noop}123").roles("USER,ADMIN").and()  
        .withUser("user").password("{noop}bla").roles("USER");  
}
```

```
<authentication-manager>  
  <authentication-provider>  
    <user-service>  
      <user name="test" password="{noop}123" authorities="ROLE_USER, ROLE_ADMIN" />  
      <user name="bob" password="{noop}bobiscool" authorities="ROLE_USER" />  
    </user-service>  
  </authentication-provider>  
</authentication-manager>
```


Password Encoder

- Super important:
 - **Never store plain text** (no reading of pwds!)
 - Basic hashing isn't that great either

```
@Bean
public BCryptPasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
        .withUser("jimi").password("{bcrypt}d7e6351eaa13189a5a3641bab846c8e8c69ba39f").roles("ADMIN","USER").and()
        .withUser("bob").password("{bcrypt}4e7421b1b8765d8f9406d87e7cc6aa784c4ab97f").roles("USER");
}
```

```
<sec:authentication-manager>
<sec:authentication-provider>
  <sec:password-encoder hash="bcrypt"/>
  <sec:user-service>
    <sec:user name="jimi" password="{bcrypt}d7e6351eaa13189a5a3641bab846c8e8c69ba39f" authorities="ROLE_USER, ROLE_ADMIN" />
    <sec:user name="bob" password="{bcrypt}4e7421b1b8765d8f9406d87e7cc6aa784c4ab97f" authorities="ROLE_USER" />
  </sec:user-service>
</sec:authentication-provider>
</sec:authentication-manager>
```

Can be: md4, md5, sha, sha-256, bcrypt

Bcrypt is recommended as it also auto-salts

JDBC Authenticator

```
@Autowired
private DataSource dataSource;

@Override
public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.jdbcAuthentication()
        .dataSource(dataSource)
        .withDefaultSchema()
}
```

```
<sec:authentication-manager>
  <sec:authentication-provider>
    <sec:jdbc-user-service data-source-ref="dataSource" />
  </sec:authentication-provider>
</sec:authentication-manager>

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://localhost/cs544"/>
  <property name="username" value="root"/>
  <property name="password" value="root"/>
</bean>
```

Standard Authentication Tables

JDBC authentication expects the following tables:

```
create table users(  
    username varchar_ignorecase(50) not null primary key,  
    password varchar_ignorecase(50) not null,  
    enabled boolean not null  
);  
create table authorities (  
    username varchar_ignorecase(50) not null,  
    authority varchar_ignorecase(50) not null,  
    constraint fk_authorities_users foreign key(username) references users(username)  
);  
create unique index ix_auth_username on authorities (username,authority);
```

Values could be inserted like so:

```
Insert into users values("test", "{bcrypt}d7e6351eaa13189a5a3641bab846c8e8c69ba39f", 1);  
Insert into users values("bob", "{bcrypt}4e7421b1b8765d8f9406d87e7cc6aa784c4ab97f", 1);  
Insert into authorities values("test", "ROLE_USER");  
Insert into authorities values("test", "ROLE_ADMIN");  
Insert into authorities values("bob", "ROLE_USER");
```

Custom Tables / Queries

@Override

```
public void configure(AuthenticationManagerBuilder auth) throws Exception {  
    auth.jdbcAuthentication()  
        .dataSource(dataSource)  
        .usersByUsernameQuery("select username,password, enabled from users where username=?")  
        .authoritiesByUsernameQuery("select u.username, ur.authority from users u, user_roles ur where u.user_id = ur.user_id " +  
            " and u.username =?");  
}
```

```
<sec:authentication-manager>  
    <sec:authentication-provider>  
        <sec:jdbc-user-service data-source-ref="dataSource"  
            users-by-username-query="select username,password, enabled from users where username=?"  
            authorities-by-username-query="select u.username, ur.authority from users u, user_roles ur where u.user_id = ur.user_id  
                and u.username =?" />  
    </sec:authentication-provider>  
</sec:authentication-manager>
```

Custom Authentication Provider

```
public class CustomAuthenticationProvider implements AuthenticationProvider {  
    @Override  
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {  
        String name = authentication.getName();  
        String password = authentication.getCredentials().toString();  
        if (name.equals("test") && password.equals("123")) {  
            List<GrantedAuthority> grantedAuths = new ArrayList<>();  
            grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));  
            Authentication auth = new UsernamePasswordAuthenticationToken(name, password, grantedAuths);  
            return auth;  
        } else {  
            return null;  
        }  
    }  
    @Override  
    public boolean supports(Class<?> authentication) {  
        return authentication.equals(UsernamePasswordAuthenticationToken.class);  
    }  
}
```

```
@Bean  
public CustomAuthenticationProvider customAuthenticationProvider() {  
    return new CustomAuthenticationProvider();  
}
```

```
<sec:authentication-manager>  
    <sec:authentication-provider ref="customAuthenticationProvider"/>  
</sec:authentication-manager>
```

Multiple Authentication Providers

- Spring will try each one, in the order found

```
<sec:authentication-manager>

  <sec:authentication-provider>
    <sec:user-service>
      <sec:user name="test" password="{noop}123" authorities="ROLE_USER, ROLE_ADMIN" />
      <sec:user name="bob" password="{noop}bobiscool" authorities="ROLE_USER" />
    </sec:user-service>
  </sec:authentication-provider>

  <sec:authentication-provider>
    <sec:jdbc-user-service data-source-ref="dataSource" />
  </sec:authentication-provider>

  <sec:authentication-provider ref="customAuthenticationProvider" />
</sec:authentication-manager>
```

I have not tested this yet for Java Config

Applications



CS544 EA

Spring Security: Method Security

Method Security

- Method security is important: defense in depth!
- 3 types of security annotations supported:
 - @Secured
 - JSR-250 annotations
 - @PreAuthorize and @PostAuthorize

Nice tutorial at: <https://www.baeldung.com/spring-security-method-security>

Enabling Method Security

Maven dependency for Method Security

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
</dependency>
```

```
@Configuration
@EnableWebMvc
@EnableGlobalMethodSecurity(
    securedEnabled = true,
    jsr250Enabled = true,
    prePostEnabled = true)

@ComponentScan("cs544")
public class WebConfig implements WebMvcConfigurer{
```

Can be added to any @Configuration class

Enable which annotations you want

```
<sec:global-method-security
    secured-annotations="enabled"
    jsr250-annotations="enabled"
    pre-post-annotations="enabled"/>
```

Or with XML config

@Secured

- Spring Security's original annotation
 - Specify which Roles are allowed to execute

```
@Service
@Transactional
public class ContactService {
    @Resource
    private ContactDao contactDao;

    @Secured({ "ROLE_USER", "ROLE_ADMIN" })
    public Contact get(Long id) {
        return contactDao.getOne(id);
    }

    @Secured("ROLE_ADMIN")
    public void add(Contact contact) {
        contactDao.save(contact);
    }
}
```

As always you can also add them to the class level to apply to all methods

JSR-250

- Very similar to `@Secured` (but Java standard)

```
@Service
@Transactional
public class ContactService {
    @Resource
    private ContactDao contactDao;

    @RolesAllowed({ "ROLE_USER", "ROLE_ADMIN" })
    public Contact get(Long id) {
        return contactDao.getOne(id);
    }

    @RolesAllowed("ROLE_ADMIN")
    public void add(Contact contact) {
        contactDao.save(contact);
    }
}
```

@Pre / @PostAuthorize

- Modern Spring annotations
 - Can use security expressions
 - Can access arguments / return values

```
@Service
@Transactional
public class ContactService {
    @Resource
    private ContactDao contactDao;

    @PreAuthorize("hasRole('USER')")
    public Contact get(Long id) {
        return contactDao.getOne(id);
    }

    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public void add(Contact contact) {
        contactDao.save(contact);
    }
}
```

Pre or Post

- @PreAuthorize has access to incoming params to make an authorization decision

```
@PreAuthorize("#id < 100")  
public Contact get(Long id) {  
    return contactDao.getOne(id);  
}
```

- @PostAuthorize executes method and then has access to the return (to make a decision)

```
@PostAuthorize("returnObject.name != 'bob'")  
public void add(Contact contact) {  
    contactDao.save(contact);  
}
```

@Pre / @PostFilter

- Filter lets you remove items from a collection
 - @PreFilter can remove from parameter collection
 - @PostFilter can remove from returned collection

```
@PreFilter(value = "filterObject != authentication.principal.username",  
    filterTarget = "usernames")  
public String joinUsernamesAndRoles(List<String> usernames, List<String> roles) {  
    return usernames.stream().collect(Collectors.joining(";"))  
        + ":" + roles.stream().collect(Collectors.joining(";"));  
}  
  
@PostFilter("filterObject != authentication.principal.username")  
public List<String> getAllUsernamesExceptCurrent() {  
    return userRepository.getAllUsernames();  
}
```

Security Summary

- **We've seen:**
 - Authorizing web requests
 - Customizing Login / Logout
 - Spring Security Taglib
 - Authentication Providers
 - Method Security

Applications



CS544 EA

Validation

Validation Overview

- User input is often not reliable
 - Regardless of the mistake being honest / dishonest
 - Can be a real problem to system functionality
- Data Validation is critical for any serious system
 - Can be done by specifying constraints
 - Data is generally stored in Entities
 - Hibernate Validator has become a Java Standard

About Validation

- Validation is the act of ensuring that your data is what you expect it to be
- Ensuring that the program operates on clean, correct and useful data
- Important for:
 - Consistency, reliability, security

See: http://en.wikipedia.org/wiki/Data_validation

Hibernate Bean Validation

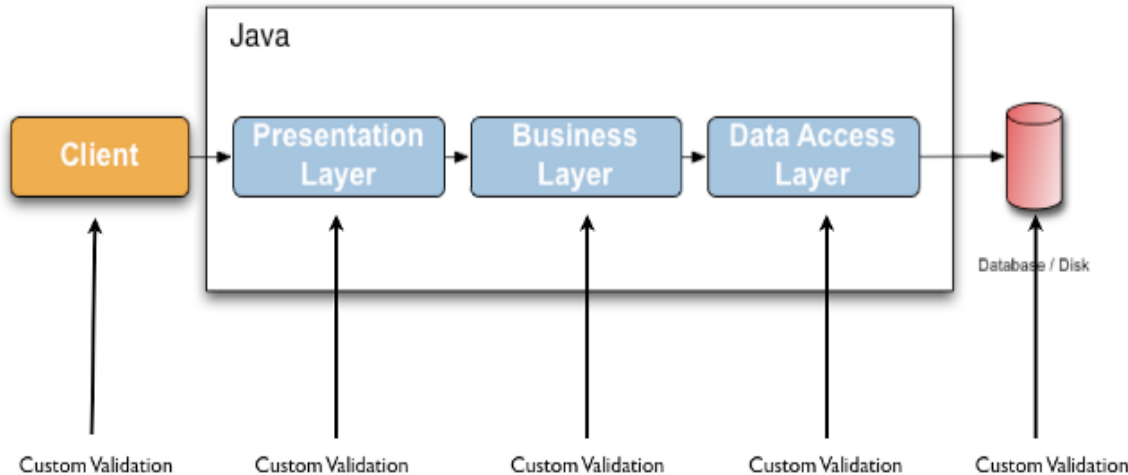
- Validating data is a common task that occurs throughout the application layers
 - Presentation Layer (user input)
 - Business Layer (input from clients)
 - Data Access Layer (check before persisting)

Many of the slides are based on the hibernate validation reference documentation:

http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/

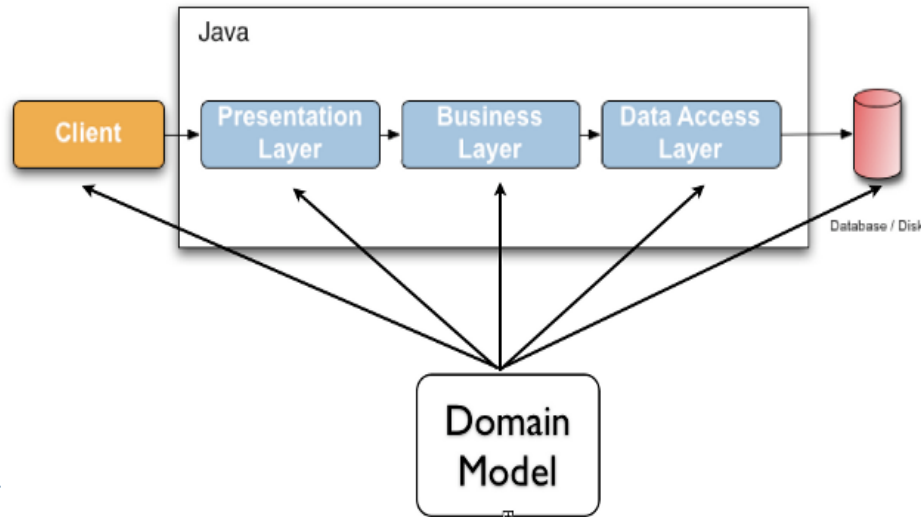
Validation Logic

- Same validation logic could be implemented in each layer
 - Time consuming



Validation in Domain

- Validation logic can be added to the domain
 - But code would clutter domain classes
 - Such code is really just metadata about each class
 - Metadata is best expressed by annotations!



Code Validation VS Annotations

Code

```
@Entity
public class Car {

    ...

    private int year;

    public void setYear(int year) {
        if (year > 1940 && year < 2015){
            this.year = year;
        } else {
            throw new IllegalArgumentException
                ("Invalid year for a Car");
        }
    }
}
```

Annotations

```
@Entity
public class Car {

    ...

    @Range(min = 1940, max = 2015)
    private int year;
}
```

Declaring Bean Constraints

- Constraints can be declared on:
 - Fields (validator framework will use reflection)
 - Properties (Class needs to adhere to JavaBean)
 - Constraint Inheritance (super class / interface)
 - Reference / creating a valid Object Graph
 - Class Level Constraints (always custom)
-

Provided Constraints 1/3

Annotation	Data Types	Description
@Null	Any	Check if it's null (affects column)
@NotNull	Any	Check that it's not null
@NotBlank	String	Not null, trimmed length > 0
@Valid	Any non-primitive	Go into the object and validate it
@AssertFalse	Boolean	Check that it's false
@AssertTrue	Boolean	Check that it's true
@Future	Date or Calendar	Check that it's in the future
@Future OrPresent	Date or Calendar	Future or Present
@Past	Date or Calendar	Check that it's in the past
@PastOrPresent	Date or Calendar	Past or Present
@Size(min=,max=)	String / Collection	Check size is >= min and <= max, column length set to max
@Pattern(regex=,flag=)	String	Check that it matches the regex

Numeric Constraints (2/3)

Annotation	Data Types	Description
@Positive	Numeric types	
@PositiveOrZero	Numeric types	
@Negative	Numeric types	
@NegativeOrZero	Numeric types	
@Min(value=)	Numeric types	Check that it's not lower
@Max(value=)	Numeric types	Check that it's not higher
@DecimalMin(value=,inclusive=)	Numeric types	Check that it's not lower
@DecimalMax(value=,inclusive=)	Numeric types	Check that it's not higher
@Digits(integer=,fraction=)	Numeric types	Checks if it has less digits / fractional points then given

@Min @Max and @Digits also affect DDL, adding constraints on the table column

@DecimalMin and @DecimalMax do not, but their min/max values can be specified as a string which allows you to check beyond Long.MAX_VALUE / Long.MIN_VALUE

Annotation	Data Types	Description
@CreditCardNumber(..)	String	Credit Cards
@EAN	String	Barcode
@Email	String	Email address
@URL(...)	String	URL
@Length(min=,max=)	String	Column length set to max
@LuhnCheck(...)	String	Checksum (mod 10) CC
@Mod10Check(...)	String	Checksum (mod 10)
@Mod11Check(...)	String	Checksum (mod 11) (also used in ISBN)
@ISBN	String	Checks if valid ISBN number
@NotEmpty	String / Collection	Not null or empty
@Range(min=,max=)	Numeric	Checks >= min and <= max
@SafeHtml(...)	String	Requires jsoup, checks for <script> etc
@ScriptAssert(...)	Any Type	Executes JSR 233 script against target

Fields and Properties

Fields

```
public class Car {  
  
    @NotNull  
    private String manufacturer;  
  
    @AssertTrue  
    private boolean isRegistered;  
  
    public Car(String manufacturer,  
                boolean isRegistered) {  
        this.manufacturer = manufacturer;  
        this.isRegistered = isRegistered;  
    }  
  
    //getters and setters...  
}
```

Properties

```
public class Car {  
    private String manufacturer;  
    private boolean isRegistered;  
    public Car(String manufacturer, boolean isRegistered) {  
        this.manufacturer = manufacturer;  
        this.isRegistered = isRegistered;  
    }  
  
    @NotNull  
    public String getManufacturer() {  
        return manufacturer;  
    }  
    public void setManufacturer(String manufacturer) {  
        this.manufacturer = manufacturer;  
    }  
  
    @AssertTrue  
    public boolean isRegistered() {  
        return isRegistered;  
    }  
    public void setRegistered(boolean isRegistered) {  
        this.isRegistered = isRegistered;  
    }  
}
```

Container Types (Collections)

- **Bean Validation 2.0 also adds support for:**
 - Container constraints
 - Container cascades
 - Example:

```
private Map<@Valid @NotNull OrderCategory, List<@Valid @NotNull Order>> OrderByCategory
```

Inheritance

```
public class Car {  
    private String manufacturer;  
  
    @NotNull  
    public String getManufacturer() {  
        return manufacturer;  
    }  
  
    // ...  
}
```

```
public class RentalCar extends Car {  
  
    private String rentalStation;  
  
    @NotNull  
    public String getRentalStation() {  
        return rentalStation;  
    }  
  
    //...  
}
```

When validating RentalCar
both manufacturer and rentalStation
will be validated

Also works with interfaces

Object Graph

```
public class Car {
```

```
public class Car {
```

```
@NotNull
```

```
@Valid
```

```
private Person driver;
```

```
//...
```

```
}
```

```
public class Person {
```

```
@NotNull
```

```
private String name;
```

```
//...
```

```
}
```

When validating Car
the @Valid makes the validator
cascade into Person and check
that its name is @NotNull

Class Level

```
@ValidPassengerCount
```

```
public class Car {
```

```
    private int seatCount;
```

```
    private List<Person> passengers;
```

```
    //...
```

```
}
```

You can make a custom class level annotations to check the relationship between properties

@ValidPassengerCount checks that:
passengers.size() <= seatCount
(see next slide)

Custom Constraint Annotation

```
@Target({ TYPE, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Constraint(validatedBy = { ValidPassengerCountValidator.class })
@Documented
public @interface ValidPassengerCount {

    String message() default "{org.hibernate.validator.referenceguide.chapter06.classlevel." +
        "ValidPassengerCount.message}";

    Class<?>[] groups() default { };

    Class<? extends Payload>[] payload() default { };
}
```


Custom Validator

```
public class ValidPassengerCountValidator
    Implements ConstraintValidator<ValidPassengerCount, Car> {

    @Override
    public void initialize(ValidPassengerCount constraintAnnotation) {
    }

    @Override
    public boolean isValid(Car car, ConstraintValidatorContext context) {
        if (car == null) {
            return true;
        }
        return car.getPassengers().size() <= car.getSeatCount();
    }
}
```

Applications



CS544 EA

Validation: Programmatic Validation

Programmatic Validation

```
public class App {  
    public static void main(String[] args) {  
        ValidatorFactory factory = Validation.buildDefaultValidatorFactory();  
        Validator validator = factory.getValidator();  
  
        Car car = new Car( null, true );  
  
        Set<ConstraintViolation<Car>> constraintViolations =  
            validator.validate( car );  
  
        assertEquals( 1, constraintViolations.size() );  
        assertEquals( "may not be null",  
            constraintViolations.iterator().next().getMessage() );  
    }  
}
```

Create a validator

Car with @NotNull manufacturer and @AssertTrue registered

Validate the car

Check if it worked

Checking a Single Prop / Value

▶ If you don't want to validate an entire object:

- ▶ You can validate individual values

Field using reflection

- ▶ You can validate individual properties

Field using getter

 - ▶ Uses JavaBean property name (no get, lowercase first)

- ▶ These will not follow @Valid annotations

```
Set<ConstraintViolation<Car>> constraintViolations = validator.validateValue(  
    Car.class,  
    "manufacturer",  
    null );
```

Checks if manufacturer is null

```
Set<ConstraintViolation<Car>> constraintViolations = validator.validateProperty(  
    Car.class,  
    "manufacturer",  
    null );
```

Checks if manufacturer is null



Constraint Violation Methods

Method	Description	Example
getMessage()	The error message	“may not be null”
getMessageTemplate()	The name in the bundle	{...NotNull.message}
getRootBean()	Root of object graph	Car
getRootBeanClass()	Class or root bean	Car.class
getLeafBean()	‘leaf’ the constraint is on	Person
getPropertyPath()	From root to property	Car.Person.name
getInvalidValue()	Value failing the constraint	null
getConstraintDescriptor()	Access to annotation etc.	@NotNull

Applications



CS544 EA

Validation: SpringMVC Integration

Pom.xml

- If SpringMVC detects a validator implementation on the classpath
 - It automatically integrates with it

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>6.0.9.Final</version>
</dependency>
```

WebAppInitializer

```
public class MyWebAppInitializer implements WebApplicationInitializer {  
  
    @Override  
    public void onStartUp(ServletContext container) throws ServletException {  
        // Create the 'root' Spring application context  
        AnnotationConfigWebApplicationContext rootContext =  
            new AnnotationConfigWebApplicationContext();  
        rootContext.register(WebConfig.class);  
        container.addListener(new ContextLoaderListener(rootContext));  
  
        // Create the dispatcher servlet  
        ServletRegistration.Dynamic appServlet = container.addServlet("mvc",  
            new DispatcherServlet(new GenericWebApplicationContext()));  
        appServlet.setLoadOnStartup(1);  
        appServlet.addMapping("/");  
    }  
}
```

Regular Initializer
nothing extra needed
for validation

@Configuration

```
@Configuration
@EnableWebMvc
@ComponentScan("cs544")
public class WebConfig implements WebMvcConfigurer{
    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
        messageSource.setBasename("i18/errormsg");
        return messageSource;
    }

    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver bean = new InternalResourceViewResolver();

        bean.setViewClass(JstlView.class);
        bean.setPrefix("/WEB-INF/view/");
        bean.setSuffix(".jsp");

        return bean;
    }
}
```

Optional bean for
custom validation messages

XML version

```
<bean id="messageSource1"
      class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="messages" />
</bean>
```

Car Class

With validation annotations

@Entity

public class Car {

 @Id

 @GeneratedValue

private int id;

 @NotEmpty

private String make;

 @NotEmpty

private String model;

 @Range(min = 1940, max = 2015)

private int year;

private String color;

Controller

@Controller

```
public class CarController {  
    @Resource  
    private CarService carService;  
  
    @GetMapping("/addCar")  
    public String addCar(@ModelAttribute("car") Car car) {  
        return "addCar";  
    }  
  
    @PostMapping("/addCar")  
    public String add(@Valid Car car, BindingResult result, RedirectAttributes attr){  
        if (result.hasErrors()) {  
            attr.addFlashAttribute("org.springframework.validation.BindingResult.car", result);  
            attr.addFlashAttribute("car", car);  
            return "redirect:/addCar";  
        } else {  
            carService.add(car);  
            return "redirect:/cars";  
        }  
    }  
}
```

ModelAttribute needed
because view has to have
an object to pull from

Order of args important!
BindingResult has to be
right after @Valid arg

Package name is needed for the
binding result to show after redirecting

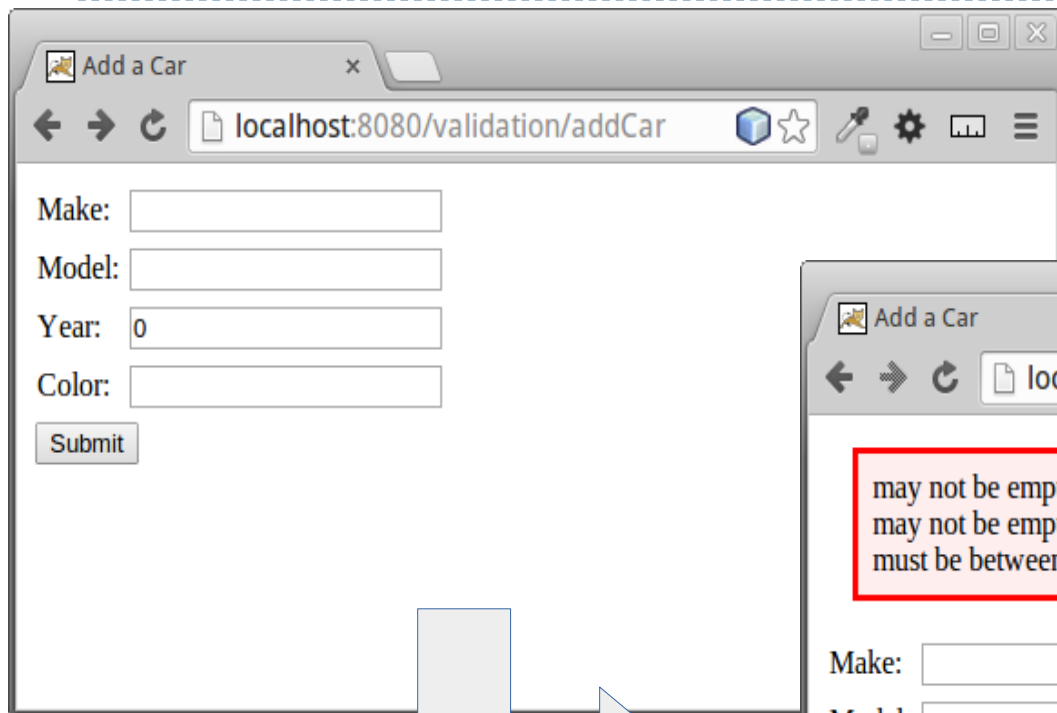
```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Add a Car</title>
    <link href="resources/style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <form:form modelAttribute="car" action="addCar" method="post">
      <form:errors path="*" cssClass="errorblock" element="div" />
      <table>
        <tr>
          <td>Make:</td>
          <td><form:input path="make" /> </td>
          <td><form:errors path="make" cssClass="error" /> </td>
        </tr>
        <!-- Model year and color removed to keep the slide shorter -->
      </table>
      <input type="submit"/>
    </form:form>
  </body>
</html>
```

ModelAttribute or
CommandName refers
to the bean from which
data should be used

Path attribute specifies
which field on that bean

Spring form error tags
display validation errors
for their fields

Result



Add a Car

localhost:8080/validation/addCar

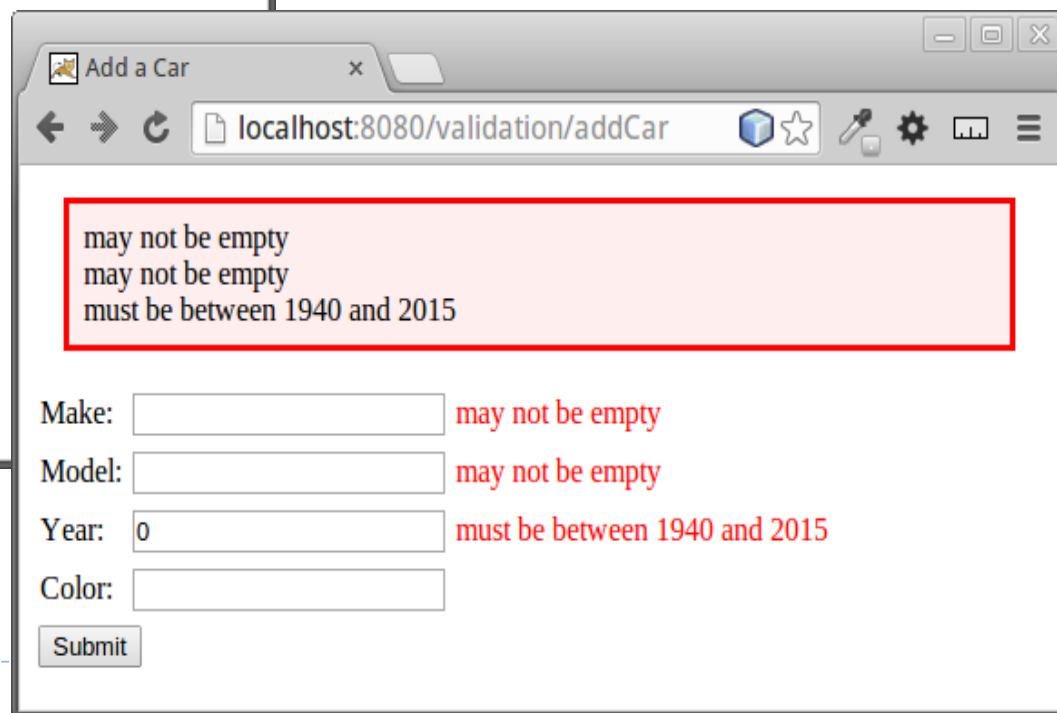
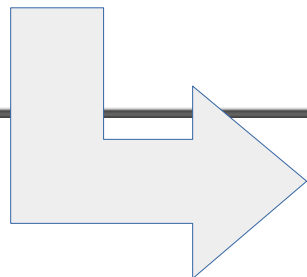
Make:

Model:

Year:

Color:

Submit



Add a Car

localhost:8080/validation/addCar

may not be empty
may not be empty
must be between 1940 and 2015

Make: may not be empty

Model: may not be empty

Year: must be between 1940 and 2015

Color:

Submit

Spring Form Tags

Validation Summary

- Validation ensures that we operate on clean, correct, and useful data.
- With annotations we declare what constraints should be applied.
- There are built in annotations, and it's easy to create custom annotations.
- A `ConstraintViolation` object is created for every item that fails to validate.
- Spring MVC will integrate without configuration if it detects a validator implementation on the classpath
- Spring Form error tags display any error messages related to their field