

Exam Objectives

Spring 3.0 Exam Objectives

The following detailed objectives are from [Spring 3 Certification Study Notes.pdf](#) and are **copied / reposted** from [Jeanne Boyarsky's Spring 3.X Certification Experiences](#). They are similar to those in the official [Spring Certification Study Guide](#), with some objectives reworded and split into multiple rows for clarity. All meanings have (hopefully) stayed intact. A couple of objectives that looked important, but were not in the study guide were added. These are identified by {these} and are probably not necessary for the test.

Objective	Notes
1. Container	
1.1 General	
The potential advantages of using Spring's dependency injection	Code is more flexible, unit testable, loosely coupled and maintainable. Can autowire using no, byName, byType, constructor or autodetect (constructor or byType depending on class) Can do dependency checking by none, simple (everything except collaborators), object (collaborators), all (both). @Required equivalent to "all"
Dependency injection in XML, using constructor	<bean id="b" class="x"> <constructor-arg ref="other" /> </bean> Can take optional index (0 based) or type parameters
Dependency injection in XML, using setter injection	<bean id="b" class="x"> <property name="my Field" value="a" /> </bean>
Default scope for Spring beans.	Singleton
What are the main scopes that can be used alternatively?	Prototype – new each time, init called, destroy not called, ignores lazy-init, session, request singleton – one per container [also globalSession (portlets) and custom which I think are out of scope] [web flow scopes which are out of scope]
1.2 Lifecycle	
How to declare an initialization method in a Spring bean [in XML]	init-method attribute in bean tag – can be a private method because reflection used, can return value which is ignored, must take 0 parameters
How to declare a destroy method in a Spring bean [in XML]	destroy-method attribute in bean tag It gets called when you call (ConfigurableApplicationContext ctx).close()
Default bean instantiation policy: when are beans instantiated? Note: you should have a good understanding of the lifecycle in general.	For singleton beans, when the application context is created/parsed <ol style="list-style-type: none"> 1. Load all bean definitions creating ordered graph 2. Instantiate and run BeanFactoryPostProcessors (can update bean definitions here) 3. Instantiate each bean

	<ol style="list-style-type: none"> 4. set properties 5. BeanPostProcessors – postProcessBeforeInitialization 6. @PostConstruct 7. InitializingBean – afterPropertiesSet 8. init-method 9. BeanPostProcessors – postProcessAfterInitialization 10. ... do stuff 11. @PreDestroy 12. DisposableBean – destroy 13. destroy-method
BeanFactoryPostProcessors and BeanPostProcessors? When are they called in the startup process?	<p>BeanFactoryPostProcessors – run once after XML is parsed. Can change the configuration/definition before the beans are instantiated. Property Placeholder configurator is an example of one.</p> <p>BeanPostProcessors – run multiple times (one for each bean) – contain postProcessBeforeInitialization and postProcessAfterInitialization returning original bean</p>
{What is a BeanFactory?}	Factory bean that creates another type of bean. Contains getObject(), getObjectType() and isSingleton(). Use "&name" to get reference to bean factory itself rather than created object.
1.3 Annotations	
Enabling component-scanning	<context:component-scan base-package="mine" /> looks for @Component (general), @Controller (web), @Repository (dao) and @Service (service) can set context:include-filter and context:exclude-filter
Behavior of the annotation @Autowired with regards to field injection, constructor injection and method injection	<p>Default is by type</p> <pre>@Autowired public MyClass(fields) {} @Autowired private MyClass c; - if annotate field, can't have setter present @Autowired public setValue(field) {}</pre> <p>If > 1 match, throws exception (unless injecting into Collection). If no matches, sets to null unless using @Required or required attribute</p>
How does the @Qualifier annotation complement the use of @Autowired	<p>@Autowired @Qualifier("name") or @Resource (not Spring specific) chooses specific bean when more than one exists of same type Can also put on specific parameter</p>
What is the role of the @PostConstruct and @PreDestroy annotations	Equivalent to init-method and destroy-method
Enabling the scanning of annotations such as @Required and @PreDestroy	<context:annotation-config /> or <context:component-scan /> which includes it @Required goes on setters, not fields @PreDestroy goes on methods
{Other interesting annotations}	<p>@Value("#{systemProperties['name']}") to inject SpEL value @Scheduled @Async</p>
1.4 Miscellaneous	
How to inject scalar/literal values into Spring Beans	<pre><value>one</value> <bean id="a" value="b" /> <null /></pre>

How to refer to a collection in a Spring bean definition	<pre><list> <ref bean="one" /> </list> <set> <ref bean="one" /> </set> <props> <prop key="a" value="b" /> </props> <map> <entry key="a" value="b" /> </map></pre> <p>Note: If need to refer to this outside a <bean /> or need to specify list/set/map type, it needs to util: namespace.</p>
How to create an ApplicationContext..	<pre>ApplicationContext ctx = new ClasspathXmlApplicationContext("one.ctx", "two.ctx");</pre>
What are the resource prefixes that can be used	<pre>classpath: file: http:</pre>
How to refer to a Spring configuration file inside a package.	<pre>new ClasspathXmlApplicationContext("/com/mine/app-config.xml");</pre>
The different implementations of ApplicationContext that can be used in an application.	<p>ClasspathXmlApplicationContext – defaults to classpath:, starts looking from root of classpath regardless of whether specify "/"</p> <p>FileSystemXmlApplicationContext – defaults to file:, uses relative path even ifspecify "/". But "file:/mine" forces an absolute path</p> <p>XmlWebApplicationContext – defaults to file: in /WEB-INF/applicationContext.xml, uses path relative to web application</p>
How to externalize constants from a Spring XML configuration file into a .properties file	<pre><context:property-placeholder location="/myFile.properties" /> <bean id="mind" class="Mine"> <property name="a" value="{value.from.property.file}" /> </bean></pre>
Purpose and usage of bean definition Inheritance	<p>Extract common bean setup into one "super" bean to reduce duplicate setup code</p> <pre><bean id="parent" class="Mine" abstract="true" /> <bean id="child" parent="parent" class="Mine"/></pre> <p>A bean without a class or factory is implicitly abstract.</p>
How to use the p namespace	<p>Set property as attributes</p> <pre><bean id="a" class="Mine" p:propName="value" /> <bean id="a" class="Mine" p:propName-ref="otherBean" /></pre>
Difference between "id" and "name" attributes in the <bean> tag	<p>id – must be unique</p> <p>name – can specify multiple names separated by comma, semicolon or space, allows more characters in name such as slashes</p>
Purpose of the <aop:scoped-proxy/> tag	<p>If need to inject a bean with request/session scope into a bean with singleton scope, the proxy intercepts it and makes sure you get the right bean for each call.</p>
{How does factory bean work – from 2.5 sample questions}	<p>Class name in bean.xml is that of the factory, setters called on the factory rather than created bean, but bean id maps to type returned by factory's getObject()</p>

{How to inject using a factory method}	factory-bean/factory-method for instance method class/factory-method for static method – uses constructor-arg
{How to use a PropertyEditor}	Extend PropertyEditorSupport and implement setAsText(String text) calling setValue() and String getAsText() calling getValue().
{Splitting the XML}	<import resource="local path" /> use classpath:/path if need absolute path
{Other util namespace items}	<util:constant static-field="fully qualified name" /> <util:properties location="property file" />
{Spring EL}	#{systemProperties.name}
1.5 JavaConfig	http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html
Usage of the @Bean and @Configuration annotations	@Bean – declared on a method that creates a bean in a Java based configuration. Takes properties initMethod (redundant because can call method), destroyMethod, name (aliases), autowire (defaults to no) @Configuration – declares a class to be a Java based configuration, must have default constructor @Scope("prototype") if want to change scope @Primary if want to specify which bean to inject @Import to reference another @Configuration – can't just call because must be in correct order in dependency graph
How to write a bean definition method	@Bean public String myString() { return "hi"; } defines a String bean with the id "myString" and the value "hi"
1.6 Testing	http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/testing.html
How to use Spring's integration testing support in a JUnit 4 test	@RunWith(SpringJUnit4ClassRunner) @ContextConfiguration("config.xml") @ContextConfiguration(locations={"config.xml"}) Defaults to testclassname-context.xml The context is cached across tests/test classes that use same config files unless you use @DirtiesContext.
How to declare that a test should run in a transaction in spring-enabled JUnit 4	@TransactionConfiguration – on class to set transaction manager and default rollback policy @Transactional – on class or method, can set propagation/isolation/etc @Rollback – on class or method @BeforeTransaction (@Before is inside txn) @AfterTransaction (@After is inside txn)
Differences between a Unit test and an Integration test. Which of them is supposed to interact with Spring?	Unit test tests one class in isolation with mocks. Integration test tests multiple classes together. Only integration tests interact with Spring.
2. AOP	

<h2>2.1 Recommendations</h2>	
<p>In order to work successfully with AOP, it is important to understand the theory behind the scenes. Consequently you should understand the four main AOP keywords taught in this course:</p> <ul style="list-style-type: none"> Aspect Advice Pointcut Joinpoint 	<p>Packaged advice and pointcut Code to execute Expression to identify join point. Place in code where code can be injected. In Spring this is just before/after methods.</p>
<p>{What problems occur with cross cutting concerns – from Spring 2.5 sample questions}</p>	<p>Code tangling – method does unrelated things Code scattering – duplication</p>
<h2>2.2 Pointcuts</h2>	
<p>You should have a deep understanding of pointcut expressions. It is important to understand all pointcut examples in the slides (apart from the ones in the “advanced topics” section at the end of the module).</p>	<p>execution([Modifiers such as annotations] ReturnType [FullyQualifiedClassName.] MethodName ([Arguments]) [throws ExceptionType]) * exactly 1 of anything (argument, characters in package name, etc) .. 0 or more (arguments or packages) + this class or any implementors (no matter how deep) &&, , ! (and, or and not) other expressions I think are outside the scope of the exam: within, target, this,...</p>
<h2>2.3 Advice</h2>	
<p>Different kinds of Advice. How do they differ from each other?</p>	<p>@Before – before call code – aborts if throw exception @After – after unconditionally @AfterReturning – after on success – takes JoinPoint and return value, can change return value thrown @AfterThrowing – after on failure – takes JoinPoint and exception, can change exception type thrown @Around – surrounds – takes ProceedingJoinPoint, can intercept call or eat exception prefer least powerful advice that can do the job</p>
<h2>2.4 Configuration</h2>	
<p>Enabling the detection of @Aspect annotated classes.</p>	<pre><aop:aspectj-autoproxy /></pre>
<p>Is it possible to set up Spring AOP using XML configuration only (as opposed to annotations)?</p>	<pre>Yes. <aop:config> <aop:aspect ref="adviceBean"> <aop:before pointcut="execution..." /> </aop:config></pre>
<h2>2.5 Proxies</h2>	
<p>When are they generated in the Spring lifecycle?</p>	<p>Bean Post Processing</p>

How do they resemble the target object that they advice?	They have the same public interface if there is an interface. You must have an interface or non-final class to create a proxy.
What limitations does Spring-AOP's proxy-based approach have?	Can only be used for methods, limited pointcut syntax, doesn't get called if caller and callee are methods in same class
3. Data Access and transactions	
<u>3.1 General</u>	
The rationale for having the DataAccessException hierarchy Definition of a DataSource in the Spring configuration	Runtime exception – doesn't force developers to catch and rethrow Consistent/non vendor specific messages – isolates from database Clearer names – easy to catch and handle specific errors If in JNDI: <jee:jndi-lookup id="ds" jndi-name="java:comp/env/jdbc/DS" /> If not in JNDI: <bean id="ds" class="DataSourceClass">properties for standard jdbc properties</bean>
<u>3.2 The JdbcTemplate</u>	
Usage of the JdbcTemplate with regards to exception handling	JdbcTemplate jdbc = new JdbcTemplate(datasource); Callback methods throw SQL Exception and Spring converts to DataAccessException Or use SimpleJdbcTemplate when can Or NamedParameterJdbcTemplate when want param names rather than #s
With regard to querying [generically]	jdbc.queryForInt("select count(*) from table"); generic queryFor_____ methods available for: primitives, Object Map – column name is key List – list of maps
With regard to querying [with parameters]	jdbc.queryForInt("select count(*) from table where col > ?", id);
With regard to result set parsing [when need to map to custom object]	RowMapper<A> mapper = new RowMapper<A>() { public A mapRow(ResultSet rs, int row) throws SQLException {} }; jdbc.query(sql, mapper); - for list jdbc.queryForObject(sql, mapper); - for single row
With regard to result set parsing [when need to write out data to file but not return it]	RowCallbackHandler handler = new RowCallbackHandler() { public void processRow(ResultSet rs) {} }; jdbc.query(sql, handler);
With regard to result set parsing [merging rows into a list or other return object]	ResultSetExtractor<A> extractor = new ResultSetExtractor<A>() { public A extractData(ResultSet rs) {} }; jdbc.query(sql, extractor);

With regard to querying [for insert/update/delete row]	<code>jdbc.update(sql);</code>
With regard to running DDL	<code>jdbc.execute(sql);</code>
With regard to batch updates	<pre>BatchPreparedStatementSetter setter = new BatchPreparedStatementSetter() { public void setValues(PreparedStatement stmt, int row) {} public int getBatchSize() { return 0; } }; with jdbcTemplate.batchUpdate(sql, setter) or pass Object[] as second parameter</pre>
With regard to when need more control	PreparedStatementCreator and CallableStatementCreator create from connection SimpleJdbcInsert and SimpleJdbcCall use a map rather than callbacks relying on database metadata.
<u>3.3 Hibernate</u>	
Configuration of a SessionFactoryBean in xml [listing each entity]	<pre><bean id="f" class="os.orm.hibernate3.annotation.AnnotationSessionFactoryBean"> <property name="dataSource" ref="d" /> <property name="annotatedClasses"> <list><value>Class1</value><value>Class2</value></list> </bean></pre>
Configuration of a SessionFactoryBean in xml [scanning for annotations]	<pre><bean id="f" class="os.orm.hibernate3.annotation.AnnotationSessionFactoryBean"> <property name="dataSource" ref="d" /> <property name="packagesToScan"> <list><value>com/mine/*/entity</value></list> </bean></pre>
Configuration of a SessionFactoryBean in xml [listing each hbm file]	<pre><bean id="f" class="os.orm.hibernate3.LocalSessionFactoryBean"> <property name="dataSource" ref="d" /> <property name="mappingLocations"> <list><value>classpath:/package/hbm.xml</value></list> </bean></pre>
{What are benefits of transactions for Hibernate?}	Read only transactions prevent Hibernate from flushing session
<u>3.4 Transactions</u>	
Configuration to declare a local transaction manager	<pre><bean id="mgr" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"> <property name="dataSource" ref="dataSource"/> </bean></pre> <p> HibernateTransactionManager – supports Hibernate and JDBC JdbcTransactionManager – only supports JDBC </p>
Configuration to declare a JTA transaction manager	<pre><bean id="mgr" class="org.springframework.transaction.jta.JtaTransactionManager" /></pre>
Declarative transaction management with Spring [creating the transaction]	<p>If server provides JTA: <code><tx:jta-transaction-manager /></code></p> <p>Otherwise: <code><bean id="transactionManager"</code></p>

with Spring [creating the transaction manager]	<pre> <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"> <property name="dataSource" ref="ds" /> </bean> </pre>
Declarative transaction management with Spring (xml)	<pre> <aop:config> <aop:advisor pointcut-ref="p" advice-ref="ta" /> </aop:config> <tx:advice id="ta"> <tx:attributes> <tx:method name="update*" timeout="60" /> <tx:method name="*" timeout="30" read-only="true" /> </tx:attributes> </tx:advice> </pre>
Declarative transaction management with Spring (annotations)	<p>@Transactional on public methods or class (not recommend on interfaces b/c not all proxies will take them). @Transactional on non-public methods will not give error, but will not be in transaction.</p> <p><tx:annotation-driven /> to enable</p>
Usage of TransactionTemplate for programmatic transaction management (you simply need to understand a basic example based on the TransactionTemplate).	<p>Takes transaction manager in constructor.</p> <p>Passes TransactionStatus to TransactionCallback which lets you call status.setRollbackOnly()</p>
Transactions and exception management: what is the default rollback policy?	Rollback for a RuntimeException
Can it be overridden?	Yes. Specify rollbackFor or noRollbackFor when passing Class or rollbackForClassname/noRollbackForClassname when passing Strings
The @Transactional annotation: what are the main attributes that can be used for a transaction definition?	<p>timeout – specified in seconds</p> <p>readOnly – true or false</p> <p>isolation – Isolation.READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ or SERIALIZABLE</p> <p>propagation – Propagation.REQUIRES_NEW, REQUIRED, etc</p> <p>rollbackFor – list of exceptions</p> <p>noRollbackFor – list of exceptions</p>
Can this annotation also be used with a JTA Transaction Manager?	Yes
Regarding propagation modes, what is the difference between PROPAGATION_REQUIRED and PROPAGATION_REQUIRES_NEW	<p>Requires new creates a new transaction regardless of whether one exists.</p> <p>Required uses the existing transaction and throws an exception if there isn't one.</p> <p>Required is the default. The others are MANDATORY, NESTED (uses savepoints so only for JDBC, NEVER, NOT_SUPPORTED, and SUPPORTS).</p>
Regarding isolation levels, what is the difference between READ_COMMITTED and READ_UNCOMMITTED?	<p>Read uncommitted allows reading dirty/uncommitted data. Read committed does not. The default is DEFAULT - "default isolation level for your database".</p> <p>While this is usually READ_COMMITTED, the actual default is DEFAULT.</p> <p>The others are REPEATABLE_READ which guarantees the same result within a transaction and SERIALIZABLE which prevents phantom reads too.</p>
4. Spring MVC and REST	

4.1 General configuration	
This module shows how to configure a ViewResolver, a HandlerMapping and the DispatcherServlet. You won't be asked any question on how to configure those classes. However, you need to know what the goal of each of those components is.	View Resolver – Maps returned view name to view implementation. Can also return null/void (to use default view) or a concrete class such as new JstlView(path) Handler Mapping – Identifies correct controller to call. Spring 3 uses the default one which goes by the @RequestMapping annotation defined in the @Controller Dispatcher Servlet – front controller delegating to web infrastructure beans (handler mappings, view resolvers, type converters) and calling controllers/views
You also need to understand the relationship between a DispatcherServlet ApplicationContext and a root ApplicationContext.	DispatcherServletApplicationContext can see the root context's configuration, but the root context can not see DispatcherServletApplicationContext
{How access app context from a servlet}	WebApplicationContextUtils.getRequiredWebApplicationContext(servletContext)
{How turn on annotation scanning}	<mvc:annotation-driven /> needed in addition to <context:component-scan />
4.2 Controllers	
Bootstrapping a root WebApplicationContext using the ContextLoaderListener	The DispatcherServlet takes a contextConfigLocation parameter in the web.xml or uses the default of name-servlet.xml Also need to define listener so loads the root ApplicationContext before initializing the servlet. <listener><listener-class>org.springframework.web.context.ContextLoaderListener</listener-class></listener>
General usage of the @Controller annotations	Annotates a class containing @RequestMapping on methods returning ModelAndView, String, void, etc.
General usage of the @RequestMapping annotations	Takes value of path of URL to match on method level and optionally class level adding slashes as needed. Combined, they form the absolute URL path including Ant style wildcards. Can also pass method=RequestMethod.GET (or the like) to restrict by type or filter by params/headers.
A method annotated with @RequestMapping can return a String. What does it refer to?	The view name. Or more specifically the name passed to the view resolver to determine the view name. For example, one can add a prefix/suffix to all strings returned.
What are the main parameter types that this method can accept? (based on what you've seen in the class)	Model – set attributes to be used in the view HttpSession HttpServletRequest HttpServletResponse
Goal of the @RequestParam annotation	To map request parameters to method parameters for use in the controller's methods. Can pass parameter name if doesn't match one in method.
4.3 REST	
Differences between GET, POST, PUT and DELETE	GET = select (read only) POST = create PUT = update

	DELETE = delete
Usage of the <code>@PathVariable</code> annotation	Maps parts of the url in <code>@RequestMapping</code> to a parameter. For example the URL <code>/account/{id}</code> goes with <code>@PathVariable id</code> in the method signature.
What is the <code>RestTemplate</code> ?	Programmatic client for calling RESTful URLs.
How should it be used?	<pre>RestTemplate t = new RestTemplate(); t.getForObject(uri, Mine.class, id); t.postForLocation(uri, mine, id); (or postForObject) t.put(uri, mine); t.delete(uri);</pre>
Purpose of the <code>@ResponseStatus</code> Annotation	Send an HTTP response code with the response. If defined on the method, sends that header. If defined on an exception, only sends if that error occurs. For example, <code>@ResponseStatus(HttpStatus.CREATED)</code> An empty body can be used for REST so no view is used. Annotation can go on <code>@RequestMapping</code> method or an exception class.
Purpose of the <code>@ExceptionHandler</code> Annotation	If cannot annotate the actual exception, defines a response status in controller. For example, <code>@ResponseStatus(HttpStatus.NOT_FOUND)</code> <code>@ExceptionHandler({MyException.class})</code>
{What are <code>HttpMessageConverters</code> }	Map from HTTP request/response to Java object. Use <code>@RequestBody</code> to map param and <code>@ResponseBody</code> to map return value
5. Advanced topics	
<u>5.1 Remoting</u>	
Advantages of using Spring Remoting rather than plain RMI?	Hide plumbing/complexity, support multiple protocols, simplify things when both ends of remote call run Spring
Goal of the RMI Service Exporter	Handle server side of RMI interaction – bind to registry, expose endpoint (used for everything except EJB; EJB already has mechanism to export remote)
General configuration of the RMI Service Exporter	<pre><bean class="org.springframework.remoting.rmi.RmiServiceExporter"> and set properties: serviceName (name in registry) serviceInterface (interface classname) service (reference to POJO implementing serviceInterface) </bean></pre>
Goal of RMI Proxy Generator	Handle client side of RMI interaction – communicate with endpoint, convert remote exception to runtime exception, can treat remote EJBs as POJOs
General configuration of RMI Proxy Generator	<pre><bean class="org.springframework.remoting.rmi.RmiProxyFactoryBean"> and set properties: serviceInterface (interface classname) serviceUrl (rmi://server:port/serviceName) </bean></pre>

Difference between the RMI Service Exporter and the HttpInvoker	HttpInvoker uses HTTP POST while RMI one uses RMI. To use create bean HttpInvokerServiceExporter with service (reference) and service interface (class name). The serviceName is specified as the name of the bean because default is to use BeanNameUrlHandlerMapping. Also create bean HttpInvokerProxyFactoryBean with same properties as for RMI.
Does the HttpInvoker require to run a web server on the client side? On the server side?	Needs web server on the server side, but not the client side. XML over HTTP. Hessian is binary, Burlap is textual. Proprietary approach to serialization
{What about Hessian and Burlap?}	To use, see instructions for HttpInvoker.
5.2 Security	
What is the "Security filter chain" used in Spring Security?	Series of servlet filters that must be passed before/after resource is accessed. These include loading HttpSession's context, logging out if needed, authenticating if needed, throwing proper exception (after only) and checking role. Can add your own or replace an existing filter. In web.xml: <filter> <filter-name>springSecurityFilterChain</filter-name> <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter>
Syntax to configure Spring Security to intercept particular URLs? (you need to have a good understanding of the various examples using intercept-url in the course)	<security:http> <security:intercept-url pattern="/stuff/**" [method="GET"] access="role1, role2" /> Pattern uses Ant syntax (** is any level of directories), can change to reg exp Access can get IS_AUTHENTICATED_FULLY or a list of roles If more than one intercept-url, matched top to bottom filters="none" - no security applied
Syntax to configure method level security Method level security using @Secured or @RolesAllowed	<security:global-method-security secured-annotations="enabled" /> <security:global-method-security jsr250-annotations="enabled" /> or without annotations: <security:global-method-security> <security:protect-pointcut expression="execution(* doStuff())" access="role" /> </security:global-method-security> Pass allowed roles. @Secured is Spring's annotation and takes an array of roles @RolesAllowed is the JSR one.
Possible mechanisms to store user details: database? LDAP? Others?	<security:authentication-manager> <security:authentication-provider> Database: <security:jdbc-user-service data-source-ref="ds" /> In memory: <security:user-service properties="a.properties" /> hard coded in xml, LDAP, JAAS, SSO, JDBC, Open Id, etc
When working with an authentication provider, is it possible to use a password encoder?	Yes. <security:password-encoder hash="abc" />
In the security tag library, what is the purpose of the <security:authorize /> tag?	To say what roles are permitted access to a section of the JSP. <security:authorize ifAnyGranted="roles">do stuff</security:authorize>

5.3 JMS	
Purpose of the JmsTemplate	Simplify usage of JMS, reduce boilerplate code, handle exceptions, etc
{What properties does JmsTemplate require}	Connection Factory defaultDestination – not required, but common
{What callbacks are available}	MessageConverter - if need to map object to/from Message for nonstandard type DestinationResolver – map destination name to actual destination
{How to send a message}	template.convertAndSend(); - takes message and optional Destination name/object (uses default if not specified) Or call send() if need MessageCreator callback or execute() if need advanced callback during template All send methods put message on queue and resume
{How to receive a message}	template.receive() take optional destination or receiveAndConvert() if set MessageConverter All receive methods are blocking/synchronous
How to declare a JMS Listener?	Implement MessageListener or SessionAwareMessageListener <jms:listener-container connection-factory="connFactoryRef"> <jms:listener destination="dest" ref="listenerBeanRef" /> </jms:listener-container>
Can a JMS Listener be a POJO?	Yes, just register in XML inside a listener-container: <jms:listener ref="pojo" method="m" destination="d" response-destination="rd" /> destination is where we listen for a message response-destination is needed if the method doesn't return void
Is it possible to run JMS inside a plain Servlet container without full Java EE support (such as Tomcat or Jetty)?	Yes. Can use SimpleMessageListenerContainer for basics or DefaultMessageListenerContainer if need transactions Takes as properties: connectionFactory, destination, messageListener
{what configuration is necessary}	Create connection factory and queues by creating bean for standalone or <jee:jndi-lookup jndi-name="jms/aaa" /> Can wrap with CachingConnectionFactory if need caching
5.4 JMX	
Role of the MBeanExporter	Expose POJO as MBean
Using Spring JMX, is it possible to export a POJO as an MBean?	Yes. Register as a bean and use MBeanExporter <bean class="org.springframework.jmx.export.MBeanExporter"> <property name="beans"> <util:map> <entry key="bean:name=mine" value-ref="beanRef" /> </util:map> </property> </bean>
Using Spring JMX, is it possible to automatically register an existing MBean with an MBeanServer?	Yes, just register them as a bean in the XML config and Spring will notice they are already MBeans. (these are classes implementing class ending in MBean)

{How to declare an MBeanServer}	<context:mbean-export /> turns on looking for MBeans <bean id="mine" class="ExistingMBean" /> <context:mbean-server /> - or declare as bean so can control whether to use existing server
Purpose of the @ManagedResource, @ManagedAttribute and @ManagedOperation annotations	<context:mbean-export /> turns on annotation scanning @ManagedResource(objectName="mine") - identifies class as MBean @ManagedAttribute – expose field to JMX – place on getter and setter @ManagedOperation – expose method to JMX