1. ApplicationContext is a BeanFactory
2. Autowire Attribute Description:

* byName: The container attempts to find beans with the same name as the property being wired. For example, if the property to be set is called operation, the container will look for a bean with id=”operation”. If such a bean is not found, an error is raised.
* byType: The container examines the argument type of the setter methods on the bean, and tries to locate a bean with the same type. An error is raised when more than one bean with the same type exists, or when there is no bean of the required type.
* constructor: The container checks for a constructor with a typed argument,and tries to find a bean with the same type as the argument to the constructor and to wire that bean during a call to the constructor. An error is raised when more than one bean with the required type exists, or when there is no bean of the required type.
* autodetect: The container performs either the constructor or byType setter autowiring by examining the bean to be wired. It raises an error if the bean cannot be autowired.
* no: This is the default behavior when the autowire attribute is not specified. The container does not attempt to wire the properties automatically; they must be explicitly wired. This is desirable in most cases if you want explicit documentation of the component wiring.

1. IoC(Inversion of Control): The control for the resource to use is inverted, from the component(class) to the container and its configurable
2. Dependency Injection: if code is dependent on the availability of the injected object instance to work.

Setter Injection: the use of the setter method, to inject the object.

Constructor Injection: a dependent resource is injected into a bean via its constructor.

1. AOP(aspect-oriented programming) allows you to systematically apply a set of code modules, called aspects, to another (typically larger) body of **target code** which is called separation of concerns

Pointcut in AOP describes where and how the aspect code should be inserted into the target.

1. Taking from connection pool or from DriverManager

DriverManager.registerDriver(new org.hsqldb.jdbcDriver());

/If not using default setup; change the URL to point to pix database.

return DriverManager.getConnection(“jdbc:hsqldb:hsql://localhost/pix”,“sa”, “”);

1. Spring database DAO Supports:

* JDBC JdbcDaoSupport
* Hibernate HibernateDaoSupport
* JDO JdoDaoSupport
* Oracle TopLink TopLinkDaoSupport
* Apache OJB PersistenceBrokerDaoSupport
* SQLMaps SqlMapClientDaoSupport
* JPA JPADaoSupport

1. JPA: Entity Manager

Basically, you use an entity manager to create, delete, and query entity instances. The entity manager is the interface maintaining and searching entity instances.

The group of entity instances managed by a manager is referred to as its persistence context.

There are two types of entity managers: container-managed and application-managed

1. Container-Managed Entity Managers:

Transactions with container-managed entity managers are Java Transaction API (JTA) transactions. A container-managed entity 0020 is obtained in an application through dependency injection with the @PersistenceContext annotation or through JNDI lookup

supports two kinds of persistence contexts that are defined according to the lifetime of the persistence context.

In transaction-scoped persistence contexts the lifetime is equal to a single transaction. In other words, the persistence context is alive for the length of the transaction.

The second type, the extended-scope persistence context, has an extended lifetime that spans multiple transactions.

1. Entity States:

Each entity may be in one of four possible states: new, managed, detached, or removed.

New Entity: instance is one that has not yet been managed or persisted.

Managed Entity: instance is one that has been persisted and thus is associated with a persistence context.

Detached Entity: instance is one that was previously managed but is not currently associated with a persistence context.

Removed Entity: instance is one that has a persistent identity, is associated with a persistent context, and is to be queued up for removal

1. There are two components to the JPA architecture: a container provider and a persistence provider. The container provider has control of the runtime threads and transactions. It manages the JPA entity managers and the transactions. The persistence provider implements the persistence API and implements the entity manager. The container and persistence providers are integrated through a service provider interface (SPI). In the case of Spring and JPA, Spring is the container that manages the entity managers and transactions from the provider (say a Hibernate JPA provider, or an ORACLE Toplink JPA provider).
2. There are two ways to define the mapping: field access(using the entity member variables) or property access(using JavaBean-style getter/setter methods on the properties). Entities can have either persistent fields or persistent properties but not both. In the case of field access all the non-transient member variables are persisted.

When you add annotations to an entity’s member variables, it is called persistent field annotation. In this case, the persistence implementation accesses entity member variables directly. In the case of property access, all public, protected, and non-transient properties are persisted.

When you add annotations to the entity’s getter/setter methods for JavaBeans-style properties, it is called persistent properties annotation. In this case, the entity coding must conform to that of JavaBeans.

The default type of access for persistent entities is property-level access.

1. The Spring framework has the @Repository annotation, which informs the Spring container that the class is a persistence repository and needs to have exception translation performed on it. To get the exception translation feature you must add the @Repository annotation to the classes affected, and also create a Spring bean instance.

@Repository

public class AlbumJpaRepository implements AlbumRepository {

private EntityManager entityManager;

@PersistenceContext

public void setEntityManager(EntityManager entityManager) {

this.entityManager = entityManager;

}

public void persistAlbum(Album album) {

entityManager.persist(album);

}

public Album retrieveAlbumById(Integer albumId) {

return entityManager.find(Album.class, albumId);

}

@SuppressWarnings(“unchecked”)

public List<Album> retrieveUserAlbums(PixUser user) {

Query q = entityManager.createNamedQuery(“userAlbums”);

q.setParameter(1, user.getUserName());

return q.getResultList();

}

//..remaining methods commented

}

This is the alternate way to do operation on DB for Spring Templates.

1. The @Transient annotation can be used to prevent fields from being persisted i.e. saved into database while the @GeneratedValue annotation for specifying the primary-key-generation technique to employ. The specified technique is based on actual primary-key-generation technique supported by the database.
2. The DispatcherServlet is the first place a request meets the application. Front Controller pattern implementation uses a HandlerMapping implementation to figure out which Controller class should process the request. The ModelAndView class defines a logical view name, which is resolved to an actual view implementation with the help of a ViewResolver.
3. ContextLoaderListener takes care of starting the Spring container’s WebApplicationContext which takes care of configuring the web application’s Spring environment. ContextLoaderListener can automatically detect your WebApplicationContext configuration file named <servletnameofDispatcher>-servlet.xml, in case you want to explicitly define the location of a context file or are using multiple context files, you can pass a list of context files within web.xml’s context-param element. The param-name should be set to contextConfigLocation and the param-value as the list of ApplicationContext files separated by a space or comma.
4. Spring provides multiple mechanisms to establish the link between an incoming URL request and a controller class like ControllerClassNameHandlerMapping inspects the URL path and maps it to a controller whose class name matches the URL path. e.g. a URL ending with albums.htm points to a controller class whose class name is AlbumsController. To enable this mapping strategy, you must add the ControllerClassNameHandlerMapping bean to the application’s configuration file.
5. While you can explicitly set the view name in the Controller, it also possible to fall back to the default view name convention. ModelAndView is actually capable of providing default view names and model keys in case these are not explicitly defined.
6. When no explicit view name is set by the controller (AlbumController), the view defaults to the same name as the URL path (album as view name) that was used by the ControllerClassNameHandlerMapping.
7. In ModelAndView name, forward: prefix is used here to instruct the controller to forward the request to another controller while redirect: prefix to point the user to another controller. With a fresh request. If you simply want to show another page, without invoking another controller, you shouldn’t use any prefix.
8. Spring MVC takes care of generating an appropriate key if you are providing any explicit key for the model data by inspecting the model’s object type and using the object’s class name as the model’s key and in case of a collection of objects, the class name of the first object found in the collection has List appended to it like stringList for list of String. $(key-name) is used to access value in view.
9. Spring’s mock package contains a collection of mock objects that can be used to simulate servlet infrastructure code like MockHttpServletRequest and MockHttpServletResponse objects in the org.springframework.mock.web package can be used for testing Spring MVC controllers. The other dependency like data from database while you are executing a unit test can be simulating with EasyMock.
10. InternalResourceViewResolver tells Spring how view names map to the web application’s JSPs. JSP pages stored under the WEB-INF folder can never be accessed directly, as the application server does not allow direct access to the WEB-INF folder.
11. SimpleFormController can be used if we have form data which needs to validate before sending it to backend. Spring form tag will map field name with Pojo’s properties following java standard. Method descriptions as follows:

* referenceData() method can be used to pass data to front end using ModelMap class.
* initBinder() is used to register any custom property editor with its format with binder like CustomDateEditor.
* setCommandClass() is used to set command class.
* setCommandName() is used to set command class object instance name.
* formBackingObject() is used to load customize object in the form for editing or other purpose
* doSubmitAction() is used to retrieve the requested data returned by the formBackingObject method.
* setSuccessView() is used to set success view.

1. Class that contains the form’s data is known in Spring MVC as the **command** class.
2. AbstractWizardFormController class can be used in case of multi wizard in a single form. The \_target prefix is interpreted by the wizard controller as a page change so in JSP page identifier like \_target0 can be used to go backward or forward while \_finish defines end of the form and \_cancel defines cancelling the form. Method description is as follows:

* setPages(String[]) defines wizard page order.
* processFinish():This method handles the final step in the workflow. It is invoked when the user clicks the Finish button.
* processCancel():This method cancels the workflow and is invoked when the user hits the Cancel button.
* postProcessPage():The postProcessPage method is invoked whenever the user clicks the Next or Back button in the workflow.
* processFormSubmission() method is used to save the command to the database
* formBackingObject() method is used to populate the command object from the database.

1. MultipartResolver is used by Spring MVC to automatically detect file uploads and provide support for processing those files like this

*<bean id=”multipartResolver” class=”org.springframework.web.multipart.commons.CommonsMultipartResolver”*

*p:maxUploadSize=”1000000” />*

You must also make sure to set the bean’s ID to multipartResolver: Spring MVC looks for beans by this name and file element name should be same as the command property name whose type should be MultipartFile.

1. **AbstractController** provides a single method **handleRequestInternal**(request, response), which is invoked for all requests passed to the controller while **MultiActionController** enables mapping multiple urls to invoke multiple methods (having HttpServletRequest and HttpServletResponse as a parameter and void, ModelAndView or Map as return type). These methods are called directly by appending the method name (in lower case) to the controller’s URL mapping (controller name in lower case).
2. The abstract **buildPdfDocument** method from the **AbstractPdfView** class has all the necessary ingredients to build a PDF document to its subclasses and **ResourceBundleViewResolver** (configured with p:basename=”<property file name>”) can be used to map view names to view classes (apart for JSPs) by defining the mapping in a properties file. The pattern to use for this mapping is <view name>.class=<view class> like pdf.class=<pdf generating class>.
3. Spring provides two implementations of the **MessageSource** interface: **ResourceBundleMessageSource** and **ReloadableResourceBundleMessageSource**. The latter can be used to automatically reload messages in a running application. Both implementations load messages from properties files. Following is an example of a properties file. It also manage the locale so if you want to add a French translation for p:basename=labels.properties, you add a file named labels\_fr.properties with French translation.
4. Adding cookies is simple as **CookieLocaleResolver** is responsible for determining the user’s locale by looking at a browser cookie, which is created by Spring MVC and stores the locale to be used while **LocaleChangeInterceptor** needs to be configured to change the default locale to whatever the user selected which is configured with a **HandlerMapping** implementation as interceptor so that requests can be inspected for the presence of a parameter that holds the new value of the locale.
5. Theme settings are stored in properties files. A **CookieThemeResolver** stores user preferences across sessions. To trigger a theme change, you add the **ThemeChangeInterceptor** to the **HandlerMapping’s** implementation list of interceptors.
6. The **FlowController** handles all flow execution for SWF (Swing Web Flow) and is configured as a normal JavaBean. It is an extension to the Spring AbstractController that serves as a point of integration between Spring MVC and SWF. Its job is to handle the execution of flows that are stored in the FlowRegistry. The flow registry serves as an index of flows.
7. Following are the scopes supported by SWF:

* Request: A scope that is local to a single request in a web flow; once that request completes, the scope is destroyed.
* Flash: A scope that is available until the next user event that survives refreshes is signaled.
* Flow: This scope exists for the life of a flow session.
* Conversation: A scope that is available until the root session for flow ends

1. Whenever a SWF start state is view (not action), it’s referring to a page that is coded using one of Spring’s supported view technologies (JSP, Velocity, Freemarker, XSLT, et cetera).
2. REST or Representational State Transfer is an architectural style for exposing applications; it’s not a specification or a standard.