***Spring and the Java Persistence API***

* JPA is a POJO-based persistence mechanism that draws ideas from both Hibernate and Java Data Objects (JDO) and mixes Java 5 annotations.
* The first step toward using JPA with Spring is to configure an entity manager factory as a bean in the Spring application context.

***Configuring an entity manager factory***

JPA-based applications use an implementation of EntityManagerFactory to get an instance of an EntityManager. The JPA specification defines two kinds of entity managers:

***Application-managed***—Entity managers are created when an application directly requests one from an entity manager factory. With application-managed entity managers, the application is responsible for opening or closing entity managers and involving the entity manager in transactions. This type of entity manager is most appropriate for use in standalone applications that don’t run in a Java EE container.

***Container-managed***—Entity managers are created and managed by a Java EE container. The application doesn’t interact with the entity manager factory at all. Instead, entity managers are obtained directly through injection or from JNDI. The container is responsible for configuring the entity manager factories. This type of entity manager is most appropriate for use by a Java EE container that wants to maintain some control over JPA configuration beyond what’s specified in persistence.xml.

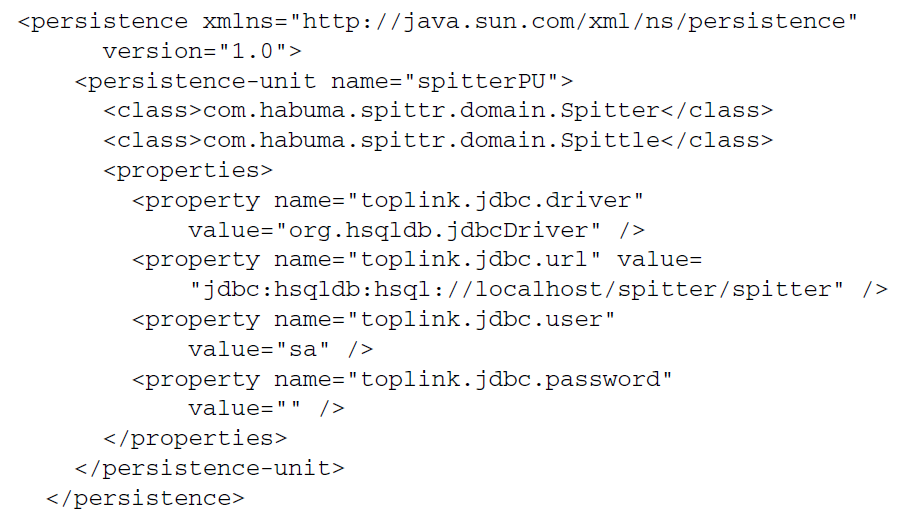
* Both kinds of entity manager implement the same EntityManager interface. The key difference isn’t in the EntityManager itself, but rather in how the EntityManager is created and managed.
* Application-managed EntityManagers are created by an EntityManagerFactory obtained by calling the **createEntityManagerFactory()** method of the PersistenceProvider.
* Container-managed EntityManagerFactorys are obtained through PersistenceProvider’s **createContainerEntityManagerFactory()** method.

Each flavor of entity manager factory is produced by a corresponding Spring factory bean:

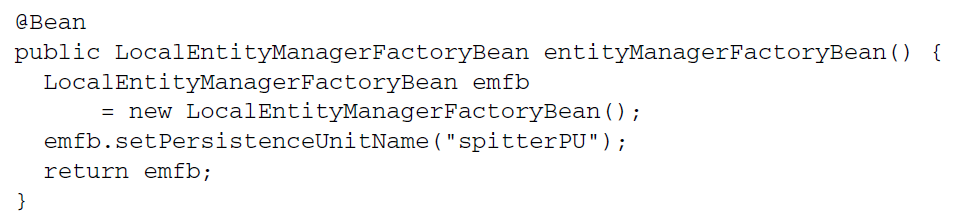
* LocalEntityManagerFactoryBean produces an application-managed EntityManagerFactory.
* LocalContainerEntityManagerFactoryBean produces a container-managed EntityManagerFactory.

**CONFIGURING APPLICATION-MANAGED JPA**

* Application-managed entity-manager factories derive most of their configuration information from a configuration file called persistence.xml. This file must appear in the META-INF directory in the classpath.
* persistence.xml enumerates one or more persistent classes along with any additional configuration such as data sources and XMLbased mapping files.

****

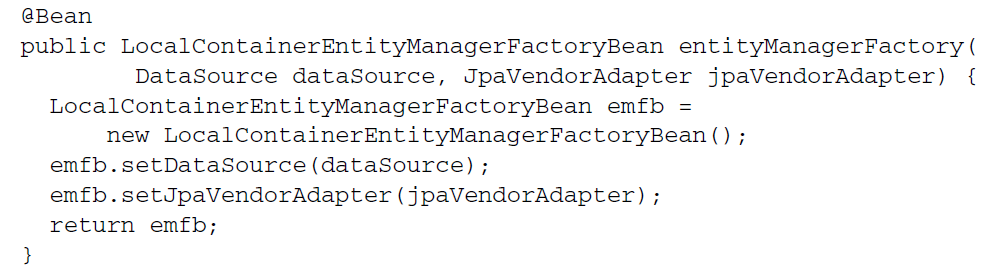
* The following <bean> declares a LocalEntityManagerFactoryBean in Spring:

****

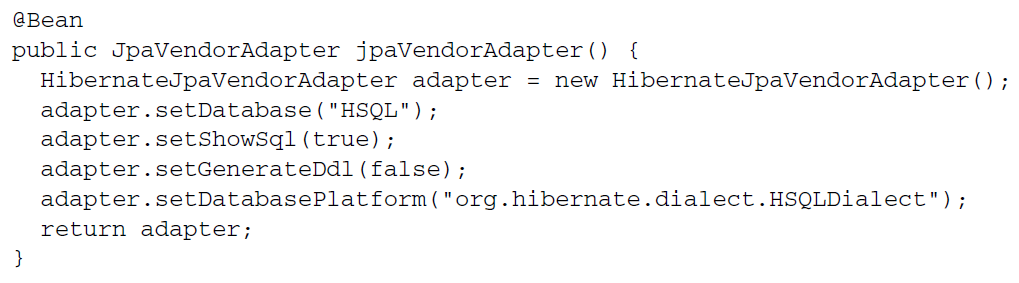
* The value given to the persistenceUnitName property refers to the persistence unit name as it appears in persistence.xml.

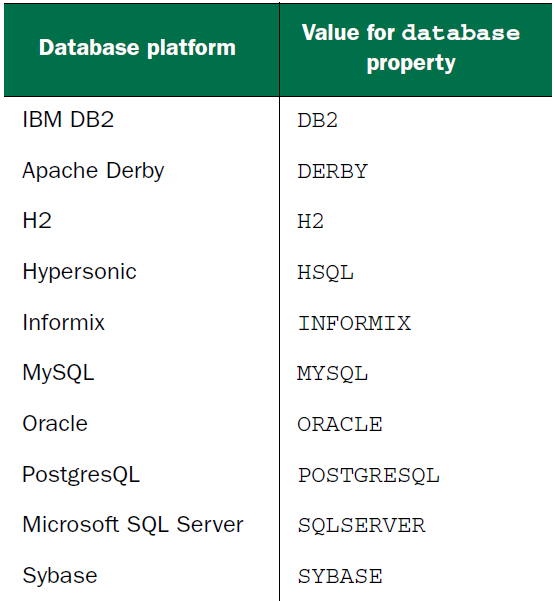
**CONFIGURING CONTAINER-MANAGED JPA**

* Container-managed JPA takes a different approach. When running in a container, an EntityManagerFactory can be produced using information provided by the container.
* Container-managed JPA takes a different approach. When running in a container, an EntityManagerFactory can be produced using information provided by the container— Spring, in this case.
* Instead of configuring data-source details in persistence.xml, you can configure this information in the Spring application context. For example, the following <bean> declaration shows how to configure container-managed JPA in Spring using LocalContainerEntityManagerFactoryBean:

****

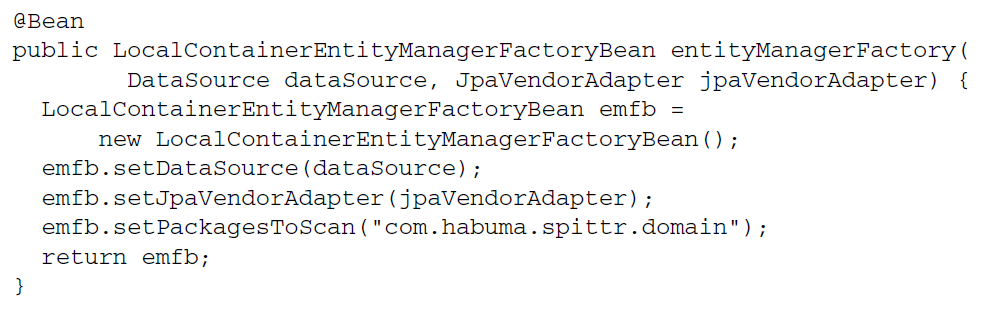
* Use the jpaVendorAdapter property as HibernateJpaVendorAdapter

****

****

* The primary purpose of the persistence.xml file is to identify the entity classes in a persistence unit. But as of Spring 3.1, you can do that directly with LocalContainerEntityManagerFactoryBean by setting the packagesToScan property:

The primary purpose of the persistence.xml file is to identify the entity classes in a persistence unit. But as of Spring 3.1, you can do that directly with LocalContainerEntityManagerFactoryBean by setting the packagesToScan property:

****

* configured here, LocalContainerEntityManagerFactoryBean will scan the com.habuma.spittr.domain package for classes that are annotated with @Entity. Therefore, there’s no need to declare them explicitly in persistence.xml. And because the DataSource is also injected into LocalContainerEntityManagerFactoryBean, there’s no need to configure details about the database in persistence.xml.