[Spring](http://springframework.org/), a popular framework for developing lightweight J2EE applications, uses Aspect Oriented Programming (AOP) and dependency injection at its core. It supports various frameworks, including object-relational mapping (ORM) tools such as the open-source [Oracle TopLink](http://www.oracle.com/technetwork/middleware/toplink/overview/index.html) and JBoss Hibernate. In this article, you will learn how to use the Spring Framework's support for TopLink. ( [Click here](http://www.oracle.com/technology/tech/java/spring.html) to visit the Oracle + Spring page on OTN.)

<http://www.oracle.com/technetwork/articles/dikmans-spring-toplink-089059.html>

To provide further explanations, JPA is an API specified in the frame of the JCP as an answer to a request (e.g [JSR 338](https://jcp.org/en/jsr/detail?id=338) for JPA 2.1).

Several implementations of that specification exist, the main are:

* [EclipseLink](http://eclipse.org/eclipselink/jpa.php)
* [Hibernate](http://hibernate.org/orm/)
* [OpenJPA](http://openjpa.apache.org/)
* [DataNucleus](http://datanucleus.org/)

In the frame of the Java platform, when a standard API is implemented, this is specified via a system called [SPI](https://docs.oracle.com/javase/tutorial/ext/basics/spi.html) (for Service Provider Interfaces). Each "vendor" of an implementation has to provide a specific component which is a single interface as a starting point for the implementing classes. The Java tutorial includes an [example for the sound API](https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html). An implementing class has to be mentioned in a file available to the ClassLoader after the name *META-INF/services/{{MyFullInterfaceName}}*.

For the JPA API, this starting point is the [PersistenceProvider](https://docs.oracle.com/javaee/6/api/javax/persistence/spi/PersistenceProvider.html) interface (note the spi section in the package name). Each implementation includes the declaration of the implementing class, for exemple in the *eclipselink.jar* you can find a file *META-INF/services/javax.persistence.spi.PersistenceProvider* (named after the full interface name) which contains only the full name of the provider implementation class, in the case of EclipseLink:

org.eclipse.persistence.jpa.PersistenceProvider

Most of time the application client of the API does not have to care about that declaration because it is included in the implementation JAR. The only case in which an application has to use that kind of file is when multiple implementations have to be used, for example with EclipseLink and Hibernate:

org.eclipse.persistence.jpa.PersistenceProvider

org.hibernate.ejb.HibernatePersistence

You find the implementing class also specified in the persistence.xml file (<provider/> tag).

Sometime, the JPA provider expression is used to refer to the "vendor" (EclipseLink, Hibernate, etc.) and not to the software component. Both can be considered as valid, depending on the context.

<https://stackoverflow.com/questions/27420513/what-is-a-jpa-provider>

Delivering on its commitment to the Java developer community, Oracle today announced the immediate availability of [Oracle TopLink](http://www.oracle.com/tools/toplink_adf.html) 11g, a component of [Oracle® Fusion Middleware](http://www.oracle.com/products/middleware/index.html).

Oracle TopLink 11g, an advanced object-relational persistence framework based on the open source EclipseLink project, introduces seamless integration with Oracle Coherence, flexible Object-XML binding, and provides out-of-the-box compatibility with Oracle WebLogic Server and Java frameworks including Oracle Application Development Framework (ADF) 11g.

Oracle will also now provide professional commercial support for EclipseLink.

EclipseLink is based on a previous version of Oracle TopLink, which Oracle contributed to the Eclipse Foundation in 2006.

<http://www.oracle.com/us/corporate/press/017498_en>