Persistence in Java

a) Serialization
b) Jdbc
c) EJB 2 Entity Beans

Serialization

Serialization allows transforming an object graph into a series of bytes, which can then be sent over the network or stored in a file.

Limitations of serialization

a) **Unsuitable for Large amount of data**:- it must store and retrieve the entire object graph at once. This makes it unsuitable for dealing with large amount of data.

b) **File system can not be fully reliable.**

c) **No querying capability**:- it provides no query capabilities.

JDBC

JDBC overcomes a lot of limitations that serialization has such as:

a) It can handle large amount of data.

b) Ensures data integrity.

c) Supports concurrent access to information.

d) Provides a sophisticated query language in SQL.

JDBC

Unfortunately, JDBC does not provide ease of use.

JDBC was not designed for storing objects. JDBC allows the Java programs to fully interact with the database. However, this interaction is heavily reliant on SQL. It requires a considerable amount of code spec that deals with taking tabular row and column data and converting it back and forth into objects.

EJB 2 Entity Beans

EJB 2 entities are components that represent persistent information in a data store.

Advantages:-

a) Provide an object oriented view of persistent data.

b) Use a strict standard,making them portable across vendors.

EJB 2 Entity Beans

Drawbacks:-

a) Are slow

b) Are difficult to code

c) Require expensive application servers to run, as they have to reside within a J2EE application server environment.

Object Relational Mapping
ORM

The object oriented model use **classes** whereas the relational databases use **tables**.

Getting the data and associations from objects into relational table structure and vice-versa requires a lot of tedious programming due to the difference between the two. This difference is called **The Impedance Mismatch**.

Developers need something simple to covert from one to the other automatically.

Bridging the gap between the object model and the relational model is known as **Object –Relational Mapping. (ORM)**

Object Relational Mapping
ORM

Advantages:-

a) Eliminates writing SQL to load and persist object state, leaving the developer free to concentrate on the business logic.

b) Enables creating an appropriate **DOMAIN** model, after which , the developer only needs to think in terms of **objects**, rather than **tables** , **rows** and **columns**.

c) Reduces dependence on database specific SQL and thus provides portability across databases.

d) Reduces more than 30% of the amount of Java Code spec that needs to be written by adopting an ORM

ORM tools

a) Hibernate
b) Enterprise Java Beans Entity Beans
c) TopLink
d) EclipseLink
e) OpenJPA

JPA

ORM tools allow the developers to focus on the object model instead of dealing with the mismatch between the object-oriented and relational model. Unfortunately , each of these tools has its own set of APIs. This ties the application code spec to the proprietary interfaces of a specific vendor (ORM tool). Since the code spec is tied to a specific vendor (referred to as Vendor Lock-In) , switching to another tool is not possible without rewriting all the persistence code spec.

JPA

JPA defines a persistence framework which provides a way of automatically mapping normal java objects to an SQL database. In other words, it helps load, search and save the data model objects.

JPA

JPA is a specification from Sun, which is released under Java EE 5 specification.

It is not

- An implementation
- a product

Hence it cannot be used as it is for persistence.

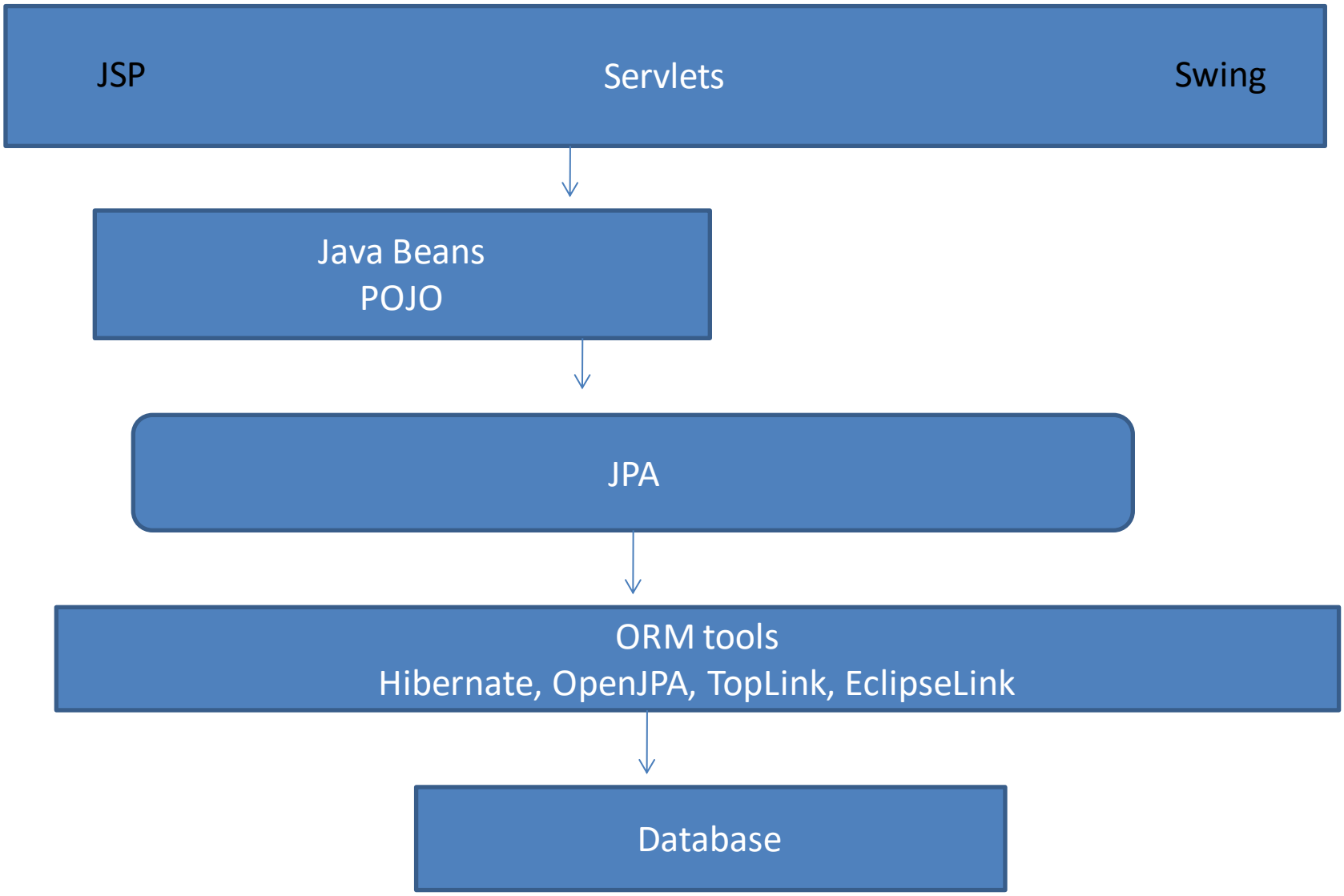
It needs an **ORM implementation** to work with and **persist Java Objects**.

JPA

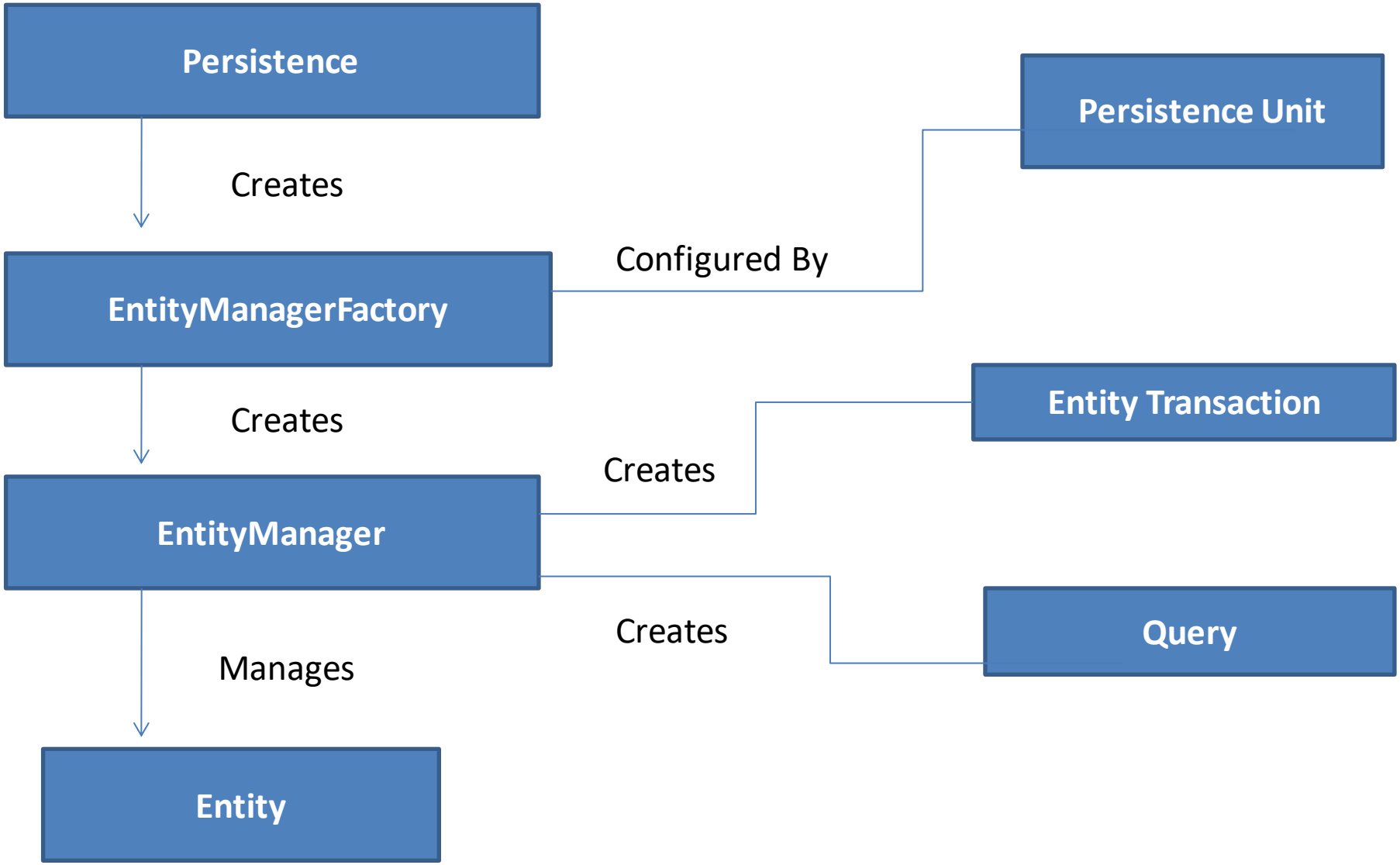
Technically , JPA is just a set of interfaces (a specification) and thus requires an **implementation**. Using JPA, therefore requires picking up an implementation (ORM tool such as Hibernate, TopLink, OpenJPA or any other ORM that implements JPA). Here the whole point of having a standard set of interface is that **users can in principle, switch between implementations of JPA without changing their code**.

Even though , the JPA is defined as part of the EJB 3.0 specification, it is not needed to have an EJB container or a java EE application server in order to run applications that use persistence.

Application, JPA,ORM and Database



Architecture of JPA



Persistence

The javax.persistence.Persistence class contains static helper methods to obtain EntityManagerFactory instances in a vendor-neutral fashion.

EntityManagerFactory

The **EntityManagerFactory** is created with the help of a **Persistence Unit** during the application start up. It serves as a factory for creating **EntityManager** objects when required. It is created once (one **EntityManagerFactory** object per database) and kept alive for later use.

EntityManager

The **EntityManager** object is lightweight and inexpensive to create. It provides the main interface to perform actual database operations. All the POJOs i.e. persistent objects are saved and retrieved with the help of an **EntityManager** object. Typically, **EntityManager** objects are created as needed and destroyed when not required. EntityManager manages a set of persistent objects and has APIs to insert new objects and delete existing ones.

Entity

Entities are **persistent objects (POJOs)** that represent datastore records.

EntityTransaction

A **transaction** represents a unit of work with the database. Any kind of modifications initiated via the EntityManager object are placed within a transaction. An EntityManager object helps creating an **EntityTransaction** object. Transaction objects are typically used for a short time and are close by either committing or rejecting.

Query

Persistent objects are retrieved using a Query object. Query objects allows using SQL or Java Persistence Query Language (JPQL) queries to retrieve the actual data from the database and create objects.

Multiple Query instances can be obtained using an EntityManager.

How JPA Works ?

An XML is created or annotations are added to POJOs, which inform the JPA provider (e.g. Hibernate) about :

- The classes needed to store the data.
- How the classes are related to the tables and columns in the database.

This way , all the necessary information is provided to the JPA provider.

How JPA Works ?

During the runtime, the JPA provider reads the XML and / or annotations and dynamically builds Java Classes to manage the transaction bet'n the database and java objects.

How JPA Works ?

An EntityManagerFactory is created from the compiled collection of mapping metadata. The EntityManagerFactory provides the mechanism for managing persistent classes and the EntityManager interface.

How JPA Works ?

EntityManager provides the interface bet’n persistent data store and the application. It is used to save and retrieve POJOs from the database.