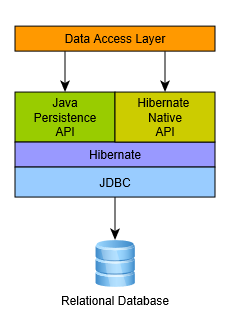
Hibernate



<https://docs.jboss.org/hibernate/stable/orm/userguide/html_single/Hibernate_User_Guide.html>

Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables using (XML) configuration files. Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

The main advantage of ORM like hibernate is that it shields developers from messy SQL. Apart from this, ORM provides following benefits:

* **Improved productivity**
  + High-level object-oriented API
  + Less Java code to write
  + No SQL to write
* **Improved performance**
  + Sophisticated caching
  + Lazy loading
  + Eager loading
* **Improved maintainability**
  + A lot less code to write
* **Improved portability**
  + ORM framework generates database-specific SQL for you

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

* Session interface
* SessionFactory interface
* Configuration interface
* Transaction interface
* Query and Criteria interfaces

Session Factory

The main contract here is the creation of [Session](https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/Session.html) instances. Usually an application has a single [SessionFactory](https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/SessionFactory.html) instance and threads servicing client requests obtain [Session](https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/Session.html) instances from this factory.

The internal state of a [SessionFactory](https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/SessionFactory.html) is immutable. Once it is created this internal state is set. This internal state includes all the metadata about Object/Relational Mapping.

Implementers **must** be thread safe.

As explained in this forum [post](https://forum.hibernate.org/viewtopic.php?p=2384979&sid=8367751b54bf160003b867f858393398#p2384979), 1 and 2 are related. If you set hibernate.current\_session\_context\_class to thread and then implement something like a servlet filter that opens the session - then you can access that session anywhere else by using the SessionFactory.getCurrentSession().

SessionFactory.openSession() always opens a new session that you have to close once you are done with the operations. SessionFactory.getCurrentSession() returns a session bound to a context - you don't need to close this.

If you are using Spring or EJBs to manage transactions you can configure them to open / close sessions along with the transactions.

You should never use one session per web app - session is not a thread safe object - cannot be shared by multiple threads. You should always use "one session per request" or "one session per transaction"

The Hibernate persistent or entity object can live in the following three states:  
1) transient  
2) persistent  
3) detached

A Session object uses the first-level cache. the second level cache is used at SessionFactory level.

HibernateTemplate:

org.springframework.orm.hibernate.HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked Hibernate Exceptions into unchecked DataAccessExceptions.

Hibernate categorizes types into two groups:

* [Value types](https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#categorization-value)
* [Entity types](https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#categorization-entity)

Value Types

A value type is a piece of data that does not define its own lifecycle. It is, in effect, owned by an entity, which defines its lifecycle.

Value types are further classified into three sub-categories:

1. Basic Types
2. Embeddable Types
3. Collection Types

Basic Types:

Basic value types usually map a single database column, to a single, non-aggregated Java type.

Internally Hibernate uses a registry of basic types when it needs to resolve a specific org.hibernate.type.Type.

These mappings are managed by a service inside Hibernate called the org.hibernate.type.BasicTypeRegistry, which essentially maintains a map of org.hibernate.type.BasicType (a org.hibernate.type.Type specialization) instances keyed by a name. That is the purpose of the "BasicTypeRegistry key(s)" column in the previous tables.

The @Basic annotation defines 2 attributes.

optional - boolean (defaults to true)

Defines whether this attribute allows nulls. JPA defines this as "a hint", which essentially means that it effect is specifically required. As long as the type is not primitive, Hibernate takes this to mean that the underlying column should be NULLABLE.

fetch - FetchType (defaults to EAGER)

Defines whether this attribute should be fetched eagerly or lazily. JPA says that EAGER is a requirement to the provider (Hibernate) that the value should be fetched when the owner is fetched, while LAZY is merely a hint that the value is fetched when the attribute is accessed. Hibernate ignores this setting for basic types unless you are using bytecode enhancement.

**Embeddable types**

are used to group multiple basic type mappings and reuse them across several entities.

An embeddable type is another form of a value type, and its lifecycle is bound to a parent entity type, therefore inheriting the attribute access from its parent

Hibernate called these components. JPA calls them embeddables