**Hibernate** is one of the most widely used **Java ORM** tool. Most of the applications use relational databases to store application information and at the low level we use [JDBC API](https://www.digitalocean.com/community/tutorials/jdbc-tutorial) for connecting to databases and perform CRUD operations.

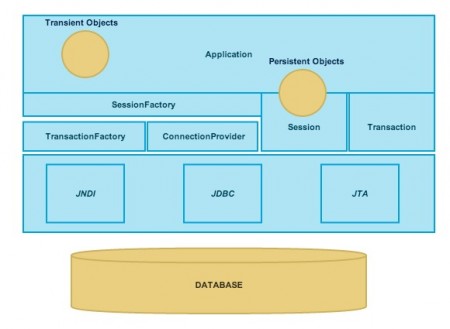
[**Hibernate Tutorial for Beginners**](https://www.digitalocean.com/community/tutorials/hibernate-tutorial-for-beginners#hibernate-tutorial-for-beginners)

If you look at the JDBC code, there is so much of boiler plate code and there are chances of resource leak and data inconsistency because all the work needs to be done by the developer. This is where an ORM tool comes handy. **Object-relational mapping** or **ORM** is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa. Some of the benefits of using Hibernate as ORM tool are:

1. Hibernate supports mapping of java classes to database tables and vice versa. It provides features to perform CRUD operations across all the major relational databases.
2. Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business use cases rather than making sure that database operations are not causing resource leaks.
3. Hibernate supports transaction management and make sure there is no inconsistent data present in the system.
4. Since we use XML, property files or annotations for mapping java classes to database tables, it provides an abstraction layer between application and database.
5. Hibernate helps us in mapping joins, collections, inheritance objects and we can easily visualize how our model classes are representing database tables.
6. Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.
7. Hibernate also offers integration with some external modules. For example Hibernate Validator is the reference implementation of Bean Validation (JSR 303).
8. Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small and there are tons of online documentations and help is easily available in forums.
9. Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that [Spring Framework](https://www.digitalocean.com/community/tutorials/spring-jdbc-example) provides built-in support for integrating hibernate with Spring applications.

I hope all the above benefits will convince you that Hibernate is the best choice for your application object-relational mapping requirements. Let’s look at the Hibernate Framework architecture now and then we will jump into sample project where we will look into different ways to configure Hibernate in standalone java application and use it.

[**Hibernate Architecture**](https://www.digitalocean.com/community/tutorials/hibernate-tutorial-for-beginners#hibernate-architecture)

Below image shows the Hibernate architecture and how it works as an abstraction layer between application classes and JDBC/JTA APIs for database operations. It’s clear that Hibernate is built on top of JDBC and JTA APIs.[](https://journaldev.nyc3.cdn.digitaloceanspaces.com/2014/05/Hibernate-Architecture-Diagram.jpg)Let’s look at the core components of hibernate architecture one by one.

* **SessionFactory (org.hibernate.SessionFactory)**: SessionFactory is an immutable thread-safe cache of compiled mappings for a single database. We can get instance of org.hibernate.Session using SessionFactory.
* **Session (org.hibernate.Session)**: Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction.
* **Persistent objects**: Persistent objects are short-lived, single threaded objects that contains persistent state and business function. These can be ordinary JavaBeans/POJOs. They are associated with exactly one org.hibernate.Session.
* **Transient objects**: Transient objects are persistent classes instances that are not currently associated with a org.hibernate.Session. They may have been instantiated by the application and not yet persisted, or they may have been instantiated by a closed org.hibernate.Session.
* **Transaction (org.hibernate.Transaction)**: Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases.
* **ConnectionProvider (org.hibernate.connection.ConnectionProvider)**: ConnectionProvider is a factory for JDBC connections. It provides abstraction between the application and underlying javax.sql.DataSource or java.sql.DriverManager. It is not exposed to application, but it can be extended by the developer.
* **TransactionFactory (org.hibernate.TransactionFactory)**: A factory for org.hibernate.Transaction instances.

[**Hibernate and Java Persistence API (JPA)**](https://www.digitalocean.com/community/tutorials/hibernate-tutorial-for-beginners#hibernate-and-java-persistence-api-jpa)

Hibernate provides implementation of **Java Persistence API**, so we can use JPA annotations with model beans and hibernate will take care of configuring it to be used in CRUD operations. We will look into this with annotations example.