***Hibernate Framework Notes***

***(Database Framework)***

Table of Contents

[Hibernate Description 1](#_Toc792666213)

[Hibernate Architecture 2](#_Toc956793883)

[SessionFactory 2](#_Toc2080281760)

[Session 2](#_Toc23537617)

[Transaction 3](#_Toc2014770691)

[Key Features of Hibernate 3](#_Toc646241945)

[Data Manipulation 3](#_Toc1007867782)

[Session Management 3](#_Toc1097030156)

[Query Options 3](#_Toc2046335549)

[Caching 3](#_Toc687351887)

[Transaction Management 4](#_Toc685218525)

[Automatic Schema Generation 4](#_Toc687240193)

[Advantages of Using Hibernate 4](#_Toc1822274335)

[Portability 4](#_Toc153945315)

[Productivity 4](#_Toc125167283)

[Maintainability 4](#_Toc356828837)

[Performance 4](#_Toc2009292734)

[Use Cases of Hibernate 4](#_Toc2031742788)

[Enterprise Applications 5](#_Toc936212856)

[Web Applications 5](#_Toc351550762)

[How to Use Hibernate 5](#_Toc1242035770)

[Adding Dependencies 5](#_Toc1478063814)

[Configuring Hibernate 5](#_Toc1085085029)

[Creating Entity Classes 6](#_Toc99307637)

[Using Hibernate for Database Operations 7](#_Toc1326193562)

[Why using Hibernate, not JDBC 8](#_Toc2030478916)

# Hibernate Description

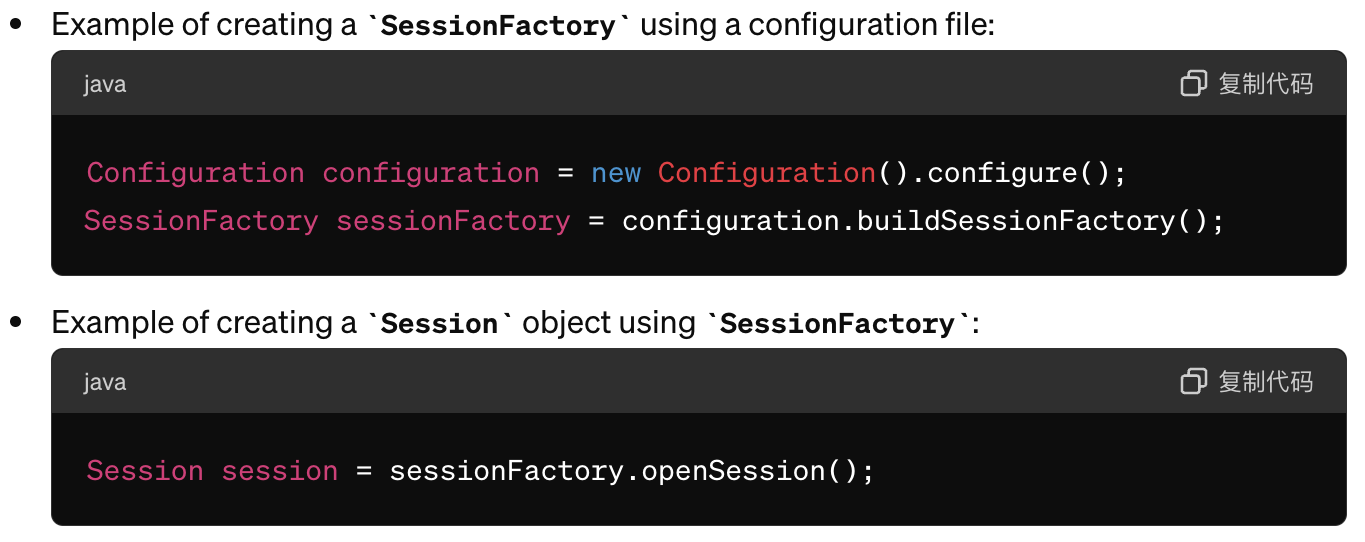
* A popular open-source Object-Relational Mapping (ORM) tool for Java applications, offering a framework for mapping an object-oriented domain model to a traditional relational database
* Provides a bridge between the object-oriented world of Java and the relational world of databases, enhancing the capability to develop more efficient and effective data-access layers in Java applications

Hibernate Architecture

* The Hibernate architecture consists of three core components:
  + SessionFactory
  + Session
  + Transaction

## SessionFactory

* SessionFactory is the top-level component of the Hibernate architecture and acts as a singleton factory for Hibernate applications.
  + Create Session objects, which are one of the most important components in Hibernate.
  + It is typically instantiated only once, and then reused whenever needed.
  + Reduces the overhead of the application and improves its performance.
* Creating a SessionFactory usually requires some initialization configuration, such as database connection information and mapping metadata.
  + It can be configured and created using Hibernate's configuration file (e.g., hibernate.cfg.xml) or programmatically.
  + Example of code:



## Session

* Session is the second core component of the Hibernate architecture. It represents a single session between a Java application and a relational database.
  + Thread-safe and is usually associated with a database transaction.
  + Is typically used to perform CRUD (Create, Read, Update, Delete) operations and to query the database for retrieving required data.
  + Provides methods for managing object states, allowing developers to fetch persistent objects from the database into memory and return them to the database after processing.
  + Supports associating objects with other objects, thereby building complex data models.
* Hibernate Session offers many advanced features:
  + Using caching mechanisms to improve performance and supporting lazy loading to reduce the number of database queries.
  + Supports integration with JTA (Java Transaction API) for transaction management in distributed environments.

## Transaction

* Transaction is the third core component of the Hibernate architecture. It allows for secure and efficient communication between Java applications and databases.
  + Ensures that all operations are atomic, meaning they either all succeed or all fail.
  + Are managed by the Session object, which simplifies transaction management.

# Key Features of Hibernate

## Data Manipulation

* Hibernate provides a way to translate data between the application and the database by abstracting away the complexities of these interactions
  + Turns java classes into database tables and Java types into SQL data types
    - Provides data query and retrieval facilities
  + It is most useful for object-oriented domain models and business logic within a Java-based middle tier. Nevertheless, Hibernate can certainly help you eliminate or encapsulate vendor-specific SQL code and simplify the common tasks of converting result sets from tabular representation to object graphs.

## Session Management

* Hibernate manages database sessions and provides data query and update capabilities
  + It creates SQL calls and relieves the developer from manual handling and object conversion of the result set

## Query Options

* Supports SQL and its own query language called Hibernate Query Language (HQL), which is object-oriented and understands concepts like inheritance, polymorphism, and association

## Caching

* Hibernate includes a powerful internal caching mechanism, which can significantly boost the performance of applications by reducing the number of queries made to the database

## Transaction Management

* Abstracts the complexity of transaction management from the developer, providing a more intuitive way to handle database transactions

## Automatic Schema Generation

* Hibernate can generate database schemas based on the Java classes, making it easier to maintain the database structure

# Advantages of Using Hibernate

## Portability

* Hibernate-generated database independent queries
  + It is not tied to a specific database
    - Your code can work with any database that hibernate supports with minimal changes

## Productivity

* Since it handles most of the database operations, developers can focus more on business logic

## Maintainability

* Changes in the database schema or in the domain model can be managed more easily

## Performance

* Advanced caching mechanisms help improve application performance and reduce the load on the database

# Use Cases of Hibernate

## Enterprise Applications

* Hibernate is widely used in enterprise environments where robustness and scalability are critical, especially when dealing with complex transactions and vast amounts of data

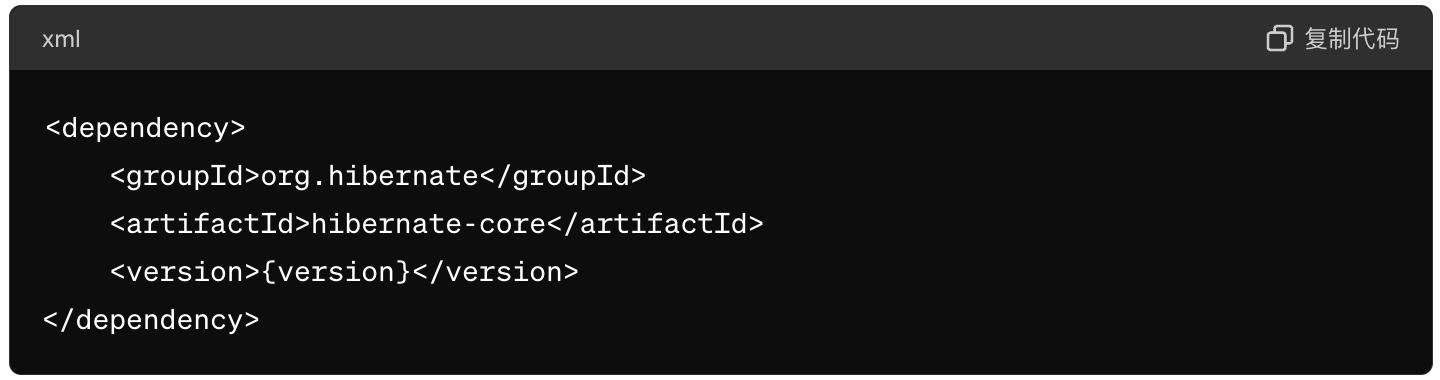
## Web Applications

* Commonly used in Java-based web applications where database interaction is a key component

# How to Use Hibernate

## Adding Dependencies

* To use Hibernate, you need to add Hibernate dependencies to your project. You can add the following dependency in build tools such as Maven or Gradle:



## Configuring Hibernate

* Create a *hibernate.cfg.xml* file in your project to configure Hibernate. In this file, you need to specify database connection information and other Hibernate configurations.
  + The configuration file can set various Hibernate properties, including caching strategies and database dialects. Additionally, you can specify the mapping relationships for entity classes, enabling Hibernate to correctly map entity class objects to data in the database tables.
  + In the configuration file, you can set different configurations for different environments, such as enabling SQL output in the development environment but disabling it in the production environment. You can also configure data sources, transaction managers, and more.
* Below is an example of a simple *hibernate.cfg.xml* file:

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property> <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property> <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydatabase</property>

<property name="hibernate.connection.username">root</property> < property name="hibernate.connection.password">password</property> </session-factory>

</hibernate-configuration>

## Creating Entity Classes

* In Hibernate, an entity class is a Java class mapped to a database table. An entity class should contain properties that correspond to the columns in the database table:
  + A no-argument constructor so that Hibernate can use it to create instances of the entity class.
  + Add annotations to describe the properties of the entity class, such as the @*Column* annotation, which can be used to specify the column name and data type in the database table.
  + Add getter and setter methods to access the properties of the entity class. These methods should follow the JavaBean conventions.
  + Add methods in the entity class to perform business logic, such as calculating or validating properties.
* Below is an example of a simple entity class:



## Using Hibernate for Database Operations

* Hibernate is a popular ORM (Object-Relational Mapping) framework that helps simplify interactions with the database in Java programs.
* Below is an example of a simple Hibernate query:



# Why using Hibernate, not JDBC

* Facilitates maintenance and allows Java programmers to fully implement object-oriented programming principles when manipulating data. This makes the program easier to maintain, reduces coupling, and allows programmers to focus more on business logic rather than repetitive code.
* Before Hibernate, if a project needed to switch its database, it was necessary to rewrite SQL statements because different databases have variations in certain SQL syntax (for example, pagination in SQL Server and MySQL).
* Before Hibernate, writing SQL statements using JDBC to operate on databases was actually quite cumbersome. Moreover, it was very easy to make mistakes.