Hibernate

JDBC

- JDBC stands for **Java Database Connectivity** and provides a set of API for accessing relational DB. These APIs enables programs to execute SQL statements and interact with any SQL compliant database.

- JDBC provides a flexible architecture to write a database independent application that can run on different platforms and interact with different DBMS without any modification

Pros & Cons of JDBC

Pros

- Clean and simple SQL processing
- Good performance with large data
- Very good for small applications
- Simple syntax so easy to learn

Cons

- Complex if it is used in large projects
- Large programming overhead
- No encapsulation
- Hard to implement MVC concept
- Query is DBMS specific

Mismatch Problems

Mismatch	Description
Granularity	Sometimes you will have an object model which has more classes than the number of corresponding tables in the database.
Inheritance	RDBMSs do not define anything similar to Inheritance which is a natural paradigm in object-oriented programming languages.
Identity	A RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).
Associations	Object-oriented languages represent associations using object references where as am RDBMS represents an association as a foreign key column.
Navigation	The ways you access objects in Java and in a RDBMS are fundamentally different.

Need for Mapping

- When we work with an object-oriented systems, there's a mismatch between the object model and the relational database

- RDBMSs represent data in a tabular format whereas object-oriented languages represent it as an interconnected graph of objects.

ORM

- ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages

ORM Advantages

- Lets business code access objects rather than DB tables
- Hides details of SQL queries from OO logic
- Based on JDBC 'under the hood'
- No need to deal with the database implementation
- Entities based on business concepts rather than database structure
- Transaction management and automatic key generation
- Fast development of application

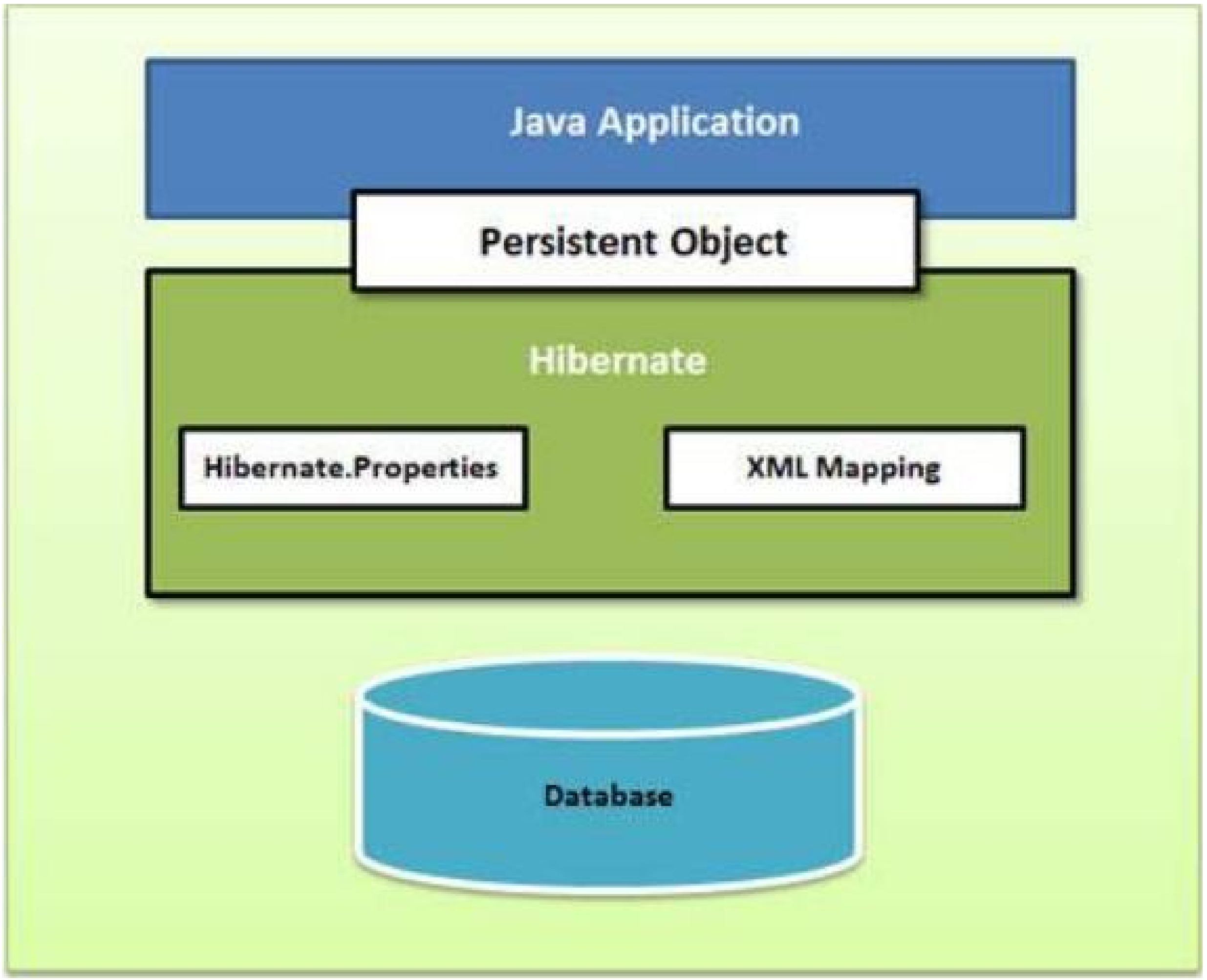
What is Hibernate

- Hibernate is an Object-Relational Mapping(ORM) solution for JAVA
- Maps Java classes to database tables and from Java data types to SQL data types
- Sits between traditional Java objects and database server to persist those

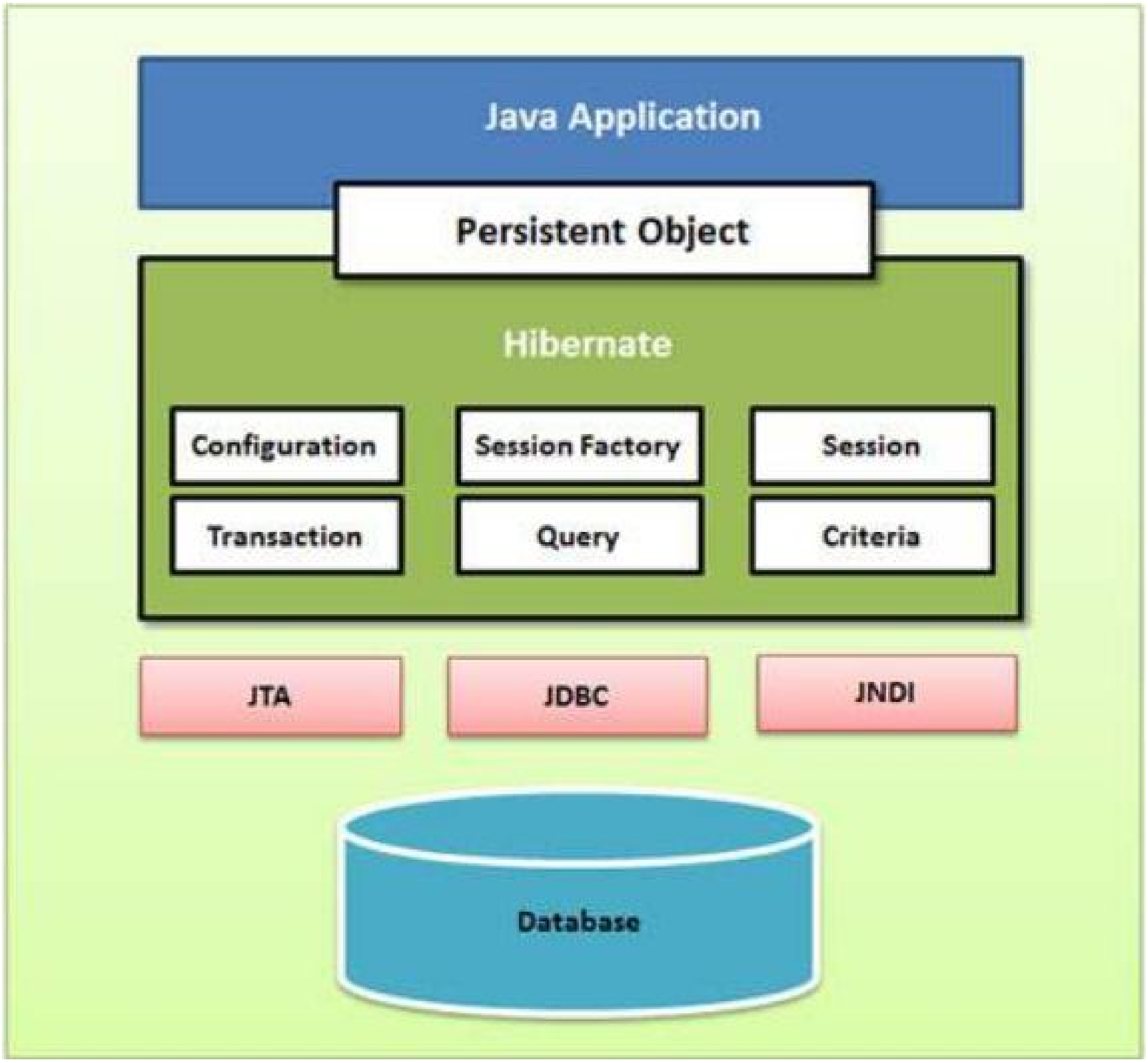
Why Hibernate

- Takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database
- If there is change in Database or in any table then the only need to change XML file properties
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects
- Does not require an application server to operate
- Manipulates Complex associations of objects of your database
- Minimize database access with smart fetching strategies
- Provides Simple querying of data

Architecture Overview



Detailed Architecture



Configuration Object

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys components:

- **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
- **Class Mapping Setup:** This component creates the connection between the Java classes and database tables.

SessionFactory Object

- Configuration object is used to create a **SessionFactory** object which in-turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The **SessionFactory** is a thread safe object and used by all the threads of an application

- The **SessionFactory** is heavyweight object so usually it is created during application start up and kept for later use. You would need one **SessionFactory** object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple **SessionFactory** objects

Session Object

- A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

- The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed as needed.

Transaction Object

- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

- This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code (but it is a good practice to use the Transaction provided by Hibernate)

Query Object

- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects

- Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query

Criteria Object

- Criteria objects are used to create and execute object oriented criteria queries to retrieve objects

Object States

- **Transient:** A new instance of a persistent class which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.

- **Persistent:** You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.

- **Detached:** Once we close the Hibernate Session, the persistent instance will become a detached instance.

Loading objects from DB

- Get
- Load
- Criteria

Get

- Used for loading objects from DB using Identifier
- It always **hits the database** and returns the real object, an object that represents the database row

Load

- Used for loading objects from DB using Identifier
- It will always return a “**proxy**” without hitting the database. In Hibernate, proxy is an object with the given identifier value, its properties are not initialized yet
- When any of the properties of the proxy object is accessed, it will hit the database and if the entry is not found, it throws a ObjectNotFoundException

Criteria

Used for loading objects from DB using other fields

- Restrictions
- Pagination
- Order
- Projections(Avg, Min, Max, Sum)

Synching up the DB

- Hibernate automatically detects that object has been modified and needs to be updated. This is called *automatic dirty checking*.
- As long as they are in *persistent* state, that is, bound to a particular Hibernate org.hibernate.Session, Hibernate monitors any changes and executes SQL in a write-behind fashion.
- The process of synchronizing the memory state with the database, usually only at the end of a unit of work, is called *flushing*.
- The unit of work usually ends with a commit, or rollback, of the database transaction.

Fetching Strategies

Hibernate uses a *fetching strategy* to retrieve associated objects if the application needs to navigate the association. Fetch strategies can be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query.

- ***Join fetching*:** Hibernate retrieves the associated instance or collection in the same SELECT, using an OUTER JOIN.
- ***Select fetching*:** a second SELECT is used to retrieve the associated entity or collection. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you access the association.
- ***Subselect fetching*:** a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you access the association.
- ***Batch fetching*:** an optimization strategy for select fetching. Hibernate retrieves a batch of entity instances or collections in a single SELECT by specifying a list of primary or foreign keys

Pagination

- **setFirstResult(int firstResult)** This method takes an integer that represents the first row in your result set, starting with row 0.
-
- **setMaxResults(int maxResults)** This method tells Hibernate to retrieve a fixed number **maxResults** of objects.

HQL

- FROM
- AS
- SELECT
- WHERE
- ORDER BY
- GROUP BY

Native SQL Query

- Used when you want to use database specific features in the query
- Will send the query directly to the DB
- Eg: Uses
  - Connect by in Oracle
  - Hints in Oracle

Annotations

- @Entity
- @Id
- @Column
- @OneToOne
- @OneToMany
- @ManyToMany
- @ManyToOne

Caching

- Session/First Level Cache: Caches object within the current session
- Query Cache: Responsible for caching queries and their results.
- Second Level Cache: Responsible for caching objects across sessions.

Session/First Level Cache

- Enabled by default
- All entries are evicted once session is closed
- Can use session.evict(Object) to evict a single entity
- Can use session.clear() to evict all entities
- No way to disable L1 cache
- Either use short lived sessions or L2 cache to maintain sync with DB

Query Cache

- Disabled by Default
- Enable it by setting the property hibernate.cache.use_query_cache=true
- Also need to set Query.setCacheable(true)
- Most applications will not benefit from this
- Has overhead to maintain the sync with DB
- Only the query gets cached. If the corresponding entities are not cached in L2, then a query will be triggered to load the entities

Second Level Cache

- External cache that needs to be supplied to Hibernate
- Some of the supported caches are
  - EHCache
  - warmCache
  - JBoss Cache
  - OSCache

Caching Concurrency Strategies

- **Read-Only:** Suitable for data which never changes. Use it for reference data only
- **Nonstrict-read-write:** Makes no guarantee of consistency between the cache and the database. Use if data hardly ever changes and a small likelihood of stale data is not of critical concern. Never locks the entity
- **Read-write:** Read-mostly data where it is critical to prevent stale data in concurrent transactions, in the rare case of an update. Soft locks the entity
- **Transactional:** Read-mostly data where it is critical to prevent stale data in concurrent transactions, in the rare case of an update. It is synchronous

Thank You!