Hibernate Basics

Mudassar Hakim
mudassar.trainer@gmail.com

Why Use ORM?

mudassar.trainer@gmail.com

Why Object/Relational Mapping (ORM)?

- A major part of any enterprise application development project is the persistence layer
  - Accessing and manipulating persistent data typically with relational database
- ORM handles Object-Relational impedance mismatch
  - Relational database is table driven (with rows and columns)
    - Designed for fast query operation of table-driven data
  - We, Java developers, want to work with classes/objects, not rows and columns
  - ORM handles the mapping between the two

mudassar.trainer@gmail.com

What is & Why Hibernate?

mudassar.trainer@gmail.com

What is Hibernate?

- The most popular ORM framework for enabling transparent POJO persistence
  - Let you work without being constrained by table-driven relational database model – handles Object-Relational impedance mismatch
- Lets you build persistent objects following common OO programing concepts
  - Because they are POJOs

mudassar.trainer@gmail.com

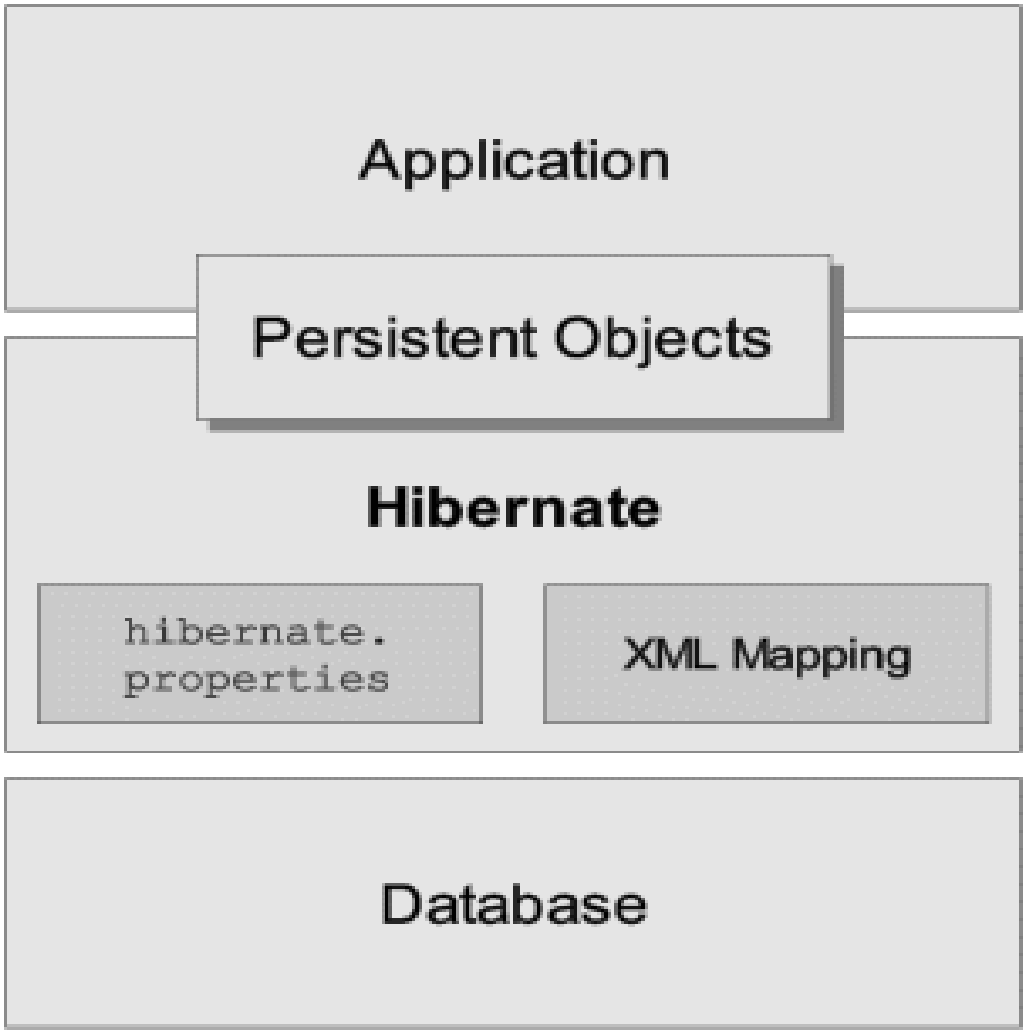
Why use Hibernate?

- Allows developers focus on domain object modelling not the persistence plumbing
- Performance
  - High performance object caching
  - Configurable materialization strategies
- Sophisticated query facilities
  - Criteria API
  - Query By Example (QBE)
  - Hibernate Query Language (HQL)
  - Native SQL

Hibernate Architecture

mudassar.trainer@gmail.com

Hibernate Architecture



Hibernate Framework Classes

mudassar.trainer@gmail.com

Hibernate Framework Classes

- *org.hibernate.SessionFactory*
- *org.hibernate.Session*
- *org.hibernate.Transaction*

(We will cover Hibernate annotation in another session)

mudassar.trainer@gmail.com

org.hibernate.Session

- A single-threaded, short-lived object representing a conversation between the application and the persistent store
- A session represents a persistence context
- The life of a Session is bounded by the beginning and end of a logical transaction.
- Handles life-cycle operations- create, read and delete operations - of persistent objects
- Factory for *Transaction*

mudassar.trainer@gmail.com

org.hibernate.Transaction

- A single-threaded, short-lived object used by the application to specify atomic units of work
- Abstracts application from underlying JDBC, JTA or CORBA transaction.
- However, transaction demarcation, either using the underlying API or Transaction, is never optional!

(We will cover Transaction in detail in Hibernate Transaction)

mudassar.trainer@gmail.com

Domain Classes

mudassar.trainer@gmail.com

Domain Classes

- Domain classes are classes in an application that implement the entities of the business domain (e.g. Customer and Order in an E-commerce application)
- Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model.

mudassar.trainer@gmail.com

Steps to write a Domain Class

- Step 1: Implement a no-argument constructor
  - All persistent classes must have a default constructor so that Hibernate can instantiate them
- Step 2: Provide an identifier property
  - This property maps to the primary key column of a database table.
  - The property can be called anything, and its type can be any primitive type, any primitive "wrapper" type, *java.lang.String* or *java.util.Date*
  - Composite key is possible
- Step 3: Declare getter/setter methods for persistent fields

mudassar.trainer@gmail.com

Instance States

mudassar.trainer@gmail.com

Instance States

- An instance of a domain class may be in one of three different states, which are defined with respect to a persistence context
  - transient (does not belong to a persistence context)
  - persistent (belongs to a persistence context)
  - detached (used to belong to a persistence context)
- The persistence context is represented by Hibernate Session object
  - In JPA, the persistence context is represented by EntityManager, which plays same role of Session in Hiberrnate

mudassar.trainer@gmail.com

“transient” state

- The instance is not, and has never been associated with any session (persistence context)
- It has no persistent identity (primary key value)
- It has no corresponding row in the database
- ex) When POJO instance is created outside of a session meaning before it is persisted
- Changes made to transient objects do not get reflected to the database table - They need to be persisted before the change get reflected to the database table (when committed)

mudassar.trainer@gmail.com

“persistent” state

- The instance is currently associated with a single session (persistence context).
- It has a persistent identity (primary key value) and likely to have a corresponding row in the database (if it has been committed before or read from the table)
- Changes made to persistent objects (objects in “persistent” state) are reflected to the database tables when they are committed
- ex) When an object is created within a session or a transient object gets persisted

mudassar.trainer@gmail.com

“detached” state

- The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process
- It has a persistent identity and, perhaps, a corresponding row in the database
- Used when POJO object instance needs to be sent over to another program for manipulation without having persistent context
- Changes made to detached objects do not get reflected to the database table - They need to be merged before the change get reflected to the database table

mudassar.trainer@gmail.com

State Transitions

- Transient instances may be made persistent by calling *save()*, *persist()* or *saveOrUpdate()*
- Persistent instances may be made transient by calling *delete()*
- Any instance returned by a *get()* or *load()* method is persistent
- Detached instances may be made persistent by calling *update()*, *saveOrUpdate()*, *lock()* or *replicate()*
- The state of a transient or detached instance may also be made persistent as a new persistent instance by calling *merge()*.

mudassar.trainer@gmail.com

Methods of Session Interface

mudassar.trainer@gmail.com

Types of Methods in Session Class

- Life cycle operations
- Transaction and Locking
- Managing resources
- JDBC Connection

mudassar.trainer@gmail.com

Lifecycle Operations

mudassar.trainer@gmail.com

Lifecycle Operations

- *Session* interface provides methods for lifecycle operations
- Result of lifecycle operations affect the instance state
  - Saving objects
  - Loading objects
  - Getting objects
  - Refreshing objects
  - Updating objects
  - Deleting objects
  - Replicating objects

mudassar.trainer@gmail.com

Saving Objects

- An object remains to be in “transient” state until it is saved and moved into “persistent” state
- The class of the object that is being saved must have a mapping file (*myclass.hbm.xml*)

mudassar.trainer@gmail.com

Java methods for saving objects

- From Session interface

```
// Persist the given transient instance,
// first assigning a generated identifier.
// Returns generated identifier.
public Serializable save(Object object)
```


mudassar.trainer@gmail.com

Example: Saving Objects

- Note that the *Person* is a POJO class with a mapping file (*person.hbm.xml*)

```
Person person = new Person();  // transient state
person.setName("Mudassar Hakim");
session.save(person);                // persistent state

// You can get an identifier
Object identifier = session.getIdentifier(person);
```


mudassar.trainer@gmail.com

Loading Objects

- Used for loading objects from the database
- Each *load(..)* method requires object's primary key as an identifier
  - The identifier must be *Serializable* – any primitive identifier must be converted to object
- Each *load(..)* method also requires which domain class or entity name to use to find the object with the id
- The returned object, which is returned as *Object* type, needs to be type-casted to a domain class

mudassar.trainer@gmail.com

Java methods for loading objects

- From Session interface

```
// Return the persistent instance of the given entity
// class with the given identifier, assuming that the
// instance exists.
public Object load(Class theClass, Serializable id)
```


mudassar.trainer@gmail.com

Getting Objects

- Works like load() method

mudassar.trainer@gmail.com

load() vs. get()

- Only use the *load()* method if you are sure that the object exists
  - *load()* method will throw an exception if the unique id is not found in the database
- If you are not sure that the object exists, then use one of the *get()* methods
  - *get()* method will return null if the unique id is not found in the database

mudassar.trainer@gmail.com

Java methods for getting objects

- From Session interface

```
// Return the persistent instance of the given entity
// class with the given identifier, or null if there is no
// such persistent instance.
public Object get(Class theClass, Serializable id)
```


mudassar.trainer@gmail.com

Example: Getting Objects

```
Person person = (Person) session.get(Person.class, id);
if (person == null){
    System.out.println("Person is not found for id " + id);
}
```


mudassar.trainer@gmail.com

Refreshing Objects

- Used to refresh objects from their database representations in cases where there is a possibility of persistent object is not in sync. with the database representation
- Scenarios you might want to do this
  - Your Hibernate application is not the only application working with this data
  - Your application executes some SQL directly against the database
  - Your database uses triggers to populate properties on the object

mudassar.trainer@gmail.com

Java methods for Refreshing objects

- From Session interface

  // Re-read the state of the given instance from the

  // underlying database.

  public void refresh(Object object)

mudassar.trainer@gmail.com

Updating Objects

- Hibernate automatically manages any changes made to the persistent objects
  - The objects should be in “persistent” state not transient state
- If a property changes on a persistent object, Hibernate session will perform the change in the database when a transaction is committed (possibly by queuing the changes first)
- From developer perspective, you do not have to any work to store these changes to the database
- You can force Hibernate to commit all changes using *flush()* method
- You can also determine if the session is dirty through *isDirty()* method

mudassar.trainer@gmail.com

Deleting Objects

- From Session interface

```
// Remove a persistent instance from the datastore.
// The argument may be an instance associated with
// the calling Session or a transient instance with
// an identifier associated with existing persistent
// state. This operation cascades to associated
// instances if the association is mapped with
// cascade="delete".
public void delete(Object object)
```


mudassar.trainer@gmail.com

Thank you!

Mudassar Hakim
mudassar.trainer@gmail.com