JAVA FULL STACK DEVELOPMENT PROGRAM

Session 12: Hibernate

OUTLINE

- Introduction
  - JDBC & Terminologies
- Hibernate
  - Introduction & features
  - Configuration & Mapping
  - SessionFactory & Session
  - Transaction
  - Query

JDBC

- Before we go to Hibernate, let's review JDBC
  - What is JDBC?
  - How to use JDBC?

TERMINOLOGIES

- Persistence

- Connection Pool

PERSISTENCE

- Persistence — The process of storing data to permanent place and retrieving data from permanent place.

- Persistent logic — the required logic to add, remove, read and modify.

- Persistent store — the place where data will be stored permanently.

CONNECTION POOL

- As we know, we have to open connection to a database whenever we are using JDBC.

- But database connections are fairly expensive operations, and as such, should be reduced to a minimum in every possible use case.

- Here is where connection pool comes into the play

CONNECTION POOL

- How to configure connection pool?
  - JDBC Datasource (Apache DBCP)
  - JNDI — Java Naming and Directory Interface
    - JNDI is a Java API for a directory service that allows Java software clients to discover and look up data and resources (in the form of Java objects) via a name
    - i.e. connecting a Java application to an external directory service (such as an address database)
    - i.e. allowing a Java Servlet to look up configuration information provided by the hosting web container

CONNECTION POOL

```
public class DBCPDataSource {

    private static BasicDataSource ds = new BasicDataSource();

    static {
        ds.setUrl("jdbc:h2:mem:test");
        ds.setUsername("user");
        ds.setPassword("password");
        ds.setMinIdle(5);
        ds.setMaxIdle(10);
        ds.setMaxOpenPreparedStatements(100);
    }

    public static Connection getConnection() throws SQLException {
        return ds.getConnection();
    }

    private DBCPDataSource(){ }
}
```

Connection con = DBCPDataSource.getConnection();

CONNECTION POOL

- Connection pooling is a well-known data access pattern, whose main purpose is to reduce the overhead involved in performing database connections and read/write database operations.

- By just simply implementing a database connection container, which allows us to reuse a number of existing connections, we can effectively save the cost of performing a huge number of expensive database trips, hence boosting the overall performance of our database-driven applications.

- It is very hard to set up connection pool with JDBC.

DRAWBACKS OF JDBC

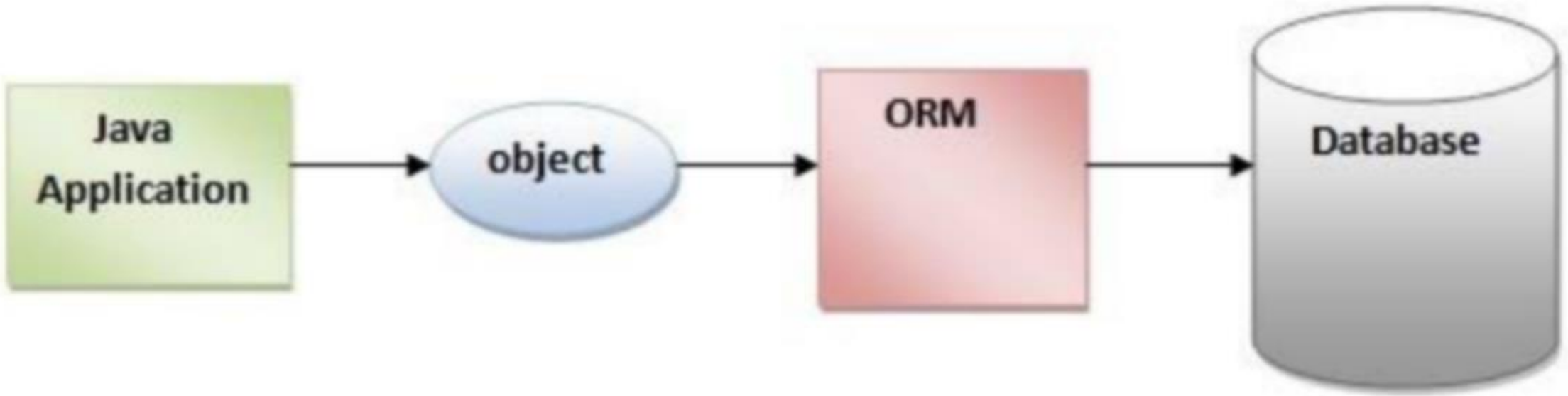
- JDBC used SQL quires to implement persistence logic. JDBC based persistence logic is becomes database dependent.
- Change of database software becomes complex and disturbs persistence logic.
- Programmer is responsible to take about exception handing and transaction management.
- ResultSet Object is not serializable object, we cannot send this object over the network.
- We need to write additional code to have connection pooling.

HIBERNATE

- Hibernate is an open source, light weight ORM tool to develop DB independent persistence logic in java based enterprise application.
    - Hibernate also provides query service along with persistence.
    - This gives developers a way to map the object structures in Java classes to relational database tables.
- ORM — Object - Relational mapping
    - ORM framework eases to store the data from object instances into persistence data store and load that data back into the same object structure

ORM

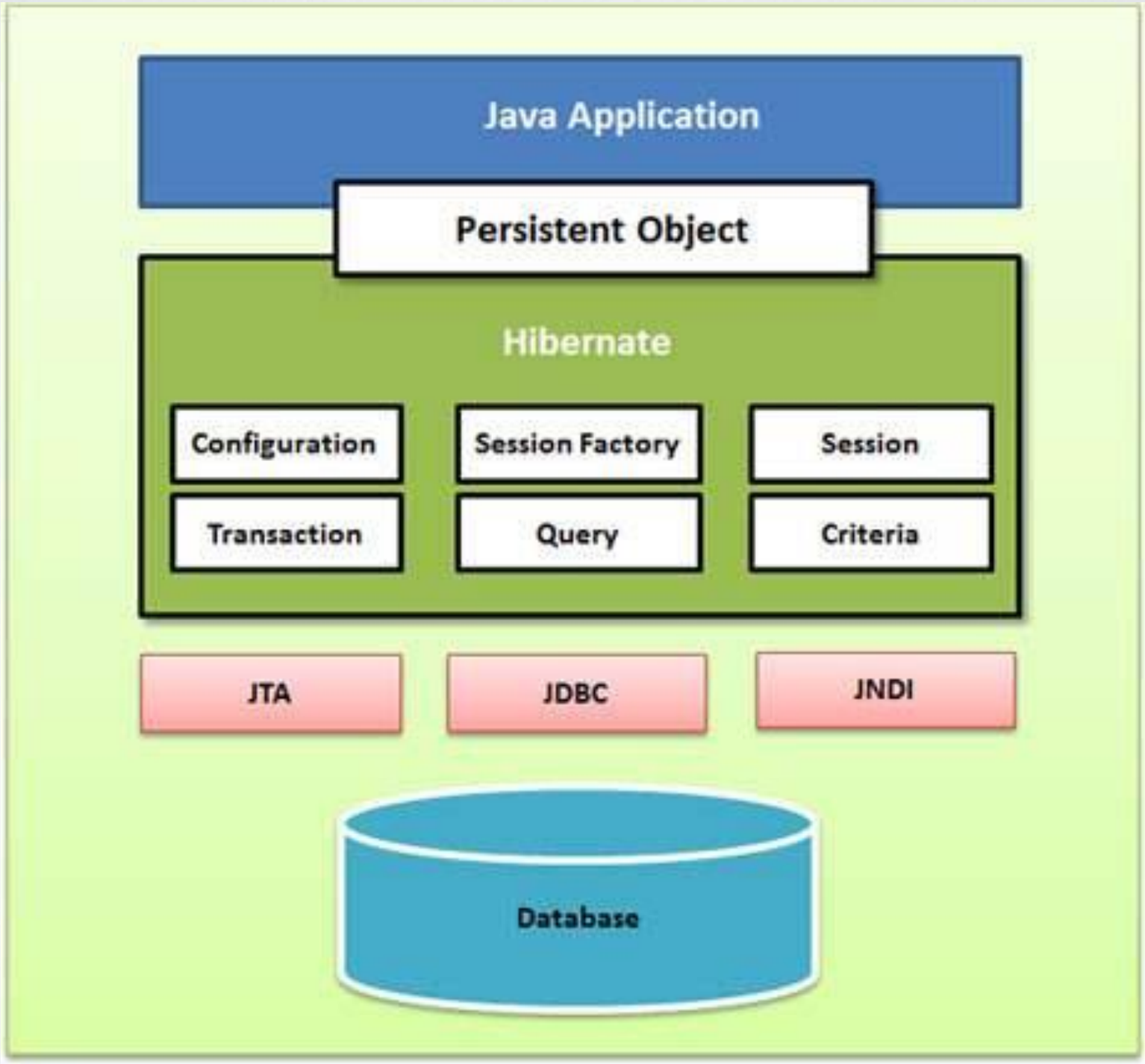
- The process of mapping java class with database table, java class members with database table columns



HIBERNATE FEATURES

- O-R mapping using ordinary JavaBeans
- Database independent persistence logic in OR Mapping style.
- Pluggable with any Java/J2EE based frameworks.
- Can set attributes using private fields or private setter methods
- object-oriented query language
- It provides APIs for storing and retrieving objects directly to and from the database.
- Transaction management with rollback

HIBERNATE STRUCTURE



HIBERNATE CONFIGURATION

- Hibernate needs to know where it can look for mapping between Java classes and relational database tables.

- Along with this mapping, Hibernate needs some database configuration settings and parameters. This information is provided

through *hibernate.cfg.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">123456</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydb</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <mapping resource="Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

HIBERNATE MAPPING

- Hibernate provides a way to map Java objects to relational database tables through an XML file. This mapping file tells hibernate how to map the defined class or classes to the database table.

```java
public class Employee implements java.io.Serializable {

    private int eid;
    private String firstname;
    private String lastname;
    private String email;
```

```xml
<hibernate-mapping>
    <class name="com.hibernate.Employee" table="employee" catalog="mydb" optimistic-lock="version">
        <id name="eid" type="int">
            <column name="eid" />
            <generator class="sequence" />
        </id>
        <property name="firstname" type="string">
            <column name="firstname" length="20" />
        </property>
        <property name="lastname" type="string">
            <column name="lastname" length="20" />
        </property>
        <property name="email" type="string">
            <column name="email" length="20" />
        </property>
    </class>
</hibernate-mapping>
```

HIBERNATE ANNOTATION

- Instead of XML configuration, there is an alternative way to configure Hibernate Mapping by Java Annotations.

```
import javax.persistence.*;
@Entity
@Table(name="employee")
public class Employee implements java.io.Serializable
{
    @Id
    @GeneratedValue
    @Column(name="eid")
    int no;

    @Column(name="firstname")
    String fname;

    @Column(name="lastname")
    String lname;

    @Column(name="email")
    String email;
```

HIBERNATE ANNOTATION

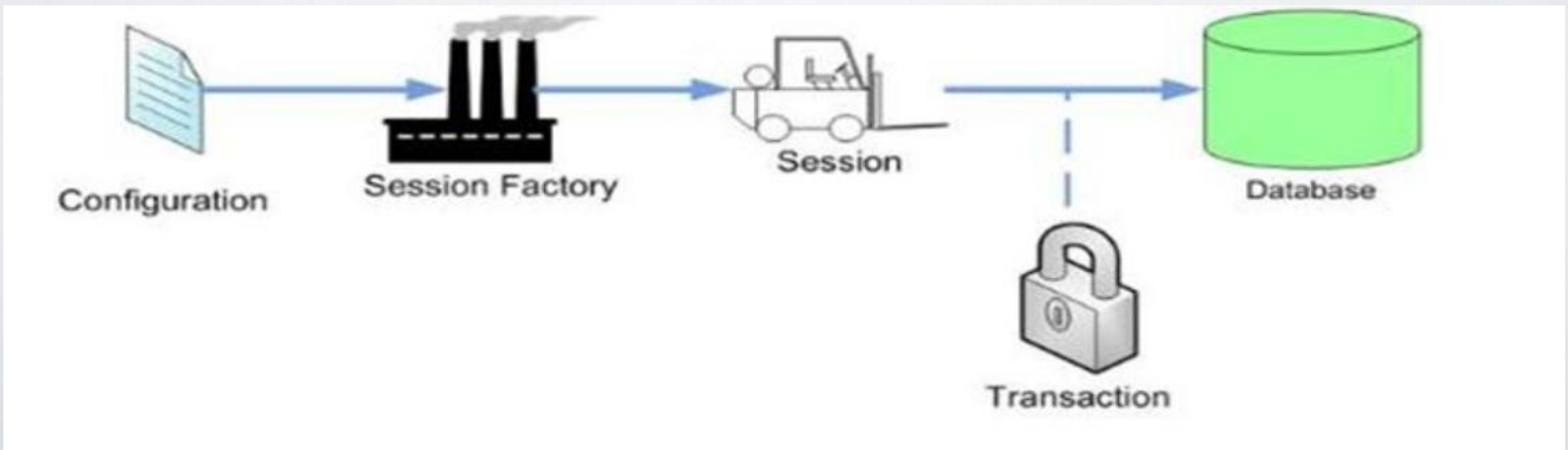
- @Entity declares the class as an entity (i.e. a persistent POJO class)
- @Table is set at the class level; it allows you to define the table, catalog, and schema names for your entity mapping. If no @Table is defined the default values are used: the unqualified class name of the entity.
- @Id declares the identifier property of this entity.
- @Generated Value annotation is used to specify the primary key generation strategy to use. If the strategy is not specified by default AUTO will be used.
- @Column annotation is used to specify the details of the column to which a field or property will be mapped. If the @Column annotation is not specified by default the property name will be used as the column name.

HIBERNATE ANNOTATION

- JPA — Java Persistence Annotation
  - JPA entities are plain POJOs. (Plain Old Java Object — not bound by any special restriction)
  - Their mappings are defined through JDK 5.0 annotations instead of hbm.xml files
  - JPA annotations are in the *javax.persistence.** package
  - https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/

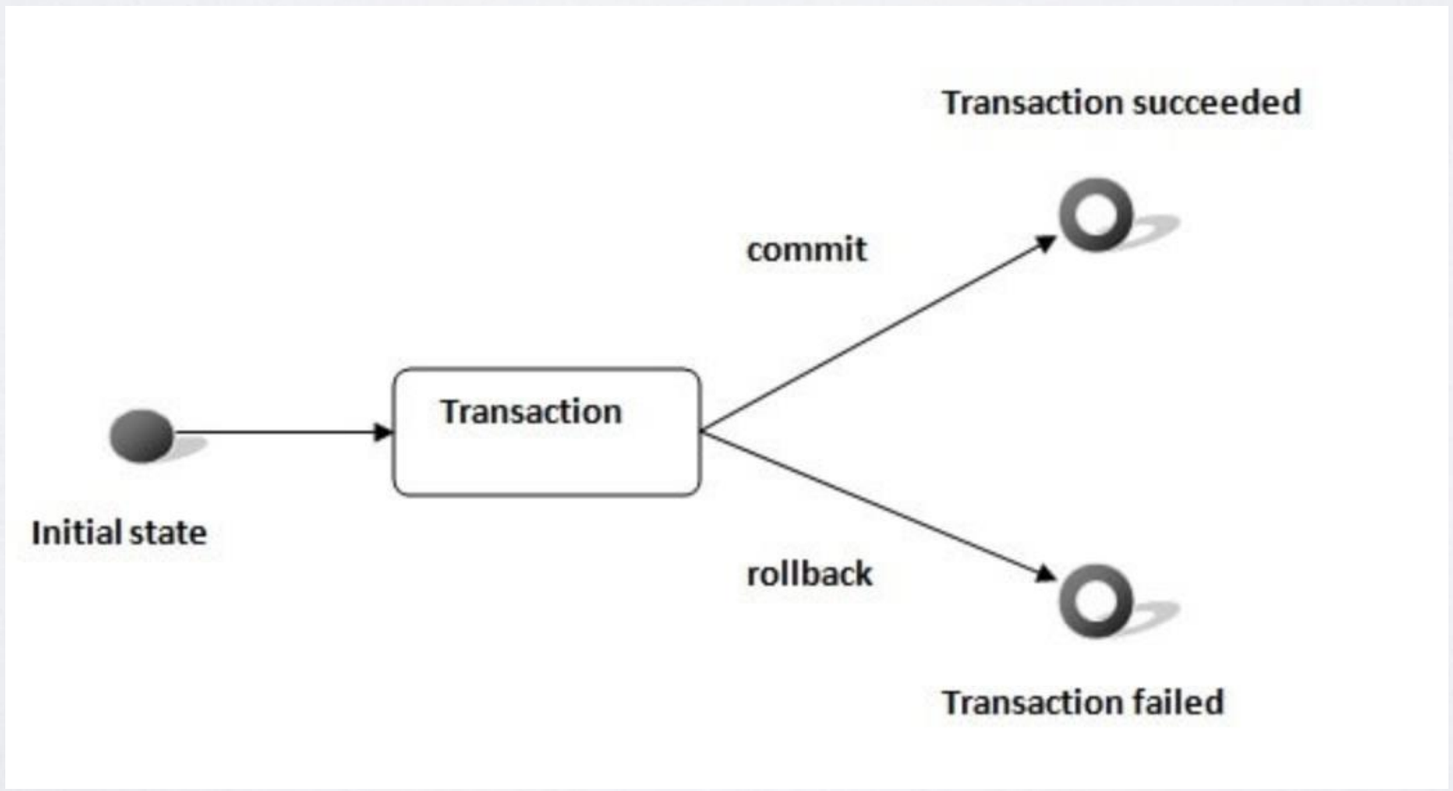
HIBERNATE SESSION & SESSION FACTORY

- The session object provides an interface between the application and data stored in the database.
  - A Session is a light weight and a non-threadsafe object that represents a single unit-of-work with the database.
  - It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria.
- SessionFactory is Hibernate's concept of a single datastore and is thread-safe so that many threads can access it concurrently and request for sessions and immutable cache of compiled mappings for a single database.



HIBERNATE TRANSACTION

- A **transaction** simply represents a unit of work.
  - In such case, if one step fails, the whole transaction fails (which is termed as atomicity). A transaction can be described by ACID properties



HIBERNATE TRANSACTION MANAGEMENT

- Hibernate provide us a collection of APIs to manage the transaction
  - void begin() — starts a new transaction.
  - void commit() — ends the unit of work unless we are in FlushMode.NEVER.
  - void rollback() — forces this transaction to rollback.
  - void setTimeout(int seconds) — it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
  - boolean isAlive() — checks if the transaction is still alive.
  - boolean wasCommited() — checks if the transaction is committed successfully.
  - boolean wasRolledBack() — checks if the transaction is rolled back successfully.

HIBERNATE SESSION INTERFACE

- Hibernate Session Interface provides us following APIs to save/get/delete data into/from database:
  - save()
  - persist()
  - update()
  - merge()
  - delete()
  - get()

LET'S TAKE A LOOK AT EXAMPLE