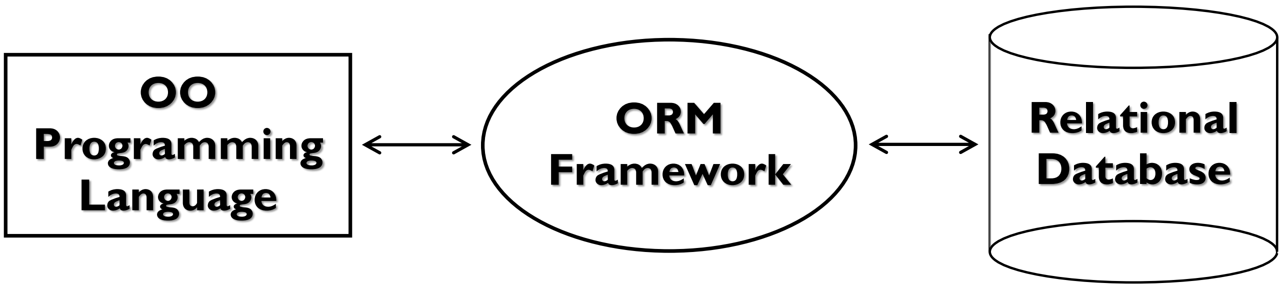
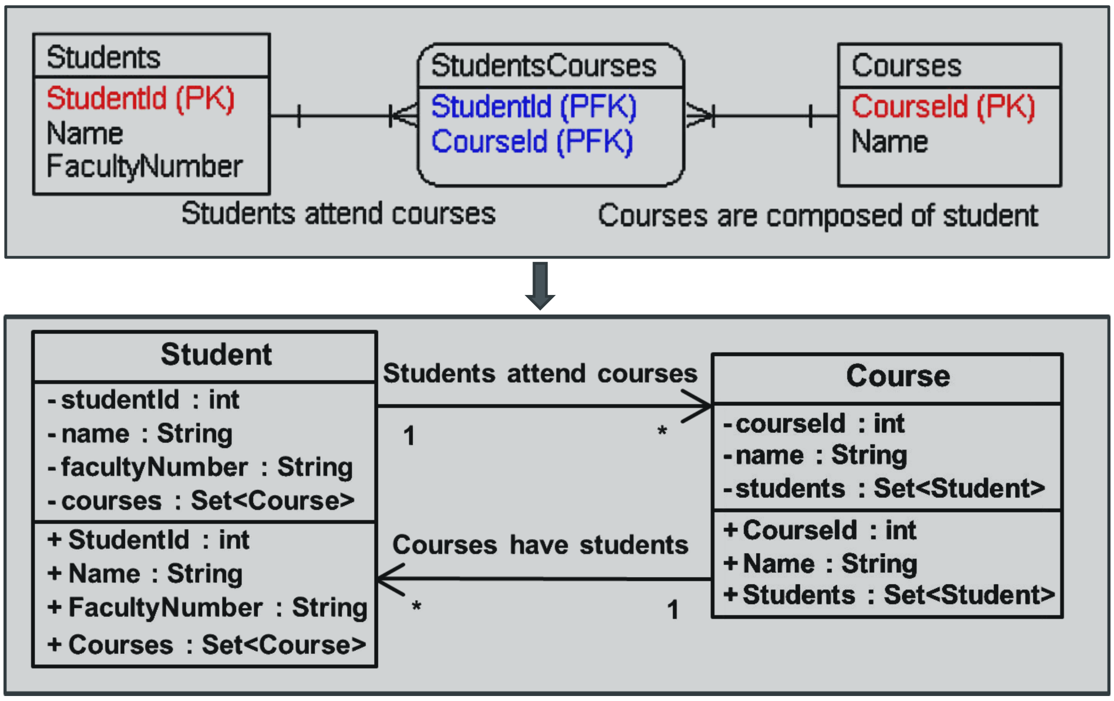
HIBERNATE

WHAT IS ORM?

- ❖ In relational databases, business entities are represented as tables + relationships
- ❖ In object-oriented languages, business entities are represented as classes
- ❖ Object relational mapping frameworks (ORMs) are used for mapping business entities to database tables



ORM – EXAMPLE



ORM TECHNOLOGIES

- ORM (Object-Relational Mapping) technologies
  - Map database tables to objects and enables CRUD operations, queries, concurrency, transactions, etc.
  - Dramatically simplifies the development of DB applications
- ORM technologies in the Java world
  - Hibernate – the most popular ORM library for Java (open source)
  - EclipseLink – ORM for Java by Eclipse foundation (open source)
  - Java Persistence API (JPA) – the standard for ORM in Java

ORM IN JAVA: PRODUCTS AND HISTORY

- ❖ Hibernate ORM – http://hibernate.org/orm/
  - The first popular ORM framework in the Java world (2001)
  - Alternative to J2EE persistence technology called "EJB“
- ❖ EclipseLink – https://eclipse.org/eclipselink/
  - ORM for Java by Eclipse foundation
  - Maps classes to database, XML and Web services
- ❖ JDO (Java Data Objects) – http://db.apache.org/jdo/
  - Java ORM persistence framework (retired)

JAVA PERSISTENCE API (JPA)

- ❖ The official standard for ORM in Java and Java EE (JSR 338)
- ❖ Unifies JDO (Java Data Objects) and EJB CMP (Enterprise JavaBeans, container-managed persistence Entity Beans)
- ❖ Implemented by most Java ORMs like Hibernate ORM, EclipseLink, OpenJPA, Apache JDO, Oracle TopLink, DataNucleus, …
- ❖ The **javax.persistence** package contains the JPA classes and interfaces.

JAVA ORM APPROACHES

Different approaches to Java ORM:

- **POJO (Plain Old Java Objects) + XML mappings**
  - A bit old-fashioned, but very powerful
  - Implemented in the "classical" Hibernate
- **Annotated Java classes (POJO) mapped to DB tables**
  - The modern approach, based on Java annotations
  - Easier to implement and maintain
- **Code generation**
  - A tool generates classes based on some ORM / persistence framework

ORM APPROACHES: POJO + XML MAPPINGS

POJO (Plain Old Java Objects) + XML mappings

```
public class Post {
  private int id;
  private String title;
  private Set<Tag> tags;
  public int getId() { … }
  public void setId(…) { … }
  public int getTitle() …
  public void setTitle() …
  public int getTags() …
  public void setTags() …
}
```

```
<hibernate-mapping>
  <class name="model.Post" table="POSTS">
        <id name="id" column="POST_ID">…</id>
        <property name="title" column="TITLE" />
        <set name="tags" table="POST_TAGS">
              <key column="POST_ID"/>
              <many-to-many class="model.Tag"
                column="TAG_ID"/>
        </set>
        …
  </class>
</hibernate-mapping>
```

ORM APPROACHES: ANNOTATED JAVA CLASSES

Java classes (POJO) + annotations

```
@Entity
public class Post {
  @Id private int id;
  private String title;

  @OneToMany(mappedBy="posts")
  private Set<Tag> tags;

  public int getId() { … }
  public void setId(int id) {…}
  …
}
```

```
@Entity
public class Tag {
  @Id private int id;
  private String text;
  public int getId() { … }
  public void setId(int id) {…}
  public int getText() { … }
  public void setText(…) {…}
  …
}
```

GROUP BY CLAUSE

HIBERNATE

HIBERNATE ORM

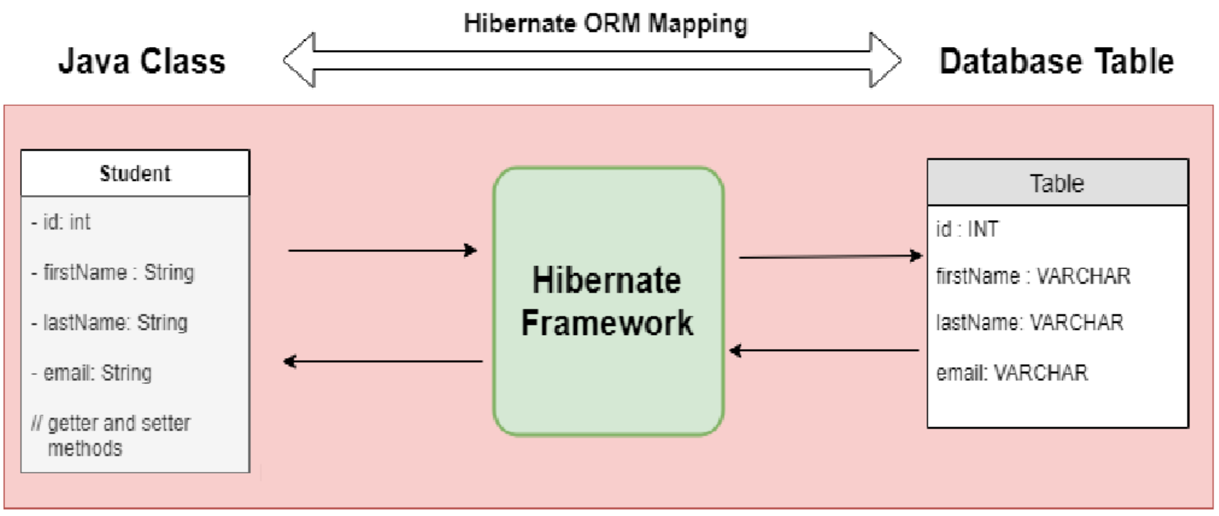
- Object-Relational Persistence for Java

HIBERNATE

- Hibernate is an **O**bject-**R**elational **M**apping (ORM) solution for JAVA.
- It is an open source persistent framework created by Gavin King in 2001.
- Hibernate is probably the most popular JPA implementation and one of the most popular Java frameworks in general.
- Hibernate acts as an additional layer on top of JDBC and enables you to implement a database-independent persistence layer.
- It provides an object-relational mapping implementation that maps your database records to Java objects and generates the required SQL statements to replicate all operations to the database.
- Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.

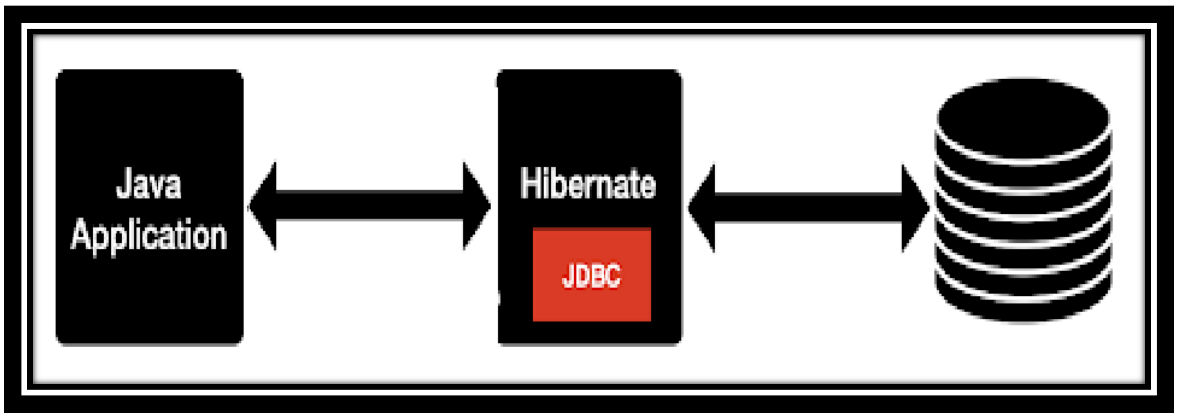
HIBERNATE ….CONT

- **Object Relational Mapping between Student Java class and student table in the database..**



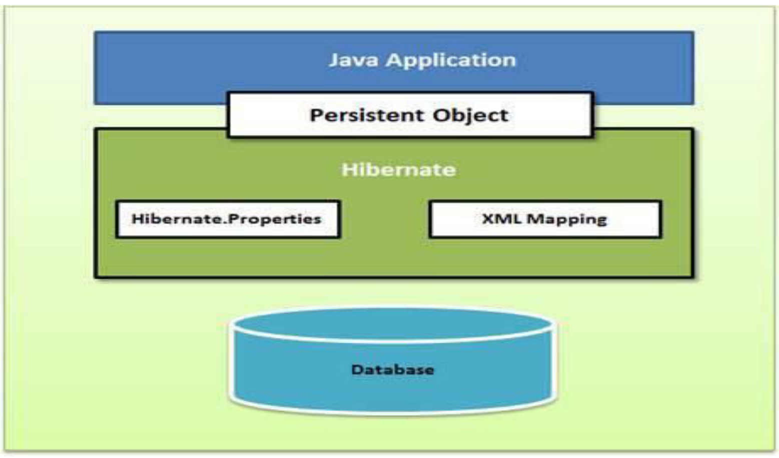
HOW DOES HIBERNATE RELATE TO JDBC?

- Hibernate uses JDBC for all database communications.
- Hibernate uses JDBC to interact with the database.
- Hibernate acts as an additional layer on top of JDBC and enables you to implement a database-independent persistence layer

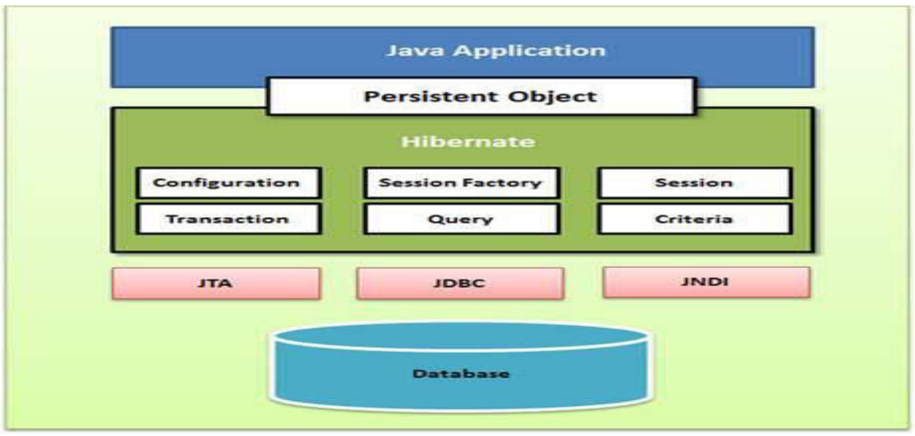


HIBERNATE ARCHITECTURE

- Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs.
- Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.



HIBERNATE ARCHITECTURE WITH ITS IMPORTANT CORE CLASSES



- Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI)

PAGINATION USING QUERY

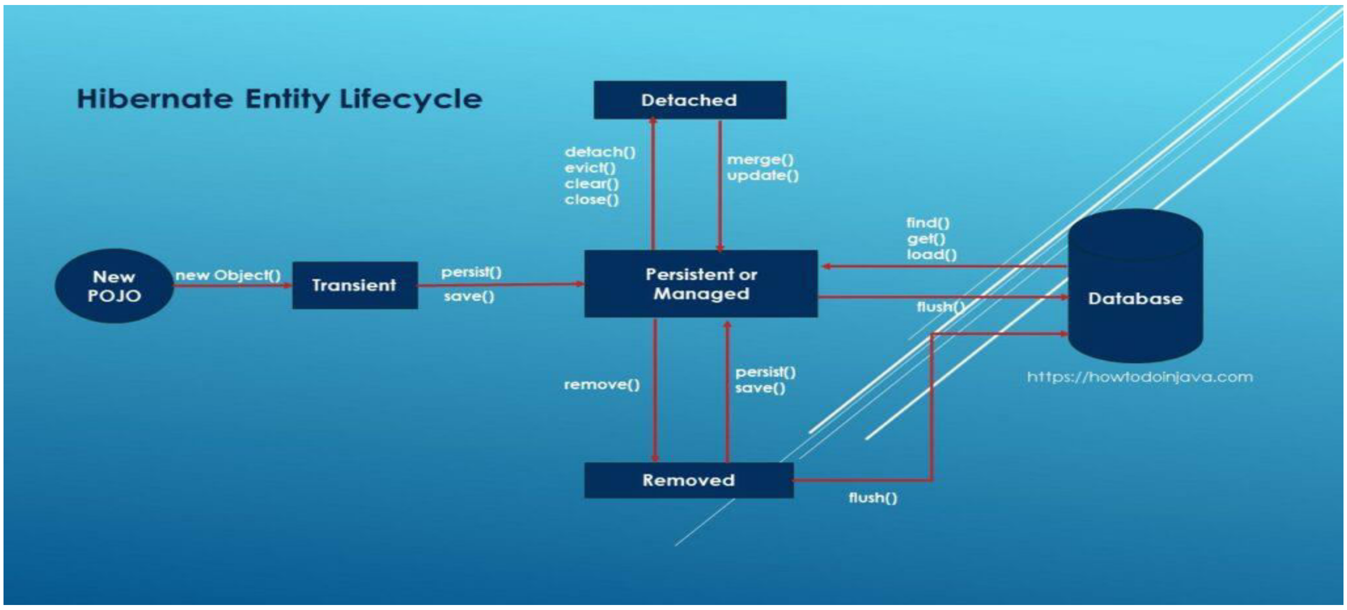
- There are two methods of the Query interface for pagination.
  - **Query setFirstResult(int startPosition) -** This method takes an integer that represents the first row in your result set, starting with row 0.
  - **Query setMaxResults(int maxResult) -** This method tells Hibernate to retrieve a fixed number **maxResults** of objects.
- Following is the example which you can extend to fetch 10 rows at a time:

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
query.setFirstResult(1);
query.setMaxResults(10);
List results = query.list();
```

HIBERNATE ENTITY LIFECYCLE STATES

- Instance of a POJO class can be in any one of **four different persistence states** (known as **hibernate entity lifecycle states**):
  - Transient
  - Persistent or Managed
  - Detached
  - Removed

HIBERNATE ENTITY LIFECYCLE STATES ..CONT



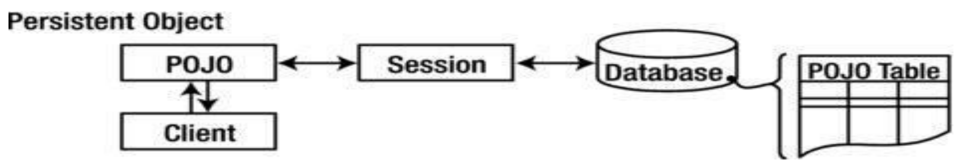
TRANSIENT STATE

- **Transient state**
  - Transient entities exist in heap memory as normal Java objects.
  - Hibernate does not manage transient entities.
  - The persistent context does not track the changes done on them.
  - In simple words, a transient entity has neither any representation in the datastore nor in the current *Session*.
  - A transient entity is simply a POJO without any identifier.

PERSISTENT OR MANAGED STATE

❖ **Persistent state**

- Persistent entities exist in the database.
- Hibernate’s persistent context tracks all the changes done on the persistent entities by the client code.
- A persistent entity is mapped to a specific database row, identified by the ID field.
- Hibernate’s current running Session is responsible for tracking all changes done to a managed entity and propagating these changes to database.



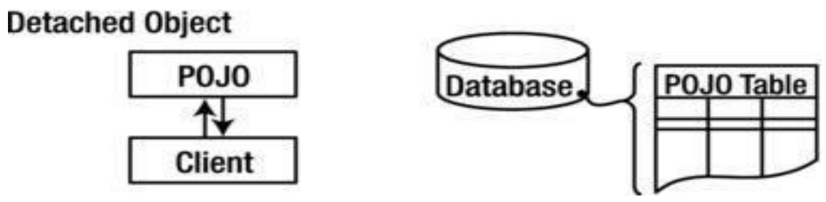
PERSISTENT OR MANAGED STATE …CONT

❖ **We can get persistent entity in either of two ways:**

- Load the entity using get() or load() method.
- Persist the transient or detached entity using persist(), save(), update() or saveOrUpdate() methods.

DETACHED STATE

- ❖ Detached entities have a representation in the database but these are currently not managed by the Session.
- ❖ Any changes to a detached entity will not be reflected in the database, and vice-versa.
- ➢ A detached entity can be created by **closing the session** that it was associated with, or by evicting it from the session with a call to the session's **evict()** method.



REMOVED STATE

- ❖ Removed entities are objects that were being managed by Hibernate and now those have been passed to the session's remove() method.
- ❖ When the application marks the changes held in the Session as to be committed, the entries in the database that correspond to removed entities are deleted.