JAVA PERSISTENCE API (JPA)

JPA NOTE : Part-1

P.K PRUSTY

linkedin.com/in/pkprusty999

INDEX

1. JPA Introduction (Java Persistence API)

- JPA stands for Java Persistence API.
- It is a Java specification for accessing, managing, and persisting data between Java objects/classes and a relational database.
- JPA provides a way to map Java objects to database tables and vice versa.
- It simplifies the development of applications that need to interact with a database by providing a set of standardized interfaces and annotations for managing relational data.
- JPA is a Java specification for object-relational mapping (ORM) and managing relational data in Java applications.
- Persistence:
  It deals with the long-term storage and retrieval of data, ensuring that valuable information isn't lost when programs or systems are stopped or restarted

1.1. Key components of JPA:

1. **Entity:** An entity is a Java class that represents a table in the database. Each instance of an entity class corresponds to a row in the table.
2. **EntityManager:** EntityManager is the main interface for interacting with the persistence context. It manages the lifecycle of entities, including creating, updating, deleting, and querying them.
3. **Persistence Unit:** A persistence unit is a configuration unit defined in the `persistence.xml` file. It defines the set of entity classes and their related settings, such as the database connection information and ORM options.
4. **Annotations:** JPA uses annotations to define the mapping between Java classes and database tables. Common annotations include **@Entity**, **@Table**, **@Column**, **@Id**, and various relationship annotations like **@OneToOne**, **@OneToMany**, **@ManyToOne**, and **@ManyToMany**.
5. **Primary Key:** The **@Id** annotation is used to designate a field in an entity class as the primary key.
6. **Relationships:** JPA supports various types of relationships between entities, such as one-to-one, one-to-many, many-to-one, and many-to-many relationships.
7. **JPQL (Java Persistence Query Language):** JPQL is a SQL-like query language used to retrieve data from the database using entity objects. It operates on entities and their relationships rather than database tables.

2. JPA BASIC ANNOTATIONS :

@Entity	Marks a class as an entity, representing a table in the database
@Table:	Specifies the name of the database table associated with an entity.
@Id	Denotes a field as the primary key of an entity.
@GeneratedValue	Configures the way primary key values are generated.
@Column	Maps a field to a database column and allows customization of column attributes.
@Basic	Specifies the default fetching strategy for an attribute.
@Transient	Marks a field to be not persisted to the database.
@Temporal	Specifies the type of a temporal attribute (DATE, TIME, TIMESTAMP).
@Enumerated	Defines the mapping for an enum type attribute.
@Version	Enables optimistic locking using a version number/timestamp
@OneToOne	Defines a one-to-many relationship between entities
@ManyToOne	Defines a many-to-one relationship between entities.
@OneToMany	Defines a one-to-many relationship between entities.
@ManyToMany	Defines a many-to-many relationship between entities.
@JoinTable	Specifies the details of a join table for a many-to-many relationship.
@JoinColumn	Specifies a column for joining an entity association.
@NamedQueries	Declares named queries for an entity.
@NamedQuery	Defines a named query for an entity
@NamedNativeQueries	Declares named native SQL queries for an entity
@NamedNativeQuery	Defines a named native SQL query for an entity.
@EntityListeners	Specifies callback listener classes for an entity.
@PrePersist	Marks a method to be executed before an entity is persisted
@ **PostPersist**	Marks a method to be executed after an entity is persisted
@PreUpdate	Marks a method to be executed before an entity is updated
@PostUpdate	Marks a method to be executed after an entity is updated.
@PreRemove	Marks a method to be executed before an entity is removed.
@PostRemove	Marks a method to be executed after an entity is removed.
@Inheritance	Specifies the inheritance strategy for an entity class hierarchy
@ DiscriminatorColumn	Configures the discriminator column for an entity hierarchy.

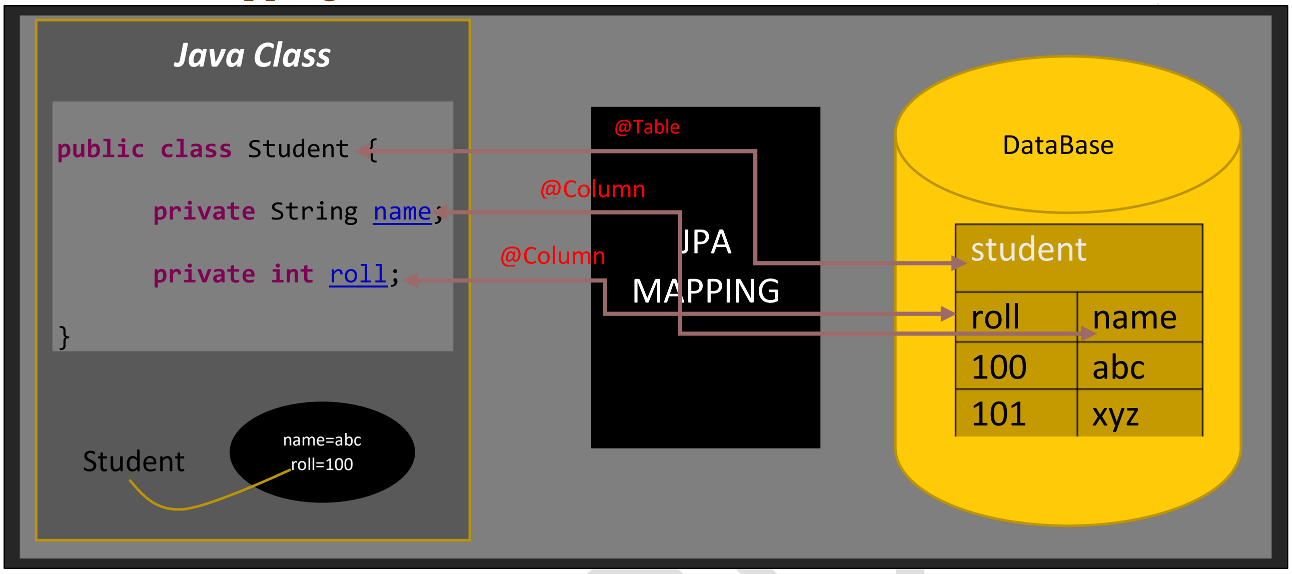
3. Introduction to Object-Relational Mapping (ORM)

Object-Relational Mapping, refers to the technique of mapping data between an Java object and a relational database.

ORM Performs database operations in Object Format ,like

save(Object obj);
update(Object obj);

3.1) ORM Mapping(JPA Architecture):



notes :

*** JPA(Interface) : {Provides specifications (rules And guidelines) Sunmicro System(Oracle)}

⬇

Implementation provides by various third party Frameworks

- Hibernate With JPA
- EclipseLink
- Apache OpenJPA

*** Difference between JDBC, ORM, JPA, Hibernate

JDBC : Used to perform Database Operations using Database Connectivity API and SQL queries.

ORM : Theory that says do DB operation in Object Format.

JPA : Specification given by sunmicro system(Oracle), provides interfaces and annotations to performs DB operations

Hibernate : Implementation of JPA

*** All these things we are using for Rapid Application development, reducing no. of lines of codes, As of now, any database programming finally executed as JDBC logic only, even any Web Application finally executed as Servlet concept only.

4. Important Interface and Classes of JPA

a. Persistence (class):

`javax.persistence.Persistence` class is a part of the JPA bootstrap API, which is used to obtain an EntityManagerFactory.

```
public class Persistence {

    /*
     * Create and return an EntityManagerFactory for the named persistence unit.
     */
    public static EntityManagerFactory createEntityManagerFactory(String persistenceUnitName)
      {
        return createEntityManagerFactory(persistenceUnitName, null);
    }

    /*
     * Create and return an EntityManagerFactory for the named persistence unit
     * using the given properties.
     */
    public static EntityManagerFactory createEntityManagerFactory(String persistenceUnitName, Map properties)
      {
      ...
      }

    /**
     * Return the PersistenceUtil instance
     */
    public static PersistenceUtil getPersistenceUtil() {
       return new PersistenceUtilImpl();
    }
}
```

Example:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("persistence-unit-name");
```

b. EntityManagerFactory (Interface):

The `EntityManagerFactory` interface is used to create `EntityManager` instances. It represents a factory for `EntityManager` instances, which are used to interact with the persistence unit.

```
public interface EntityManagerFactory {

     * Create a new application-managed <code>EntityManager</code>.
     * This method returns a new <code>EntityManager</code> instance each time it is invoked.
    public EntityManager createEntityManager();

     * Create a new application-managed <code>EntityManager</code> with the  specified Map of
properties.
       * This method returns a new <code>EntityManager</code> instance each time it is invoked.
    public EntityManager createEntityManager(Map map);

     * Indicates whether the factory is open. Returns true until the factory has been closed.
       * @return boolean indicating whether the factory is open
    public boolean isOpen();

     * Close the factory, releasing any resources that it holds.
     * After a factory instance has been closed, all methods invoked
    public void close();
}
```

c.EntityManager:

⇒ The EntityManager interface is the primary interface used to interact with the persistence context.

⇒ It is responsible for performing operations such as create, retrieve, update, delete (CRUD)and managing entity transactions.

```
public interface EntityManager {

     * This method is used to make an entity instance managed and persistent. The
     * entity is added to the persistence context, and an identifier is assigned to it.
    public void persist(Object entity);

     * It updates the entity in the database if it already exists, or it creates a
     * new entity if it doesn't exist.
    public <T> T merge(T entity);


     * This method is used to remove an entity from the persistence context and the
     * database.
    public void remove(Object entity);

     * This method is used to find an entity by its primary key. It returns the
     * entity with the specified primary key, or null if the entity does not exist.
    public <T> T find(Class<T> entityClass, Object primaryKey);

     * This method is used to synchronize the persistence context with the
     * underlying database. It ensures that all changes made to managed entities are
     * written to the database.
    public void flush();

     * This method is used to clear the persistence context, detaching all managed
     * entities. Any changes made to entities that have not been flushed to the
     * database will be lost.
    public void clear();

     * Refresh the state of the instance from the database, using the specified
     * properties, and overwriting changes made to the entity, if any.
    public void refresh(Object entity);

     * This method checks if the specified entity is associated with the current
     * persistence context. If the entity is managed by the EntityManager, this
     * method returns true; otherwise, it returns false.
    public boolean contains(Object entity);


     * This method is used to create an instance of Query for executing a Java
     * Persistence query language (JPQL) query.
    public Query createQuery(String qlString);

     * method is used to obtain the EntityTransaction instance, which is then used to
    * begin and commit the transaction. Additionally, the EntityTransaction interface
    * provides methods for controlling the transaction, such as begin(), commit(), and
    * rollback().
```

Example:

```
EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("jpa-app");
EntityManager entityManager = entityManagerFactory.createEntityManager();
```

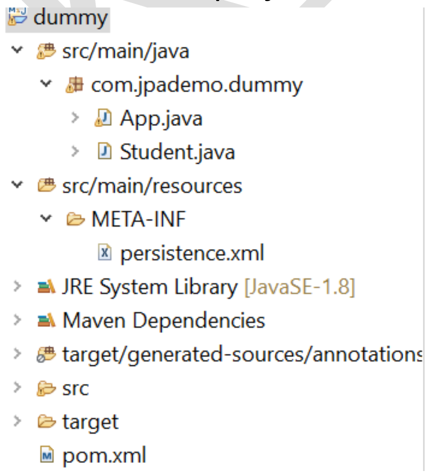
d. EntityTransaction (Interface):

The `EntityTransaction` interface in JPA (Java Persistence API) represents an active transaction between the application and the database. It provides methods to begin, commit, and roll back transactions, allowing you to manage transaction boundaries in a JPA application.

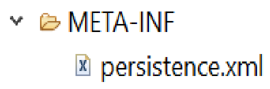
```
public interface EntityManager {

	/*
	 * This method is used to start a new transaction. It marks the beginning of the
	 * transaction boundary, and any subsequent database operations will be part of
	 * this transaction until it is committed or rolled back.
	 */
	public void begin();

	/*
	 * The commit method is used to complete the transaction and persist the changes
	 * to the database. It makes all the changes made within the transaction
	 * permanent.
	 */
	public void commit();

	/*
	 * This method is used to discard the changes made within the transaction and
	 * roll back to the state before the transaction began. It reverts any database
	 * changes made within the scope of the transaction.
	 */
	public void rollback();

	/*
	 * The isActive method checks if the transaction is currently active. It returns
	 * true if a transaction is in progress, and false if there is no active
	 * transaction.F
	 */
	public boolean isActive();
}
```

5. CRUD application Example: (Eclipse-Maven project)

- Create maven project



```
dummy
  src/main/java
    com.jpademo.dummy
      App.java
      Student.java
  src/main/resources
    META-INF
      persistence.xml
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  target/generated-sources/annotations
  src
  target
  pom.xml
```


META-INF
persistence.xml

```
Path should be: /src/main/resources/META-INF/persistence.xml
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" version="2.0">

    <persistence-unit name="jpa-app" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <class>com.jpademo.dummy.Student.java</class>

        <properties>

      <!—Database properties properties -->
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/studentdb"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="root"/>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>

            <!-- Other JPA properties -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>

        </properties>
    </persistence-unit>
</persistence>
```

here is an example of a simple **persistence.xml** file. You need to create this file in the **src/main/resources/META-INF** directory of your Eclipse project.

*** Make sure the necessary dependencies are added to the pom.xml file as well. With these configurations in place, your JPA application should be able to connect to the database and perform the necessary operations.

Explanation:

```xml
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect"/>
```

Dilect is a class that generate SQL query when we perform Database operation. Queries are database dependent so use database specific Dilect.for my sql here using *org.hibernate.dialect.MySQL8Dialect*

```xml
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

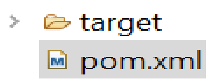
It will create table automatically based on value. Value may be(update, create, create-drop, validate)

```xml
<property name="hibernate.show_sql" value="true"/>
```

This property is used to print SQL queries on console/logfile. Default value is false.

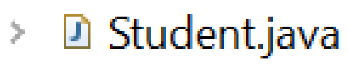
---

javax.persistence.jdbc.url => Default used to provide database URL
javax.persistence.jdbc.user=> Default used to provide userid
javax.persistence.jdbc.password => Default used to provide password
javax.persistence.jdbc.driver => Default used to provide FQN of Driver Class

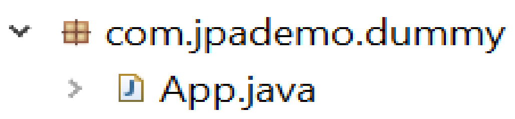

target
pom.xml

Add all dependencies to your project >>>>

```xml
		<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>8.0.33</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>6.3.0.CR1</version>
			<type>pom</type>
		</dependency>
<!--
		https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-entitymanager</artifactId>
			<version>5.6.15.Final</version>
		</dependency>


		<dependency>
			<groupId>javax.persistence</groupId>
			<artifactId>javax.persistence-api</artifactId>
			<version>2.2</version>
		</dependency>
```


Student.java

```java
package com.jpademo.dummy;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table
public class Student {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)   //follow all annotation uses
	private int id;
	@Column(name="name")
	private String name;
	@Column(name="age")
	private int age;

	public Student() {
		super();
	}
	/*
	 * Getter and Setter
	 */
}
```



com.jpademo.dummy
App.java

```java
package com.jpademo.dummy;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
public class App
{
    public static void main( String[] args )
    {
           EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("jpa-app");
            EntityManager entityManager = entityManagerFactory.createEntityManager();

            // Creating a new student
            //id will auto generate
            Student student = new Student();
            student.setName("rocky");
            student.setAge(23);

            // Persisting the student entity
            EntityTransaction transaction = entityManager.getTransaction();
            transaction.begin();
            System.out.println("=====================save====================");
            entityManager.persist(student);
            transaction.commit();


            // Retrieving the student by id
            System.out.println("=====================retrive====================");
            Student retrievedStudent = entityManager.find(Student.class, student.getId());
            System.out.println("Retrieved Student: " + retrievedStudent.getName());


            // Updating the student's age
            System.out.println("=====================Updating====================");
            transaction.begin();
            retrievedStudent.setAge(28);
            entityManager.merge(retrievedStudent);
            transaction.commit();

            //Deleting the student
            System.out.println("=====================Deleting====================");
            transaction.begin();
            entityManager.remove(retrievedStudent);
            transaction.commit();

            entityManager.close();
            entityManagerFactory.close();
    }
}
```

Output:

```
Hibernate: create table Student (id integer not null, age integer, name varchar(255), primary key (id)) engine=InnoDB
=====================save====================
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into Student (age, name, id) values (?, ?, ?)
=====================retrive====================
Retrieved Student: rocky
=====================Updating====================
Hibernate: update Student set age=?, name=? where id=?
=====================Deleting====================
Hibernate: delete from Student where id=?
```

Part-2 (all about Relationship) cont...