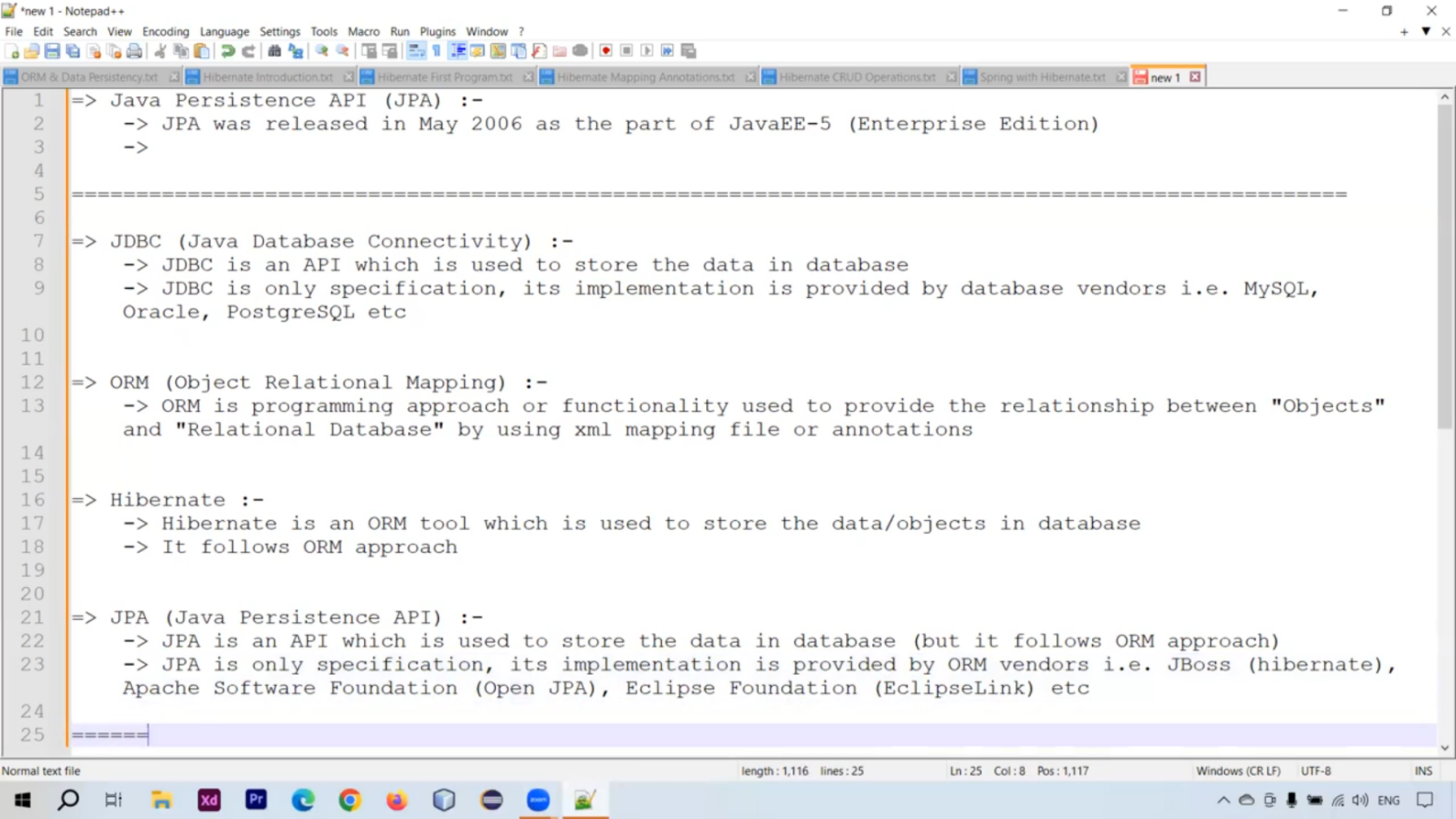

*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
ORM & Data Persistency.txt
Hibernate Introduction.txt
Hibernate First Program.txt
Hibernate Mapping Annotations.txt
Hibernate CRUD Operations.txt
Spring with Hibernate.txt
new 1
1 => Java Persistence API (JPA) :-
2 -> JPA was released in May 2006 as the part of JavaEE-5 (Enterprise Edition)
3 ->
4
5 ==========
6
7 => JDBC (Java Database Connectivity) :-
8 -> JDBC is an API which is used to store the data in database
9 -> JDBC is only specification, its implementation is provided by database vendors i.e. MySQL, Oracle, PostgreSQL etc
10
11
12 => ORM (Object Relational Mapping) :-
13 -> ORM is programming approach or functionality used to provide the relationship between "Objects" and "Relational Database" by using xml mapping file or annotations
14
15
16 => Hibernate :-
17 -> Hibernate is an ORM tool which is used to store the data/objects in database
18 -> It follows ORM approach
19
20
21 => JPA (Java Persistence API) :-
22 -> JPA is an API which is used to store the data in database (but it follows ORM approach)
23 -> JPA is only specification, its implementation is provided by ORM vendors i.e. JBoss (hibernate), Apache Software Foundation (Open JPA), Eclipse Foundation (EclipseLink) etc
24
25 ======
Normal text file
length : 1,116 lines : 25
Ln : 25 Col : 8 Pos : 1,117
Windows (CR LF) UTF-8 INS
ENG

```
=> Java Persistence API (JPA) :-
    -> JPA was released in May 2006 as the part of JavaEE-5 (Enterprise Edition)
    -> JPA is specification that simplifies the interaction between java "Objects" and "Relational
    Database"

    -> NOTE :
        = JPA is only specification, its implementation is provided by ORM vendors i.e. JBoss
        (hibernate), Apache Software Foundation (Open JPA), Eclipse Foundation (EclipseLink) etc
        = To use JPA, we have to use any one ORM tool for eg. hibernate, EclipseLink etc

    -> Advantages of JPA :-
        1. Abstraction over database :-
            = JPA provides a higher-level, object-oriented way to interact with databases, allowing
            developers to work with java objects rather than dealing with SQL queries and
            database-specific code
        2. Improves Productivity :-
            = By using JPA, developers can focus on business-logic rather than database interaction,
            leading to faster development and reduce the code complexity
        3. ORM (Object-Relational Mapping) :-
            = JPA enables ORM which means it maps Java Objects to Database and vice versa. This
            simplifies the storage and retrival of java objects in database eliminating the need to
            write low-level SQL queries. JPA allows to developers to switch between different ORM vendors
        4. Database Portability :-
            = JPA abstracts away the database-specific details, making it wasier to switch between
            different databases
        5. Scalability :-
            = JPA enables the development of scalable applications by managing database connections,
            pooling, optimizing performance etc
```

```
=> What is specification ?
    -> Specification means guidelines or blueprint for how a particular technology or API should work
    -> Specification means formal document that defines a set of rules, behaviour and interfaces that
    software application implements
```

```
=> Persisting an Entity :-
    = Persistence means "store permanently"
    = Entity means "objects"
    -> Persisting an entity means "storing the object in database permanently
```

=> JPA is an API which contains interfaces and classes which are as follows :-

1. Persistence <class>
2. EntityManagerFactory <interface>
3. EntityManager <interface>
4. EntityTransaction <interface>

5. Query <interface>
6. TypedQuery <interface>

7. CriteriaBuilder <interface>
8. CriteriaQuery <interface>

   etc

=> NOTE :-

= All above classes and interfaces are present in "javax.persistence" package

= The "Java Persistence API (JPA)" name has been changed or rebanded to "Jarkarta Persistence API (JPA)" in 2019

=> Persistence <class> :-
    -> It provides a static method i.e. createEntityManagerFactory() for obtaining EntityManag Fa tory
    instances

=> EntityManagerFactory <interface> :-
    -> It is used to create EntityManager instance for database operations

=> EntityManager <interface> :-
    -> It manages the lifecycle of entities and provides methods for CRUD operations

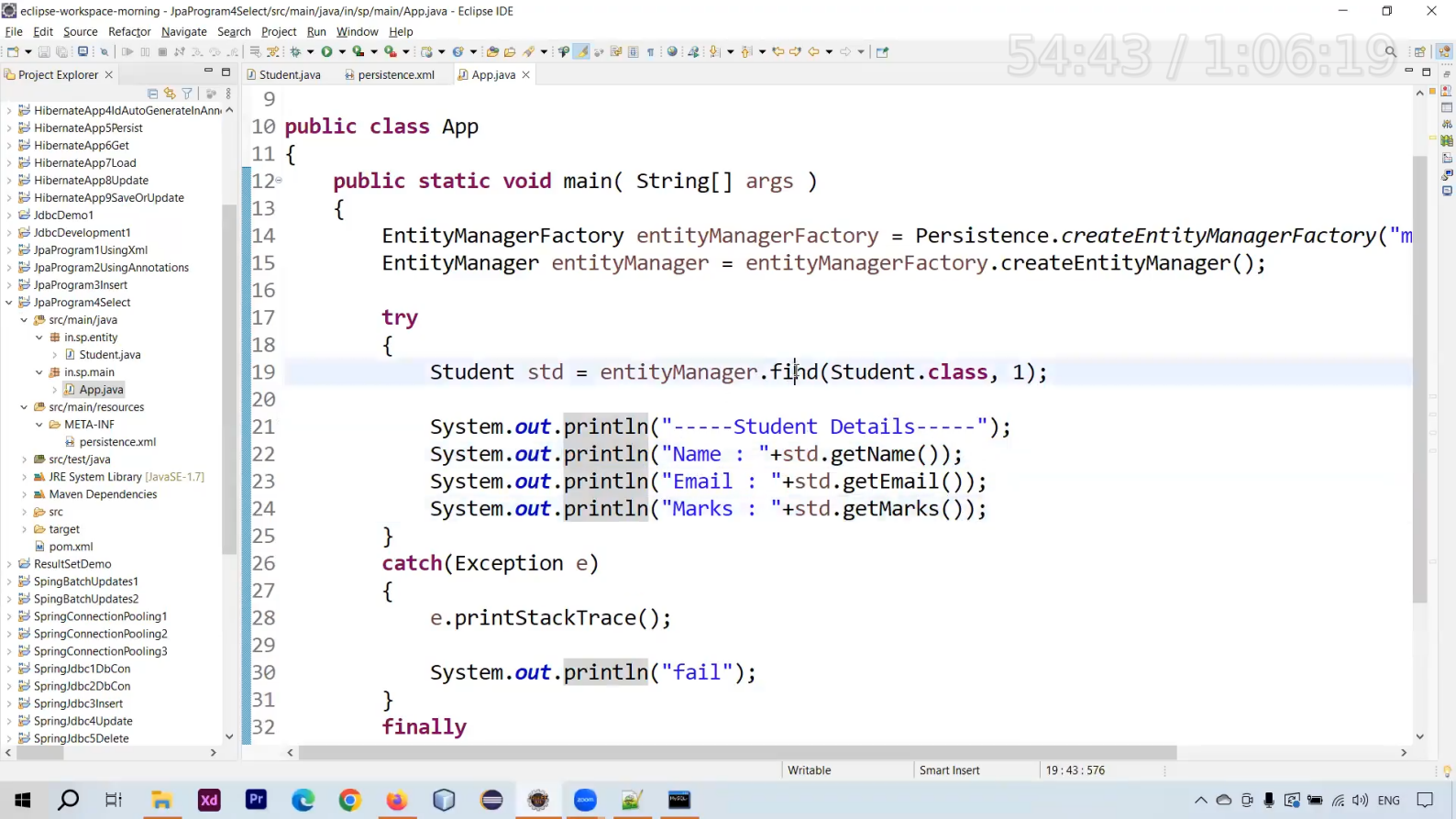
=> EntityTransaction <interface> :-
    -> It is used for transaction management including starting, committing and rolling back the
    transactions

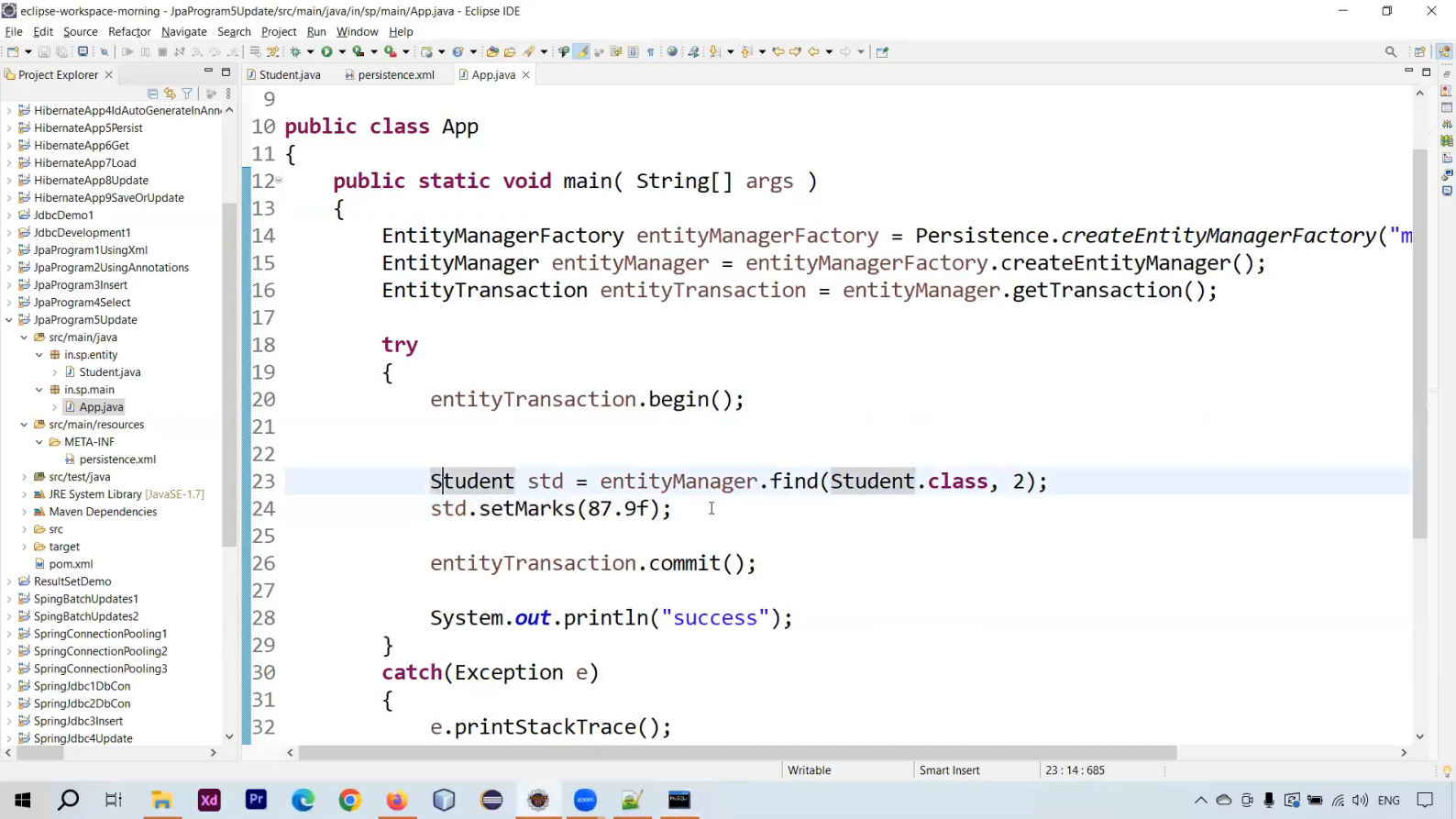
==========

=> Steps to create JPA first program :-

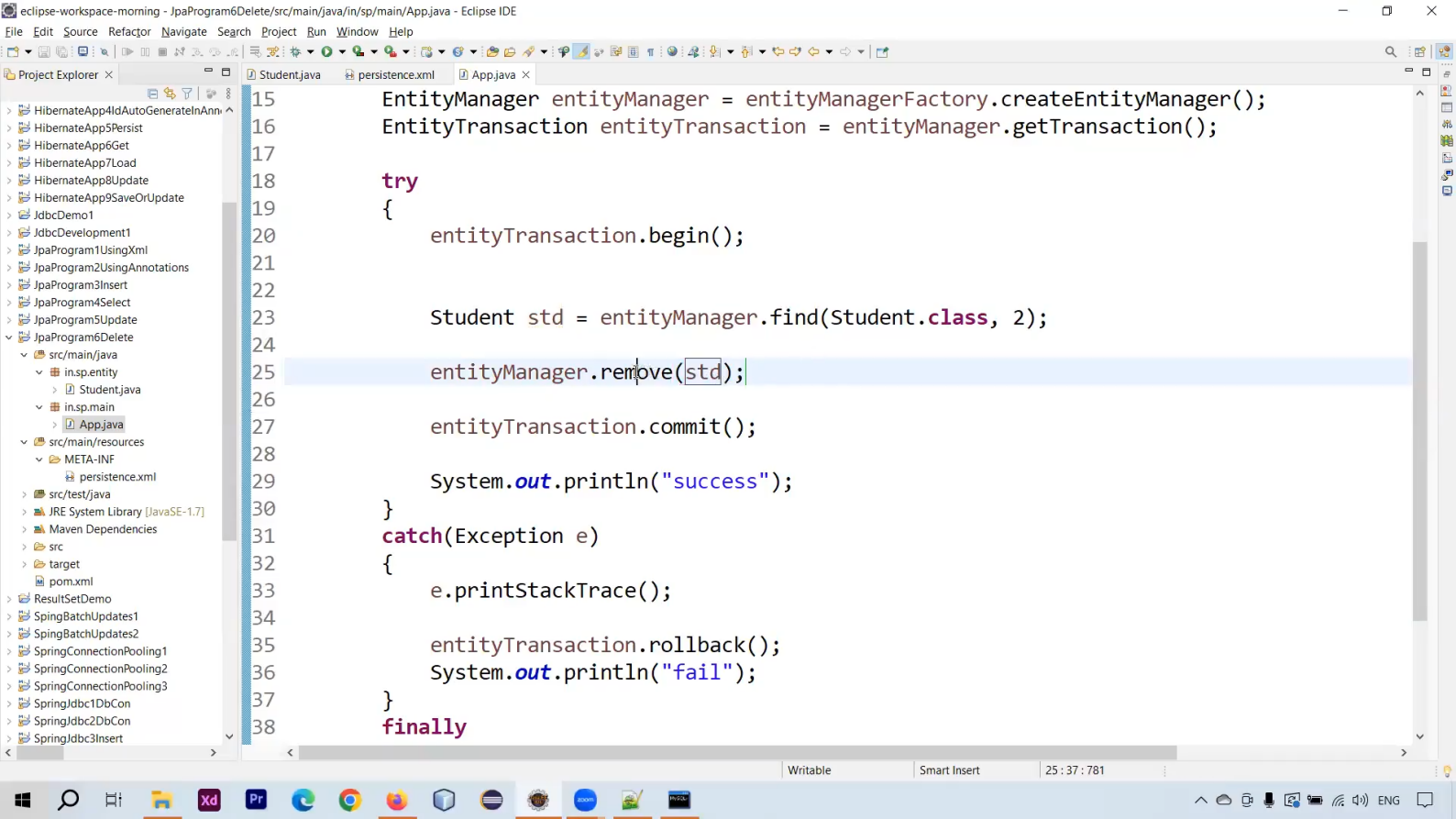
1. Download and Install any one IDE (eclipse)
2. Create "Simple Java Project" or "Maven Project"
3. Add "jars" (for simple java project) or "provide dependencies" (for maven project)
   - = javax.persistence-api.jar
   - = JPA provider jar (hibernate-core)
   - = JDBC Driver jar (mysql-connector jar)
4. Create POJO (Plain Old Java Object) class
5. Create mapping file (or provide annotations in POJO class)
6. Create Persistence Unit Configuration (persistence.xml) in META-INF folder
7. Create main class and execute the application
8. Close the resources

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
          http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
        version="2.2">

    <persistence-unit name="">
        
    </persistence-unit>

</persistence>
```


eclipse-workspace-morning - JpaProgram4Select/src/main/java/in/sp/main/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
54:43 / 1:06:19
Project Explorer
HibernateApp4IdAutoGenerateInAnn
HibernateApp5Persist
HibernateApp6Get
HibernateApp7Load
HibernateApp8Update
HibernateApp9SaveOrUpdate
JdbcDemo1
JdbcDevelopment1
JpaProgram1UsingXml
JpaProgram2UsingAnnotations
JpaProgram3Insert
JpaProgram4Select
src/main/java
in.sp.entity
Student.java
in.sp.main
App.java
src/main/resources
META-INF
persistence.xml
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
ResultSetDemo
SpingBatchUpdates1
SpingBatchUpdates2
SpringConnectionPooling1
SpringConnectionPooling2
SpringConnectionPooling3
SpringJdbc1DbCon
SpringJdbc2DbCon
SpringJdbc3Insert
SpringJdbc4Update
SpringJdbc5Delete
Student.java
persistence.xml
App.java
9
10 public class App
11 {
12 public static void main(String[] args)
13 {
14 EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("m
15 EntityManager entityManager = entityManagerFactory.createEntityManager();
16
17 try
18 {
19 Student std = entityManager.find(Student.class, 1);
20
21 System.out.println("-----Student Details-----");
22 System.out.println("Name : "+std.getName());
23 System.out.println("Email : "+std.getEmail());
24 System.out.println("Marks : "+std.getMarks());
25 }
26 catch(Exception e)
27 {
28 e.printStackTrace();
29
30 System.out.println("fail");
31 }
32 finally
Writable
Smart Insert
19 : 43 : 576
ENG


eclipse-workspace-morning - JpaProgram5Update/src/main/java/in/sp/main/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
HibernateApp4IdAutoGenerateInAnn
HibernateApp5Persist
HibernateApp6Get
HibernateApp7Load
HibernateApp8Update
HibernateApp9SaveOrUpdate
JdbcDemo1
JdbcDevelopment1
JpaProgram1UsingXml
JpaProgram2UsingAnnotations
JpaProgram3Insert
JpaProgram4Select
JpaProgram5Update
src/main/java
in.sp.entity
Student.java
in.sp.main
App.java
src/main/resources
META-INF
persistence.xml
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
ResultSetDemo
SpingBatchUpdates1
SpingBatchUpdates2
SpringConnectionPooling1
SpringConnectionPooling2
SpringConnectionPooling3
SpringJdbc1DbCon
SpringJdbc2DbCon
SpringJdbc3Insert
SpringJdbc4Update
Student.java
persistence.xml
App.java
9
10 public class App
11 {
12 public static void main(String[] args)
13 {
14 EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("m
15 EntityManager entityManager = entityManagerFactory.createEntityManager();
16 EntityTransaction entityTransaction = entityManager.getTransaction();
17
18 try
19 {
20 entityTransaction.begin();
21
22
23 Student std = entityManager.find(Student.class, 2);
24 std.setMarks(87.9f);
25
26 entityTransaction.commit();
27
28 System.out.println("success");
29 }
30 catch(Exception e)
31 {
32 e.printStackTrace();
Writable
Smart Insert
23 : 14 : 685
ENG

```
//entityManager.merge(std);     //not needed because entity will be udpated autom
```


eclipse-workspace-morning - JpaProgram6Delete/src/main/java/in/sp/main/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
HibernateApp4IdAutoGenerateInAnn
HibernateApp5Persist
HibernateApp6Get
HibernateApp7Load
HibernateApp8Update
HibernateApp9SaveOrUpdate
JdbcDemo1
JdbcDevelopment1
JpaProgram1UsingXml
JpaProgram2UsingAnnotations
JpaProgram3Insert
JpaProgram4Select
JpaProgram5Update
JpaProgram6Delete
src/main/java
in.sp.entity
Student.java
in.sp.main
App.java
src/main/resources
META-INF
persistence.xml
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
ResultSetDemo
SpingBatchUpdates1
SpingBatchUpdates2
SpringConnectionPooling1
SpringConnectionPooling2
SpringConnectionPooling3
SpringJdbc1DbCon
SpringJdbc2DbCon
SpringJdbc3Insert
Student.java
persistence.xml
App.java
15 EntityManager entityManager = entityManagerFactory.createEntityManager();
16 EntityTransaction entityTransaction = entityManager.getTransaction();
17
18 try
19 {
20 entityTransaction.begin();
21
22
23 Student std = entityManager.find(Student.class, 2);
24
25 entityManager.remove(std);
26
27 entityTransaction.commit();
28
29 System.out.println("success");
30 }
31 catch(Exception e)
32 {
33 e.printStackTrace();
34
35 entityTransaction.rollback();
36 System.out.println("fail");
37 }
38 finally
Writable
Smart Insert
25 : 37 : 781
ENG

*new 1 - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

ORM & Data Persistency.txt | Hibernate Introduction.txt | Hibernate First Program.txt | Hibernate Mapping Annotations.txt | Hibernate CRUD Operations.txt | Spring with Hibernate.txt | JPA Introduction.txt | new 1

```
    5. Create mapping file (or provide annotations in POJO class)
    6. Create Persistence Unit Configuration (persistence.xml) in META-INF folder
    7. Create main class and execute the application
        = Create EntityManagerFactory object :-
            -> EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-persistence-unit");
        = Create EntityManager object :-
            -> EntityManager em = emf.createEntityManager();
        = Create EntityTransaction object and begin the transaction
            -> EntityTransaction et = em.getTransaction();
               et.begin();
        = Perform database operations using EntityManager instance
            -> Insert operation
                = persist()
            -> Retrival operation :-
                = find()
            -> Update operation :-
                = merge()
            -> Delete operation :-
                = remove()
    8. Commit the transaction
    9. Close the resources


=> NOTE :-
    -> All the above CRUD operation methods are used to work with individual record or to perform single
    record manipulation only
    ->
```

Normal text file | length : 2,792 lines : 73 | Ln : 57 Col : 1 Sel : 167 | 8 | Windows (CR LF) | UTF-8 | INS

ENG

=> NOTE :-
-> All the above CRUD operation methods are used to work with individual record or to perform single record manipulation only
-> If we want to work with multiple records (perform manipulations with multiple records) like select, insert, update, delete multiple records then we have to use :-

1. JPQL (Java Persistence Query Langauge)
2. Native SQL
3. JPA Criteria API (less used)

```
16      EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
17      EntityManager em = emf.createEntityManager();
18
19      try
20      {
21          String jpql_query = "SELECT s FROM Student s";
22          Query query = em.createQuery(jpql_query);
23
24          List<Student> std_list = query.getResultList();
25
26          for(Student std : std_list)
27          {
28              System.out.println("Id : "+std.getId());
29              System.out.println("Name : "+std.getName());
30              System.out.println("Email : "+std.getEmail());
31              System.out.println("Marks : "+std.getMarks());
32              System.out.println("-----------------------------------");
33          }
34      }
35      catch(Exception e)
36      {
37          e.printStackTrace();
38      }
39      finally
```

```
try
{
    String jpql_query = "SELECT s FROM Student s WHERE s.id = :stdId";
    TypedQuery<Student> query = em.createQuery(jpql_query, Student.class);
    query.setParameter("stdId", 102);

    Student std = query.getSingleResult();

    if(std != null)
    {
        System.out.println("Id : "+std.getId());
        System.out.println("Name : "+std.getName());
        System.out.println("Email : "+std.getEmail());
        System.out.println("Marks : "+
        System.out.println("----------                    ----");
    }
```

Student std - in.sp.main.App.main(String[])

Press 'F2' for focus

```
try
{
    String jpql_query = "SELECT s.id, s.name FROM Student s";
    TypedQuery<Object[]> query = em.createQuery(jpql_query, Object[].class);

    List<Object[]> obj_list = query.getResultList();

    for(Object[] obj : obj_list)
    {
        System.out.println("Id : "+obj[0]);
        System.out.println("Name : "+obj[1]);

        System.out.println("--------------------------");
    }
}
catch(Exception e)
```

Student.java App.java

```
3  import java.util.List;
13
14 public class App
15 {
16     public static void main( String[] args )
17     {
18         EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
19         EntityManager em = emf.createEntityManager();
20         EntityTransaction et = em.getTransaction();
21
22         try
23         {
24             et.begin();
25
26             String jpql_query = "UPDATE Student s SET s.marks = :newMarks WHERE s.id = :stdId";
27
28             Query query = em.createQuery(jpql_query);
29             query.setParameter("newMarks", 66.77);
30             query.setParameter("stdId", 103);
31
32             int count = query.executeUpdate();
33             if(count > 0)
34             {
35                 System.out.println("success");
```

*new 1 - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Spring with Hibernate.txt | JPA Introduction.txt | JPA API & JPA First Program.txt | new 1

```
=> JPQL (Java Persistence Query Language) :-
    -> JPQL is a query language used for querying and manipulating java objects with relational database
    -> It allows the developers to write the queries in an object-oriented way, treating database
    records as objects

    -> Some JPQL queries are as follows :-
        1. Select query :-
            = SELECT en FROM EntityName en
            = SELECT en FROM EntityName en WHERE en.attribute = :value
        2. Update query :-
            = UPDATE EntityName en SET en.attribute = :newValue WHERE en.id = :entityId
        3. Delete query :-
            = DELETE FROM EntityName en WHERE en.id = :entityId;
        etc

    => NOTE :-
        = There is no INSERT query in JPQL. If we have to insert new entity in database then we have to
        use EntityManager [persist()] or native SQL query
        = There are many other queries i.e. COUNT, JOIN, GROUP ID, ORDER BY, DISTINCT etc
        = Each JPQL query should be specific to a single entity

    =>


==========================================================================================
```

Normal text file | length : 1,038 lines : 23 | Ln : 12 Col : 65 Sel : 50 | 1 | Windows (CR LF) | UTF-8 | INS

ENG

eclipse-workspace-morning - JpqlProgram3Delete/src/main/java/in/sp/main/App.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Student.java App.java

```
8
9  public class App
10 {
11     public static void main( String[] args )
12     {
13         EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
14         EntityManager em = emf.createEntityManager();
15         EntityTransaction et = em.getTransaction();
16
17         try
18         {
19             et.begin();
20
21             String jpql_query = "DELETE FROM Student s WHERE s.id = :stdId";
22
23             Query query = em.createQuery(jpql_query);
24             query.setParameter("stdId", 103);
25
26             int count = query.executeUpdate();
27             if(count > 0)
28             {
29                 System.out.println("success");
30                 et.commit();
31             }
```

Writable | Smart Insert | 22 : 13 : 666

ENG

==========

=> Query (Interface) :-
    -> It is the fundamental JPA interface that represents a database query
    -> It is used to execute dynamic queries including both JPQL and native SQL queries
    -> It allows parametrization, result retrival and is used for various querying operations

=> TypedQuery (Interface) :-
    -> It is a specilized subtype of the Query interface of JPA
    -> It enhances type safety by demanding us to specify the expected result type when creating a query. This ensures that query results are automatically cast to the specified type, reducing the risk of type-related errors.
    -> It is useful when working with entity types in JPA as it eliminates the need for explicitly casting when retriving the results

==========

=> Why we should use JPQL :-

1. Advanced Quering :-
   = If we want to use complex queries i.e. joins, aggregations etc then we can use JPQL (as standard CRUD operations are not suitable)
2. Improve Performance :-
   = If we are fetching large datasets or executing complex queries then JPQL is used to improve the performance (as standard CRUD operations reduces the performance)
3. Aggregations :-
   = JPQL can perform aggregation functions eg SUM, COUNT etc for reporting and analytics
4. Type Safety :-
   = JPQL provides type-safety which is helpful for catching the errors at compile time
5. Other benifits :-
   = JPQL provides more flexiblity, enabling custom queries and advanced filtering for a wider range of use case

etc

```
8
9  @Entity
10 @Table(name = "std_details")
11 @NamedQuery(name = "getAllStudentDetails", query = "SELECT s FROM Student s")
12 public class Student
13 {
```

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
EntityManager em = emf.createEntityManager();

try
{
    TypedQuery<Student> query = em.createNamedQuery("getAllStdDetails", Student.class);
    List<Student> std_list = query.getResultList();
    for(Student std : std_list)
    {
        System.out.println("Id : "+std.getId());
        System.out.println("Name : "+std.getName());
        System.out.println("Rollno : "+std.getRollno());
        System.out.println("Marks : "+std.getMarks());
        System.out.println("------------------------------");
    }
}
```

```
@Entity
@Table(name = "std_details")
@NamedQuery(name = "getAllStdDetails", query = "SELECT s FROM Student s")
@NamedQuery(name = "getStdDetailsById", query = "SELECT s FROM Student s WHERE s.id = :stdId")
public class Student
```

eclipse-workspace-morning - JpaNamedQueries2/src/main/java/in/sp/main/App.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Student.java App.java

```
17      EntityManager em = emf.createEntityManager();
18
19      try
20      {
21          TypedQuery<Student> query1 = em.createNamedQuery("getAllStdDetails", Student.class);
22          List<Student> std_list1 = query1.getResultList();
23          for(Student std1 : std_list1)
24          {
25              System.out.println("Id : "+std1.getId());
26              System.out.println("Name : "+std1.getName());
27              System.out.println("Rollno : "+std1.getRollno());
28              System.out.println("Marks : "+std1.getMarks());
29              System.out.println("-----------------------------");
30          }
31
32          //-----------------------------------------------------------------------------------------
33
34          TypedQuery<Student> query2 = em.createNamedQuery("getStdDetailsById", Student.class);
35          query2.setParameter("stdId", 3);
36          Student std2 = query2.getSingleResult();
37          System.out.println("Hello "+std2.getName()+", you got "+std2.getMarks()+" marks");
38      }
39      catch(Exception e)
40      {
```

Writable | Smart Insert | 37 : 92 : 1430

ENG

```
@Entity
@Table(name = "std_details")
@NamedQueries({
    @NamedQuery(name = "getAllStdDetails", query = "SELECT s FROM Student s"),
    @NamedQuery(name = "getStdDetailsById", query = "SELECT s FROM Student s WHERE s.id = :stdId")
})
```

=> Named Queries in JPA :-

-> Named queries in JPA are queries to which we assign a particular name, making them easier to reference and reuse throughout our application

-> We can declare named queries in an entity class or in xml file

-> Syntax :-

= In an entity class :-

```
@NamedQuery(name="queryName", query="----JPQL-----")
public class Student
{
    //----
}
```

= In xml file (orm.xml)

```
<named-query name="queryName">
    <query> ----JPQL----- </query>
</named-query>
```

-> NOTE :-

= Named queries in JPA are primarily used for selecting data (SELECT query). These are not used for inserting, updating or deleting data (INSERT, UPDATE or DELETE queries)

-> Advantages of Named Queries :-

1. Code Reusability :-
   = Named Queries can be reused in different parts of our application, reducing the code duplication. This enhances the consitency and readability of our code
2. Code Maintability :-
   = Named Queries centralize query definitions, making it easier to maintain and update queries across our application. When a query needs modification, we can update it in one place and all the references to that query will automatically updated
3. Type Safety :-
   = Named Queries are checked for correctness at compile time, which helps to catch the errors early in the development process

etc

eclipse-workspace-morning - JpaNativeSqlQuery1Select1/src/main/java/in/sp/main/App.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Student.java persistence.xml App.java

```
13 {
14     public static void main( String[] args )
15     {
16         EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
17         EntityManager em = emf.createEntityManager();
18
19         try
20         {
21             String native_sql_query = "SELECT * FROM std_details";
22             Query query = em.createNativeQuery(native_sql_query, Student.class);
23             List<Student> std_list = query.getResultList();
24             for(Student std : std_list)
25             {
26                 System.out.println("Id : "+std.getId());
27                 System.out.println("Name : "+std.getName());
28                 System.out.println("Rollno : "+std.getRollno());
29                 System.out.println("Marks : "+std.getMarks());
30
31                 System.out.println("------------------------------------");
32             }
33         }
34         catch(Exception e)
35         {
36             e.printStackTrace();
```

Writable Smart Insert 31 : 73 : 1065

ENG

Student.java App.java

```
18      EntityManager em = emf.createEntityManager();
19      EntityTransaction et = em.getTransaction();
20
21      try
22      {
23          et.begin();
24
25          String native_sql_query = "INSERT into std_details VALUES(:stdId, :stdName, :stdRollno, :stdMarks)";
26          Query query = em.createNativeQuery(native_sql_query, Student.class);
27          query.setParameter("stdId", 4);
28          query.setParameter("stdName", "ddd");
29          query.setParameter("stdRollno", 104);
30          query.setParameter("stdMarks", 74.75);
31
32          int count = query.executeUpdate();
33          if(count > 0)
34          {
35              System.out.println("success");
36              et.commit();
37          }
38          else
39          {
40              System.out.println("fail");
41              et.rollback();
```

Student.java App.java

```
15    public static void main( String[] args )
16    {
17        EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
18        EntityManager em = emf.createEntityManager();
19        EntityTransaction et = em.getTransaction();
20
21        try
22        {
23            et.begin();
24
25            String native_sql_query = "INSERT into std_details(std_id, std_name, std_rollno, std_marks) VALUES(
26            Query query = em.createNativeQuery(native_sql_query, Student.class);
27            query.setParameter("stdId", 4);
28            query.setParameter("stdName", "ddd");
29            query.setParameter("stdRollno", 104);
30            query.setParameter("stdMarks", 74.75f);
31
32            int count = query.executeUpdate();
33            if(count > 0)
34            {
35                System.out.println("success");
36                et.commit();
37            }
38            else
```

eclipse-workspace-morning - JpaNativeSqlQuery3Update/src/main/java/in/sp/main/App.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Student.java App.java

```
10
11 public class App
12 {
13     public static void main( String[] args )
14     {
15         EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
16         EntityManager em = emf.createEntityManager();
17         EntityTransaction et = em.getTransaction();
18
19         try
20         {
21             et.begin();
22
23             String native_sql_query = "UPDATE std_details SET std_marks = :stdNewMarks WHERE std_id = :stdId";
24             Query query = em.createNativeQuery(native_sql_query, Student.class);
25             query.setParameter("stdNewMarks", 55.55f);
26             query.setParameter("stdId", 4);
27
28             int count = query.executeUpdate();
29             if(count > 0)
30             {
31                 System.out.println("success");
32                 et.commit();
33             }
```

Writable Smart Insert 25 : 53 : 851

ENG

eclipse-workspace-morning - JpaNativeSqlQuery4Delete/src/main/java/in/sp/main/App.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Student.java App.java

```
1  package in.sp.main;
2
3  import javax.persistence.EntityManager;
10
11 public class App
12 {
13     public static void main( String[] args )
14     {
15         EntityManagerFactory emf = Persistence.createEntityManagerFactory("main-persistence-unit");
16         EntityManager em = emf.createEntityManager();
17         EntityTransaction et = em.getTransaction();
18
19         try
20         {
21             et.begin();
22
23             String native_sql_query = "DELETE FROM std_details WHERE std_id = :stdId";
24             Query query = em.createNativeQuery(native_sql_query, Student.class);
25             query.setParameter("stdId", 4);
26
27             int count = query.executeUpdate();
28             if(count > 0)
29             {
30                 System.out.println("success");
```

Writable | Smart Insert | 25 : 42 : 816

ENG

=> Native SQL Queries in JPA :-

-> "Native SQL Queries" refers to the SQL query that is specific to a particular database like MySQL, Oracle etc

-> Native SQL queries are specific to the database and thus they are database-dependent (But JPQL are database-independent query langauge)

-> Native SQl queries are very flexible which allows us to write complex queires that may not easily expressible in JPQL

-> NOTE :

= In case of native SQL queries, we use "Query" interface rather than "TypedQuery" interface because native SQL queries dont return managed JPA entities and therefore there is no entity type to specify as the result type

=> Spring with JPA :-
    -> Spring with JPA combines the power of the Spring Framework with the JPA to build robus and maintanable java application
    -> It simplifies configurations, enhances transaction management and reduces the boilerplate code for managing database interaction

    -> Key components for Spring with JPA :-
        1. LocalContainerEntityManagerFactoryBean
        2. JpaTransactionManager
        3. @PersistenceContext

=> LocalContainerEntityManagerFactoryBean :-
    -> It is a class which is responsible for creating and confguring the EntityManagerFactory, which is the central interface for working with JPA
    -> It allows for easy setup of JPA providers like hibernate, eclipselink etc

=> JpaTransactionManager :-
    -> It is a class which is used for managing transactions in spring-managed JPA environment
    -> It coordinates transactions across one or more EntityManager instances

=> JpaTransactionManager :-
    -> It is a class which is used for managing transactions in spring-managed JPA environment
    -> It coordinates transactions across one or more EntityManager instances

=> @PersistenceContext :-
    -> It is a JPA-specific annotation used in the JPA to inject EntityManager into spring-managed bean
    or components

```
=> Steps to create Spring with JPA program :-
    1. Create an entity class in "in.sp.entity" package
    2. Create Spring configuration file (xml) in "in.sp.resources" package
        = DataSource -> DriverManagerDataSource
        = LocalContainerEntityManagerFactoryBean
            - dataSource
            - package to scan the entity class
            - jpa vendor properties
    3. Create DAO interface in "in.sp.dao" package
    4. Create DAO implemented class in "in.sp.dao" package
    5. Create main class and execute the application
```

```
<bean class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean" id="lcEntityManagerFactoryl
    <property name="dataSource" ref="dmDataSource" />

    <property name="packagesToScan">
        <list>
            <value>in.sp.entity</value>
        </list>
    </property>

    <property name="jpaVendorAdapter">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
    </property>

    <property name="jpaProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
</bean>
```

the following snippet references the correct schema so that the tags in the `tx` namespace are available to you:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xmlns:tx="http://www.springframework.org/schema/tx" xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd"> <!-- bean definitions here -->

</beans>
```

```
<context:annotation-config />
<tx:annotation-driven/>
```

```
        Student std = new Student();
        std.setId(4);
        std.setName("ddd");
        std.setRollno(104);
        std.setMarks(82.74f);
        stdDao.insertStudent(std);
}
```

```
<bean class="org.springframework.orm.jpa.JpaTransactionManager" id="transactionManager">
    <property name="entityManagerFactory" ref="lcEntityManagerFactoryBean" />
</bean>
```

```
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http://ww
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <context:annotation-config />
    <tx:annotation-driven/>

```

```
51      @Override
52      public void updateStudent(int id, float marks)
53      {
54          try
55          {
56              Student std = entityManager.find(Student.class, id);
57              std.setMarks(marks);
58
59              System.out.println("Updation success");
60          }
61          catch(Exception e)
62          {
63              System.out.println("Updation failed");
64              e.printStackTrace();
65          }
```

```
stdDao.updateStudent(4, 75.66f);
```

```java
@Bean
public DriverManagerDataSource dmDataSource()
{
    DriverManagerDataSource dataSource = new DriverManagerDataSource();

    dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql://localhost:3306/spring_jpa_db");
    dataSource.setUsername("root");
    dataSource.setPassword("root");

    return dataSource;
}
```

```
9
0 @Configuration
1 @EnableTransactionManagement
```

```
29      }
30
31      @Bean
32      public LocalContainerEntityManagerFactoryBean lcEntityManagerFactoryBean()
33      {
34          LocalContainerEntityManagerFactoryBean lcEntityManagerFactoryBean = new LocalContainerEntityManagerFacto
35
36          lcEntityManagerFactoryBean.setDataSource(dmDataSource());
37
38          lcEntityManagerFactoryBean.setPackagesToScan("in.sp.entity");
39
40          lcEntityManagerFactoryBean.setJpaVendorAdapter(new HibernateJpaVendorAdapter());
41
42  //      Properties properties = new Properties();
43  //      properties.setProperty("hibernate.hbm2ddl.auto", "update");
44  //      properties.setProperty("hibernate.show_sql", "true");
45  //      lcEntityManagerFactoryBean.setJpaProperties(properties);
46
47
48
49          return lcEntityManagerFactoryBean;
50      }
51
52      public Properties huber
```

eclipse-workspace-morning - SpringWithJpa3Crud/src/main/java/in/sp/resources/SpringConfigFile.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

applicationContext.xml | *SpringConfigFile.java

```
36          lcEntityManagerFactoryBean.setDataSource(dmDataSource());
37
38          lcEntityManagerFactoryBean.setPackagesToScan("in.sp.entity");
39
40          lcEntityManagerFactoryBean.setJpaVendorAdapter(new HibernateJpaVendorAdapter());
41
42  //      Properties properties = new Properties();
43  //      properties.setProperty("hibernate.hbm2ddl.auto", "update");
44  //      properties.setProperty("hibernate.show_sql", "true");
45  //      lcEntityManagerFactoryBean.setJpaProperties(properties);
46
47          lcEntityManagerFactoryBean.setJpaProperties(hibernateProperties());
48
49          return lcEntityManagerFactoryBean;
50      }
51
52      public Properties hibernateProperties()
53      {
54          Properties hibernateProperties = new Properties();
55          hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "update");
56          hibernateProperties.setProperty("hibernate.show_sql", "true");
57
58          return hibernateProperties;
59      }
```

Properties jpaProperties

Writable | Smart Insert | 47 : 74 : 1687

ENG

```java
@Bean
public JpaTransactionManager transactionManager(EntityManagerFactory emf)
{
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(emf);

    return transactionManager;
}
```

```
@Bean
public StudentDaoImpl stdDaoImpl()
{
    return new StudentDaoImpl();
}
```