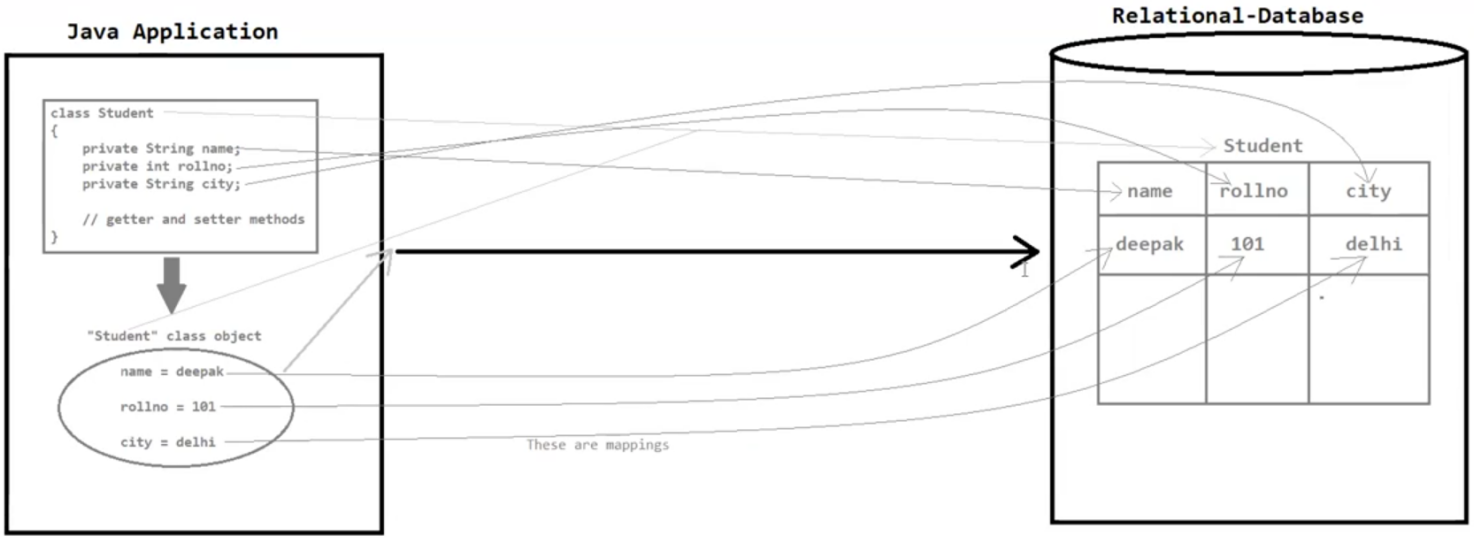
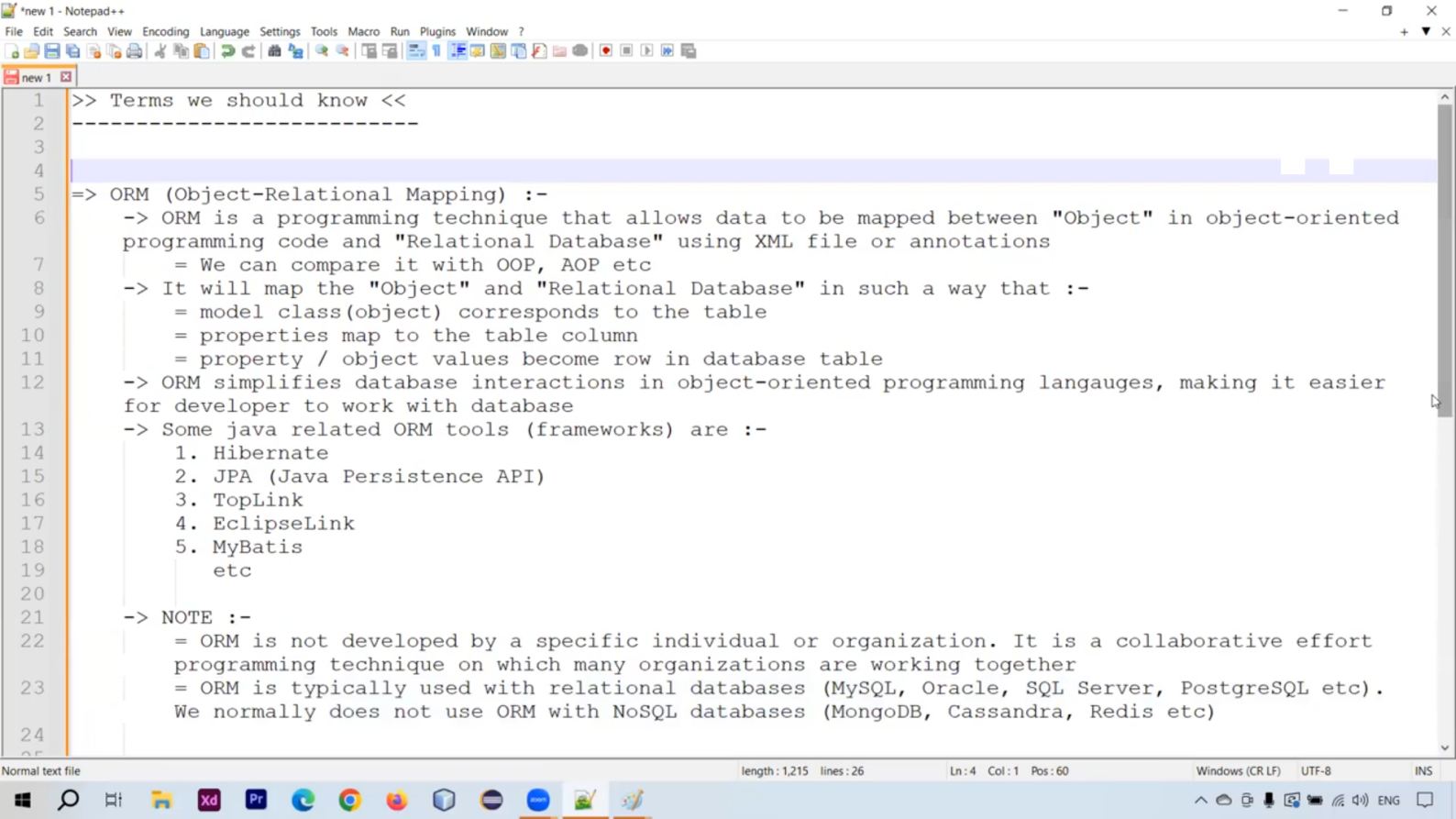

Java Application
class Student
{
private String name;
private int rollno;
private String city;
// getter and setter methods
}
"Student" class object
name = deepak
rollno = 101
city = delhi
These are mappings
Relational-Database
Student
name
rollno
city
deepak
101
delhi


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
>> Terms we should know <<

=> ORM (Object-Relational Mapping) :-
-> ORM is a programming technique that allows data to be mapped between "Object" in object-oriented programming code and "Relational Database" using XML file or annotations
= We can compare it with OOP, AOP etc
-> It will map the "Object" and "Relational Database" in such a way that :-
= model class(object) corresponds to the table
= properties map to the table column
= property / object values become row in database table
-> ORM simplifies database interactions in object-oriented programming langauges, making it easier for developer to work with database
-> Some java related ORM tools (frameworks) are :-
1. Hibernate
2. JPA (Java Persistence API)
3. TopLink
4. EclipseLink
5. MyBatis
etc
-> NOTE :-
= ORM is not developed by a specific individual or organization. It is a collaborative effort programming technique on which many organizations are working together
= ORM is typically used with relational databases (MySQL, Oracle, SQL Server, PostgreSQL etc). We normally does not use ORM with NoSQL databases (MongoDB, Cassandra, Redis etc)
Normal text file
length : 1,215 lines : 26
Ln : 4 Col : 1 Pos : 60
Windows (CR LF) UTF-8 INS
ENG

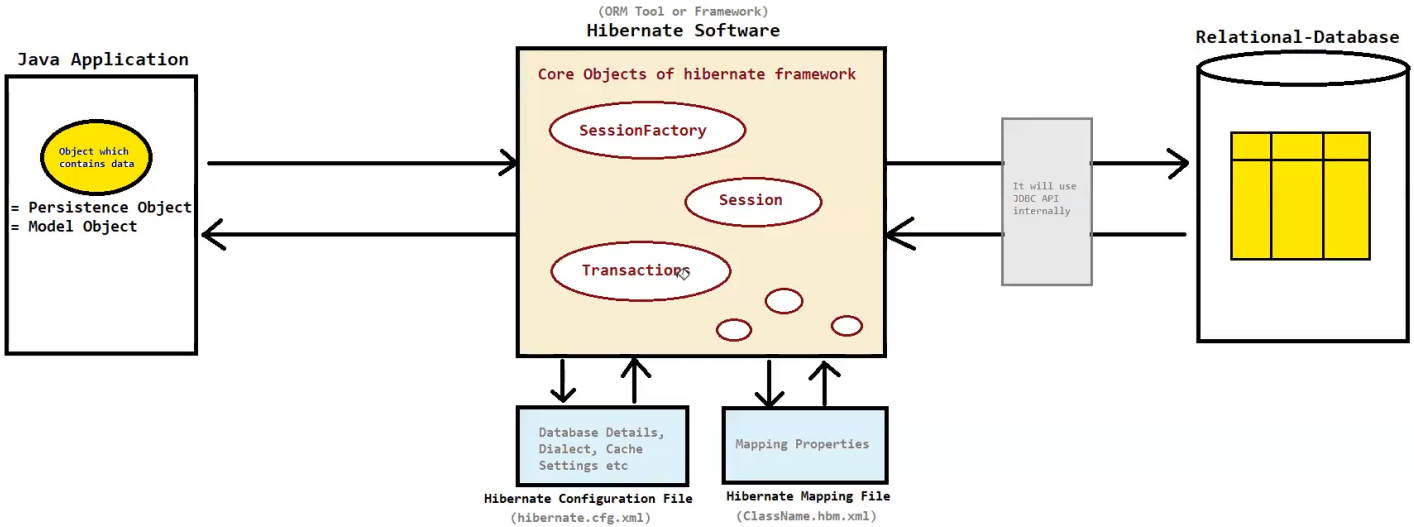
=> Data Persistency :-
    -> Data means information and persistency means permanent
    -> Data Persistency is the ability to store the data permanently in a way that it is not lost when the program or system is no longer running
    -> To achieve data-persistency wrt java we have following techniques or technologies :-
        1. Serialization and Deserialization
        2. JDBC
        3. ORM
            = Hibernate
            = JPA
            = TopLink
            = EclipseLink
            = MyBatis
              etc
        etc

=> What is hibernate :-
-> Hibernate is an open-source ORM tool or framework for Java
-> NOTE : Hibernate is not a core or official component of Java EE (Java Enterprise Edition) platform
-> Hibernate was developed by Gavin King while working in JBoss. The project was later acquired by Red Hat company.
-> Hibernate first version was released in 2001

-> It simplifies database programming in java applications by mapping java objects to database tables, making it easier to work with relational database

-> Features of Hibernate :-
1. Database independent :-
   = It supports various relational databases
2. Automatic Table Generation :-
   = It can generate databse schema from java classes
3. Query Language :-
   = Hibernate provides its own query language i.e. HQL (Hibernate Query Language). It is database-independent which allows us to write queries using a syntax that hibernate translates into database-specific SQL queries
4. Caching :-
   = It provides very good cache mechanism for better performance
5. Transaction Management :-
   = It supports ACID transaction properties
6. Connection Pooling :-
   = It manages and optimizes database connections
7. Integration :-
   = It can be easily integrated with Java EE applications and Java frameworks (Spring etc)
8.

Hibernate Architecture



7. Integration :-
= It can be easily integrated with Java EE applications and Java frameworks (Spring etc)
8. Community Support :-
= It provides active open-source community for updates and support
etc

-> Hibernate Architecture :-
= Diagram

-> Components of Hibernate Architecture :
1. Application Layer
2. Persistence Object
3. Hibernate Configuration File
4. Hibernate Mapping File
5. Hibernate Software
6. Relational Database

==========

-> Components of Hibernate Architecture :

1. Application Layer
   = In this layer persistence object is created
   = In this layer we provide business logic and persistence operations
   = In this layer hibernate software will be activated
2. Persistence Object
   = It is java object that represents the data and is designed to be persistent (means it can stored or retrieve or modify the data from relational database)
3. Hibernate Configuration File
   = It is typically an XML or Java file
   = It contains settings and properties for hibernate i.e. databse connection information, dialect, cache settings etc
4. Hibernate Mapping File
   = A hibernate mapping file (usually in XML format) defines how java object are to be mapped with database tables
5. Hibernate Software
   = When the client activates hibernate, the framework retrieves configuration details from hibernate configuration file and establish the connection with database
   = When the client initiates persistence operation, hibernate generates the appropriate database-specific SQL query and executes it.
6. Relational Database
   = The persistence object will be mapped with relation-database and new table with data will be created

==========

```
=> Steps to create Hibernate First Program :-
    1. Download & Install any IDE (Eclipse)
    2. Create "Simple Java Project" or "Maven Project"
    3. Add "hibernate jars" (for java project) or "provide dependencies" (for maven project)
    4. Create "POJO class"
    5. Create "Hibernate Mapping File"
    6. Create "Hibernate Configuration File"
    7. Create main class to execute hibernate application with following steps :-

        7.1 Create "Configuration" object :-
            -> The Configuration class is used to configure and manage hibernate settings
            -> How to get Configuration object :-
                ** Configuration cfg = new Configuration();
            -> Methods of Configuration class :-
                ** configure()  // It is used to load the hibernate configuration details from an
                external file

        7.2 Create "SessionFactory" object :-
            -> SessionFactory is an interface that represents a factory for creating "Session" objects
            -> It is used to load the configuration details from configuration file, setting up
            connection pool, manage database connections etc
            -> How to get SessionFactory object :-
                ** SessionFactory sessionFactory = cfg.buildSessionFactory();
```

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

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

<property name="show_sql">true</property>
<property name="format_sql">true</property>
```

7.3 Create "Session" object :-

-> Session is an interface that represents a single unit of work

-> It is used to perform database operations

-> How to get Session object :-

** Session session = sessionFactory.openSession();

-> Database operations that we can perform using Session object are :-

a. Insert operation
   = save()
   = persist()

b. Update operation
   = update()
   = saveOrUpdate()

c. Delete operation
   = delete()

d. Retrieve operation
   = get()
   = load()

7.4 Create "Transaction" object :-

-> Transaction is an interface that represents database transaction

-> It is used to control and manage transactions

-> How to get Transaction object :-

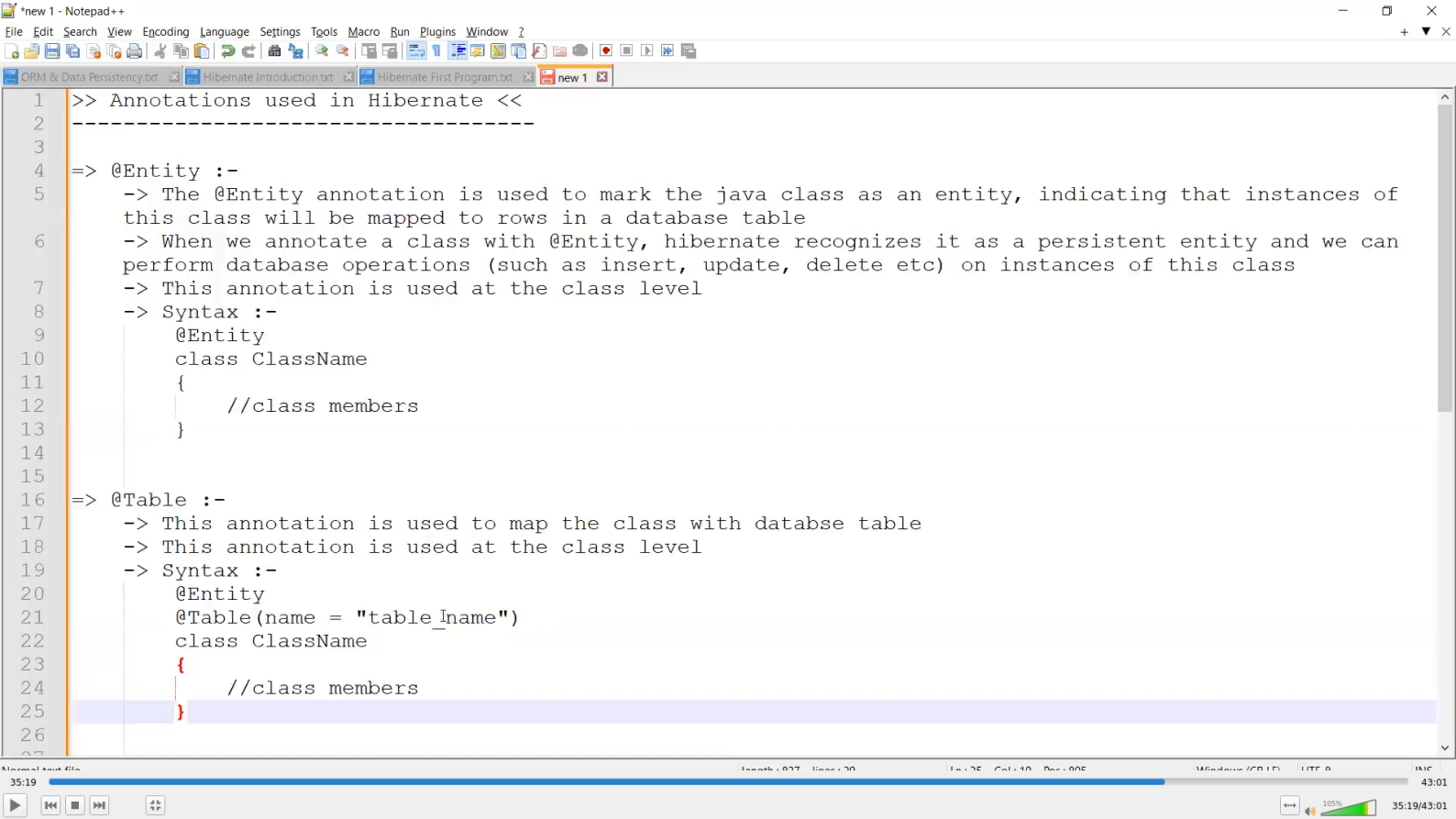
** Transaction transaction = session.getTransaction()
transaction.begin();

** Transaction transaction = session.beginTransaction();

-> NOTE : In JDBC, AutoCommit default value is "true" and we have to set it to "false" by "con.setAutoCommit(false);" statement. But in hibernate AutoCommit default value is "false"

```
<id name="stdid" column="std_id">
    <generator class="identity" />
</id>
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "emp_id")
private int empid;
```


*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
new 1
>> Annotations used in Hibernate <<
=> @Entity :-
-> The @Entity annotation is used to mark the java class as an entity, indicating that instances of this class will be mapped to rows in a database table
-> When we annotate a class with @Entity, hibernate recognizes it as a persistent entity and we can perform database operations (such as insert, update, delete etc) on instances of this class
-> This annotation is used at the class level
-> Syntax :-
@Entity
class ClassName
{
//class members
}
=> @Table :-
-> This annotation is used to map the class with databse table
-> This annotation is used at the class level
-> Syntax :-
@Entity
@Table(name = "table_name")
class ClassName
{
//class members
}
35:19
43:01
35:19/43:01

=> @Id :-
-> @Id annotation is used to specify the primary key of an entity class. It marks a field as the unique identifier for instances of that entity
-> In relational database, a primary key uniquely identifies wach row in a table and hibernate uses this annotation to map the java objects primary key to the database primary key
-> This annotation is used with field (attribute) within the entity class
-> Syntax :-

```
@Entity
@Table(name = "table_name")
class ClassName
{
    @Id
    private int empid;
}
```

```
=> @Column :-
    -> @Column is used to map the field with table column
    -> It is used with field (attribute) within the entity class
    -> Syntax :-
        @Entity
        @Table(name = "table_name")
        class ClassName
        {
            @Column(name = "column_name")
            private String propert_name;
        }
```

==========

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

=> Code to generate id :-

    -> Usig XML :-
        <id name="stdid" column="std_id">
            <generator class="identity" />
        </id>

    -> Using Annotations :-
        @GeneratedValue(strategy = GenerationType.IDENTITY)

========
```

```
=> Difference between save() and persist() method :-
    = When save() method insert the record in database, it will return primary key of the saved object
        - Serializable save(Object object);
    = When persist() method insert the record in database, it will not return any value
        - void persist(Object object);

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

=> Difference between get() and load() method :-

-> get() method :-

= The get() method is used to retrieve an object from the database by its primary key (identifier)

= If the object with the given identifier is found in the database, it will return the actual object and initializes all the properties with data from database. This is known as "eager loading" or "early loading"

= If the object is not found in the database it will return null object

-> load() method :-

= The load() method also retrieves an object by its primary key but it return a proxy object rather than the actual object from the database

= It will retrieve the values from database only when getter methods are called. This is known as "lazy loading"

= If the object is not found in the database it provides an exception i.e. "ObjectNotFoundException"

```
Employee emp = session.get(Employee.class, 102);
emp.setEmpcity("pune");

session.update(emp);
System.out.println("success");

transaction.commit();
```

=> Difference between update() and saveOrUpdate() method :-

-> update() method will update the record if the row is present in database. If row is not present in database then it will provide an exception

-> saveOrUpdate() method is the combination of save() and update() method. It will check whether the object is present in database, if it is present then it will update otherwise it will insert the values or object in database

==========

```
Employee emp = new Employee();
emp.setEmpid(104);

session.delete(emp);
System.out.println("success");

transaction.commit();
```

=> Spring with Hibernate :-

-> Spring with Hibernate combines the strength of both frameworks to create efficient, maintanable and scalable java application

-> Using hibernate with spring will simplifies the configuration, enhances transaction management etc by removing the boiler-plate code i.e. creating Configuration, SessionFactory, Session & Transaction objects

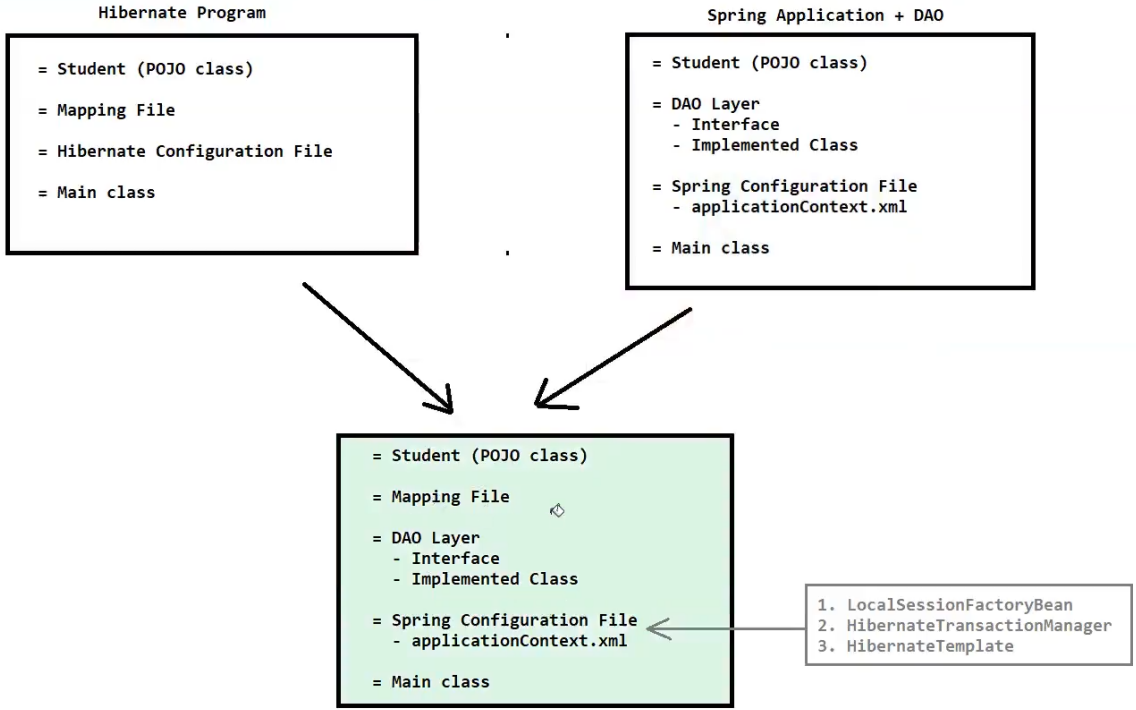
-> To achieve spring with hibernate integration, spring has provided 3 classes :-

1. LocalSessionFactoryBean
2. HibernateTransactionManager
3. HibernateTemplate

=> LocalSessionFactoryBean :-
    -> It is a class which simplifies the process of configuring hibernate within spring application
    -> Use :-
        = Configuration :-
            - It allows us to define and configure hibernate properties such as database connection details, dialect, mapping resources etc in spring context file
        = SessionFactory creation :-
            - It is responsible for creating and initializing the hibernate SessionFactory based on the provided configurations
            - The SessionFactory is a component in hibernate that manaes the lifecycle of database connections and provides a central point for creating and managing database transaction. By using LocalSessionFactoryBean, we can obtain a fully configured SessionFactory instance to work with hibernate in our spring application

```
=> HibernateTemplate :-
    -> It is a class that provides a simplified and consistent way to interact with hibernate ORM
    framework
    -> Use :-
        = Simplified Data Access :-
            - It is an abstraction layer over hibernate "Session" which provides the methods for CRUD
            operations i.e. save(), update(), delete(), get() etc
            - It abstracts away the need to manually open and close hibernate sessions
        = Exception Translations :-
            - It provides exception translation i.e. converting hibernate-specific exceptions into
            Spring's DataAccessException hierarchy

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


Hibernate Program
= Student (POJO class)
= Mapping File
= Hibernate Configuration File
= Main class
Spring Application + DAO
= Student (POJO class)
= DAO Layer
- Interface
- Implemented Class
= Spring Configuration File
- applicationContext.xml
= Main class
= Student (POJO class)
= Mapping File
= DAO Layer
- Interface
- Implemented Class
= Spring Configuration File
- applicationContext.xml
= Main class
1. LocalSessionFactoryBean
2. HibernateTransactionManager
3. HibernateTemplate

```
try
{
    hibernateTemplate.save(std);
    System.out.println("Student details added successfully");
}
catch(Exception e)
{
    System.out.println("Student details not added due to some error");
    e.printStackTrace();
}
```

```
try
{
    hibernateTemplate.update(std);
    System.out.println("Student details updated successfully");
}
catch(Exception e)
{
    System.out.println("Student details not updated due to some error");
    e.printStackTrace();
}
```

```
try
{
    hibernateTemplate.delete(std);
    System.out.println("Student details deleted successfully");
}
catch(Exception e)
{
    System.out.println("Student details not deleted due to some error");
    e.printStackTrace();
}
```

```
try
{
    Student std = (Student) hibernateTemplate.get(Student.class, id);
    System.out.println("Student details :-");
    System.out.println("Name : "+std.getName());
    System.out.println("Rollno : "+std.getRollno());
    System.out.println("City : "+std.getCity());

}
catch(Exception e)
{
    System.out.println("Student details not retrieved due to some error");
    e.printStackTrace();
}
```

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

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

    return dmDataSource;
}
```

```
@Configuration
@EnableTransactionManagement
```

```java
@Bean
public LocalSessionFactoryBean mySessionFactory()
{
    LocalSessionFactoryBean mySessionFactory = new LocalSessionFactoryBean();

    mySessionFactory.setDataSource(dmDataSource());

    Properties hibernateProperties = new Properties();
    hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "update");
    hibernateProperties.setProperty("hibernate.show_sql", "true");
    hibernateProperties.setProperty("hibernate.format_sql", "true");
    mySessionFactory.setHibernateProperties(hibernateProperties);

    mySessionFactory.setMappingResources("/in/sp/resources/student.hbm.xml");

    return mySessionFactory;
}
```

```java
@Bean
public HibernateTransactionManager transactionManager()
{
    HibernateTransactionManager transactionManager = new HibernateTransactionManager();
    transactionManager.setSessionFactory(mySessionFactory().getObject());
    return transactionManager;
}
```

```java
@Bean
public HibernateTemplate myHibernateTemplate()
{
    HibernateTemplate hibernateTemplate = new HibernateTemplate();
    hibernateTemplate.setSessionFactory(mySessionFactory().getObject());
    return hibernateTemplate;
}
```

```java
@Bean
public StudentDao stdDaoImplBean()
{
    StudentDaoImpl stdDaoImpl = new StudentDaoImpl();
    stdDaoImpl.setHibernateTemplate(myHibernateTemplate());
    return stdDaoImpl;
}
```