[https://www.java4coding.com](https://www.java4coding.com/)

**What is JPA?**

JPA is a java specification.

Which is used to access ,manage and persist data b/w java object and relation database

JPA can be seen as a bridge between object-oriented domain models and relational database systems.

**What is hibernate?**

A Hibernate is a Java framework which is used to store the Java objects in the relational database system. It is an open-source, lightweight, ORM (Object Relational Mapping) tool.

**Introduction to JPA and ORM**

* Any storage system which stores the data permanently is called as persistence store. Ex: Files, Database’s etc…
* Insert Update, delete, and select operations on the DB table are called persistence operations. The logic used for this purpose is called as persistence logic.
* The technologies which are used to develop persistence logic are called as persistence technologies. Ex: JDBC, Hibernate, JPA, IBATIS.

**Object Relational Mapping (ORM)**

>> Mapping Java Class with the Database table, member variables with the columns and making that java class objects representing rows of DB tables by having synchronization between them is called Object Relational Mapping(ORM).

>> Synchronization between objects and rows of tables is nothing but, modifications done in java objects reflect in rows of tables and vice versa.

>> ORM technologies like JPA, hibernate, TopLink, Ibatis etc., are responsible for this Synchronization and to develop objects based ORM persistence logic.

>> All ORM technologies internally uses JDBC code to perform persistence operations on DB table rows. Hence these ORM technologies are providing abstraction layer to programmers on JDBC. So ORM technologies are also called as framework technologies.

**Can JPA be used alone?**

No, JPA cannot be used alone. JPA is a specification from Oracle/Sun, and we have to use the implementation from any of the provider. Hibernate, OpenJPA and many providers have implemented this JPA specification. In our tutorials we used Hibernate implementation.

Hibernate has implemented JPA, and along with this, hibernate has its own ORM implementation.

Many organizations are using plain hibernate and hibernate came before JPA. After JPA is released, hibernate has implemented this JPA.

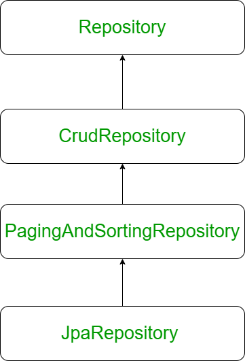
Please remember, if you are using JPA with hibernate implementation, then you should say that you are using JPA not hibernate. I have seen developers who are software architects don’t know the difference between JPA and hibernate. Now you should be proud, because you know the difference between JPA and Hibernate. JPA is a specification and Hibernate is an implementation, also remember Hibernate has its own ORM implementation, means before JPA, many developers and organizations were using hibernate own ORM implementation. So hibernate has two implementation, one is implementation for JPA and second is its own ORM implementation.

Also remember if you have learnt either hibernate or JPA, learning other is just an hour work for you.

**Difference between JPA and JDBC**

|  |  |
| --- | --- |
| **Hibernate** | **JDBC** |
| JPA is data base independent, single query will work for all ORACLE, MySQL , SQLServer etc. If we change the database, we need to change just the provider in configuration file. | In case of JDBC, query must be data base specific. . If we change the database, we need to change all the database queries. |
| JPA supports caching, two level of cache. First level and 2nd level. So we can store the data into Cache for better performance. | In case of JDBC we need to implement our java cache, and there is no built in cache support. |
| No need to create any connection pool in case of JPA. | In case of JDBC we need to create our own connection pool. |
| As JPA is set of Objects, we don't need to learn SQL language. We can treat TABLE as an Object. Only Java knowledge is need. | In case of JDBC, we need to learn SQL |
| Don't need Query tuning in case of JPA. If we use Criteria Quires then JPA automatically tunes the query and return best result with performance. | In case of JDBC we need to tune your queries |
| Productivity is good, because we don’t need to write queries | Development is slower. |
| In the xml file you can see all the relations between tables in case of JPA.Easy readability. | Using JDBC we cannot see all the relations between tables. |
| In JPA if we fetch the data from parent table, data from child table (which is related using foreign/primary key) will be fetched automatically. | JDBC Don’t have such support |

**Ques: difference b/w CrudRepository and @JPARepository?**



| **CrudRepository** | **JpaRepository** |
| --- | --- |
| It is a base interface and extends Repository Interface. | It extends PagingAndSortingRepository that extends CrudRepository. |
| It contains methods for CRUD operations. For example save(), saveAll(), findById(), findAll(), etc. | It contains the full API of CrudRepository and PagingAndSortingRepository. For example, it contains flush(), saveAndFlush(), saveAllAndFlush(), deleteInBatch(), etc along with the methods that are available in CrudRepository. |
| It doesn’t provide methods for implementing pagination and sorting | It provides all the methods for which are useful for implementing pagination. |
| It works as a marker interface. | It extends both CrudRepository and PagingAndSortingRepository. |
| To perform CRUD operations, define repository extending CrudRepository. | To perform CRUD as well as batch operations, define repository extends JpaRepository. |
| **Syntax:**  public interface CrudRepository<T, ID> extends Repository<T, ID> | **Syntax:**  public interface JpaRepository<T,ID> extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T> |

**Syntax:** **CrudRepository**

public interface CrudRepository<T, ID> extends Repository<T, ID>

public interface DepartmentRepository extends CrudRepository<Department, Long> {}

**Syntax:** JpaRepository

public interface JpaRepository<T,ID> extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>

public interface DepartmentRepository extends JpaRepository<Department, Long> {}

**Which java package is used for working with hibernate and JPA?**

* javax.persistence – JPA (JPA with any implementation)
* org.hibernate –- Hibernate (Hibernates own ORM implementation)

**Why do we need a persistence framework?**

To avoid writing Database specific SQL queries into our Java applications.

If we use JDBC, we should be familiar with relational database SQL syntax.

Java JDBC work is tedious and diverts programmer attention from programming business logic. Programmers should be careful themselves with binding query parameters, writing SQL, executing queries, scrolling the result set and retrieving the required data.

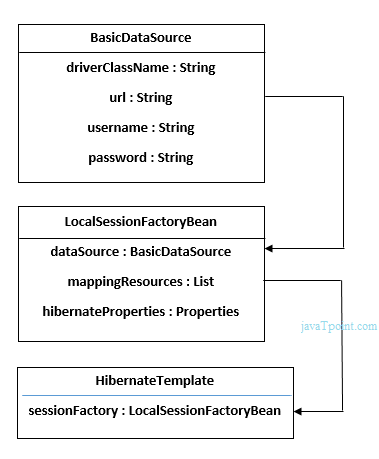
Java programs are object-oriented (or at least should be) in nature whereas relational databases are tabular in nature. This complicates the object-relational mapping and forces developers to think about problems in two different ways. A persistent framework makes coding faster and improves maintenance. The persistent framework can also help to isolate the application from database specific intricacies. Finally, a persistent framework can, in some cases, improve performance with appropriate performance enhancements and caching.

**Advatages of JPA**

1. Supports POJO-POJI model Programming
2. Light –Weight technology to develop DB independent persistence logic.
3. Allows to work with any JAVA, JEE framework softwares based applications to make them interacting with DB software.
4. When we want to do distribute transactions in JDBC we have to implement JTA API (*Java Transaction API* ). But JPA  provides built in transactions management, connection pooling.
5. Support two levels of caching (or) Buffering to reduce network round trips between client applications and Database.
6. Provides JQL (JPA Query Language) as Database independent language to perform persistence operations.
7. From Entity classes (we will read about entity classed in coming chapters) we can see all the relations between tables. Easy readability
8. Hibernate Supports automatic versioning of rows but JDBC Not.
9. Allows object level relationship in development of persistence logic, when tables are there in relationships like 1-1,1-n,n-n etc. Hibernate uses fetching strategy for retrieving associated objects if the application needs to navigate the association. You can navigate object relationships transparently. Related objects are automatically loaded as needed. For example if you load a PO (Purchase Order) and you want to access its Customer, you can simply access PO. The ORM will take care of loading the Customer data for you without any effort on your part.
10. When we are using Datasource connection, in JDBC we have to implement JNDI API Java Naming and Directory Interface *API* ). But Hibernate provides built in support for Datasource connection.

**Hibernate Tutorial (Javatpoint)**

**Hibernate Tutorial With Spring Framework**



**applicationContext.xml**

**Note:** When we use hibernate framework without Spring Framework then configuration write in hibernate.cfg.xml file.

When we are going to integrate hibernate with spring then we don’t need to write configuration in hibernate.cfg.xml. All configurations write in applicationContext.xml.

**Explain below Image:** In applicationContext file we provide all the database information in BasicDataSource object and this object is used by LocalSessionFactoryBean Class object and containing some other information such as Mapping resources and hibernate configuration. The LocalSessionFactoryBean Class Object is used in hibernate Template Class.

Example:

<?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:p="http://www.springframework.org/schema/p"      xsi:schemaLocation="http://www.springframework.org/schema/beans          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="dataSource" **class**="org.apache.commons.dbcp.BasicDataSource">

        <property name="driverClassName"  value="oracle.jdbc.driver.OracleDriver"></property>

        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"></property>

        <property name="username" value="system"></property>

        <property name="password" value="oracle"></property>

    </bean>

    <bean id="mysessionFactory"  **class**="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

        <property name="dataSource" ref="dataSource"></property>

                  <property name="mappingResources">

        <list>

        <value>employee.hbm.xml</value>

        </list>

  </property>

       <property name="hibernateProperties">

            <props>

                <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>

                 <prop key="hibernate.hbm2ddl.auto">update</prop>

                <prop key="hibernate.show\_sql">**true**</prop>

            </props>

</property>

    </bean>

    <bean id="template" **class**="org.springframework.orm.hibernate3.HibernateTemplate">

    <property name="sessionFactory" ref="mysessionFactory"></property>

    </bean>

    <bean id="d" **class**="com.javatpoint.EmployeeDao">

    <property name="template" ref="template"></property>

    </bean>

    </beans>

============================================================================

You can enable many hibernate properties like automatic table creation by hbm2ddl.auto etc. in applicationContext.xml file. Let's see the code:

<property name="hibernateProperties">

            <props>

                <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>

                <prop key="hibernate.hbm2ddl.auto">update</prop>

                <prop key="hibernate.show\_sql">**true**</prop>

            </props>

If you write this code, you don't need to create table because table will be created automatically.

**Basic Hibernate Jar’s:**

**======================================================================================**

**Interview Questions**

### 3) Explain hibernate architecture?

This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects such as session factory, session, transaction etc. along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

## Elements of Hibernate Architecture

SessionFactory, Session, Transaction, TransactionFactory,Connection Provider.

### 4) What are the core interfaces of Hibernate?

The core interfaces of Hibernate framework are:

* Configuration
* SessionFactory
* Session
* Query
* Criteria
* Transaction

#### SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

#### Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

#### Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

#### ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

#### TransactionFactory

It is a factory of Transaction. It is optional.

|  |
| --- |
| 5) Mention some of the advantages of using ORM over JDBC. ORM has the following advantages over JDBC:   * Application development is fast. * Management of transaction. * Generates key automatically. * Details of SQL queries are hidden.  6) Define criteria in terms of Hibernate. The objects of criteria are used for the creation and execution of the object-oriented criteria queries. |

### 7) List some of the databases supported by Hibernate.eeeeee

Some of the databases supported by Hibernate are:

* DB2
* MySQL
* Oracle
* Sybase SQL Server
* Informix Dynamic Server
* HSQL
* PostgreSQL
* FrontBase

### 8) List the key components of Hibernate.

Key components of Hibernate are:

* Configuration
* Session
* SessionFactory
* Criteria
* Query
* Transaction

### 9) Mention two components of Hibernate configuration object.

* Database Connection
* Class Mapping Setup

### 10) How is SQL query created in Hibernate?

The SQL query is created with the help of the following syntax:

Session.createSQLQuery

### 11) What does HQL stand for?

Hibernate Query Language

### 12) How is HQL query created?

The HQL query is created with the help of the following syntax:

Session.createQuery()

### 13) How can we add criteria to a SQL query?

A criterion is added to a SQL query by using the Session.createCriteria.

### 14) Define persistent classes.

Classes whose objects are stored in a database table are called as persistent classes.

### 15) What is SessionFactory?

SessionFactory provides the instance of Session. It is a factory of Session. It holds the data of second level cache that is not enabled by default.

### 17) What is Session?

* It maintains a connection between the hibernate application and database.
* It provides methods to store, update, delete or fetch data from the database such as persist(), update(), delete(), load(), get() etc.
* It is a factory of Query, Criteria and Transaction i.e. it provides factory methods to return these instances.

### 16) Is SessionFactory a thread-safe object?

Yes, SessionFactory is a thread-safe object, many threads cannot access it simultaneously.

### 18) Is Session a thread-safe object?

No, Session is not a thread-safe object, many threads can access it simultaneously. In other words, you can share it between threads.

### 19) What is the difference between session.save() and session.persist() method?

|  |  |  |
| --- | --- | --- |
| **No.** | **save()** | **persist()** |
| 1) | returns the identifier (Serializable) of the instance. | Return nothing because its return type is void. |
| 2) | Syn: public Serializable save(Object o) | Syn: public void persist(Object o) |

### 20) What is the difference between get and load method?

The differences between get() and load() methods are given below.

|  |  |  |
| --- | --- | --- |
| **No.** | **get()** | **load()** |
| 1) | Returns **null** if an object is not found. | Throws **ObjectNotFoundException** if an object is not found. |
| 2) | get() method always **hit the database**. | load() method **doesn't hit** the database. |
| 3) | It returns the real object, not the proxy. | It returns **proxy object.** |
| 4) | It should be used if **you are not sure** about the existence of instance. | It should be used if **you are sure** that instance exists. |

### 21) What is the difference between update and merge method?

The differences between update() and merge() methods are given below.

|  |  |  |
| --- | --- | --- |
| **No.** | **The update() method** | **merge() method** |
| 1) | Update means to edit something. | Merge means to combine something. |
| 2) | update() should be used if the session doesn't contain an already persistent state with the same id. It means an update should be used inside the session only. After closing the session, it will throw the error. | merge() should be used if you don't know the state of the session, means you want to make the modification at any time. |

**SessionFactory factory = cfg.buildSessionFactory();**

**Session session1 = factory.openSession();**

**Employee e1 = (Employee) session1.get(Employee.class, Integer.valueOf(101));//passing id of employee**

**session1.close();**

**e1.setSalary(70000);**

**Session session2 = factory.openSession();**

**Employee e2 = (Employee) session1.get(Employee.class, Integer.valueOf(101));//passing same id**

**Transaction tx=session2.beginTransaction();**

**session2.merge(e1);**

**tx.commit();**

**session2.close();**

After closing session1, e1 is in detached state. It will not be in the session1 cache. So if you call update() method, it will throw an error.

Then, we opened another session and loaded the same Employee instance. If we call merge in session2, changes of e1 will be merged in e2.

### 22) What are the states of the object in hibernate?

There are 3 states of the object (instance) in hibernate.

1. **Transient**: The object is in a transient state if it is just created but has no primary key (identifier) and not associated with a session.
2. **Persistent**: The object is in a persistent state if a session is open, and you just saved the instance in the database or retrieved the instance from the database.
3. **Detached**: The object is in a detached state if a session is closed. After detached state, the object comes to persistent state if you call lock() or update() method.

### 23) What are the inheritance mapping strategies?

We can map the inheritance hierarchy classes with the table of the database. There are three inheritance mapping strategies defined in the hibernate:

* Table Per Heirarchy
* Table Per Concrete Class
* Table Per SubClasses

#### Table Per Hierarchy

In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But nullable values are stored in the table .

**Table Per Concrete class**

In case of table per concrete class, tables are created as per class. But duplicate column is added in subclass tables.

#### Table Per Subclass

#### In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.

### 25) What is automatic dirty checking in hibernate?

The automatic dirty checking feature of Hibernate, calls update statement automatically on the objects that are modified in a transaction.

Example:

**SessionFactory factory = cfg.buildSessionFactory();**

**Session session1 = factory.openSession();**

**Transaction tx = session1.beginTransaction();**

**Employee e1 = (Employee) session1.get(Employee.class, Integer.valueOf(101));**

**e1.setSalary(70000);**

**tx.commit();**

**session1.close();**

Here, after getting employee instance e1 and we are changing the state of e1.

After changing the state, we are committing the transaction. In such a case, the state will be updated automatically. This is known as dirty checking in hibernate.

### 26) How many types of association mapping are possible in hibernate?

There can be 4 types of association mapping in hibernate.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

### 27) Is it possible to perform collection mapping with One-to-One and Many-to-One?

No, collection mapping can only be performed with One-to-Many and Many-to-Many.

### 28) What is lazy loading in hibernate?

* Lazy loading in hibernate improves the performance.
* It loads the child objects on demand.
* Since Hibernate 3, lazy loading is enabled by default, and you don't need to do lazy="true". It means not to load the child objects when the parent is loaded.

### 29) What is HQL (Hibernate Query Language)?

Hibernate Query Language is known as an object-oriented query language. It is like a structured query language (SQL).

The main advantage of HQL over SQL is:

1. You don't need to learn SQL
2. Database independent
3. Simple to write a query

### 30) What is the difference between first level cache and second level cache?

|  |  |  |
| --- | --- | --- |
| **No.** | **First Level Cache** | **Second Level Cache** |
| 1) | First Level Cache is **associated with Session**. | Second Level Cache is associated with **SessionFactory**. |
| 2) | It is **enabled** by default. | It is **not enabled** by default. |

**Describe Hibernate Persistent Class**

@OrderColumn is used to store list index. As list is ordered collection, so to maintain the list index there can be a column in database and while inserting the data, list index will be stored in that column. It is used only case of List.For this the property in entity should be annotated with

@OrderColumn (name="column-name")

**Note: List and Map are index based collection, so an extra column will be created in the table for indexing.**

**CascadeType :** If we wanted to save the mapped entity whenever relationship owner entity got saved. To enable this we had use “**CascadeType**” attribute.

## JPA Cascade Types

The cascade types supported by the Java Persistence Architecture are as below:

1. **CascadeType.PERSIST** : cascade type presist means that save() or persist() operations cascade to related entities.
2. **CascadeType.MERGE** : cascade type merge means that related entities are merged when the owning entity is merged.
3. **CascadeType.REFRESH** : cascade type refresh does the same thing for the refresh() operation.
4. **CascadeType.REMOVE** : cascade type remove removes all related entities association with this setting when the owning entity is deleted.
5. **CascadeType.DETACH** : cascade type detach detaches all related entities if a “manual detach” occurs.
6. **CascadeType.ALL** : cascade type all is shorthand for all of the above cascade operations.

There is no **default cascade type in JPA**. By default no operations are cascaded.

**Example 1:**

@Entity

@Table(name="q5991")

public class Question {

@Id

@GeneratedValue(strategy=GenerationType.TABLE)

private int id;

private String qname;

@OneToMany(cascade = CascadeType.ALL)

@JoinColumn(name="qid")

@OrderColumn(name="type") //<index column=”type”/>

private List<Answer> answers;

//setter getter

}

**Example2**

@Entity

@Table(name = "Employee")

public class EmployeeEntity implements Serializable

{

    private static final long serialVersionUID = -1798070786993154676L;

    @Id

    @Column(name = "ID", unique = true, nullable = false)

    private Integer  employeeId;

    @Column(name = "FIRST\_NAME", unique = false, nullable = false, length = 100)

    private String            firstName;

    @Column(name = "LAST\_NAME", unique = false, nullable = false, length = 100)

    private String lastName;

    @OneToMany(cascade=CascadeType.ALL, fetch = FetchType.LAZY)

    @JoinColumn(name="EMPLOYEE\_ID")

    private Set<AccountEntity> accounts;

    //Getters and Setters Ommited

}

**Annotation in Hebernate**

**Q:How to create composite key in hibernate?**

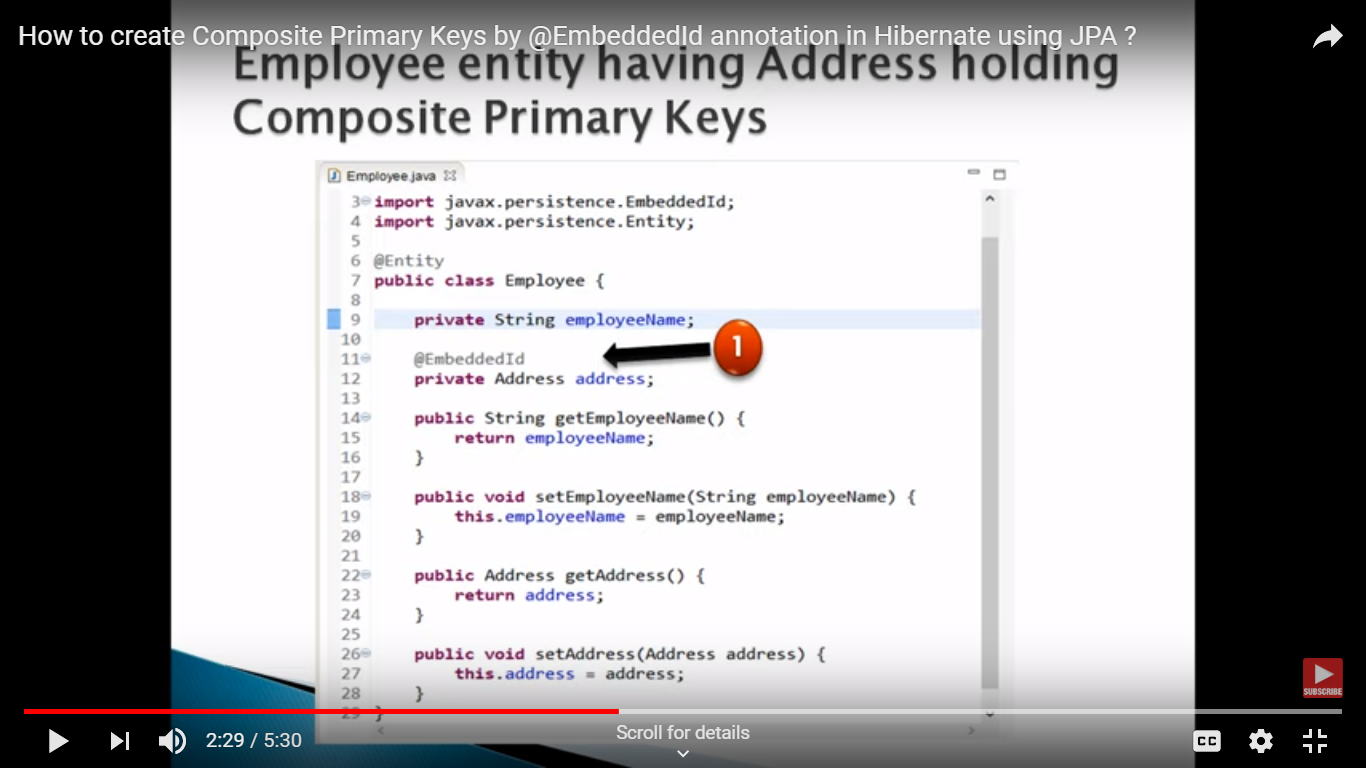
**Composite primary key using @EmblededId**

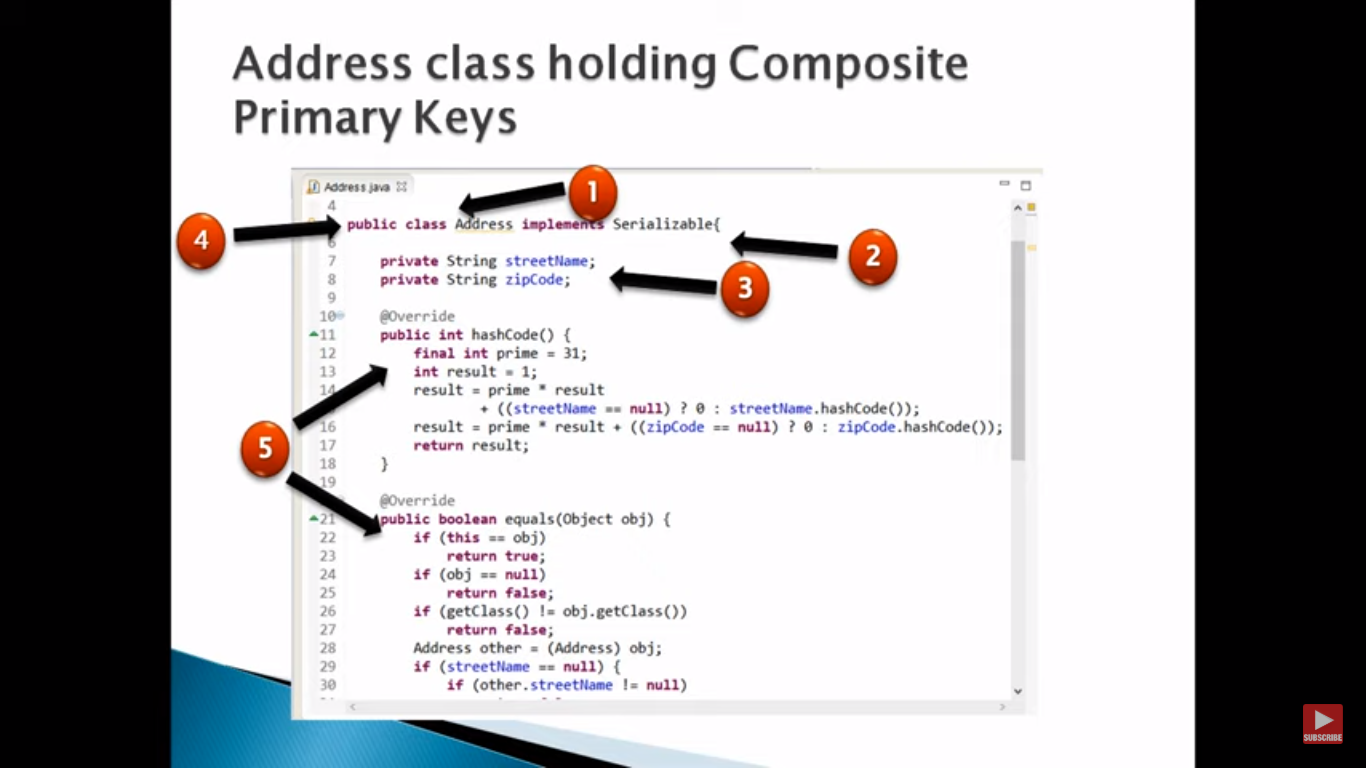
**composite key:**

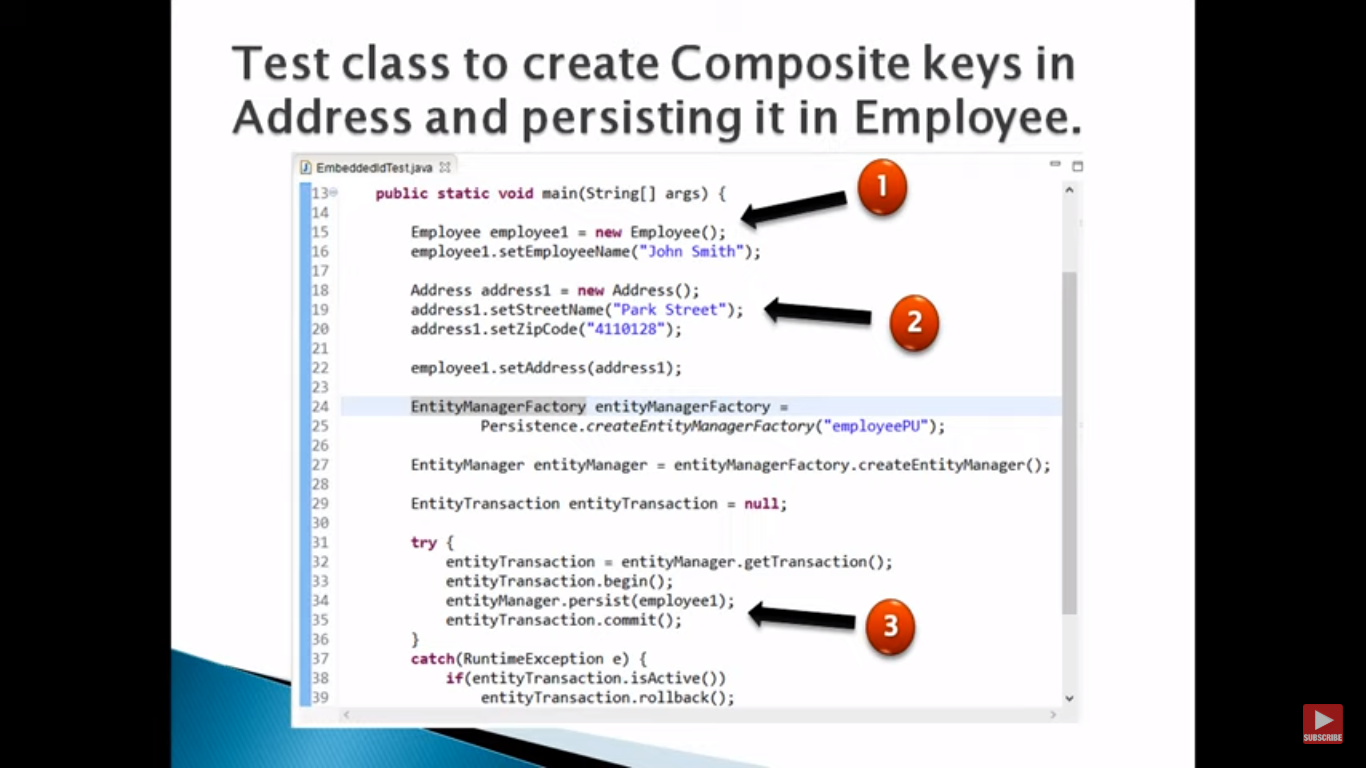
* composite primary key in the database is the group of column whose value together make unique value.
* If unique key is contained in single column than @Id is used and when the primary key consists of multiple column, these columns need to be group into a different object.
* To group multiple column in a class, some rules have to be followed.

**Rules to create class with composite key**

* class must be a public class.
* must have a default constructor.
* Must implement serialize interface.
* Class not require primary key its own.
* Class must have equals and hashcode method.

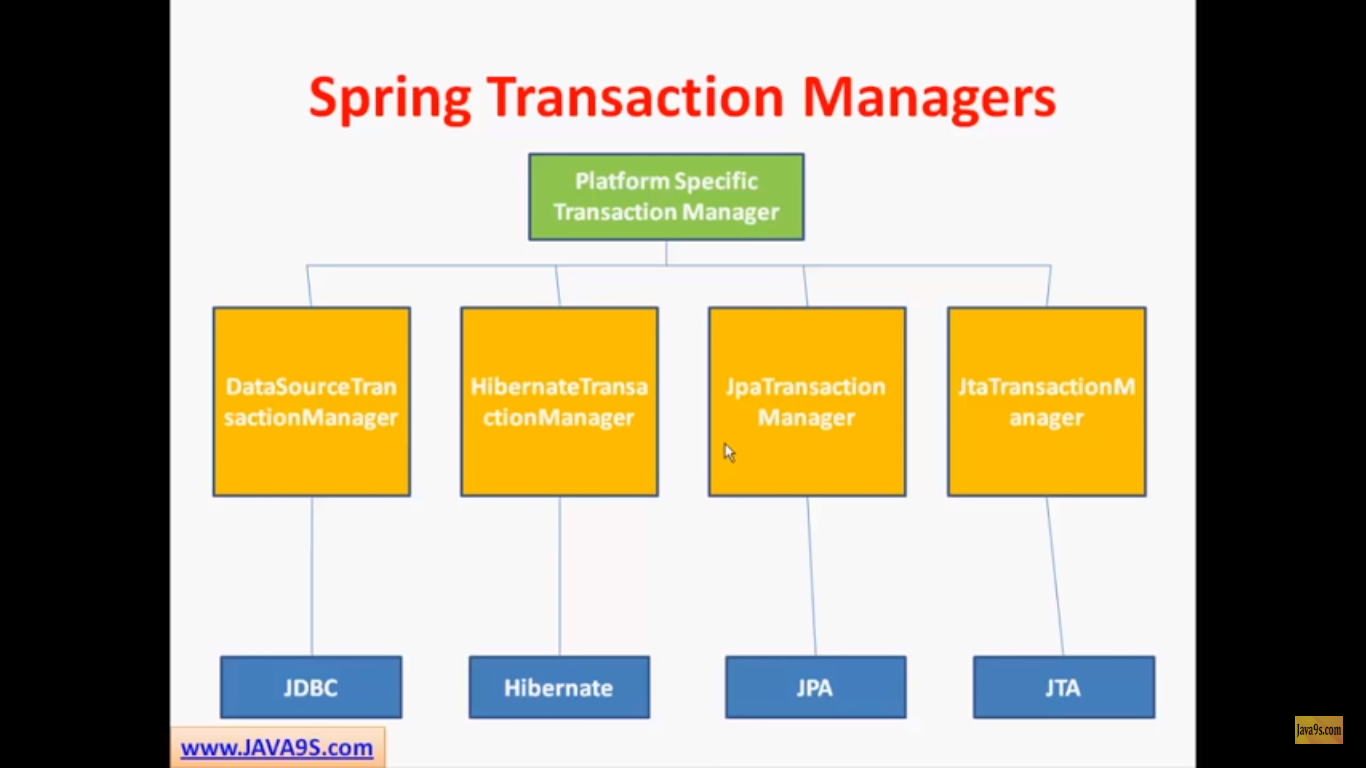


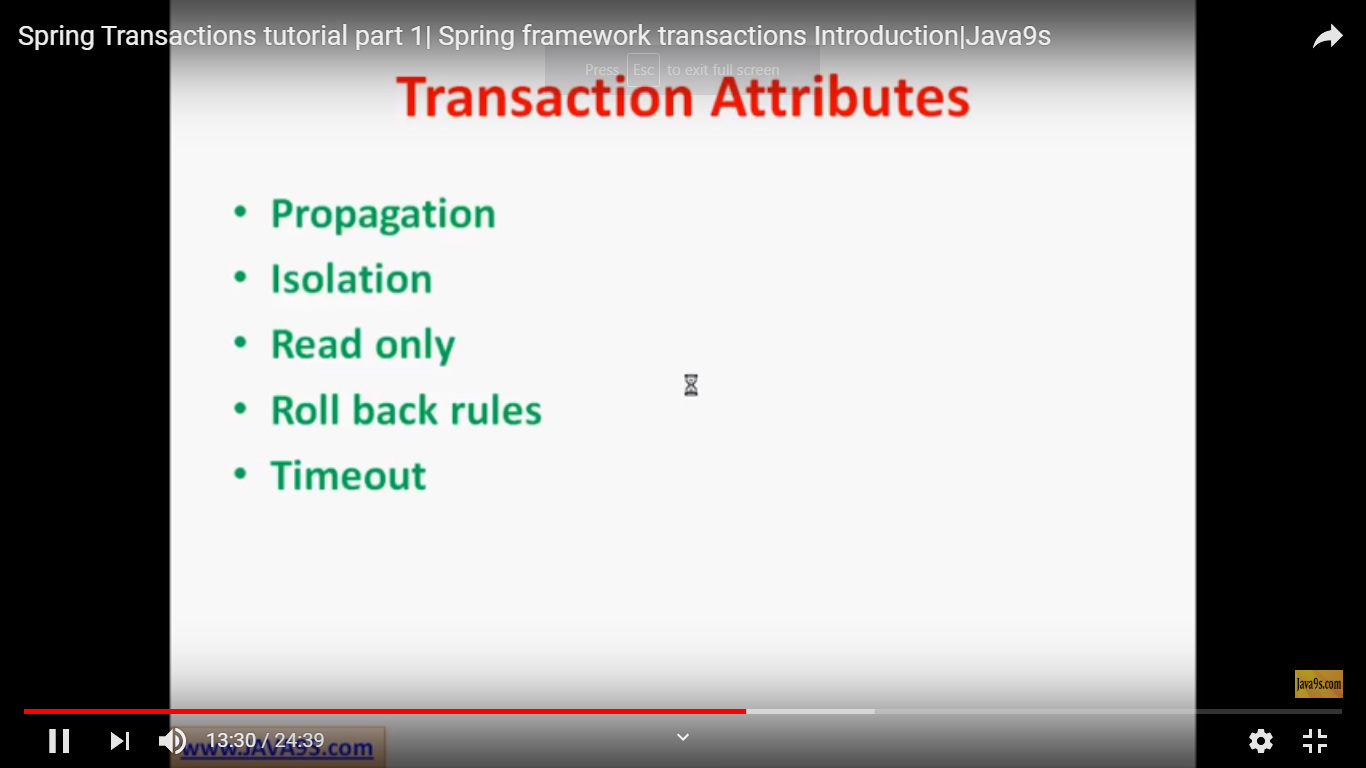


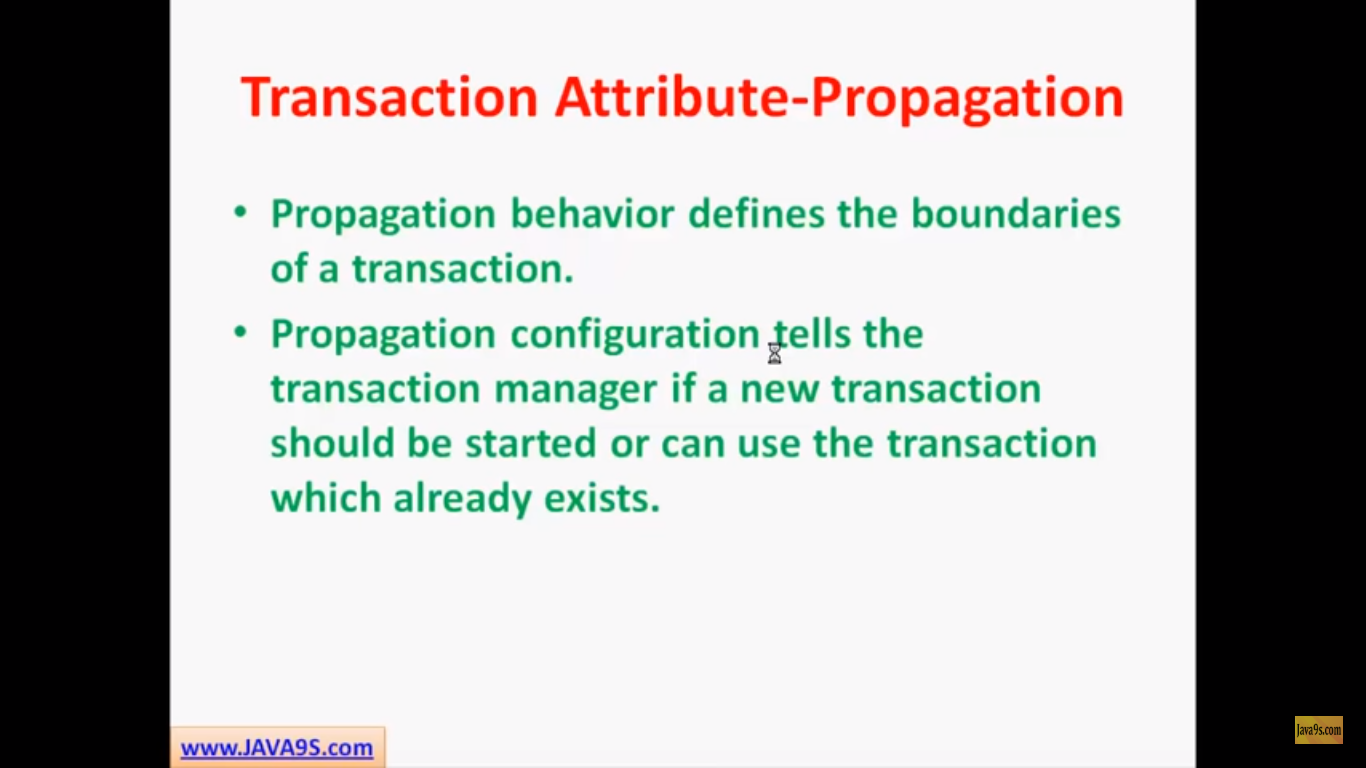


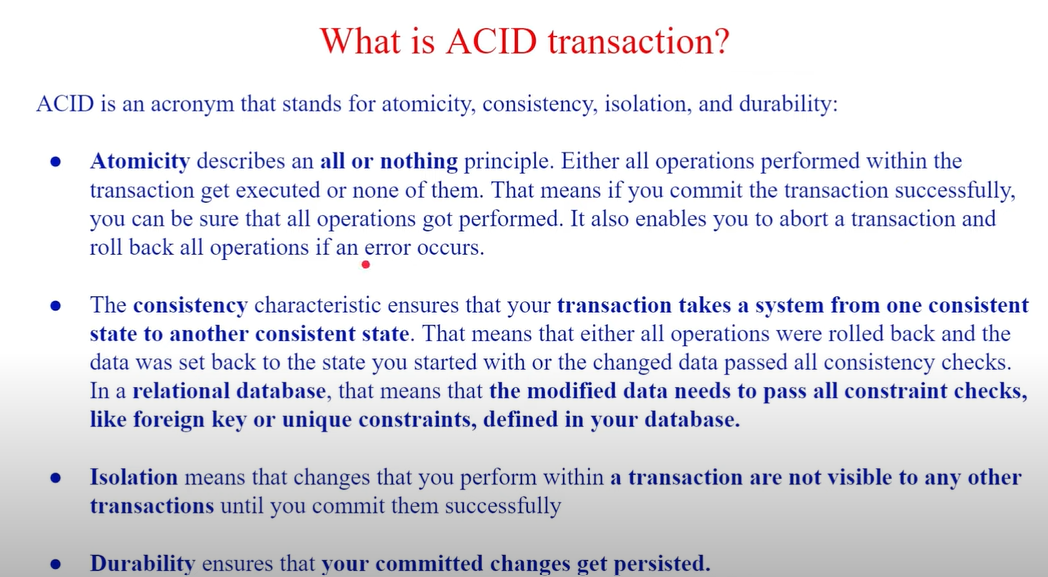
**Second Level cache best Example**<https://www.youtube.com/watch?v=7yPalDu6gx8>

Q:How to manage Transaction management in spring? <https://www.youtube.com/watch?v=C5eNBrZ0ze8>







****