JaiTechooN Software Solutions Pvt.Ltd.,
JNIT, Ramkumar

SNO	Date of Version	Document Name and Version:	Document Created By
1	15-01-2022	JDBC-Hibernate-JPA-Material-v1.0	@Ramkumar
2			

-JavaFullStack Training Material

DAO(Repository) Layer:

1. JDBC (JAVA Database Connective)
2. Java Beans
3. ORM

iMP: JDBC and Hibernate only for bank end database Understanding

Your PBL Project has to development for SpringDataJPA

Hibernate Framework

a. Hibernate with XML Mapping
b. Hibernate with Annotations Mapping
c. Configuration
d. SessionFactory
e. Session
f. Transaction

JPA (Java Persistence API)

a. JPA intefaces
g. EntityManagerFactory -> SessionFactory

b.

c. EntityManager -> Session

d. Persistence

e. getTransaction

f. JPA Entity Bean Life Cycle

   i. Transient

   ii. Persistent

   iii. Detached

g. JPA Annotations

h. CURD Operations

i. First/Second Level Caching

JPA Annotations:

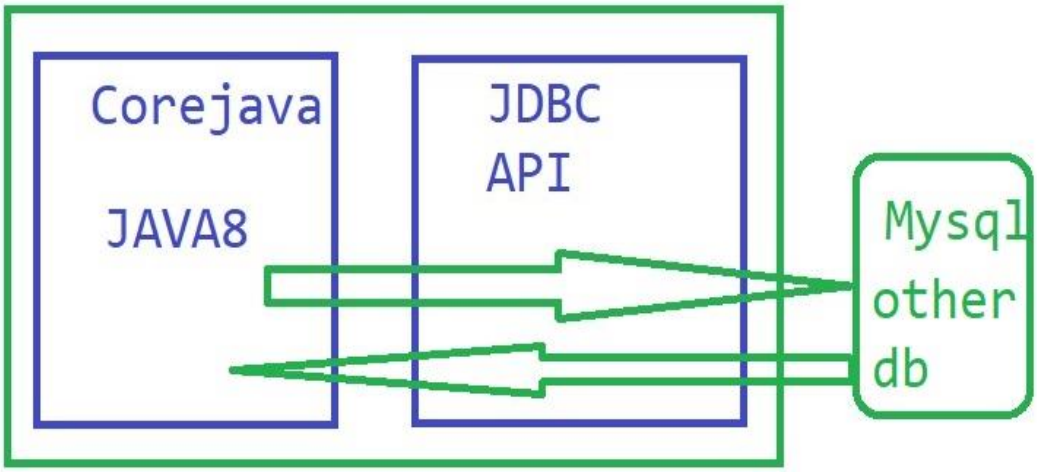
1. @Entity
2. @Table(name="Employee")
3. @Id
4. @Column
5. @GeneratedValue
6. @Transac/tional
7. @PersistenceContext
8. @OneToOne
9. @OneToMany
10. @ManyToMany
11. @ManyToOne

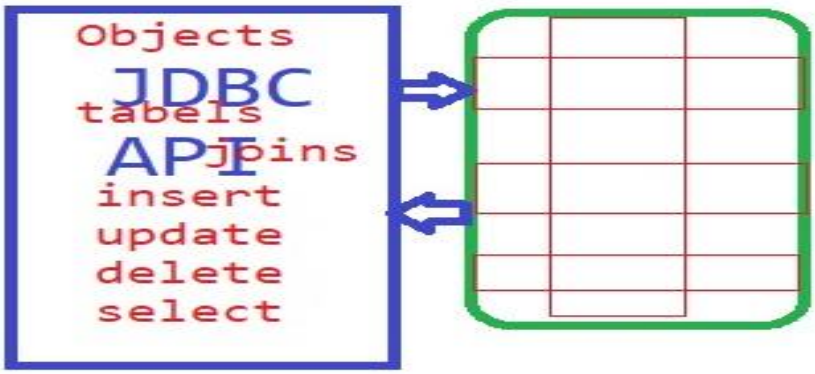
Java Database Connectivity (JDBC)

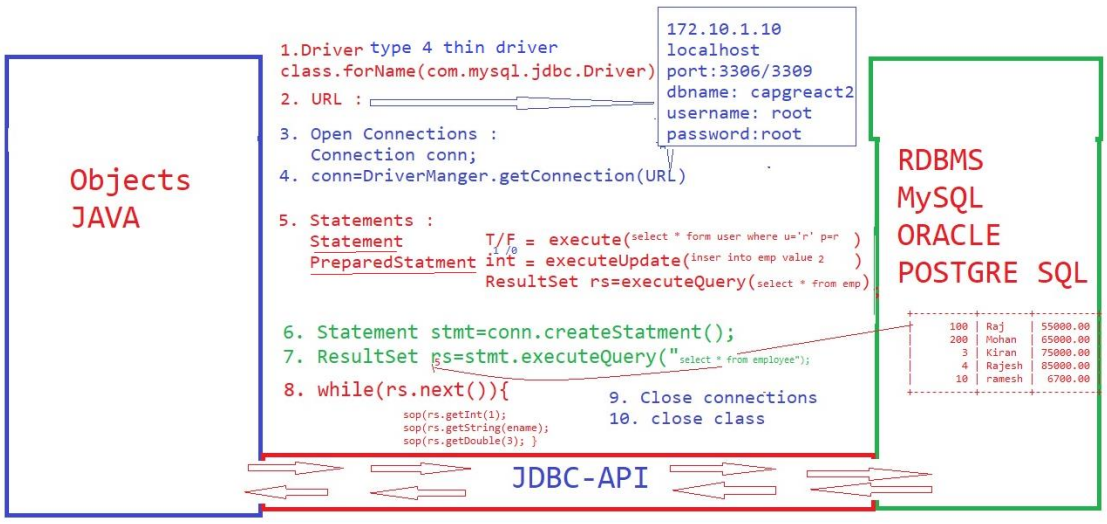
1. The JDBC API consists of a set of interfaces and classes written in the Java programming language.

2. Using these standard interfaces and classes, programmers can write applications that connect to databases, send queries written in structured query language (SQL), and process the results.

3. Since JDBC is a standard specification, one Java program that uses the JDBC API can connect to any database management system (DBMS), as long as a driver exists for that particular DBMS.







ResultSetNextExamplemMsql.java

```
1  //mysql>create table employee(no integer,ename varchar(20),sal float(8,2));
2  //mysql> insert into employee values(101,'raj1',10000);
3  package com.java.jdbc.mysql;
4  import java.sql.Connection;
10 public class ResultSetNextExamplemMsql {
11   public static void main(String[] args) {
12       Connection connection = null;
13      String url = "jdbc:mysql://localhost:3306/";
14      String dbName = "apsitdb";
15      String driverName = "com.mysql.jdbc.Driver";
16      String userName = "root";
17      String password = "root";
18      try{
19        Class.forName(driverName);
20        connection = DriverManager.getConnection(url+dbName, userName, password);
21        try{
22          Statement stmt = connection.createStatement();
23          ResultSet rs=stmt.executeQuery("Select * from employee");
24          while(rs.next()){
25              System.out.println("Employee No :" + rs.getInt(1));
26            System.out.println("Employee Name :" + rs.getString("ename"));
27
28                System.out.println("Employee sal :"+rs.getInt(3));
29          }
30        }
31        catch(SQLException s){
32          System.out.println(s);
33        }
34        connection.close();
35      }
36      catch (Exception e){
37        e.printStackTrace();
38      } }}
39
40
41
42
43
44
```

Console

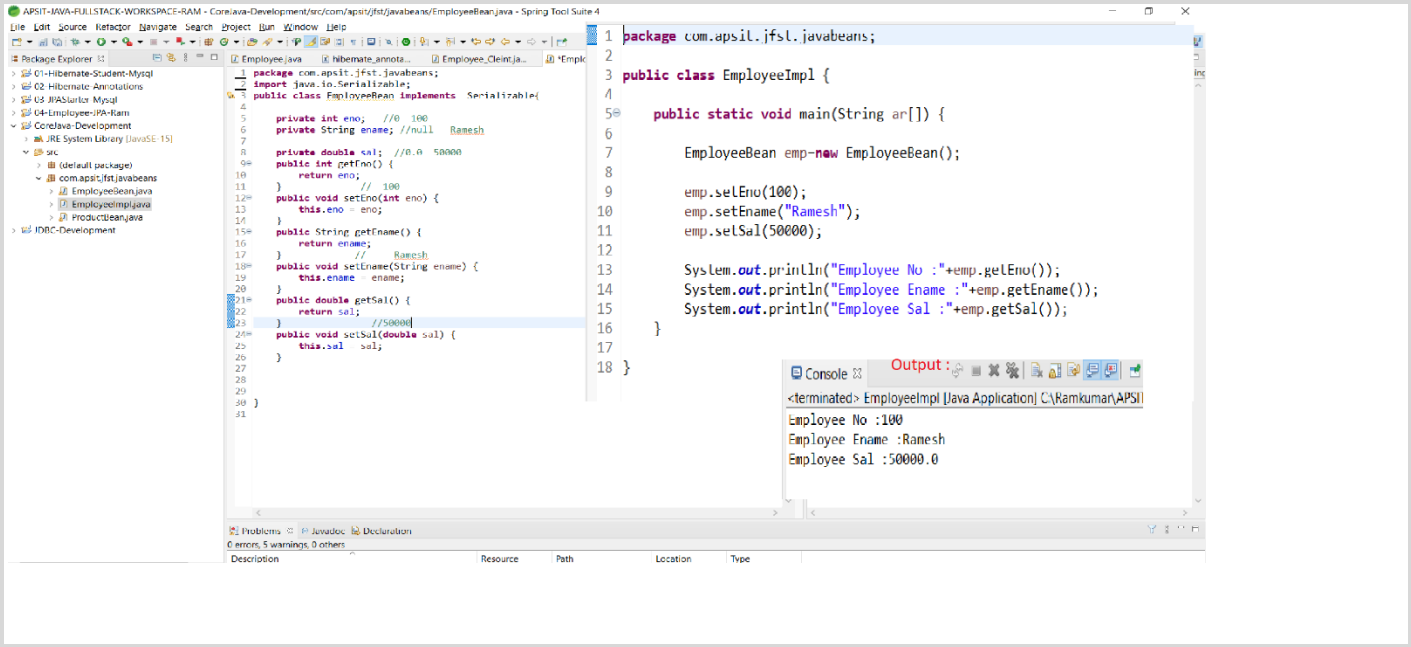
```
<terminated> ResultSetNextExamplemMsql [Java Application] C:\Ramkumar\APSIT-AdvJAVA-Ramkum
Employee No :101
Employee Name :raj1
Employee sal :10000
Employee No :102
Employee Name :raj2
Employee sal :20000
Employee No :103
Employee Name :Babu
Employee sal :20000
```

Java Beans:

All attributes are private

And get/set methods are public

Class implements java.io. Serilizable



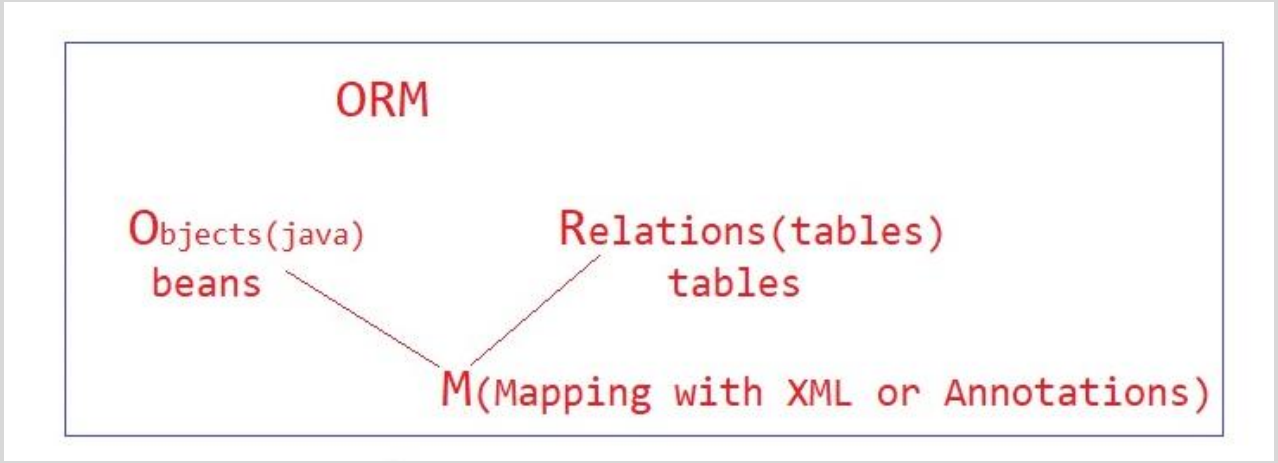
ORM

(Object Relational Mapping)

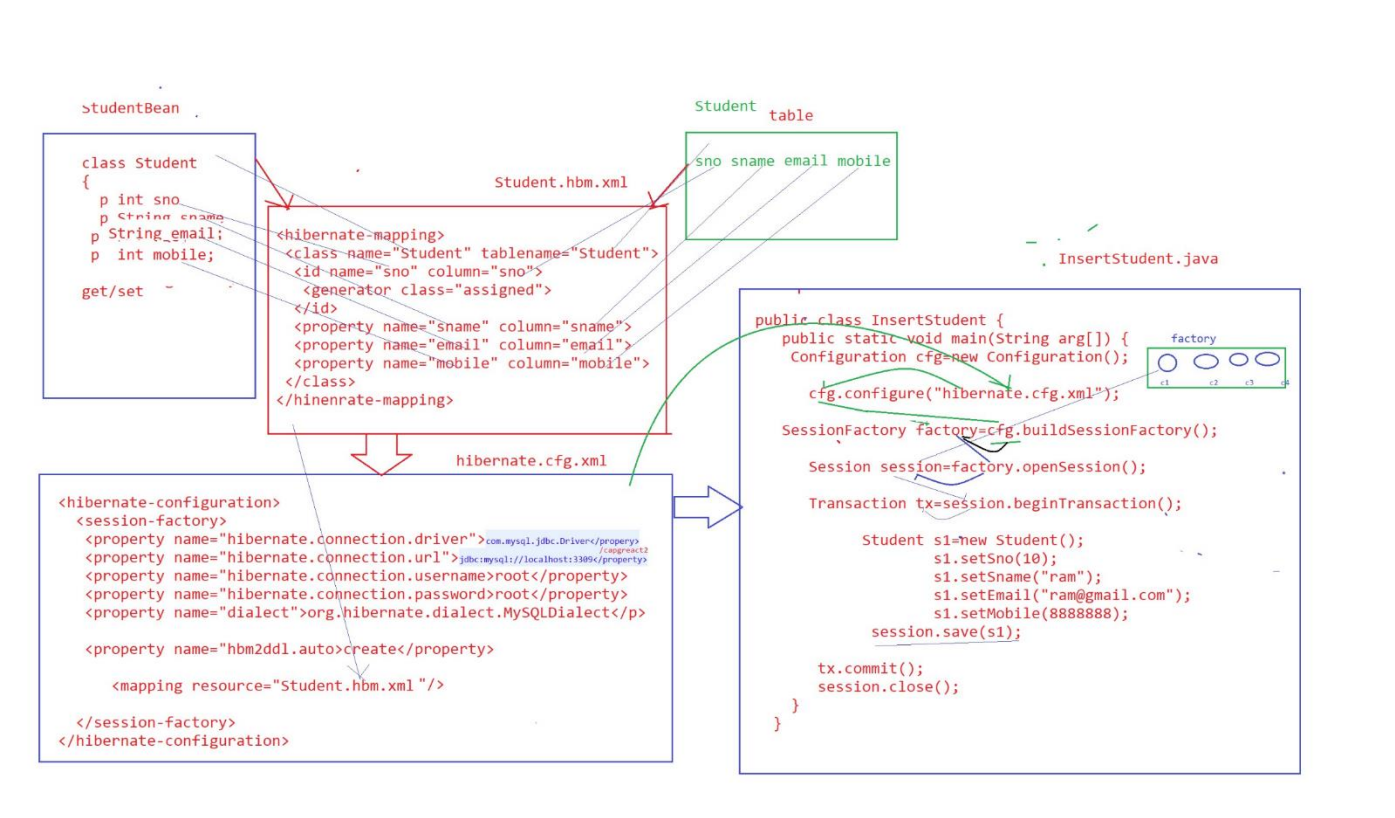
- ORM stands for object-relational mapping, where objects are used to connect the programming language on to the database systems

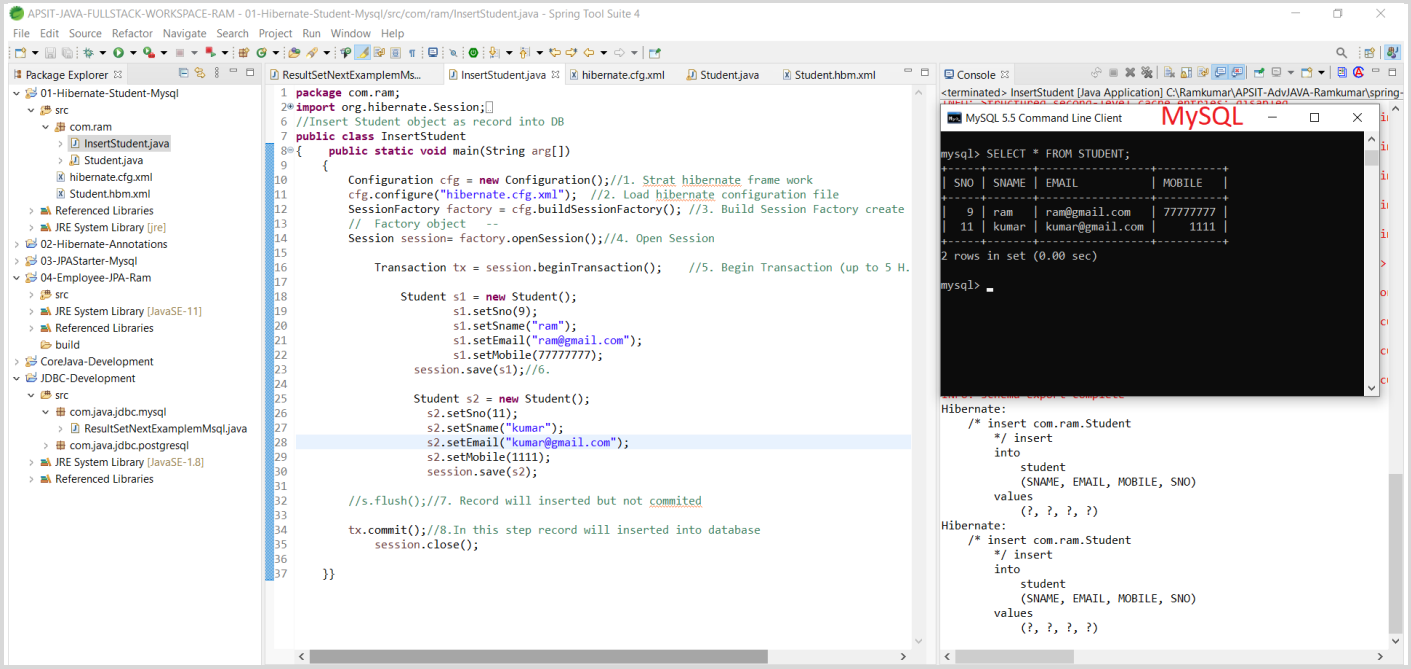
Hibernate (ORM)

- Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map bean Java objects to relational database tables using (XML/Annotation) configuration files

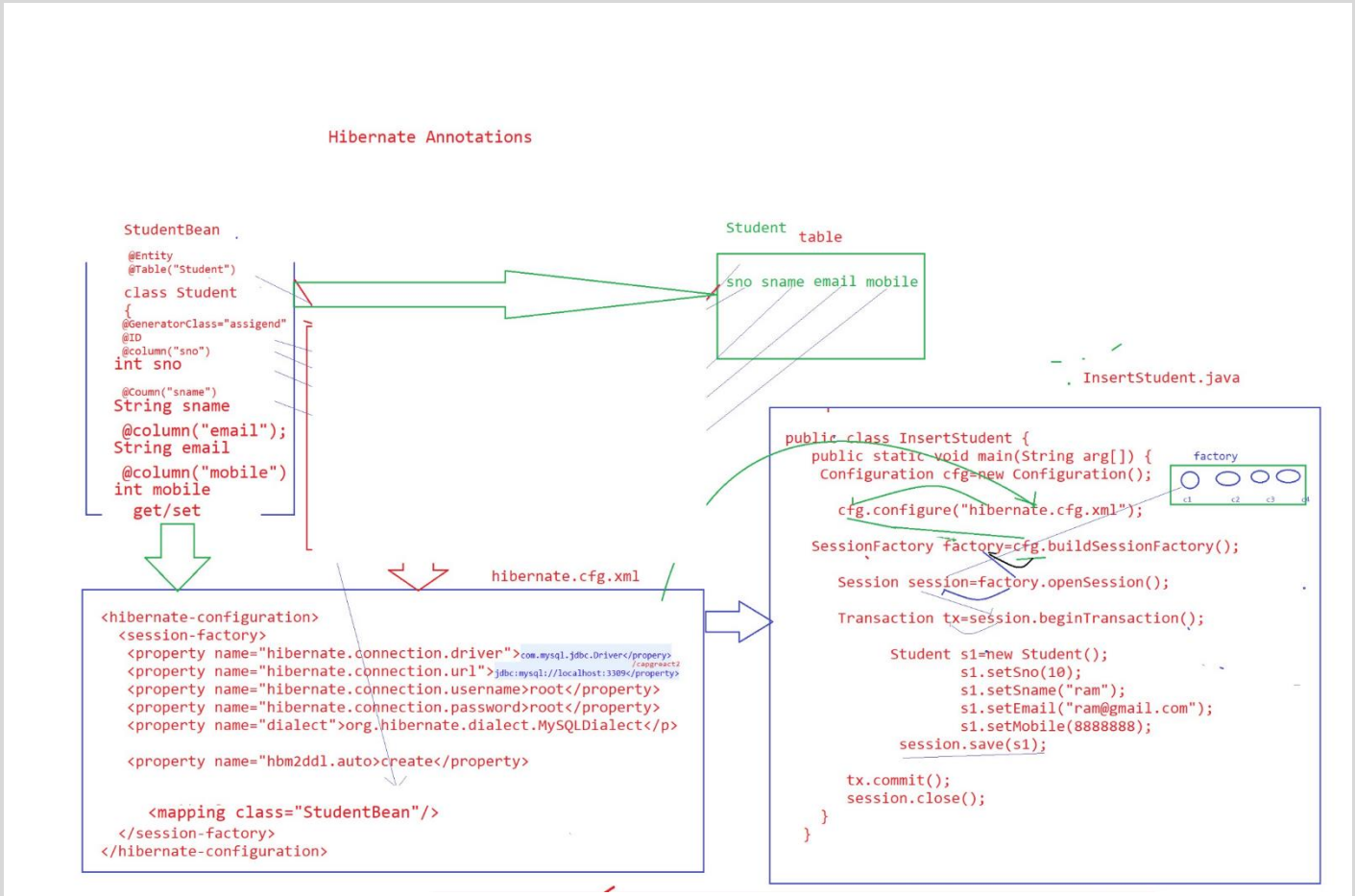


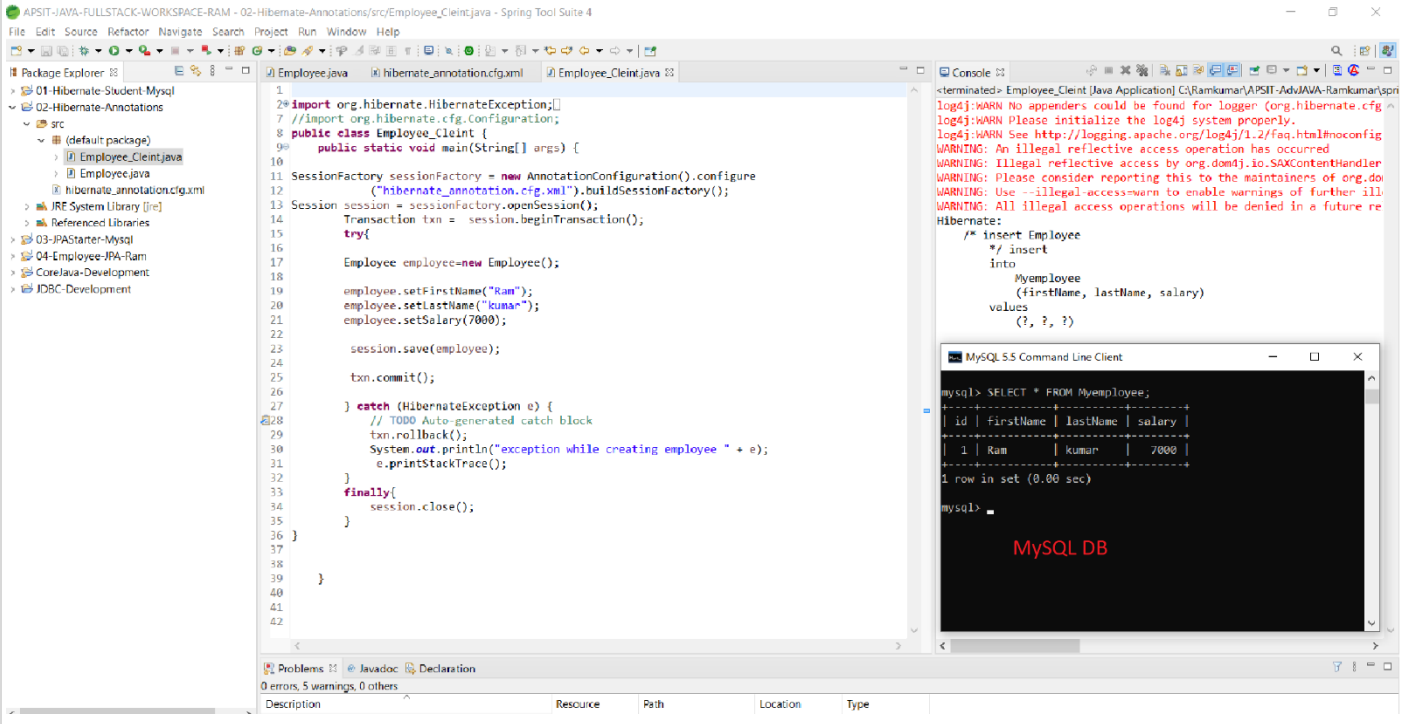
Hibernate with XML Mapping





Hibernate with Annotation Mapping





JPA (Java Persistence API)

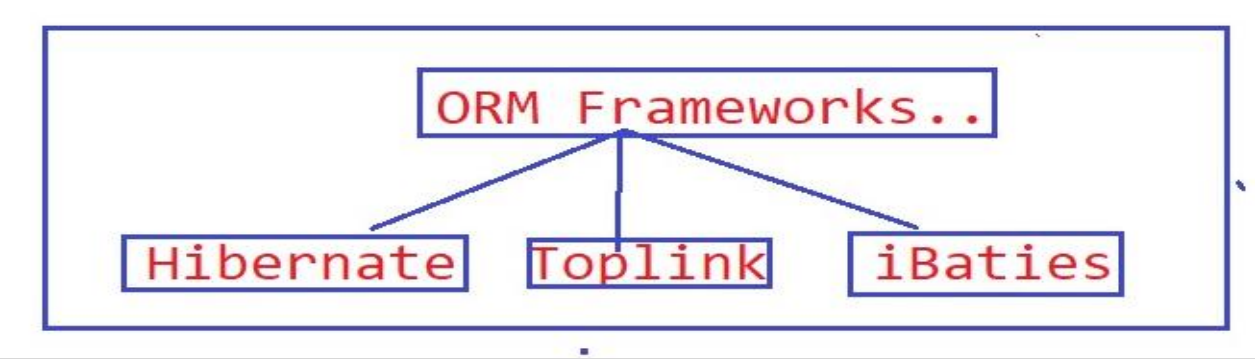
1. The Java Persistence API (JPA) is the Java standard for mapping Java objects to a relational database.

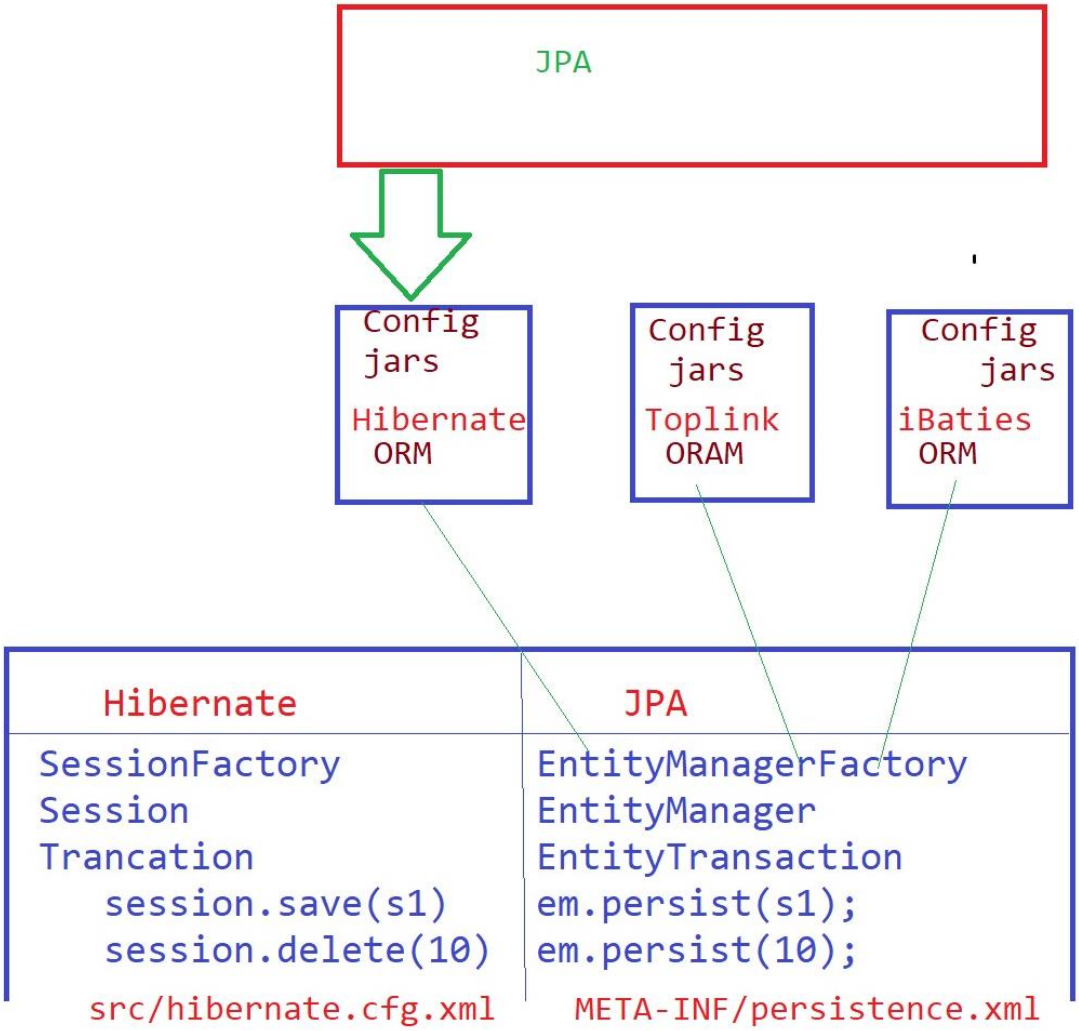
2. Mapping Java objects to database tables and vice versa is called Object-relational mapping (ORM). The Java Persistence API (JPA) is one possible approach to ORM.
3. JPA the developer can map, store, update and retrieve data from relational databases to Java objects and vice versa.
4. JPA can be used in Java-EE and Java-SE applications.

JPA is a specification and several implementations are available.

Popular implementations are:

Hibernate, Toplink , EclipseLink and Apache OpenJPA.





Entities:

1. Entities An entity is a lightweight persistence domain object.
2. Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in that table.
3. The primary programming artifact of an entity is the entity class, although entities can use helper classes.
4. The persistent state of an entity is represented through either persistent fields or persistent properties.
5. These fields or properties use object/relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.

Student Entity Bean

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;


@Entity
@Table(name="students")
public class Student implements Serializable {

	@Id
	//
@GeneratedValue(strategy=GenerationType.AUTO)
	private int studentId;
	private String name;
	public int getStudentId() {
		return studentId;
	}
	public void setStudentId(int studentId)
{
		this.studentId = studentId;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

META-INF/persistence.xml

```
</persistence>
	<properties>
			<property
name="javax.persistence.jdbc.driver"
	value="com.mysql.jdbc.Driver" />
			<property
name="javax.persistence.jdbc.url"

	value="jdbc:mysql://localhost:3306/apsitdb?
useSSL=false" />
			<property
name="javax.persistence.jdbc.user" value="root" />
			<property
name="javax.persistence.jdbc.password"
value="root" />
			<property
name="hibernate.dialect"
	value="org.hibernate.dialect.MySQLDialect"
/>
			<property
name="hibernate.hbm2ddl.auto" value="create"/>
			<property
name="hibernate.show_sql" value="true" />
		</properties>
	</persistence-unit>
</persistence>
```

StudentTest.java

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class StudentTest {

	public static void main(String[] args) {

EntityManagerFactory factory =
Persistence.createEntityManagerFactory("JPA-PU");
		EntityManager em =
factory.createEntityManager();
		em.getTransaction().begin();

		Student student = new Student();
		student.setName("Ramesh");
		student.setStudentId(33);

		em.persist(student);

		em.getTransaction().commit();

System.out.println("Added one student to database.");
		em.close();
		factory.close();
	}
}
```

Student Entity

```
package com.cg.jpastart.entities;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;


@Entity
@Table(name="students")
public class Student implements Serializable {

	@Id
	//@GeneratedValue(strategy=GenerationType.AUTO)
	private int studentId;
	private String name;
	public int getStudentId() {
		return studentId;
	}
	public void setStudentId(int studentId) {
		this.studentId = studentId;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}}
```

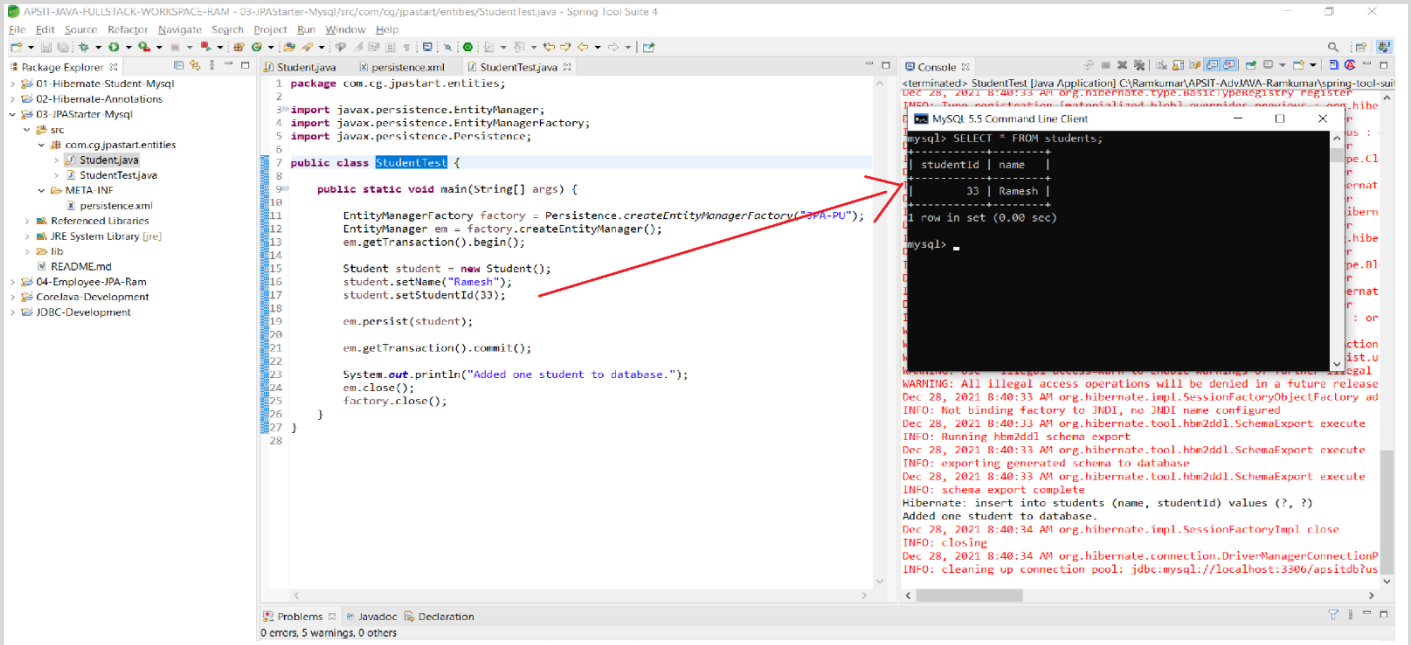
META-INF/persistence.xml

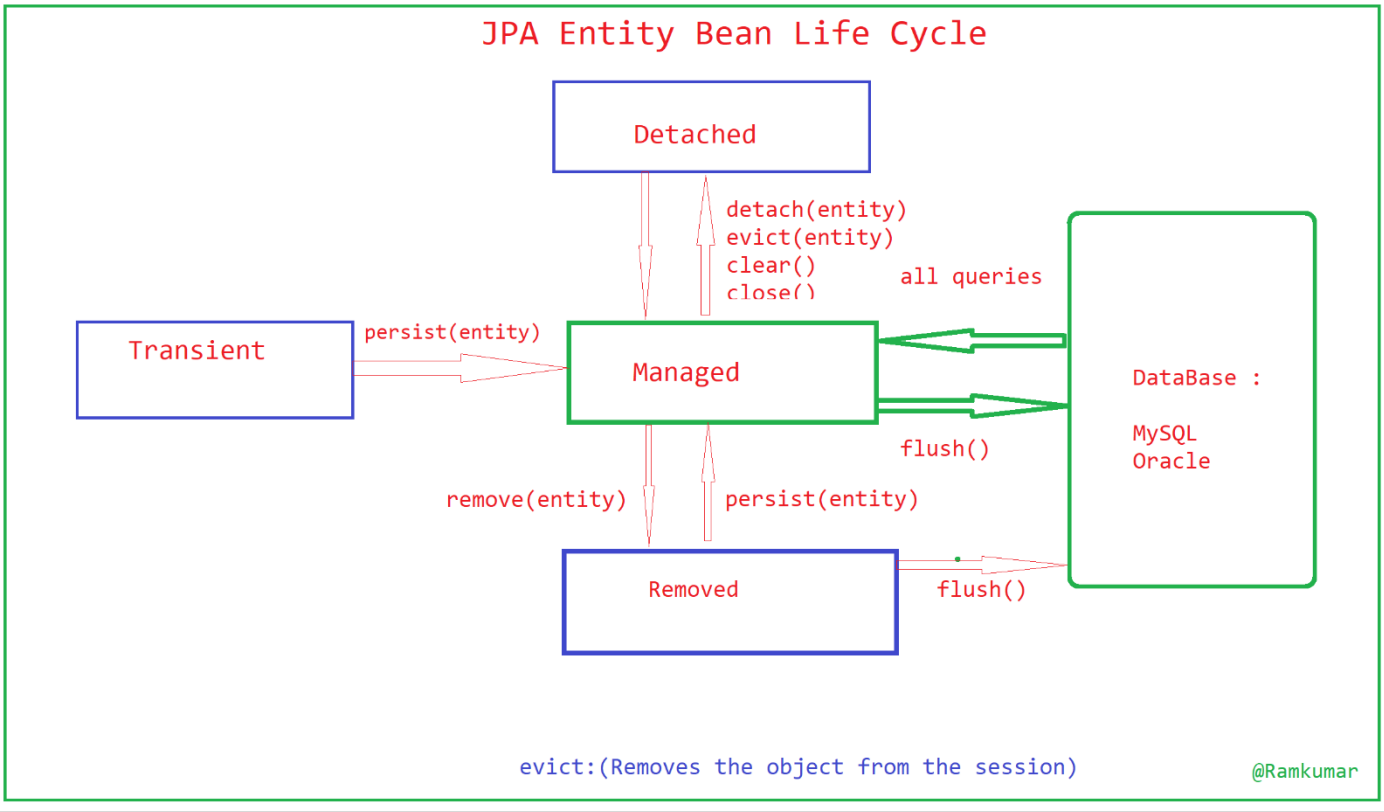
```
<persistence-unit name="JPA-PU" transaction-type="RESOURCE_LOCAL">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <properties>
	<property name="javax.persistence.jdbc.driver"
					value="com.mysql.jdbc.Driver" />
	<property name="javax.persistence.jdbc.url"
				value="jdbc:mysql://localhost:3306/apsitdb?useSSL=false"/>
	<property name="javax.persistence.jdbc.user" value="root" />
	<property name="javax.persistence.jdbc.password" value="root" />
	<property name="hibernate.dialect"
	value="org.hibernate.dialect.MySQLDialect" />
	<property name="hibernate.hbm2ddl.auto" value="create"/>
	<property name="hibernate.show_sql" value="true" />
	</properties>
	</persistence-unit>
  </persistence>
```

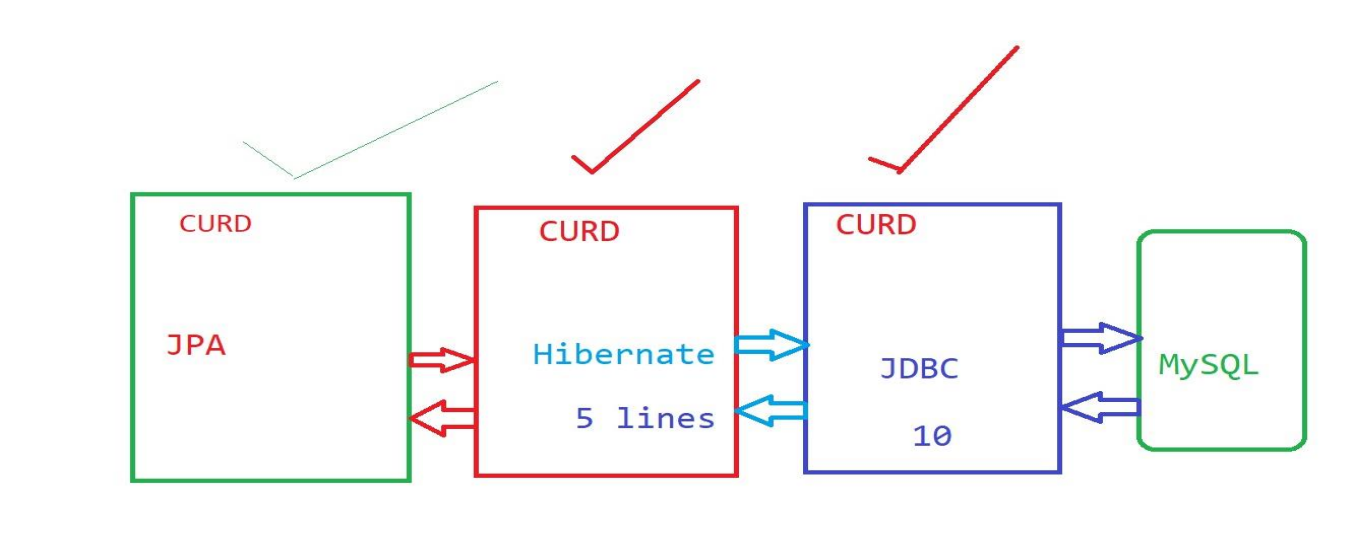
StudentTest.java

```
package com.cg.jpastart.entities;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class StudentTest {
	public static void main(String[] args) {
		EntityManagerFactory factory = Persistence.createEntityManagerFactory("JPA-PU");
		EntityManager em = factory.createEntityManager();
		em.getTransaction().begin();
			Student student = new Student();
				student.setName("Ramesh");
				student.setStudentId(33);
		em.persist(student);
		em.getTransaction().commit();
		System.out.println("Added one student to database.");
		em.close();
		factory.close();
	}}
```


APSIT-JAVA-FULLSTACK-WORKSPACE-RAM - 03-JPAStarter-Mysql/src/com/cg/jpastart/entities/StudentTest.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
01-Hibernate-Student-Mysql
02-Hibernate-Annotations
03-JPAStarter-Mysql
src
com.cg.jpastart.entities
Student.java
StudentTest.java
META-INF
persistence.xml
Referenced Libraries
JRE System Library [jre]
lib
README.md
04-Employee-JPA-Ram
CoreJava-Development
JDBC-Development
Student.java
persistence.xml
StudentTest.java
package com.cg.jpastart.entities;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
public class StudentTest {
public static void main(String[] args) {
EntityManagerFactory factory = Persistence.createEntityManagerFactory("JPA-PU");
EntityManager em = factory.createEntityManager();
em.getTransaction().begin();
Student student = new Student();
student.setName("Ramesh");
student.setStudentId(33);
em.persist(student);
em.getTransaction().commit();
System.out.println("Added one student to database.");
em.close();
factory.close();
}
}
Console
MySQL 5.5 Command Line Client
mysql> SELECT * FROM students;
| studentId | name |
| 33 | Ramesh |
1 row in set (0.00 sec)
mysql>
Problems Javadoc Declaration
0 errors, 5 warnings, 0 others


JPA Entity Bean Life Cycle
Detached
detach(entity)
evict(entity)
clear()
close()
all queries
Transient
persist(entity)
Managed
DataBase :
MySQL
Oracle
flush()
remove(entity)
persist(entity)
Removed
flush()
evict:(Removes the object from the session)
@Ramkumar



JPA Relations

Entity Relationships

Multiplicities are of the following types: one-to-one, one-to-many, many-to-one, and many-to-many:

- **One-To-One**: Each entity instance is related to a single instance of another entity. For example, to model a physical warehouse in which each storage bin contains a single widget, StorageBin and Widget would have a one-to-one relationship. One-to-one relationships use the javax.persistence.OneToOne annotation on the corresponding persistent property or field.

- **One-to-many**: An entity instance can be related to multiple instances of the other entities. A sales order, for example, can have multiple line items. In the order application, Order would have a one-to-many relationship with LineItem. One-to-many relationships use the javax.persistence.OneToMany annotation on the corresponding persistent property or field.

- **Many-to-one:** Multiple instances of an entity can be related to a single instance of the other entity. This multiplicity is the opposite of a one-to-many relationship. In the example just mentioned, the relationship to Order from the perspective of LineItem is many-to-one. Many-to-one relationships use the javax.persistence.ManyToOne annotation on the corresponding persistent property or field.

- **Many-to-many:** The entity instances can be related to multiple instances of each other. For example, each college course has many students, and every student may take several courses. Therefore, in an enrollment application, Course and Student would have a many-to-many relationship. Many-to-many relationships use the javax.persistence.ManyToMany annotation on the corresponding persistent property or field.

Cascade Operations for Entities

Cascade Operation	Description
ALL	All cascade operations will be applied to the parent entity's related entity. All is equivalent to specifying cascade={DETACH, MERGE, PERSIST, REFRESH, REMOVE}
DETACH	If the parent entity is detached from the persistence context, the related entity will also be detached.
MERGE	If the parent entity is merged into the persistence context, the related entity will also be merged.
PERSIST	If the parent entity is persisted into the persistence context, the related entity will also be persisted.
REFRESH	If the parent entity is refreshed in the current persistence context, the related entity will also be refreshed.
REMOVE	If the parent entity is removed from the current persistence context, the related entity will also be removed.

Using Collections in Entity Fields and Properties

Collection-valued persistent fields and properties must use the supported Java collection interfaces regardless of whether the entity uses persistent fields or properties.

The following collection interfaces may be used:

- java.util.Collection
- java.util.Set
- java.util.List
- java.util.Map

Transient

- All fields not annotated javax.persistence.Transient or not marked as Java transient will be persisted to the data store. The object/relational mapping annotations must be applied to the instance variables.

LAZY , EAGER

- The two attributes of @ElementCollection are target Class and fetch.
- The target Class attribute specifies the class name of the basic or embeddable class and is optional if the field or property is defined using Java programming language generics.
- The optional fetch attribute is used to specify whether the collection should be retrieved lazily or eagerly, using the javax.persistence.FetchType constants of either LAZY or EAGER, respectively.
- By default, the collection will be fetched lazily.
- The entity, Person, has a persistent field, nicknames, which is a collection of String classes that will be fetched eagerly.
- The target Class element is not required, because it uses generics to define the field.

```
@Entity
public class Person
 { ...
@ElementCollection(fetch=EAGER)
protected Set nickname = new HashSet();
```

...

}

Example:

```
3⊕import java.util.List;
12 @Entity
13 @Table(name="cart")
14 public class Cart {
15⊖    @Id
16   private String cartId;
17⊖    @OneToOne(mappedBy = "cart",fetch = FetchType.LAZY, cascade = CascadeType.ALL)
18      private Customer customer;
19
20⊖    @OneToMany(mappedBy = "cart",fetch = FetchType.LAZY, cascade = CascadeType.ALL)
21      private List<Product> products;
22
23⊖  public String getCartId() {
24         return cartId;
25    }
26⊖    public void setCartId(String cartId) {
27         this.cartId = cartId;
28    }
29⊖    public Customer getCustomer() {
30         return customer;
31    }
32⊖    public void setCustomer(Customer customer) {
33         this.customer = customer;
34    }
35⊖    public List<Product> getProducts() {
36         return products;
37    }
```

@OneToOne

Example of @OneToOne and JPA Annotation

```java
package com.cg.jpastart.entities;



import java.io.Serializable;

@Entity
@Table(name="students")
public class Student implements Serializable {


    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int studentId;

    private String name;

    @OneToOne(cascade=CascadeType.ALL)
    @JoinColumn(name="address_id")
    private Address address;


    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    public String getName() {
        return name;
```

ProductDaoImpl.java | ProductBean.java | Employee.java | Student.java | *Address.java

```java
package com.cg.jpastart.entities;

import java.io.Serializable;

@Entity
public class Address implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Column(name="ADDRESS_ID")
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int addressId;
    @Column(name="ADDRESS_STREET")
    private String street;
    @Column(name="ADDRESS_CITY")
    private String city;
    @Column(name="ADDRESS_STATE")
    private String state;
    @Column(name="ADDRESS_ZIPCODE")
    private String zipCode;

    //to create bi-directional relationship, use one to one with mappedBy
    //mappedBy attribute indicates property name of owner i.e. Student class

    @OneToOne(mappedBy="address")
    private Student student;

    public int getAddressId() {
        return addressId;
    }
    public void setAddressId(int addressId) {
        this.addressId = addressId;
    }
    public String getStreet() {
        return street;
    }
    }
```

@OneToOne : Realation

JPA ANNOTATIONS

@Entity

The @Entity annotation is used to specify that the currently annotate class represents an entity type.

@Table

The @Table annotation is used to specify the primary table of the currently annotated entity

@Id

In JPA the object id is defined through the @Id annotation and should correspond to the primary key of the object's table. An object id can either be a natural id or a generated id.

@Column

The @Column annotation is used to specify the mapping between a basic entity attribute and the database table column.

@GeneratedValue

The @GeneratedValue annotation specifies that the entity identifier value is automatically generated using an identity column, a database sequence, or a table generator.