**DAY 5**
**Implementation Of Relationship In Hibernate**

Now let's understand all CRUD operations to be performed in all different types of relationship

**Scenario 1:**
**Now let's try to fetch customer and customerDetails data from both the tables**
**Consider the table below**

**Customer Table:**

| id | name | cust_detail_id |
|---|---|---|
| 1 | alex | 501 |

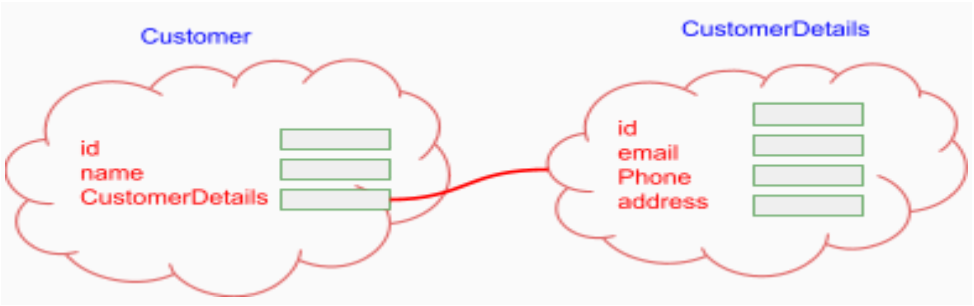
**Customer_details Table**

| id | address | mail | phone |
|---|---|---|---|
| 501 | Bangalore | alex@gmail.com | 789999909 |

**Logic:**

```
Customer customer = session.get(Customer.class, 1); // Here the get method accepts two parameters: class name and primary key. Now get() will fetch customer object which contains id, name, CustomerDetails object reference with primary key 1

CustomerDetails details = customer.getCustomerDetails();//Now using getter method of CustomerDetails in Customer class you can fetch the details of customer present in CustomerDetails
```




**OneToOne2.java**

```
package com.tap.model;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.tap.app.Customer;
import com.tap.app.CustomerDetails;

public class OneToOne2 {

	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Session session = null;

		try {
//			Create session Factory

			sessionFactory = new Configuration()
					.configure()
					.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
					.buildSessionFactory();

//			Create session
			session = sessionFactory.openSession();

//			Create Transaction
```

```
                Transaction transaction =
session.beginTransaction();

//              CRUD Operation
                Customer customer = session.get(Customer.class,
1);
                CustomerDetails details =
customer.getCustomerDetails();
                System.out.println(customer);
                System.out.println(details);;

                transaction.commit();

            } finally {
//              Closing Resources
                session.close();
                sessionFactory.close();

            }
      }

}
```

Output:

```
1, alex, CustomerDetails [501, alex@gmail.com, 789999909, Bangalore]
CustomerDetails [501, alex@gmail.com, 789999909, Bangalore]
```

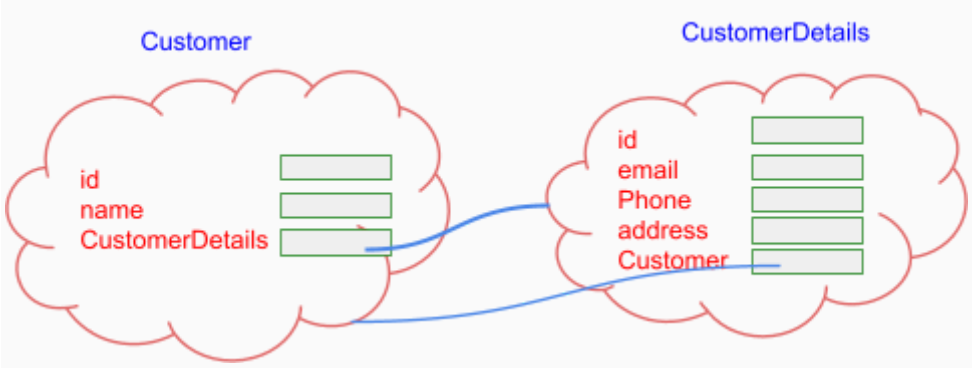
Here we have fetched the details of CustomerDetails from Customer but vise versa is not possible here because this is unidirectional

**Scenario 2:**

Now let's see how to fetch the data of customers from CustomerDetails.

Steps:

- First we need to create object reference of Customer in CustomerDetails as shown below




- Here to make bidirectional and can access Customer objects from CustomerDetails we have added @OneToOne annotations to Customer Object reference. Along with these annotations now you need to mention one more variable Mapped By .
  MappedBy signals hibernate that the key for the relationship is on the other side.
  This means that although you link 2 tables together, only 1 of those tables has a foreign key constraint to the other one. MappedBy allows you to still link from the table not containing the constraint to the other table.

```
@OneToOne(mappedBy = "customerDetails")
private Customer customer;
```

- We need to specify cascadeType so changes made in one table can reflect in another table as well

```
@OneToOne(mappedBy = "customerDetails", cascade =
CascadeType.ALL)
private Customer customer;
```

- Add getter and setter of customer in CustomerDetails class

**Adding Customer field in CustomerDetails**

**CustomerDetails.java**

```
package com.tap.app;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name = "customer_details")
public class CustomerDetails {

	@Id
	@Column(name = "id")
	private int id;

	@Column(name = "email")
	private String email;

	@Column(name = "phone")
	private String phone;

	@Column(name = "address")
	private String address;

	@OneToOne(mappedBy = "customerDetails", cascade =
CascadeType.ALL)
	private Customer customer;

	public CustomerDetails() {
	}
	public CustomerDetails(int id, String email, String
phone, String address) {
```

```
        super();
        this.id = id;
        this.email = email;
        this.phone = phone;
        this.address = address;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
```

```
        this.address = address;
    }
    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    @Override
    public String toString() {
        return "CustomerDetails [" + id + ", " + email
+ ", " + phone + ", " + address + "]";
    }

}
```

- **Now using Customer Details Object you can access Customer object**

```
CustomerDetails details =  session.get(CustomerDetails.class, 501); // using session.get() get the customerDetails object
Customer customer = details.getCustomer(); // using getter method of customer in CustomerDetails class get customer object
```

**OneToOne2.java**

```
package com.tap.model;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.tap.app.Customer;
```

```
import com.tap.app.CustomerDetails;

public class OneToOne2 {

	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Session session = null;

		try {
//			Create session Factory

			sessionFactory = new Configuration()
					.configure()

.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
					.buildSessionFactory();

//			Create session
			session = sessionFactory.openSession();

//			Create Transaction
			Transaction transaction =
session.beginTransaction();

//			CRUD Operation
			CustomerDetails details =
session.get(CustomerDetails.class, 501);
			Customer customer = details.getCustomer();
			System.out.println(details);
			System.out.println(customer);

			transaction.commit();

		} finally {
```

```
//            Closing Resources
              session.close();
              sessionFactory.close();

          }
     }

}
```

**Output:**

```
CustomerDetails [501, alex@gmail.com, 789999909, Bangalore]
1, alex, CustomerDetails [501, alex@gmail.com, 789999909, Bangalore]
```

If you can observe from the above output here we have fetched Customer details from CustomerDetails object

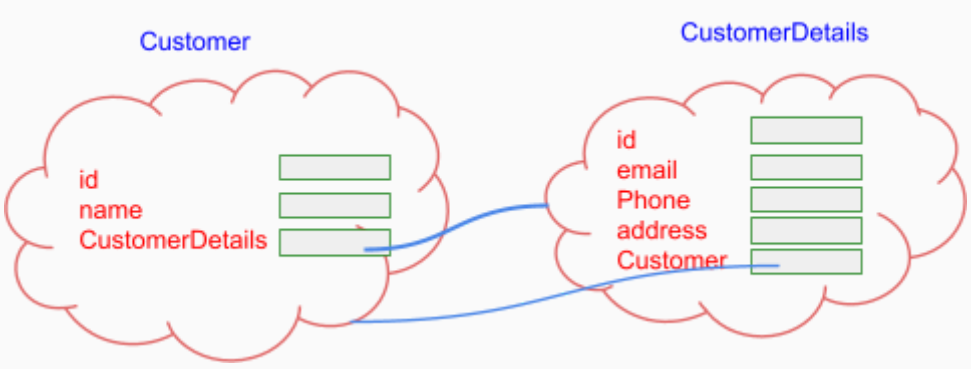
**Scenario 3:**

Now let's see how to delete the data from the database. If you delete the customer the customerDetails will also get deleted

Logic:

```
Customer customer = session.get(Customer.class, 1); // Get the Customer object using session.get()
session.delete(customer); // Delete the object using session.delete()
```

Here if you observe from below if Customer object is deleted CustomerDetails object also will get deleted. It is also because Cascadetype that is mentioned here is CascadeType.ALL it means if you remove Customer object the associated CustomerDetails object also gets deleted




```
@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "cust_detail_id")
private CustomerDetails customerDetails;
```

**OneToOneDelete.java**

```
package com.tap.model;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.tap.app.Customer;
import com.tap.app.CustomerDetails;

public class OneToOneDelete {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
                        .buildSessionFactory();

//             Create session
               session = sessionFactory.openSession();

//             Create Transaction
               Transaction transaction =
session.beginTransaction();
```

```
//            CRUD Operation
              Customer customer =
session.get(Customer.class, 1);
              session.delete(customer);

              transaction.commit();

          } finally {
//            Closing Resources
              session.close();
              sessionFactory.close();

          }
     }

}
```

Output:

```
Hibernate: delete from customer where id=?
Hibernate: delete from customer_details where id=?
```

Here if you observe the output you have deleted customer object but customerDetails object also will get deleted it is because CascadeType.ALL changes made in one entity will reflect to other entity.

**Scenario 4**
Now let's see how to update the alex phone number using Customer object

Consider the table given below

**Customer Table:**

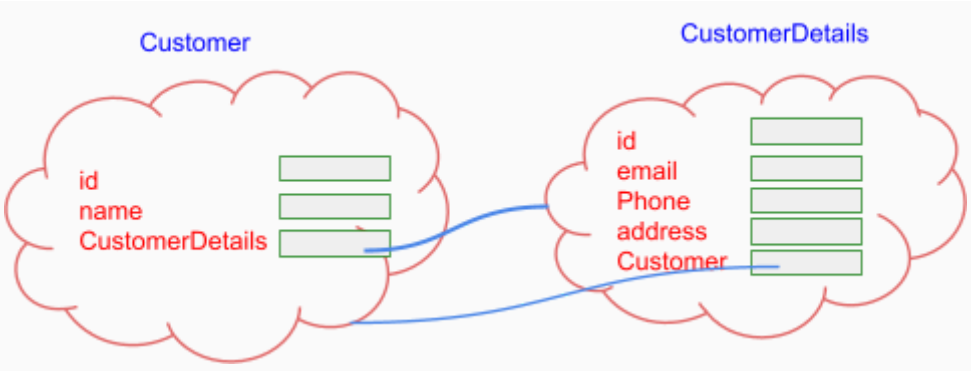
| id | name | cust_detail_id |
|---|---|---|
| 1 | bob | 501 |
| 2 | alex | 502 |

**CustomerDetails Table:**

| id | address | email | phone |
|---|---|---|---|
| 501 | Bangalore | bob@gmail.com | 7854321345 |
| 502 | Chennai | alex@gmail.com | 9845678123 |

**Logic:**

```
Customer customer = session.get(Customer.class, 1);  //
Here Customer object you are getting using session.get()
customer.getCustomerDetails().setPhone("8975323456"); //
using getter method of CustomerDetails first we are
getting customerdetails object then using setter method
of phone you can update the phone number
```



**OneToOne2.java**

```
package com.tap.model;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.tap.app.Customer;
import com.tap.app.CustomerDetails;

public class OneToOne2 {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
                         .buildSessionFactory();

//             Create session
               session = sessionFactory.openSession();

//             Create Transaction
               Transaction transaction =
session.beginTransaction();

//             CRUD Operation
```

```
            Customer customer =
session.get(Customer.class, 1);

customer.getCustomerDetails().setPhone("8975323456");

            transaction.commit();

        } finally {
//            Closing Resources
            session.close();
            sessionFactory.close();
        }
    }

}
```

Output:

```
Hibernate: update customer_details set address=?, email=?, phone=? where id=?
```

**Customer Table:**

| id | name | cust_detail_id |
|---|---|---|
| 1 | bob | 501 |
| 2 | alex | 502 |

**CustomerDetails Table:**

| id | address | email | phone |
|---|---|---|---|
| 501 | Bangalore | bob@gmail.com | 8975323456 |
| 502 | Chennai | alex@gmail.com | 9845678123 |

Here if you observe from the above table the phone number is updated using customer object.

## OneToMany

Now let's understand the OneToMany relationship.

To Understand this let's consider Customer, CustomerDetails and Orders table shown below

**Customer Table:**

| id | name | cust_detail_id |
|---|---|---|
| 1 | bob | 501 |
| 2 | alex | 502 |

**CustomerDetails Table:**

| id | address | email | phone |
|---|---|---|---|
| 501 | Bangalore | bob@gmail.com | 8975323456 |
| 502 | Chennai | alex@gmail.com | 9845678123 |

**Orders Table:**

| id | item | price |
|---|---|---|
| 701 | Burger | 300 |
| 702 | Burger | 300 |
| 703 | Pizza | 370 |
| 704 | Chapati | 150 |

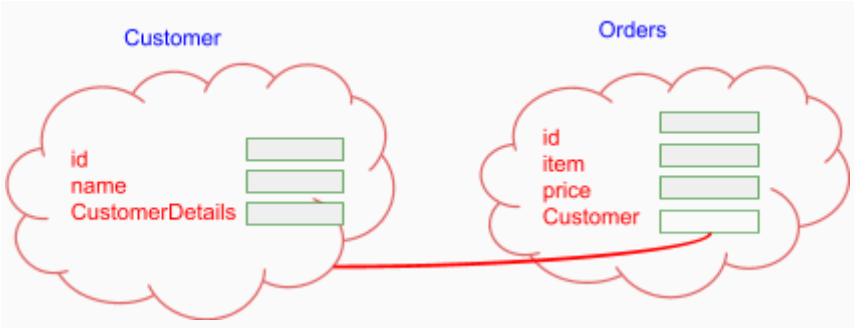
Now we need to generate a relationship between Customer and orders table, here one customer can make many orders when you look from customer table so it is **OneToMany** relationship. From the orders table there are many orders belonging to one customer so it is a **ManyToOne** relationship from the orders table.

First in order to build the relationship there should be a cust_id in the orders table which is referring to Customer table
**Orders Table:**

| id | item | price | cust_id |
|---|---|---|---|
| 701 | Burger | 300 | 2 |
| 702 | Burger | 300 | 1 |
| 703 | Pizza | 370 | 2 |
| 704 | Chapati | 150 | 2 |

Steps to generate OneToMany relationship between Customer and Orders Table

- Create orders class with id, item, price attributes
- Cust_id in the orders table refers to Customer so we will have customer object reference in Orders class




- Here cust_id helps to join orders and Customer table so annotations for this @JoinColumn(name = “cust_id”)

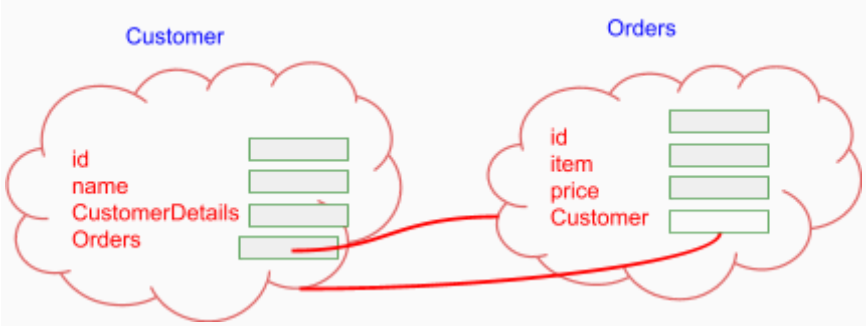
```
@JoinColumn(name = "cust_id") // Here @JoinColumn
```

```
joins two tables where cust_id is referring to primary
key of Customer table
     private Customer customer;
```

- Now we need to mention the type of relationship which is @ManyToOne(cascade = CascadeType.ALL)

```
@JoinColumn(name = "cust_id")
@ManyToOne(cascade = CascadeType.ALL) //@manyToOne means
multiple rows in the order table is referring to one row
in customer table
Private Customer customer;
```

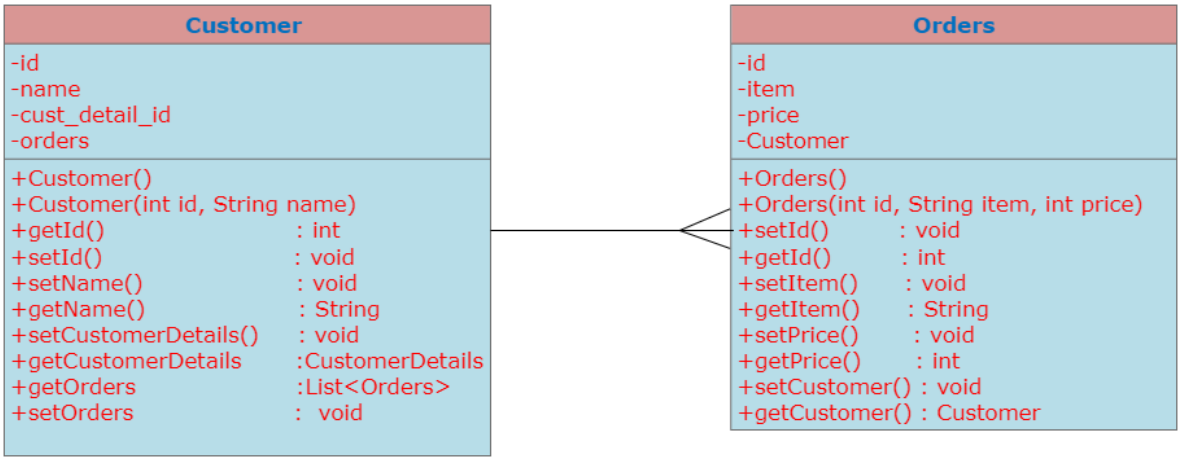
- Now from the orders you can identify who the customer is, now from the customer we need to fetch the orders details for that we need to have orders object reference in the Customer class




- The annotations for orders is @ManyToOne(mappedBy = “customer”)

```
@OneToMany(mappedBy = "customer") // @oneToMany because
one customer can make multiple orders, here we are making
it bidirectional means mappedBy is mandatory
private List<Orders> orders;
```

UML Diagram of oneToMany is shown below:




**Orders.java**

```
package com.tap.app;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "orders")
public class Orders {

	@Id
	@Column(name = "id")
	private int id;

	@Column(name = "item")
	private String item;

	@Column(name = "price")
	private int price;
```

```java
    @JoinColumn(name = "cust_id")
    @ManyToOne(cascade = CascadeType.ALL)
    private Customer customer;


    public Orders() {
    }

    public Orders(int id, String item, int price) {
         super();
         this.id = id;
         this.item = item;
         this.price = price;
    }

    public int getId() {
         return id;
    }

    public void setId(int id) {
         this.id = id;
    }

    public String getItem() {
         return item;
    }

    public void setItem(String item) {
         this.item = item;
    }

    public int getPrice() {
         return price;
    }
```

```
    public void setPrice(int price) {
        this.price = price;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    @Override
    public String toString() {
        return "Orders [" + id + ", " + item + ", " +
price + "]";
    }


}
```

**Customer.java**

```
package com.tap.app;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
```

```java
import javax.persistence.Table;

@Entity
@Table(name = "customer")
public class Customer {

    @Id
    @Column(name = "id")
    private int id;

    @Column(name = "name")
    private String name;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "cust_detail_id")
    private CustomerDetails customerDetails;

    @OneToMany(mappedBy = "customer")
    private List<Orders> orders;

    public Customer() {
    }


    public Customer(int id, String name) {
        super();
        this.id = id;
        this.name = name;
    }


    public int getId() {
        return id;
    }
```

```
    public void setId(int id) {
        this.id = id;
    }


    public String getName() {
        return name;
    }


    public void setName(String name) {
        this.name = name;
    }


    public CustomerDetails getCustomerDetails() {
        return customerDetails;
    }


    public void setCustomerDetails(CustomerDetails
customerDetails) {
        this.customerDetails = customerDetails;
    }


    public List<Orders> getOrders() {
        return orders;
    }


    public void setOrders(List<Orders> orders) {
        this.orders = orders;
    }
```

```
    @Override
    public String toString() {
        return  id + ", " + name + ", " +
customerDetails ;
    }



}
```

Now there is no orders table in the database, let's try to create an order object and set that order object to the customer.

**Logic:**

```
// creating order object
Orders o1 = new Orders(701, "Burger", 300);
Orders o2 = new Orders(702, "pizza", 350);
Orders o3 = new Orders(703, "Burger", 300);

//   fetch Customer object using get()
Customer bob = session.get(Customer.class, 1);

//   using setter method of customer in the orders class
set the orders for the customer
o1.setCustomer(bob);
o2.setCustomer(bob);
o3.setCustomer(bob);

//session.save() will help to Save into orders
Database(Into Orders Table)
System.out.println(session.save(o1));
System.out.println(session.save(o2));
System.out.println(session.save(o3));
```

**OneToMany.java**

```
package com.tap.app;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class OneToMany {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
                         .addAnnotatedClass(Orders.class)
                         .buildSessionFactory();
//             Create session
               session = sessionFactory.openSession();

//             Create Transaction
               Transaction transaction =
session.beginTransaction();
//             CRUD Operation
               Orders o1 = new Orders(701, "Burger", 300);
```

```
            Orders o2 = new Orders(702, "pizza", 350);
            Orders o3 = new Orders(703, "Burger", 300);

//          fetch Customer object
            Customer bob = session.get(Customer.class,
1);

//          Set customer To orders
            o1.setCustomer(bob);
            o2.setCustomer(bob);
            o3.setCustomer(bob);

//          Save To Database(Into Orders Table)
            System.out.println(session.save(o1));
            System.out.println(session.save(o2));
            System.out.println(session.save(o3));

            transaction.commit();
        } finally {
//          Closing Resources
            session.close();
            sessionFactory.close();

        }

    }

}
```

- Here we have created three orders object using
  Orders orders = new Orders(id, item, price);
- Next once we got customer object using get(Customer.class, 1) using which we have set order object to the customer using setter method

Output:

```
701
702
703
Hibernate: insert into orders (cust_id, item, price, id) values (?, ?, ?, ?)
Hibernate: insert into orders (cust_id, item, price, id) values (?, ?, ?, ?)
Hibernate: insert into orders (cust_id, item, price, id) values (?, ?, ?, ?)
```

**Orders table**

| id | item | price | cust_id |
|---|---|---|---|
| 701 | Burger | 300 | 1 |
| 702 | pizza | 350 | 1 |
| 703 | Burger | 300 | 1 |

**Scenario 4:**
we have understood how to assign orders to the existing customer, now we will create new user and for those user we will assign orders

Steps:

- Create new Customer and set customer details for that user
- Create orders object and assign that object to customers

**Logic:**

```
Customer customer = new Customer(3, "Charli"); // create
Customer object
CustomerDetails details = new CustomerDetails(501,
"charli@gmail.com", "9753234567", "bangalore");// create
CustomerDetails object

customer.setCustomerDetails(details); // set the
customerdetails object to customer

Orders o1 = new Orders(704, "Noodles", 300);
Orders o2 = new Orders(705, "pizza", 350);
Orders o3 = new Orders(706, "Noodles", 300);

//   Set customers object using setter of Customer
present in object class
o1.setCustomer(charli);
o2.setCustomer(charli);
o3.setCustomer(charli);

//Save To Database using session.save()
System.out.println(session.save(o1));
System.out.println(session.save(o2));
System.out.println(session.save(o3));
```

**OneToMany.java**

```
package com.tap.app;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class OneToMany {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
                         .addAnnotatedClass(Orders.class)
                         .buildSessionFactory();
//             Create session
               session = sessionFactory.openSession();

//             Create Transaction
               Transaction transaction =
session.beginTransaction();
//             CRUD Operation

               Customer customer = new Customer(3,
"Charli");
```

```
            CustomerDetails details = new
CustomerDetails(501, "charli@gmail.com", "9753234567",
"bangalore");
            customer.setCustomerDetails(details);

            Orders o1 = new Orders(704, "Noodles",
300);
            Orders o2 = new Orders(705, "pizza", 350);
            Orders o3 = new Orders(706, "Noodles",
300);

//            fetch Customer object
            Customer charli =
session.get(Customer.class, 3);

//            Set customer To orders
            o1.setCustomer(charli);
            o2.setCustomer(charli);
            o3.setCustomer(charli);

//            Save To Database(Into Orders Table)
            System.out.println(session.save(o1));
            System.out.println(session.save(o2));
            System.out.println(session.save(o3));

            transaction.commit();
        } finally {
//            Closing Resources
            session.close();
            sessionFactory.close();
        }
    }
}
```

Output:

```
704
705
706
Hibernate: insert into customer_details (address, email, phone, id) values (?, ?, ?, ?)
Hibernate: insert into customer (cust_detail_id, name, id) values (?, ?, ?)
Hibernate: insert into orders (cust_id, item, price, id) values (?, ?, ?, ?)
Hibernate: insert into orders (cust_id, item, price, id) values (?, ?, ?, ?)
Hibernate: insert into orders (cust_id, item, price, id) values (?, ?, ?, ?)
```

Database:

**Customer Table:**

| id | name | cust_detail_id |
|---|---|---|
| 1 | bob | 501 |
| 2 | alex | 502 |
| 3 | charli | 503 |

**CustomerDetails Table:**

| id | address | email | phone |
|---|---|---|---|
| 501 | Bangalore | bob@gmail.com | 8975323456 |
| 502 | Chennai | alex@gmail.com | 9845678123 |
| 503 | Bangalore | charli@gmail.com | 9753234567 |

**Orders Table:**

| id | item | price | cust_id |
|---|---|---|---|
| 701 | Burger | 300 | 1 |
| 702 | Burger | 300 | 1 |
| 703 | Pizza | 370 | 1 |

| | | | |
|---|---|---|---|
| 704 | Noodles | 300 | 3 |
| 705 | pizza | 350 | 3 |
| 706 | Noodles | 300 | 3 |

**Scenario 5:**

Now we have understood how to insert data into the database, now let's move ahead and understand how to get the data from database
Steps:

- Get the Customer object using Session.get()
- Using Getter of customer details and orders in customer class you can get the object of customer details and orders

**Logic:**

```
Customer customer = session.get(Customer.class, 3); //
get the customer object using session.get()

CustomerDetails details = customer.getCustomerDetails();
// using getter method of customerdetails get customer
details object

List<Orders> orders = customer.getOrders(); // using
getter method of orders object get the order object

System.out.println(customer);
System.out.println(details);

for (Orders order : orders) {
    System.out.println(order);
} // print all list of orders
```

**OneToMany.java**

```
package com.tap.app;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class OneToMany {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
                         .addAnnotatedClass(Orders.class)
                         .buildSessionFactory();
//             Create session
               session = sessionFactory.openSession();

//             Create Transaction
               Transaction transaction =
session.beginTransaction();
//             CRUD Operation

               Customer customer =
session.get(Customer.class, 3);
```

```
            CustomerDetails details =
customer.getCustomerDetails();
            List<Orders> orders = customer.getOrders();


            System.out.println(customer);
            System.out.println(details);

            for (Orders order : orders) {
                System.out.println(order);
            }

            transaction.commit();
        } finally {
//            Closing Resources
            session.close();
            sessionFactory.close();

        }

    }

}
```

**Output:**

```
3, Charli
CustomerDetails [503, charli@gmail.com, 9753234567, bangalore]
Orders [704, Noodles, 300]
Orders [705, pizza, 350]
Orders [706, Noodles, 300]
```

**Scenario 6:**
Now let's try to delete customer for the table below
**Customer Table:**

| id | name | cust_detail_id |
|---|---|---|
| 1 | bob | 501 |
| 2 | alex | 502 |
| 3 | charli | 503 |

**CustomerDetails Table:**

| id | address | email | phone |
|---|---|---|---|
| 501 | Bangalore | bob@gmail.com | 8975323456 |
| 502 | Chennai | alex@gmail.com | 9845678123 |
| 503 | Bangalore | charli@gmail.com | 9753234567 |

**Orders Table:**

| id | item | price | cust_id |
|---|---|---|---|
| 701 | Burger | 300 | 1 |
| 702 | Burger | 300 | 1 |
| 703 | Pizza | 370 | 1 |
| 704 | Noodles | 300 | 3 |
| 705 | pizza | 350 | 3 |
| 706 | Noodles | 300 | 3 |

**Logic:**

```
Customer customer = session.get(Customer.class, 3); //
Get the customer object using session.get()
session.delete(customer); // Delete the customer object,
associated Customer object and orders will get deleted
```

**OneToMany.java**

```
package com.tap.app;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class OneToMany {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Customer.class)

.addAnnotatedClass(CustomerDetails.class)
                         .addAnnotatedClass(Orders.class)
                         .buildSessionFactory();
```

```
//            Create session
              session = sessionFactory.openSession();

//            Create Transaction
              Transaction transaction =
session.beginTransaction();
//            CRUD Operation

              Customer customer =
session.get(Customer.class, 3);
              session.delete(customer);

              transaction.commit();
          } finally {
//              Closing Resources
              session.close();
              sessionFactory.close();

          }

     }

}
```

**Output:**
**Customer Table:**

| id | name | cust_detail_id |
|---|---|---|
| 1 | bob | 501 |
| 2 | alex | 502 |

**CustomerDetails Table:**

| id | address | email | phone |
|---|---|---|---|
| 501 | Bangalore | bob@gmail.com | 8975323456 |
| 502 | Chennai | alex@gmail.com | 9845678123 |

**Orders Table:**

| id | item | price | cust_id |
|---|---|---|---|
| 701 | Burger | 300 | 1 |
| 702 | Burger | 300 | 1 |
| 703 | Pizza | 370 | 1 |

If you can observe from the above table if a customer is deleted orders done by that customer are also deleted.

**Scenario 6:**
Now let's see how we can delete the customer, but orders should not get deleted

For that let's understand CascadeType.ALL. The meaning of CascadeType.ALL, is that the persistence will propagate (cascade) all EntityManager operations (PERSIST, REMOVE, REFRESH, MERGE, DETACH) to the related entities. So in cascadeType.all remove is also which will remove all related objects as well so to avoid order objects not to get deleted we need to specify all the cascade Type expect remove as shown below

```
@OneToMany(mappedBy = "customer",
cascade= {CascadeType.DETACH,CascadeType.MERGE,
CascadeType.PERSIST, CascadeType.REFRESH})
private List<Orders> orders;
```

**Logic:**

```
Customer customer = session.get(Customer.class, 1); // get the
customer object using getter method
session.delete(customer); // delete the customer object
```

**OneToMany.java**

```
package com.tap.app;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class OneToMany {

      public static void main(String[] args) {
            SessionFactory sessionFactory = null;
            Session session = null;

            try {
//                Create session Factory
```

```
                sessionFactory = new Configuration()
                            .configure()
                            .addAnnotatedClass(Customer.class)
                            .addAnnotatedClass(CustomerDetails.class)
                            .addAnnotatedClass(Orders.class)
                            .buildSessionFactory();
//              Create session
                session = sessionFactory.openSession();

//              Create Transaction
                Transaction transaction =
session.beginTransaction();
//              CRUD Operation

                Customer customer = session.get(Customer.class, 1);
                session.delete(customer);

                transaction.commit();
            } finally {
//              Closing Resources
                session.close();
                sessionFactory.close();

            }

      }

}
```

**Customer Table:**

| id | name | cust_detail_id |
|---|---|---|
| 2 | alex | 502 |

**CustomerDetails Table:**

| id | address | email | phone |
|---|---|---|---|
| 502 | Chennai | alex@gmail.com | 9845678123 |

**Orders Table:**

| id | item | price | cust_id |
|---|---|---|---|
| 701 | Burger | 300 | 1 |
| 702 | Burger | 300 | 1 |
| 703 | Pizza | 370 | 1 |

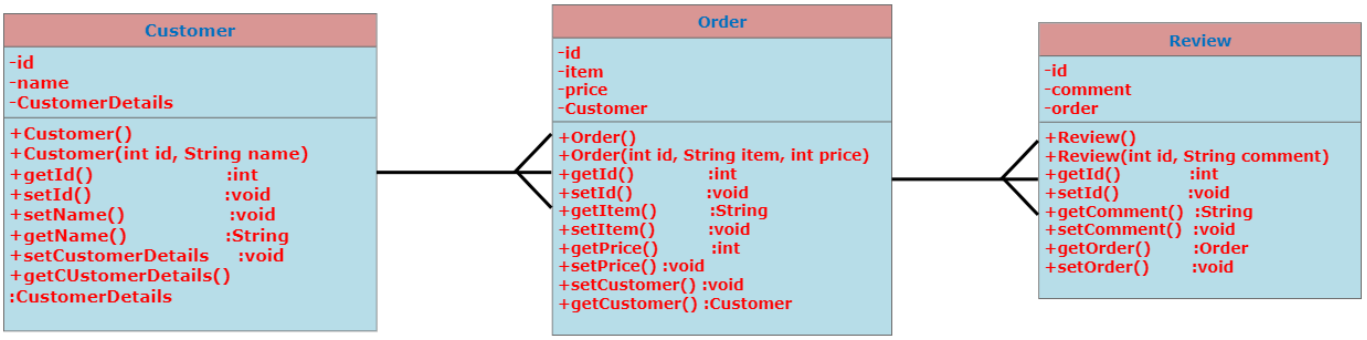
**Scenario 7:**

Now Customers can give reviews for the orders, meaning there can be multiple customer reviews for the same order.

| id | name | cust_detail_id |
|---|---|---|
| 1 | alex | 501 |
| 2 | bob | 502 |

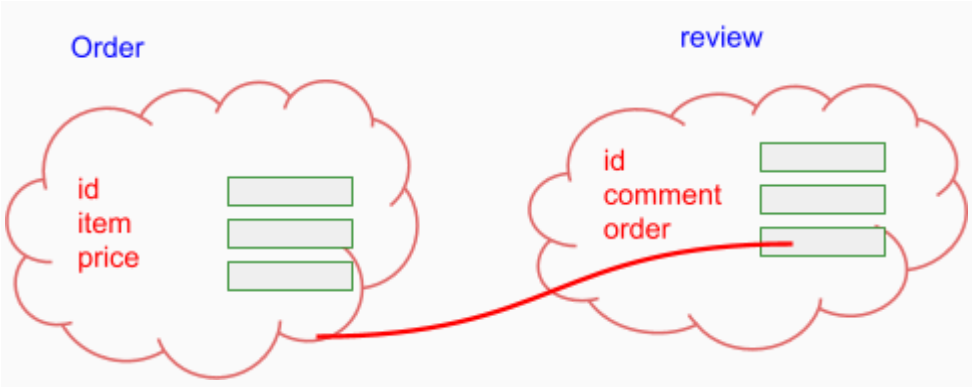
| burger | item | price | cust_id |
|---|---|---|---|
| 701 | burger | 300 | 2 |
| 702 | burger | 300 | 1 |
| 703 | pizza | 370 | 2 |
| 704 | chapati | 150 | 2 |

| id | review | order_id |
|---|---|---|
| 801 | Very Tasty Food | 701 |
| 802 | Good Quantity | 701 |
| 803 | Not Cooked Properly | 701 |




Steps:

- Now let's create a **POJO** class **review** with id, comment and orderId which is referring to the primary key of order class.



- We need to mention **@JoinColumn** annotation and also **@ManyToOne** annotations as many reviews belong to one orders

```
    @JoinColumn(name = "order_id") // @JoinColumn
annotations will join two tables
     @ManyToOne(cascade = CascadeType.ALL)//@manyToOne
means multiple rows in the review table is referring to
one row in Order table
     private Orders orders;
```

**Review.java**

```
package com.tap.app;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "review")
public class Review {

        @Id
        @Column(name = "id")
        private int id;

       @Column(name = "comment")
        private String comment;

        @JoinColumn(name = "order_id")
        @ManyToOne(cascade = CascadeType.ALL)
        private Orders orders;
```

```
public Review() {
}

public Review(int id, String comment) {
        super();
        this.id = id;
        this.comment = comment;
}

public int getId() {
        return id;
}

public void setId(int id) {
        this.id = id;
}

public String getComment() {
        return comment;
}

public void setComment(String comment) {
        this.comment = comment;
}

public Orders getOrders() {
        return orders;
}

public void setOrders(Orders orders) {
        this.orders = orders;
}
```

```
	@Override
	public String toString() {
		return "Review ["+ id + ", " + comment + "]";
	}

}
```

- Next from orders if we want to extract reviews then we need to create object reference of review in orders class.
- We need to mention **@OneToMany** annotations as one order can have many reviews

```
@OneToMany(mappedBy = "orders")
private List<Review> review;
```

**Orders.java**

```
package com.tap.app;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.MappedSuperclass;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "orders")
```

```java
public class Orders {

	@Id
	@Column(name = "id")
	private int id;

	@Column(name = "item")
	private String item;

	@Column(name = "price")
	private int price;


	@JoinColumn(name = "cust_id")
	@ManyToOne(cascade = CascadeType.ALL)
	private Customer customer;

	@OneToMany(mappedBy = "orders")
	private List<Review> review;


	public Orders() {
	}

	public Orders(int id, String item, int price) {
		super();
		this.id = id;
		this.item = item;
		this.price = price;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
```

```
    }

    public String getItem() {
        return item;
    }

    public void setItem(String item) {
        this.item = item;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public List<Review> getReview() {
        return review;
    }

    public void setReview(List<Review> review) {
        this.review = review;
    }

    @Override
    public String toString() {
        return "Orders [" + id + ", " + item + ", " + price + "]";
```

```
    }
}
```

- Now let's try to insert reviews for orders

  - ☐ First create three review object
  - ☐ Next get the order object
  - ☐ Set the review for the orders

```
//create three review object
Review r1 = new Review(801, "Very Tasty Food");
Review r2 = new Review(802, "Good Quantity");
Review r3 = new Review(803, "Not cooked properly");

// save review object into the database using session.save()
session.save(r1);
session.save(r2);
session.save(r3);

// get the order object using session.get()
Orders o1 = session.get(Orders.class, 701);
Orders o2 = session.get(Orders.class, 703);

//set the orders for review using setters of order object
present in review
r1.setOrders(o1);
r2.setOrders(o2);
r3.setOrders(o1);
```

**OneToMany.java**

```
package com.tap.app;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class OneToMany {

	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Session session = null;
		try {
//			Create session Factory
			sessionFactory = new Configuration()
					.configure()
					.addAnnotatedClass(Customer.class)
					.addAnnotatedClass(CustomerDetails.class)
					.addAnnotatedClass(Orders.class)
					.addAnnotatedClass(Review.class)
					.buildSessionFactory();
//			Create session
			session = sessionFactory.openSession();

//			Create Transaction
			Transaction transaction = session.beginTransaction();
//			CRUD Operation
			Review r1 = new Review(801, "Very Tasty Food");
			Review r2 = new Review(802, "Good Quantity");
			Review r3 = new Review(803, "Not cooked properly");

			session.save(r1);
			session.save(r2);
			session.save(r3);
```

```
                Orders o1 = session.get(Orders.class, 701);
                Orders o2 = session.get(Orders.class, 703);

                r1.setOrders(o1);
                r2.setOrders(o2);
                r3.setOrders(o1);

                transaction.commit();
            } finally {
//                  Closing Resources
                session.close();
                sessionFactory.close();

            }

        }

}
```

**Output:**
**Review table**

| id | comment | order_id |
|---|---|---|
| 801 | Very Tasty Food | 701 |
| 802 | Good Quantity | 703 |
| 803 | Not cooked properly | 701 |

**Scenario 7:**

Now let's see how to get the reviews of particular orders

- Get the order object
- Using order object get the list of reviews

Logic:

```
Orders orders = session.get(Orders.class, 701);//get
order object using session.get()

     List<Review> list = orders.getReview(); // get the
reviews of that particular object using getter method of
review present in order class

     for (Review review : list) {
          System.out.println(review);
     } // print list of review
```

**OneToMany.java**

```
package com.tap.app;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class OneToMany {

        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
```

```
		Session session = null;

		try {
//			Create session Factory

			sessionFactory = new Configuration()
					.configure()
					.addAnnotatedClass(Customer.class)
					.addAnnotatedClass(CustomerDetails.class)
					.addAnnotatedClass(Orders.class)
					.addAnnotatedClass(Review.class)
					.buildSessionFactory();
//			Create session
			session = sessionFactory.openSession();

//			Create Transaction
			Transaction transaction = session.beginTransaction();
//			CRUD Operation

			Orders orders = session.get(Orders.class, 701);
			List<Review> list = orders.getReview();

			for (Review review : list) {
				System.out.println(review);
			}

			transaction.commit();
		} finally {
//			Closing Resources
			session.close();
			sessionFactory.close();

		}

	}

}
```

Output:

```
Review [801, Very Tasty Food]
Review [803, Not cooked properly]
```