@OneToMany

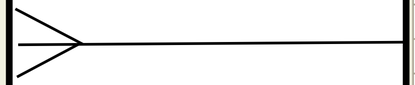
The tables

One CUSTOMER can have many PURCHASE_ORDERs

SELECT * FROM PURCHASE_ORDER;

ID	LOCAL_DATE	ORDER_AMOUNT	PRODUCT	QUANTITY	CUSTOMER_ID
1	2022-05-17	11234.00	Iphone 8697	10	1
2	2022-05-17	500.00	A bag of cats	10	1
3	2022-05-10	1124.00	Iphone 1000	1	2
4	2022-04-24	1234.00	A boat	1	2
5	2022-04-24	950.00	Some books	100	3
6	2021-05-17	50.00	Some books	10	3
7	2022-03-17	211234.00	Porsche 911	1	4
8	2022-05-17	1124.00	Iphone 1000	1	4
9	2021-05-17	5.00	A bag of potatoes	10	5
10	2022-04-12	11230.00	Samsung A500	10	5

(10 rows, 2 ms)



SELECT * FROM CUSTOMER;

ID	CUSTOMER	CUSTOMER_ADDRESS	CUSTOMER_PHONE	TAXID
1	Zazil Erhard	2435 James Avenue, Wichita	207-494-7828	AI4HD93JD
2	Knútr Madailéin	1408 Karen Lane, Louisville	816-730-7821	89DJHW3ER
3	Oleg Abhinav	1283 Cottonwood Lane, Grand Rapids	816-730-7821	S983JDI3
4	Tedore Marcelino	472 Golf Course Drive, Reston	816-730-7821	3DD93JD3
5	Darina Ragna	3907 Sarah Drive, Lafayette	816-730-7821	D83DN39

(5 rows, 1 ms)

The classes

```java
@Entity
public class PurchaseOrder {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private LocalDate localDate;
    private String product;
    private int quantity;
    private BigDecimal orderAmount;

    @ManyToOne
    private Customer customer;

    // Constructors, getters and setters
}
```

```java
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String customer;
    private String customerAddress;
    private String customerPhone;
    private String taxID;

    /*
        @OneToMany defines the relationship.
        mappedBy attribute: indicates that the entity on this side is the inverse of the relationship.
        The owner (the entity that has the foreign key) is PurchaseOrder.

        cascade = CascadeType.ALL attribute: Hibernate will propagate all actions
        e.g., in DBSeeder, we save a customer with a List of PurchaseOrders - we don't need to save them manually as
        Hibernate will make sure all the orders from the list will be saved to the corresponding table.
    */
    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<PurchaseOrder> orders;

    // Constructors, getters and setters
}
```

The classes & the tables

SELECT * FROM PURCHASE_ORDER;

ID	LOCAL_DATE	ORDER_AMOUNT	PRODUCT	QUANTITY	CUSTOMER_ID
1	2022-05-17	11234.00	Iphone 8697	10	1
2	2022-05-17	500.00	A bag of cats	10	1
3	2022-05-10	1124.00	Iphone 1000	1	2
4	2022-04-24	1234.00	A boat	1	2
5	2022-04-24	950.00	Some books	100	3
6	2021-05-17	50.00	Some books	10	3
7	2022-03-17	211234.00	Porsche 911	1	4
8	2022-05-17	1124.00	Iphone 1000	1	4
9	2021-05-17	5.00	A bag of potatoes	10	5
10	2022-04-12	11230.00	Samsung A500	10	5

(10 rows, 2 ms)

SELECT * FROM CUSTOMER;

ID	CUSTOMER	CUSTOMER_ADDRESS	CUSTOMER_PHONE	TAXID
1	Zazil Erhard	2435 James Avenue, Wichita	207-494-7828	AI4HD93JD
2	Knútr Madailéin	1408 Karen Lane, Louisville	816-730-7821	89DJHW3ER
3	Oleg Abhinav	1283 Cottonwood Lane, Grand Rapids	816-730-7821	S983JDI3
4	Tedore Marcelino	472 Golf Course Drive, Reston	816-730-7821	3DD93JD3
5	Darina Ragna	3907 Sarah Drive, Lafayette	816-730-7821	D83DN39

(5 rows, 1 ms)

PURCHASE_ORDER has a Foreign Key to CUSTOMER

```java
@Entity
public class PurchaseOrder {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private LocalDate localDate;
    private String product;
    private int quantity;
    private BigDecimal orderAmount;

    @ManyToOne
    private Customer customer;

    // Constructors, getters and setters
}
```

```java
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String customer;
    private String customerAddress;
    private String customerPhone;
    private String taxID;

    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<PurchaseOrder> orders;

    // Constructors, getters and setters
}
```

DBSeeder

One of many ways to populate the database

```java
/*
  A configuration class that implements ApplicationRunner
  Long story short - we put come code in the run method to populate the DB after Spring App is ready.
*/
@Configuration
public class DBSeeder implements ApplicationRunner {
    /*
       Spring 'glues' DBSeder with the implementation of CustomerRepository
       provided by the Spring Data Jpa project
    */
    @Autowired
    private CustomerRepository customerRepository;

    @Override
    public void run(ApplicationArguments args) throws Exception {
        Customer c1 = new Customer( id: null,  customer: "Zazil Erhard", customerAddress: "2435 James Avenue, Wichita", customerPhone: "207-494-7828", taxID: "AI4HD93JD",  orders: null);
        Customer c2 = new Customer( id: null,  customer: "Knútr Madailéin", customerAddress: "1408 Karen Lane, Louisville", customerPhone: "816-730-7821", taxID: "89DJHW3ER",  orders: null);
        Customer c3 = new Customer( id: null,  customer: "Oleg Abhinav", customerAddress: "1283 Cottonwood Lane, Grand Rapids", customerPhone: "816-730-7821", taxID: "S983JDI3",  orders: null);
        Customer c4 = new Customer( id: null,  customer: "Tedore Marcelino", customerAddress: "472 Golf Course Drive, Reston", customerPhone: "816-730-7821", taxID: "3DD93JD3",  orders: null);
        Customer c5 = new Customer( id: null,  customer: "Darina Ragna", customerAddress: "3907 Sarah Drive, Lafayette", customerPhone: "816-730-7821", taxID: "D83DN39",  orders: null);

        PurchaseOrder o1= new PurchaseOrder( id: null, LocalDate.now(),  product: "Iphone 8697",  quantity: 10, BigDecimal.valueOf(11234), c1);
        PurchaseOrder o11= new PurchaseOrder( id: null, LocalDate.now(),  product: "A bag of cats",  quantity: 10, BigDecimal.valueOf(500), c1);
        PurchaseOrder o2= new PurchaseOrder( id: null, LocalDate.now().minusWeeks( weeksToSubtract: 1),  product: "Iphone 1000",  quantity: 1, BigDecimal.valueOf(1124), c2);
        PurchaseOrder o22= new PurchaseOrder( id: null, LocalDate.now().minusDays( daysToSubtract: 23),  product: "A boat",  quantity: 1, BigDecimal.valueOf(1234), c2);
        PurchaseOrder o3= new PurchaseOrder( id: null, LocalDate.now().minusDays( daysToSubtract: 23),  product: "Some books",  quantity: 100, BigDecimal.valueOf(950), c3);
        PurchaseOrder o33= new PurchaseOrder( id: null, LocalDate.now().minusYears( yearsToSubtract: 1),  product: "Some books",  quantity: 10, BigDecimal.valueOf(50), c3);
        PurchaseOrder o4= new PurchaseOrder( id: null, LocalDate.now().minusMonths( monthsToSubtract: 2),  product: "Porsche 911",  quantity: 1, BigDecimal.valueOf(211234), c4);
        PurchaseOrder o44= new PurchaseOrder( id: null, LocalDate.now(),  product: "Iphone 1000",  quantity: 1, BigDecimal.valueOf(1124), c4);
        PurchaseOrder o5= new PurchaseOrder( id: null, LocalDate.now().minusYears( yearsToSubtract: 1),  product: "A bag of potatoes",  quantity: 10, BigDecimal.valueOf(5), c5);
        PurchaseOrder o55= new PurchaseOrder( id: null, LocalDate.now().minusWeeks( weeksToSubtract: 5),  product: "Samsung A500",  quantity: 10, BigDecimal.valueOf(11230), c5);

        c1.setOrders(List.of(o1,o11));
        c2.setOrders(List.of(o2,o22));
        c3.setOrders(List.of(o3,o33));
        c4.setOrders(List.of(o4,o44));
        c5.setOrders(List.of(o5,o55));
        /*
            Thanks to the 'cascade' attribute we save explicitly only customers

            @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
            private List<PurchaseOrder> orders;
         */
        customerRepository.saveAll(List.of(c1,c2,c3,c4,c5));

    }
}
```

Some tests :)

```java
/*
    This test class is very similar to a pure JUnit test, although it has some additional superpowers.

      @SpringBootTest - start a Spring context before the tests and
      @Autowired    - inject the repositories inside the test so that we can call/ test some methods

      H2 database is populated with DBSeeder before the tests.

 */
@SpringBootTest
class OneToManyTest {

    @Autowired
    private PurchaseOrderRepository orderRepository;
    @Autowired
    private CustomerRepository  customerRepository;


    @Test
    public void customerRepository_findAll_shouldReturn5Customers() {
        assertEquals( expected: 5, customerRepository.findAll().size());
    }

    @Test
    public void orderRepository_findAll_shouldReturn10Orders() {
        assertEquals( expected: 10, orderRepository.findAll().size());
    }

    @Test
    /*
        We need @Transactional to allow Hibernate to load the orders
        Please note that CUSTOMER table does not have the id to PURCHASE_ORDER

        Comment out  @Transactional to see what will happen
     */
    @Transactional
    public void customerWithId1_shouldHaveAnIphoneAndABagOfCats() {
        Customer customer1 = customerRepository.findById(1l).get();

        assertAll( heading: "Customr with id 1 should have 2 orders: an iphone and a bag of cats",
            ()-> assertEquals( expected: 2, customer1.getOrders().size()),
            ()-> assertEquals( expected: "Iphone 8697", customer1.getOrders().get(0).getProduct()),
            ()-> assertEquals( expected: "A bag of cats", customer1.getOrders().get(1).getProduct())
        );
    }

    @Test
    public void orderWitId1_shouldHaveCustomerWithId1() {
        PurchaseOrder purchaseOrder1 = orderRepository.findById(1l).get();
        assertEquals( expected: 1, purchaseOrder1.getCustomer().getId());
    }

}
```