Section 1 introduction

Section 2 ORM Basic

Section 3 simple crud operation

Section 4 generators

Section 5 spring data finder

Section 6 paging and sorting

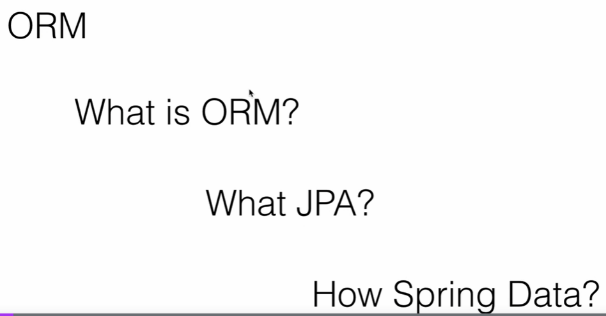
Section 7 jpql

Section 8 paging and sorting with jpql

Section 9 native sql query

Section 10 inheritance

Section 11 component mapping



**Spring boot properties**

<https://docs.spring.io/spring-boot/docs/1.4.x/reference/html/common-application-properties.html>

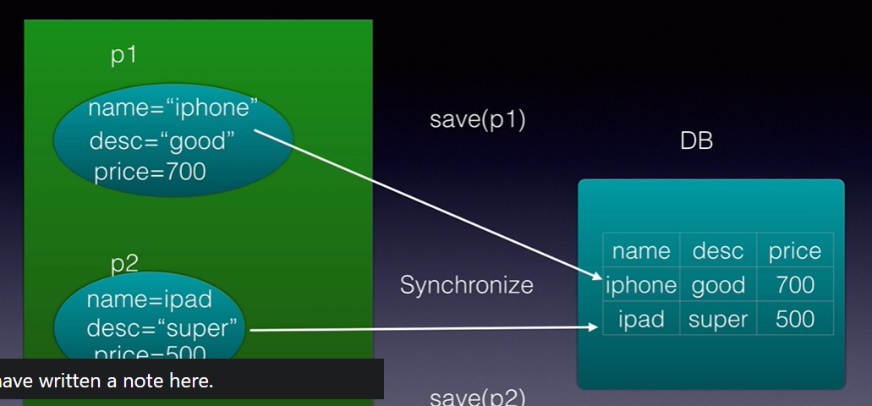
**ORM basics**

**ORM :** it is a process of mapping java class to a d/b table and its field or members to a d/b table



**Note :**

* once we do mapping we can synchronize object into rows
* We deals with object here



**1--------- JPA:** java persistence API.

JPA is just a specification, not an ORM tool. JPA is a set of classes, interfaces that helps your ORM tool to map O-R according to JPA standards. You can use Hibernate, TopLink/EclipseLink etc to have JPA implemented in your application.

You do not necessarily need JPA if you are using Hibernate in Persistence Layer of your application. But if you use JPA, it makes switching to other ORM tool easier if you plan to switch in future because the standards remain common.

(*if you remember, you do*import javax.persistence.\*;*when you use annotations for OR mapping (like @Id, @Column, @GeneratedValue etc.) in Hibernate, that's where you are using JPA under Hibernate, you can*

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

it is standard from oracle to perform orm in java ee application

**JPA come with specification and API**

**Providers :-** hibernate , open jpa, eclipse link which implement jpa api by following specification

Specification :- set of rules written in plain english

**NOTE:** now we need to learn only JPA api and all other(Hibernate, open jpa,EclipseLink )vendor using it and we can switch any other

**Hibernate is most popular jpa provider**

**Important classes in JPA api**

EntityManagerFactory

EntityManager

Map java classes to database table

@Entity

@Table

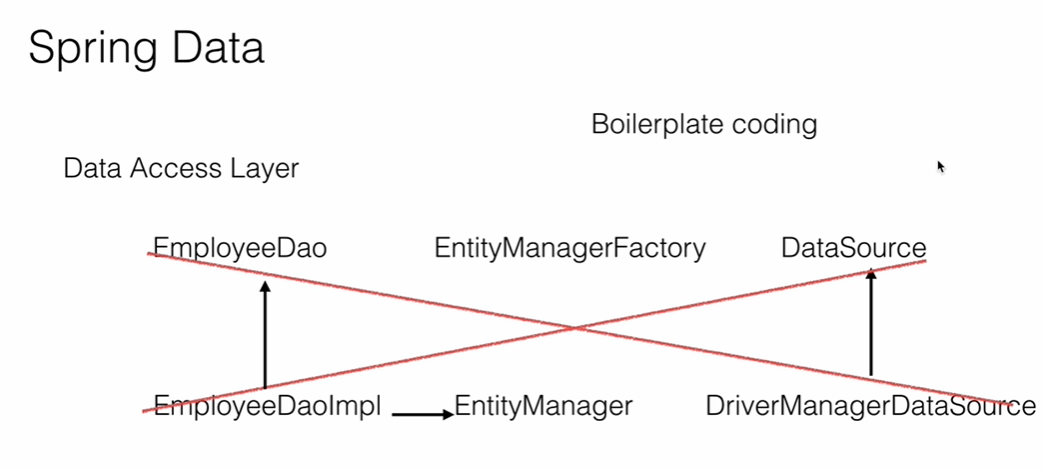
Map field to database table field

@id @Column

**Note:**

jpa is a standard for performing orm in java ee and Hibernate, open jpa,EclipseLink

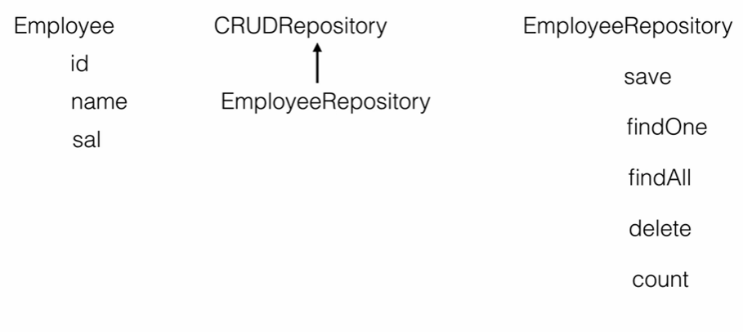
Are providers or implementation of JPA api



Spring data says don’t do all this and avoid boilerplate coding and configuration and I will take care of everything

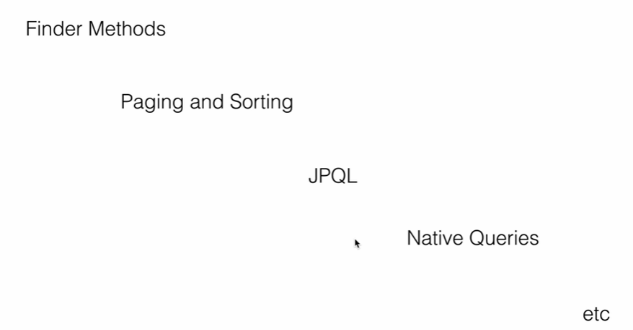
**Note:**

* in two simple steps we implement d/b access layers
* Internally spring data will use jpa , EntityManagerFactory nad EntityManager





Spring data also provides



Spring Data JPA brings in a concept of JPA Repositories a set of Interfaces which defines query methods.

Simple Crud Operation

**Use case**

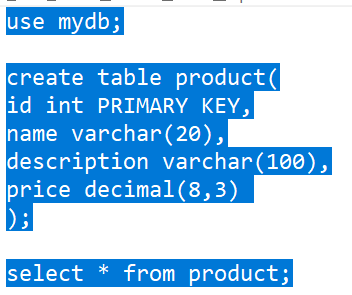
Product product-db

Product Repository

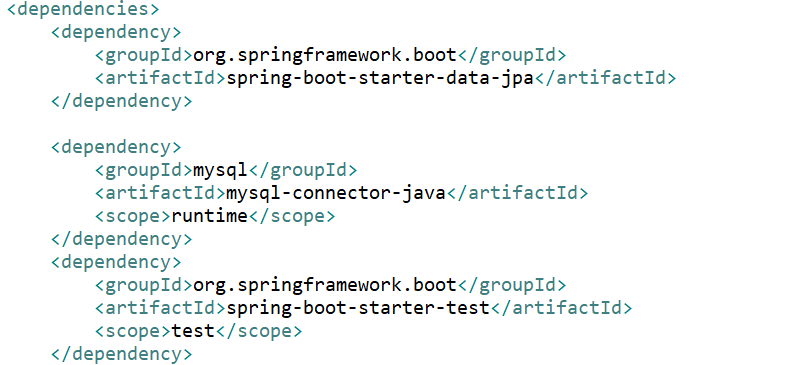
Configure the data source

Write test to perform crud operation

**Step1**



**Step 2--** add mysql and jpa dependency



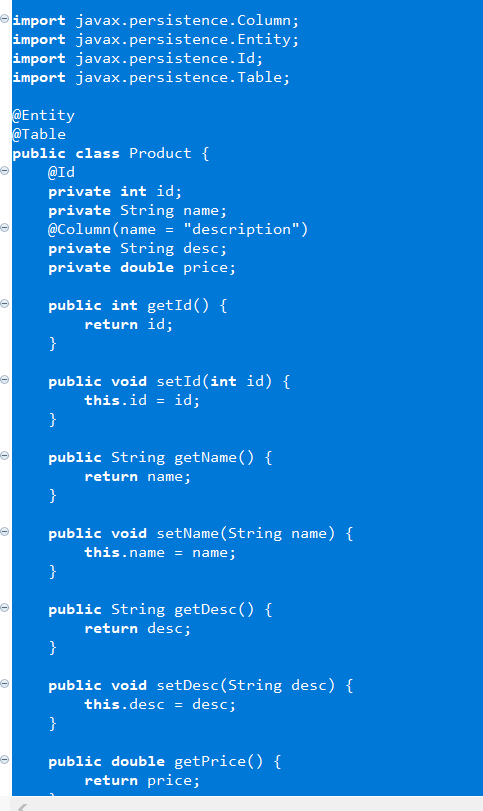
**Step 3-**

* Create the entity
* create the product class and mark it with annotation

**Note:**

product class should be mapped with database table Product

Since column name is different in d/b so we need to mark it as @Column

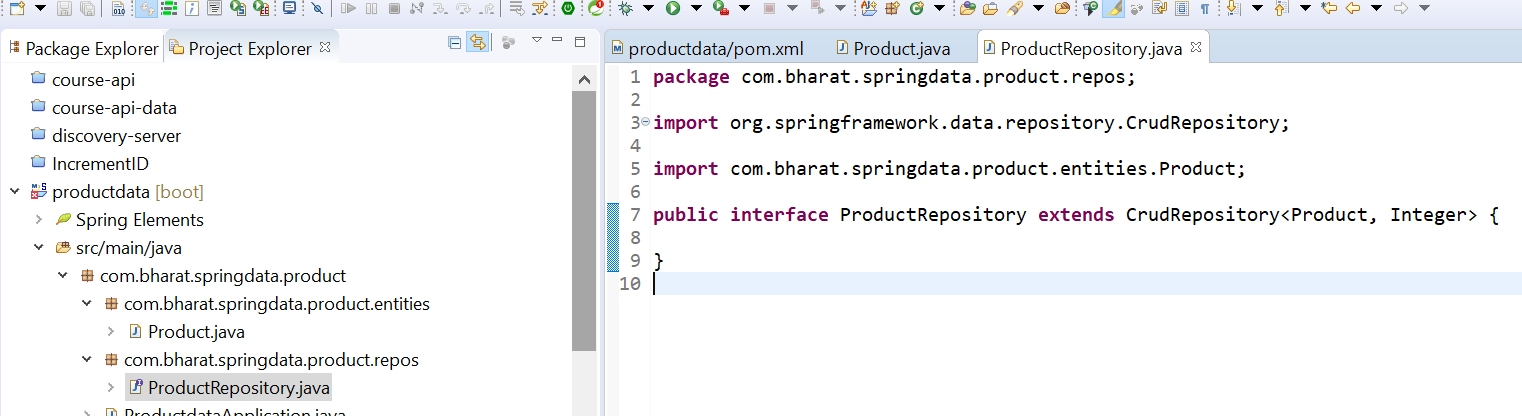


**Step 4 :-**

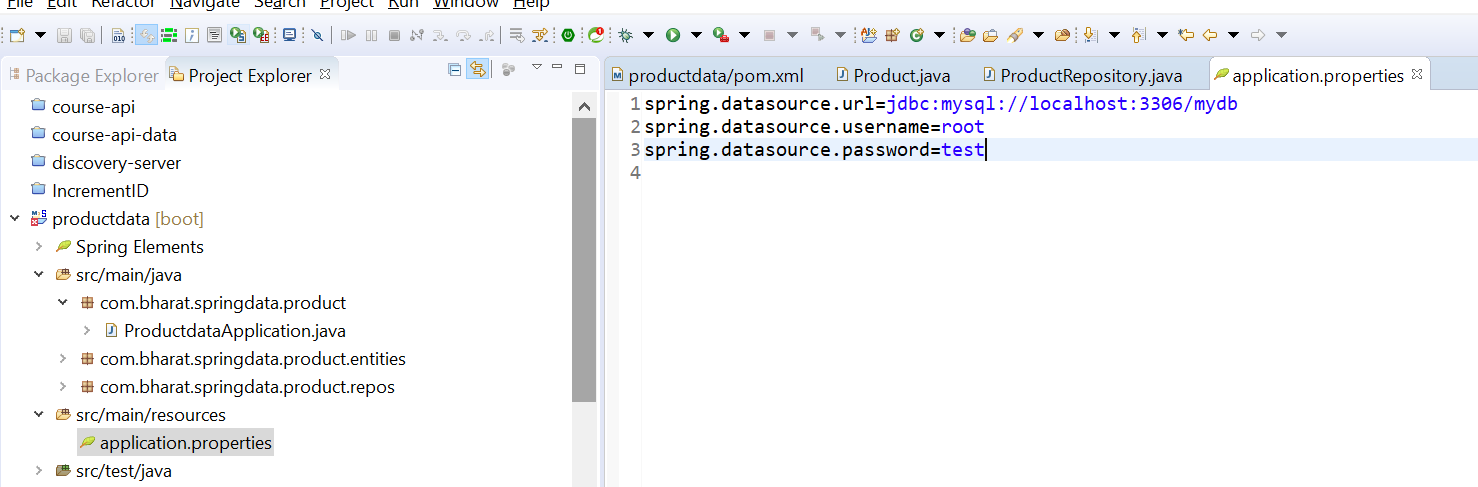
**Create the repository**

First parameter---name of entity class

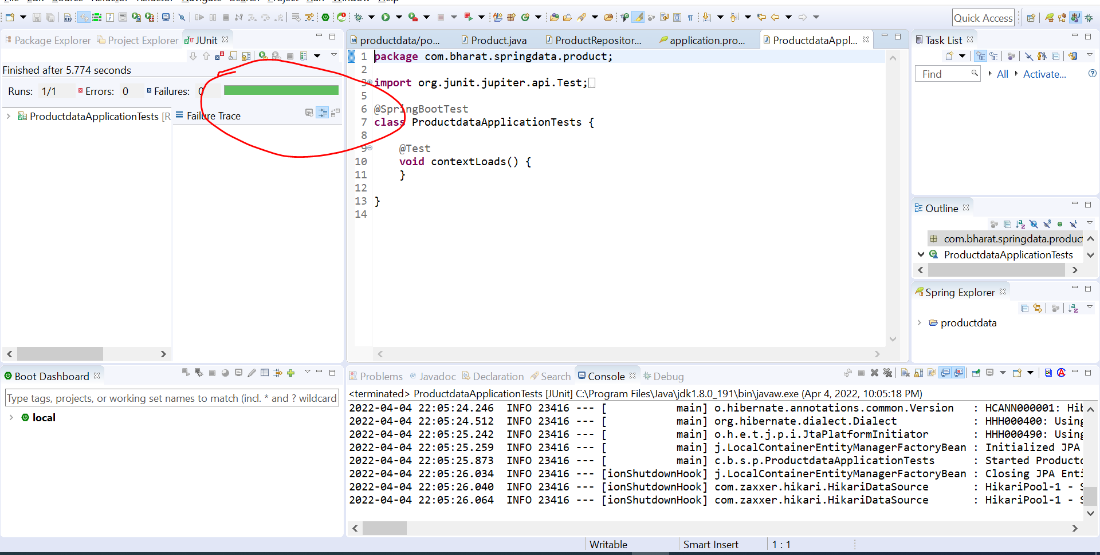
Second parameter --- data type of id



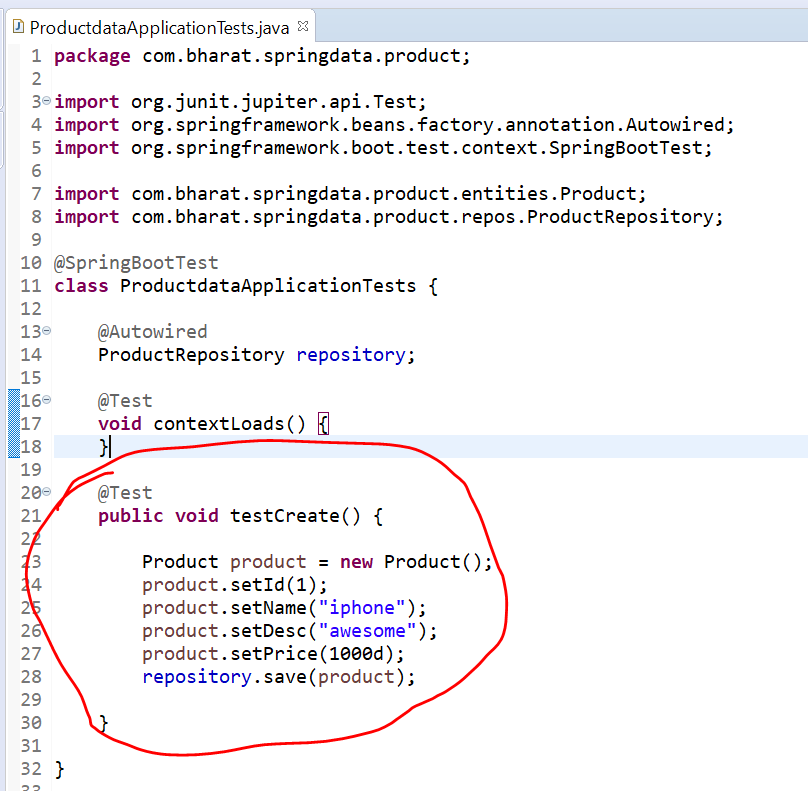
**Step 5:--- configure data source**

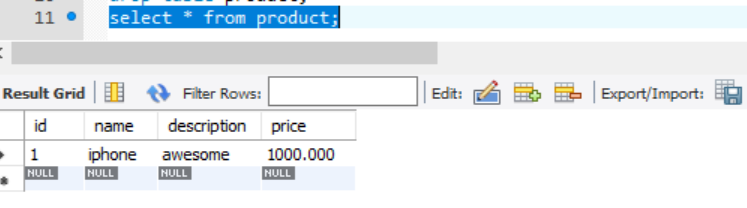


**Note:-** although no test to run but if all wiring done properly it will turn into green and no error will come



**Create product row in d/b**



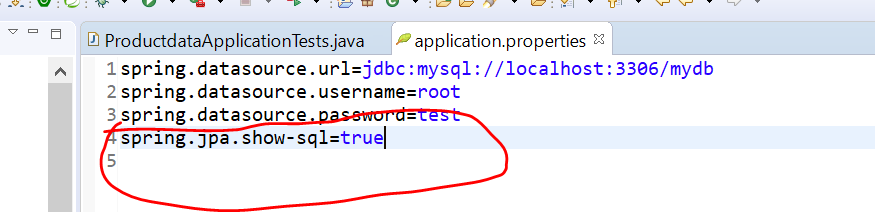


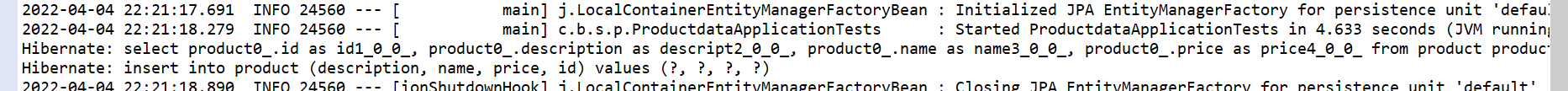
**Note:**

When repository Save is invoked the spring data jpa using hibernate behind the scene

Will generate a insert query. It will be helpful if we see the query which is generated

By default it is false but if we use this property then will be able to see query





**First it will run select query if data is not there then it will insert record**

**Read operation**



We need to use get() to get back Product

Update record

**Note:**

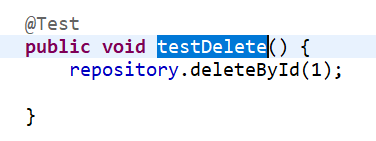
* before update we need to fetch the record
* We can update any field\

**Save()----** method will use for update also . If entity not there it will create it,

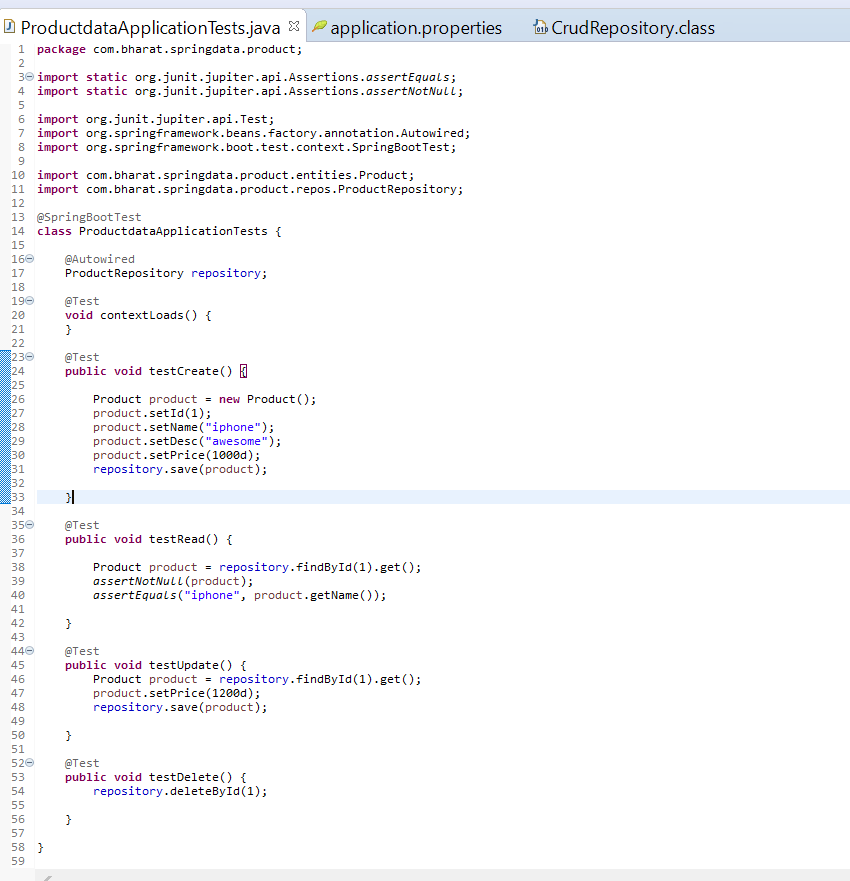
If entity is there it will update it



Delete



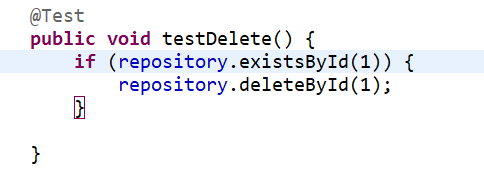
===================================================



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

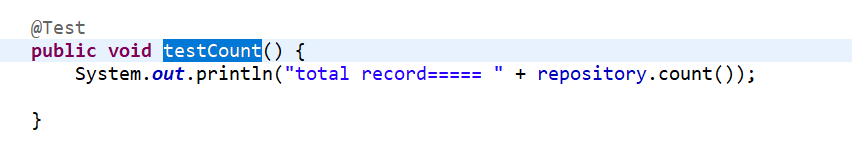
**Note:--- internally spring invoke save method on EntityManager when we call repository.save() ----** so we avoid boilerplate code

**existsById:**----- if record is present it will return true



Count

**Count** method on repository tell us total number of rows present in table



ID Generators

**Note:---** in real time we don’t hard code primary key value instead we generate it

Four different types of id generation strategy

GenerationType.AUTO

GenerationType.IDENTITY

GenerationType.SEQUENCE

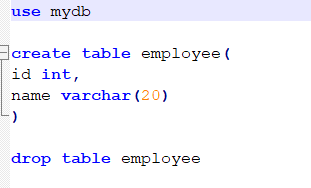
GenerationType.TABLE

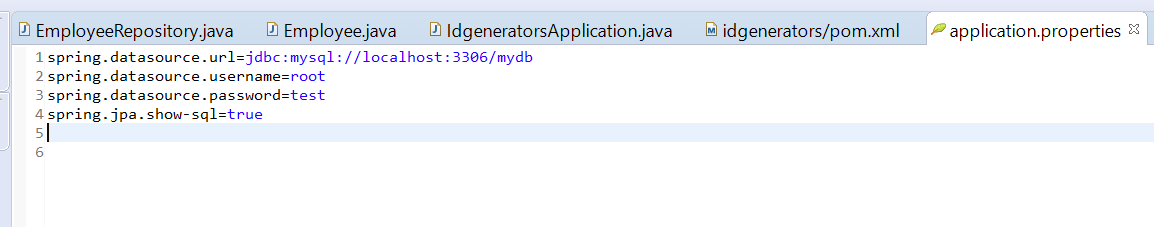
The **GenerationType**.**AUTO** is the default generation type and lets the persistence provider choose the generation strategy from remaining 3

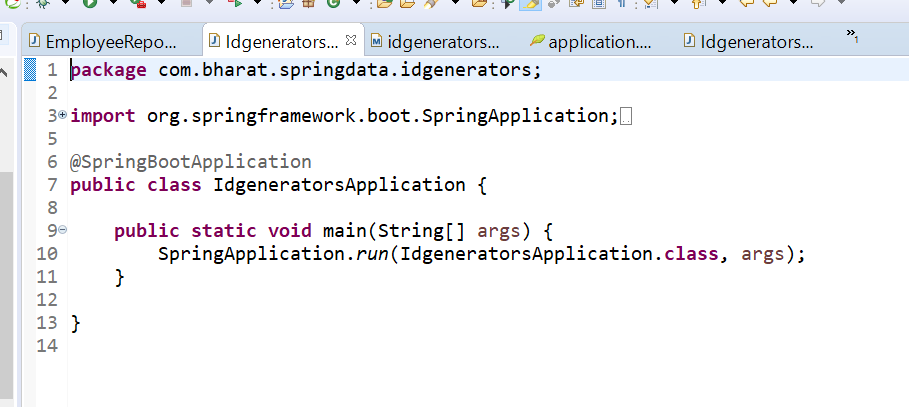
**GenerationType**. **IDENTITY** − In identity , database is responsible to auto generate the primary key

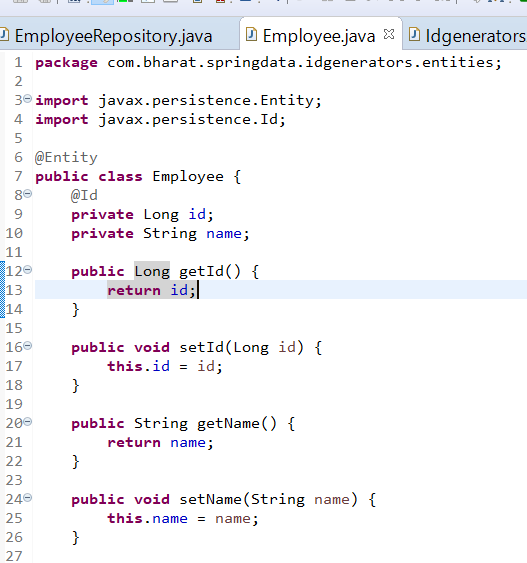
GenerationType. SEQUENCE in hibernate **generates the sequence for the primary column of the table**. We need to create a sequence generator in database and refer that name in the code. The

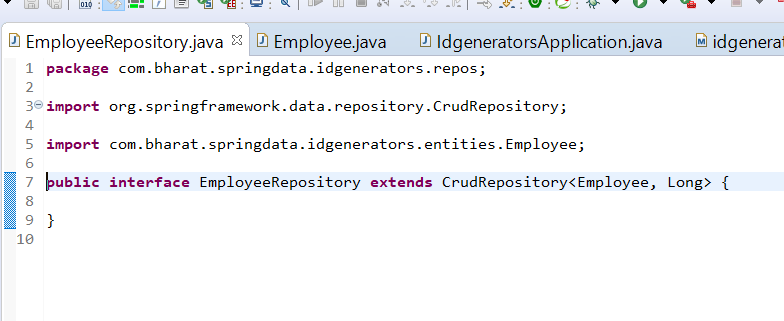
 In the case of **GenerationType**.**TABLE** the column values is filled by a table. This will be separate table. The syatx to use GenerationType.

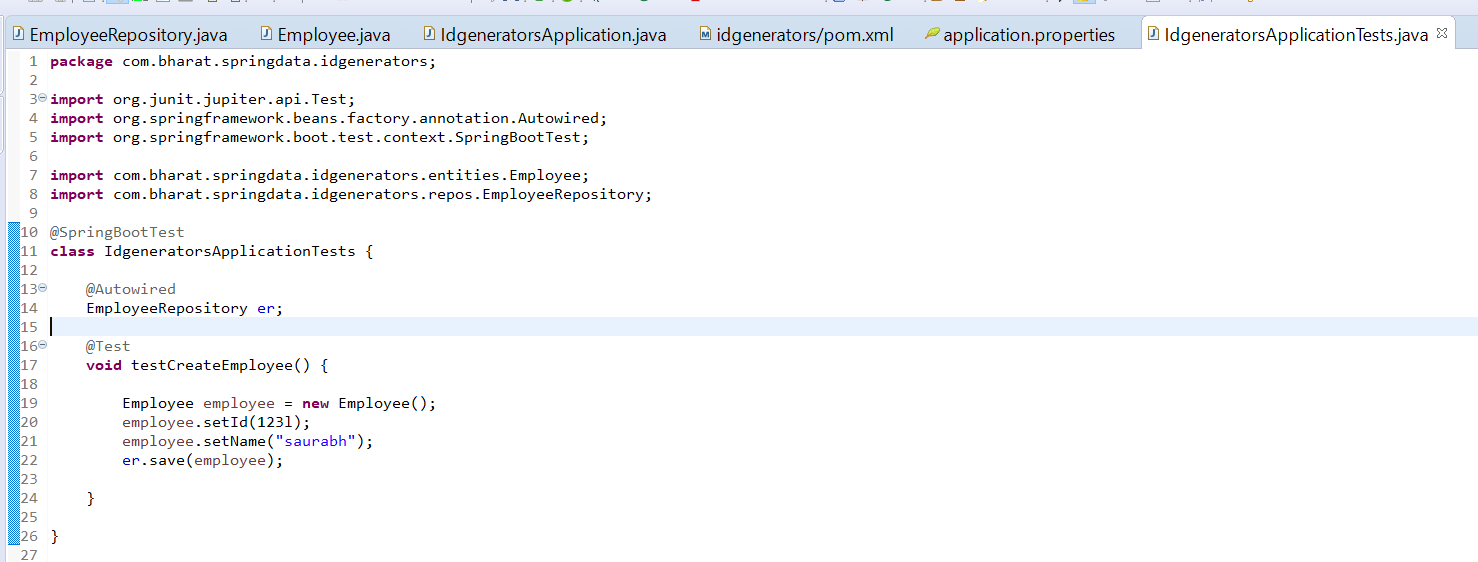




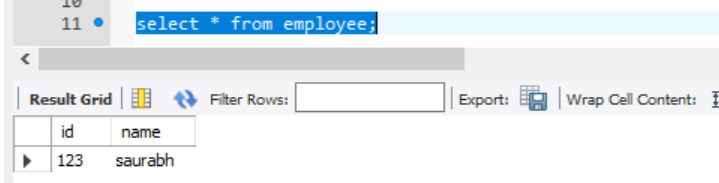






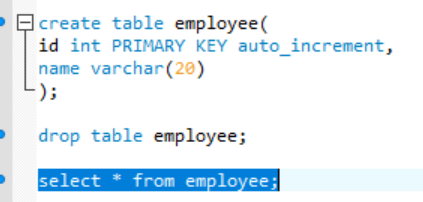


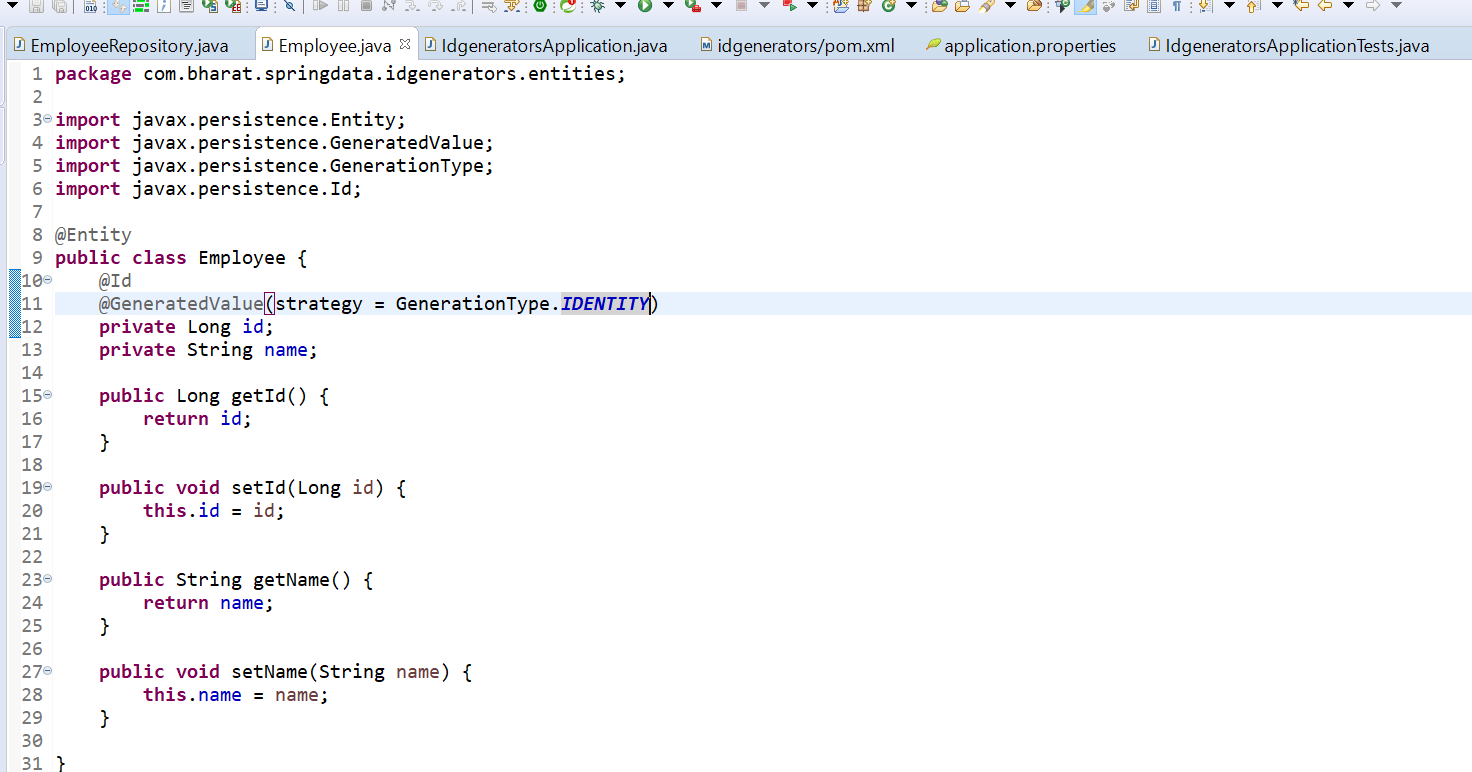
**//output**



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

**Primary key : auto increment**





Graphical user interface, text, application, email

Description automatically generated

Note:

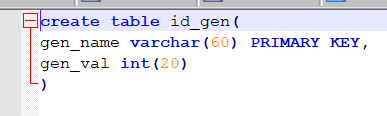
* now we don’t need to provide id , automatically database will configure for it
* Id field will increment automatically everytime we save record in db

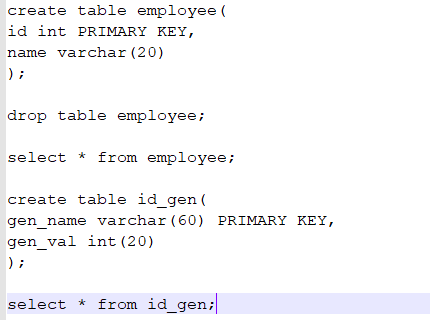
**// output**

Graphical user interface, text, application

Description automatically generated

**Note:-----update db schema for table Strategy**





**Note:--** id\_gen is sequence table

**Configure the table strategy**

**@TableGenerator --- will create sequence**

Graphical user interface, text, application, email

Description automatically generated

**Develop Custom Generator**

**Note:---** wtever this method will return that will used as a id

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated



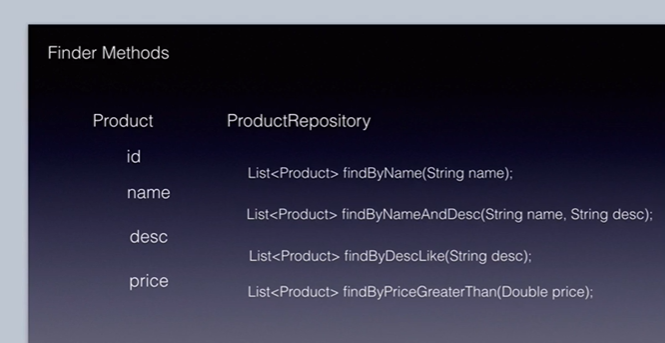
**//output**

Graphical user interface, application

Description automatically generated

**Spring data finder methods**

**Just by following some naming convention we can load data in d/b**



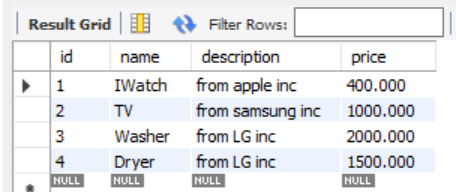
Internally java method converted into sql query

Graphical user interface, text, chat or text message

Description automatically generated

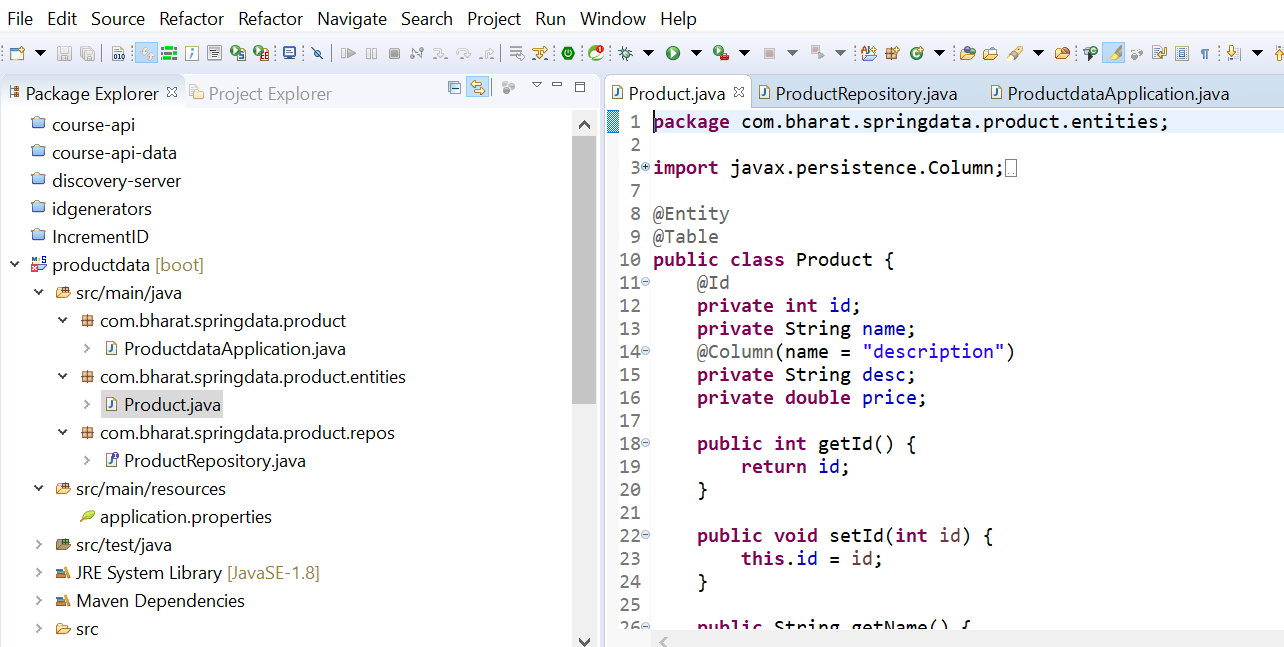
Text, letter

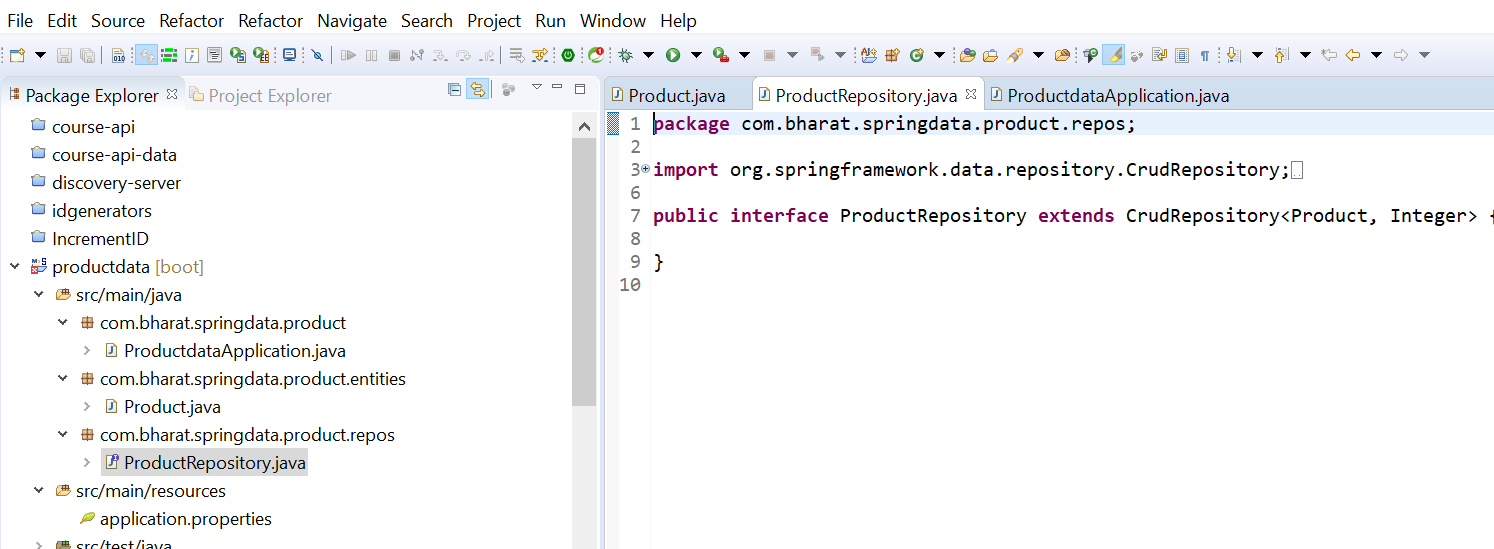
Description automatically generated



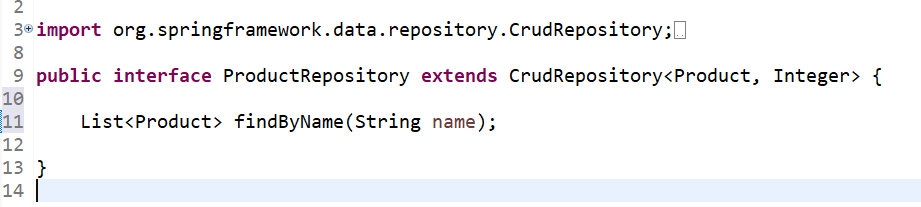
**Requirement :** find product by name without writing any query



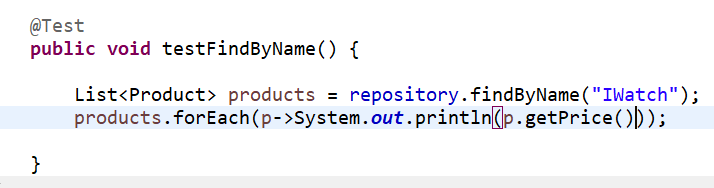




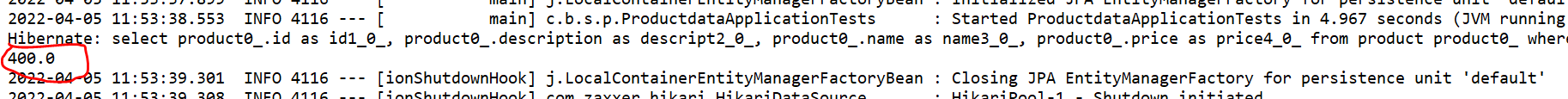
=========



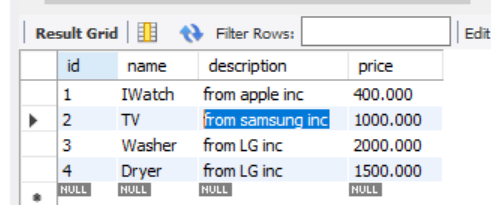
At run time select query will fire and return type is product which will add into list

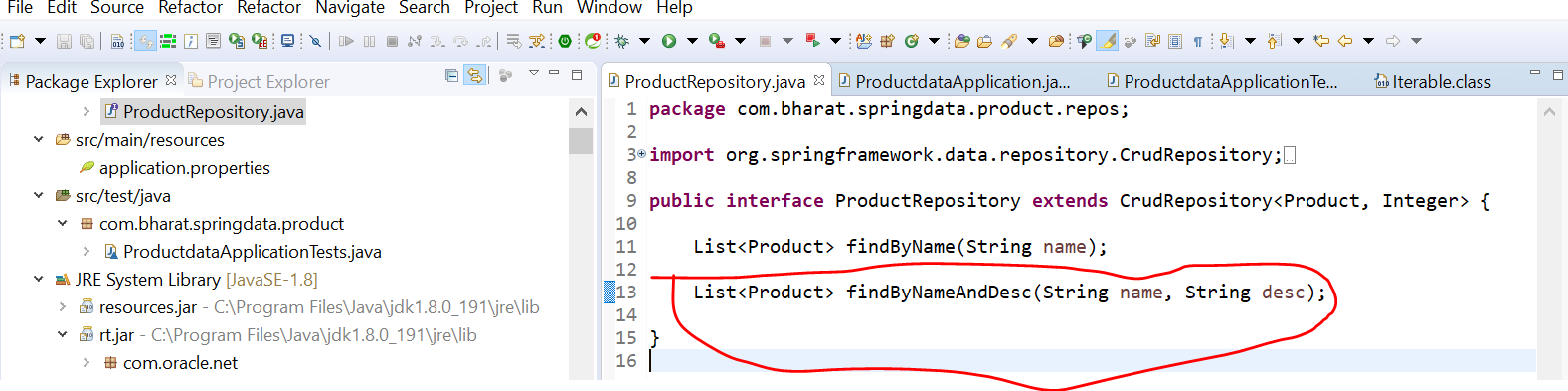


**//output**



**Find by multiple fields**

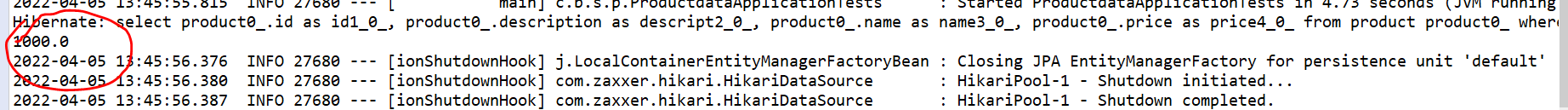




Text, letter

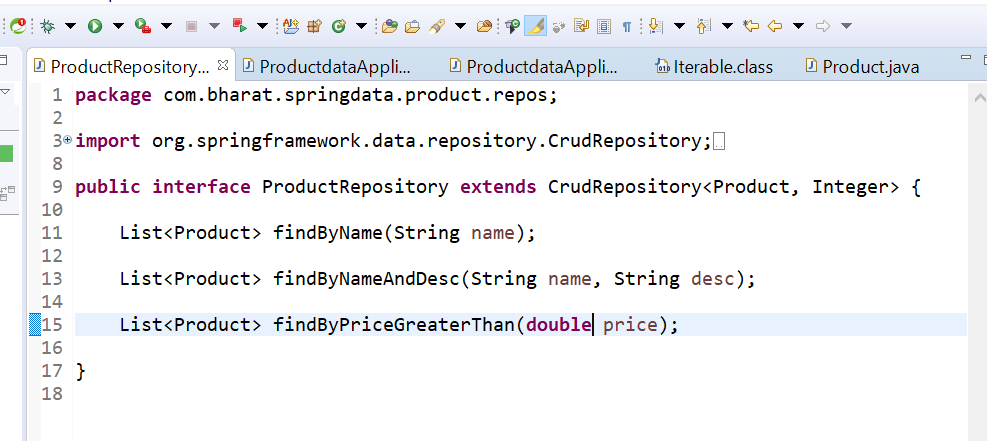
Description automatically generated

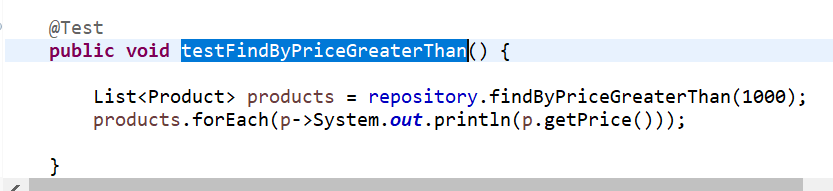
**// output**

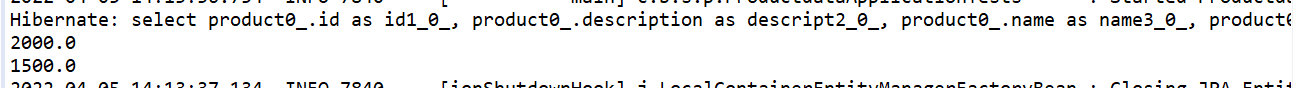


**Greater Than**

**Find all product whose price is greater than price**

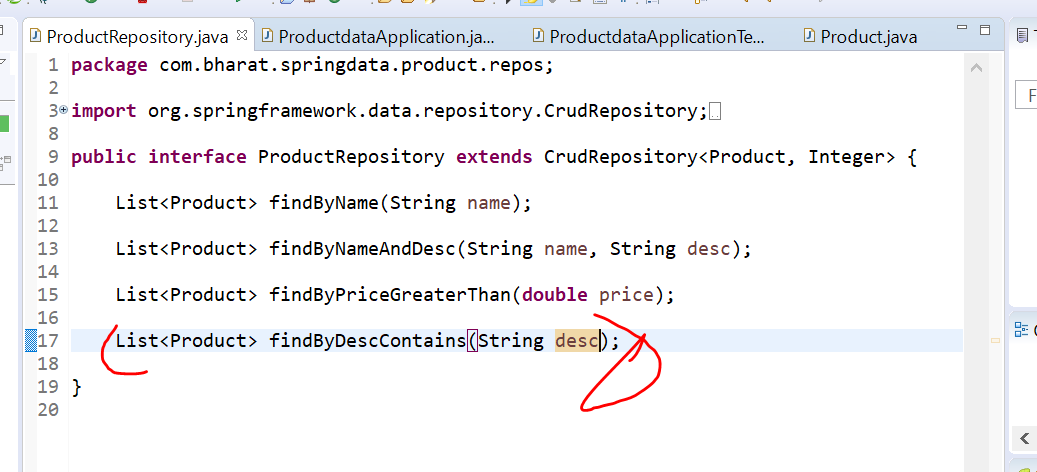






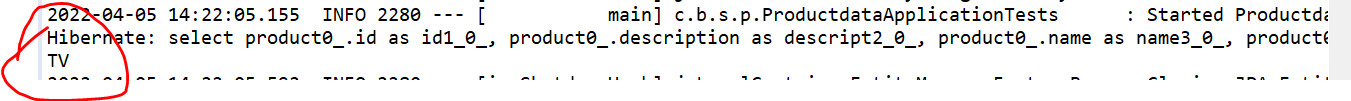
**Contains**

**Find all product that contain a particular word in description**



Graphical user interface, text, application

Description automatically generated



**Between**

**Find all products which fall between two different prices**

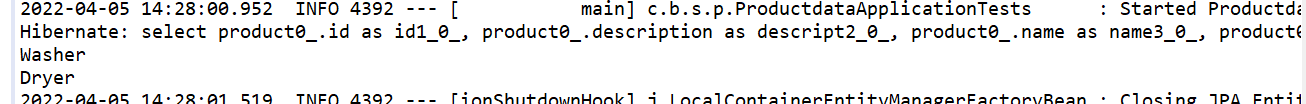
Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**//output**



**Like**

**If part of description is like then we need to find a product**

Graphical user interface, text, application, email

Description automatically generated

Always passed value in %%

Graphical user interface, text

Description automatically generated

**// output**

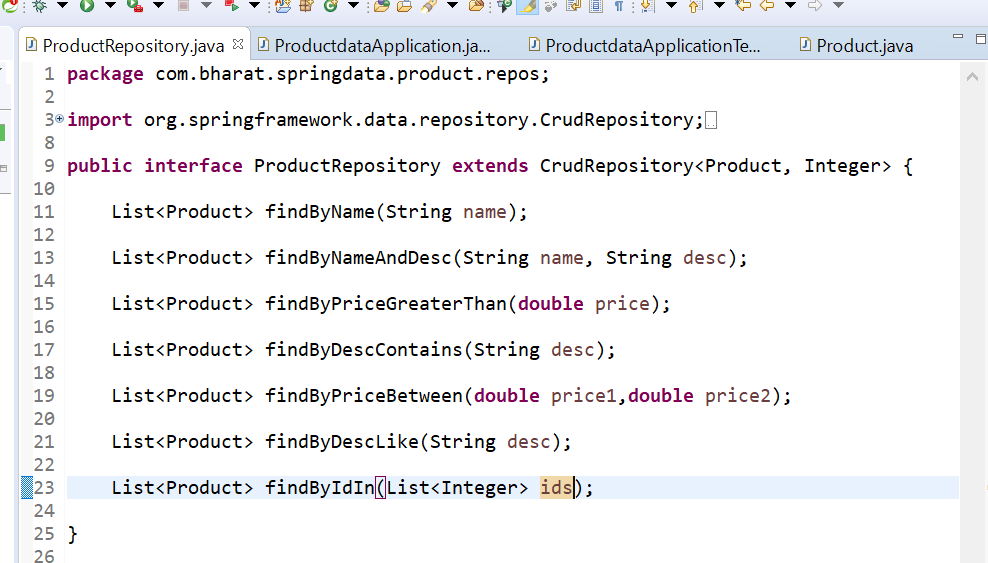
Graphical user interface, text, application

Description automatically generated

**IN**

We can filter out record based on multiple value of column by using IN and passing in multiple values

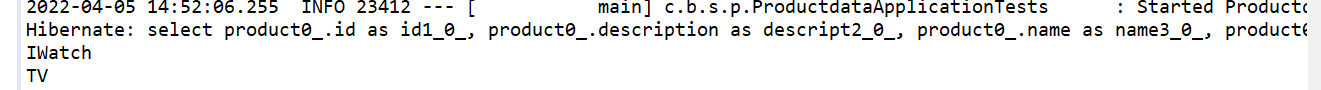
**Requirement :--- find out multiple products given by their ids**



Graphical user interface, text, application

Description automatically generated

**Output**



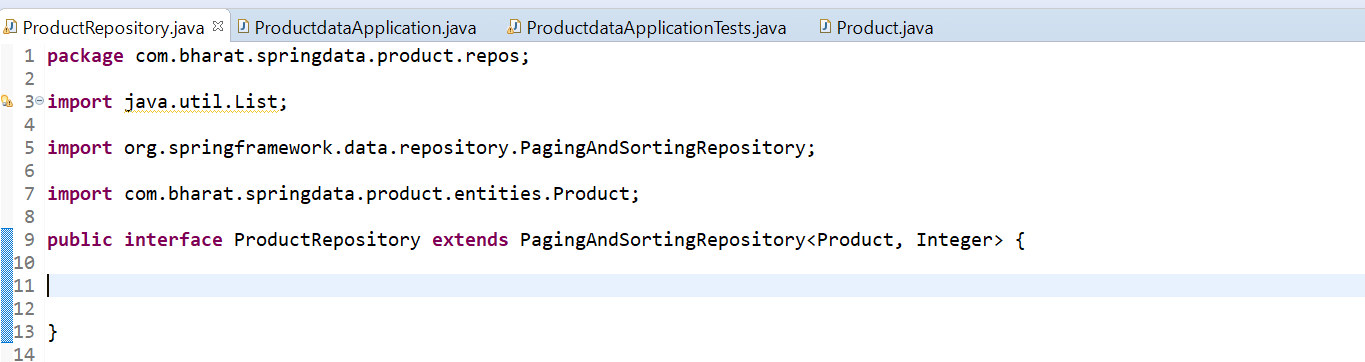
**Paging and Sorting**

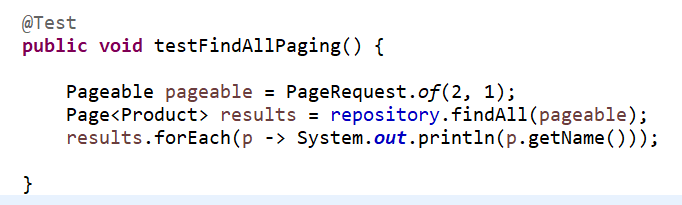
**Paging and sorting works on findAll method**

To support paging and sorting ProductRepository need to extends PagingAndSortingRepository

**To use paging -- PageRequest implements Pageable interface**

**Sorting :-** sort, direction, order which are part of spring data not jpa



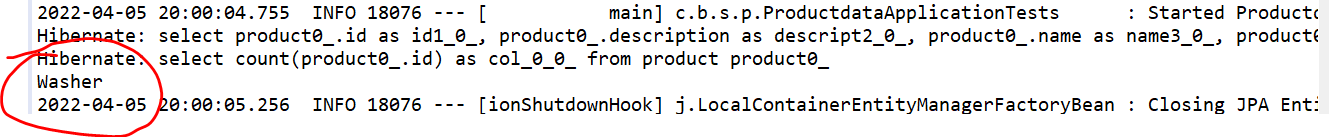


First argument :- page index and second ---in page how many records will display

Graphical user interface, application, table

Description automatically generated

**//output**



Sort by single property

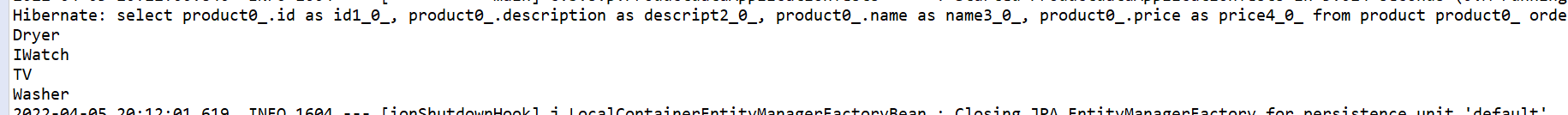
**Requirement** :- findAll method can also be sort or not

**Note: by default sort is ascending**

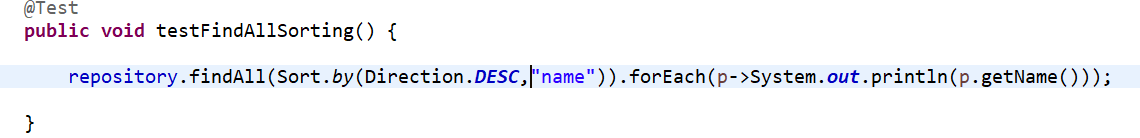
Logo

Description automatically generated with medium confidence

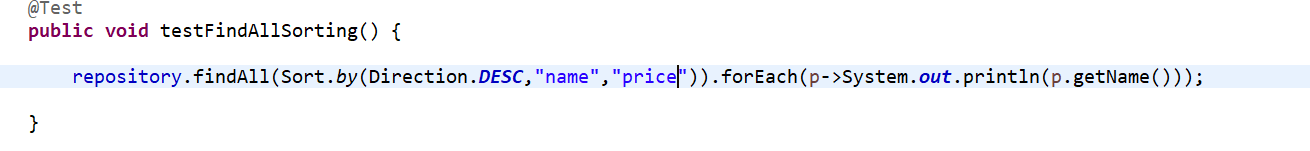
**//output**



**We can sort in descending order -----by using Direction.Desc**



**Sort by multiple properties**



**Sort by multiple properties and direction**

Text

Description automatically generated with medium confidence

**Paging and sorting in action**

Text, letter

Description automatically generated

Text

Description automatically generated

PageRequest is overloaded constructor

**Paging and sorting on a custom finder method**

Text

Description automatically generated

**We need to pass pageable then it will support paging and sorting**

Text

Description automatically generated

**//output :--- getting last 2 record since paging it**

Chart

Description automatically generated

**JPQL --- Java Persistence Query Language**

**It is standard from JPA to perform query against object and domain classes.**

**Note :-**

* We will use domain class name and field for query and not the column name
* These query are internally converted into sql query by looking into the mapping of entity and database table
* Most of the thing which is in sql is also provided by JPQL
* When we are using domain class name and field name then it is case sensitive but it is not case sensitive in case of using keyword like , count

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Add some data in student table to master JPA**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, table

Description automatically generated

**First JPQL**

**Requirement :--- f**ind all student in student table

Graphical user interface, text, application, email

Description automatically generated

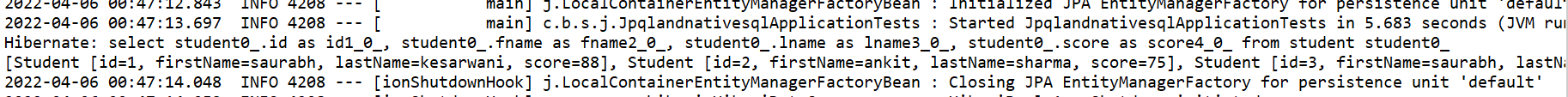
Note :--- no need to write select \* from student , if we are finding all student then

From student is enough. This is the way we write JPQL

Text

Description automatically generated with medium confidence

**//output**



**Read Partial data**

**Requirement :---** to get student firstName and lastName

Graphical user interface, text, application, email

Description automatically generated

**Note:--** in object array we will get all column name

Graphical user interface, text

Description automatically generated

**// output**

Text

Description automatically generated

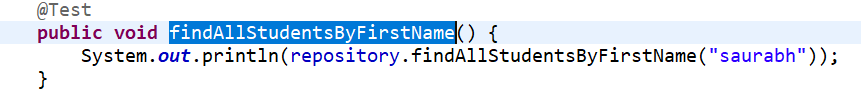
**Using Named Query Parameter**

**Requirement :-** retrieve all student for given firstname

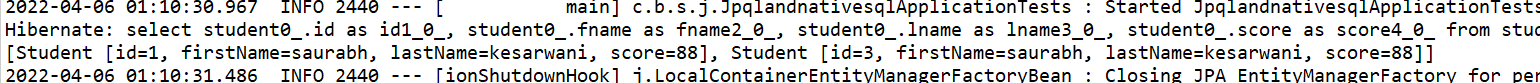
Graphical user interface, text, application

Description automatically generated

Named parameter will be bind by : and param value and value after : should be same



**// output**



**Find by Student Scores**

**Requirement :-**  find all student between score

Graphical user interface, text, application, email

Description automatically generated

**Test and output**

Graphical user interface, text, application

Description automatically generated with medium confidence

**Non Select Operation**

**JPQL also support insert,update,delete**

**Requirement :** delete all student for given firstname

**Note:--** spring data by default consider all query as read query means select query

So to perform update operation we need to use @Modifying

Text

Description automatically generated

Telling spring this particular query will modifying the query

**NOTE:--- if we want to perform DML operation we use @Transactional annotation**

Graphical user interface, text, application, email

Description automatically generated

**This Rollback** we need to use in test case only so that it should not roll back

**// output**

Graphical user interface, application

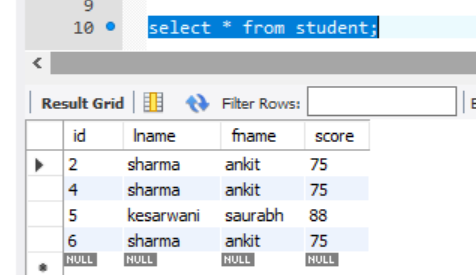
Description automatically generated

Paging and Sorting with JPQL

**Requirement :---**  Let do paging on this method

Graphical user interface, text, application

Description automatically generated



Note:--- for paging support we need to pass Pageable in method

Graphical user interface, text, application

Description automatically generated

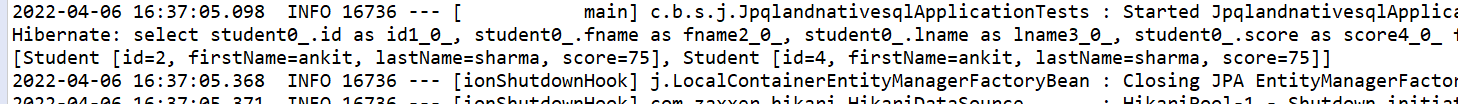
**Pageable is an interface and PageRequest is an implementation class of this interface.**

We need to pass PageRequest in this method

Graphical user interface, application, Word

Description automatically generated

**//output**

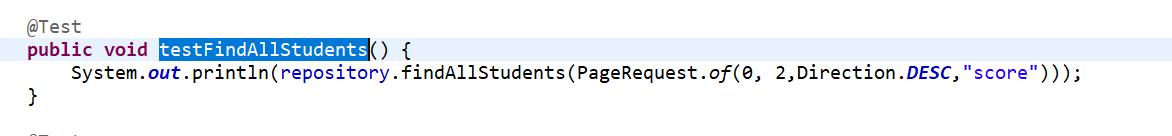


**Sorting**

**Requirement :----** sort student based on id or name

Graphical user interface, text, application

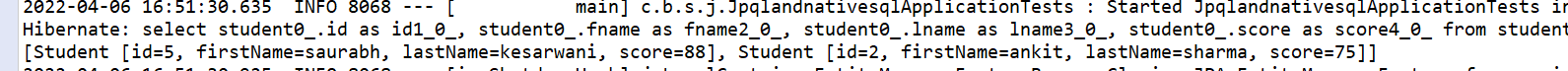
Description automatically generated



**Imp points:-**

**Note :---- first it will sort in complete d/b then page it**

**//output**



**Native SQL Query**

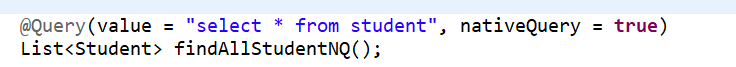
* These are the query before orm :---
* when JPQL query become complex we use this and return result into object
* DDL is not supported in JPQL but supported in native sql query
* Also can pass named parameters

This time will use database directly not entity

**Requirement** :--- retrieve all student

**Note:---**

by default NativeQuery = false , we need to turn it true , it tell spring data this is native query



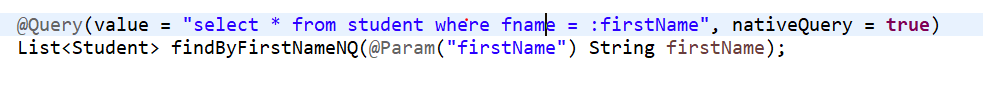
**Test case and output**



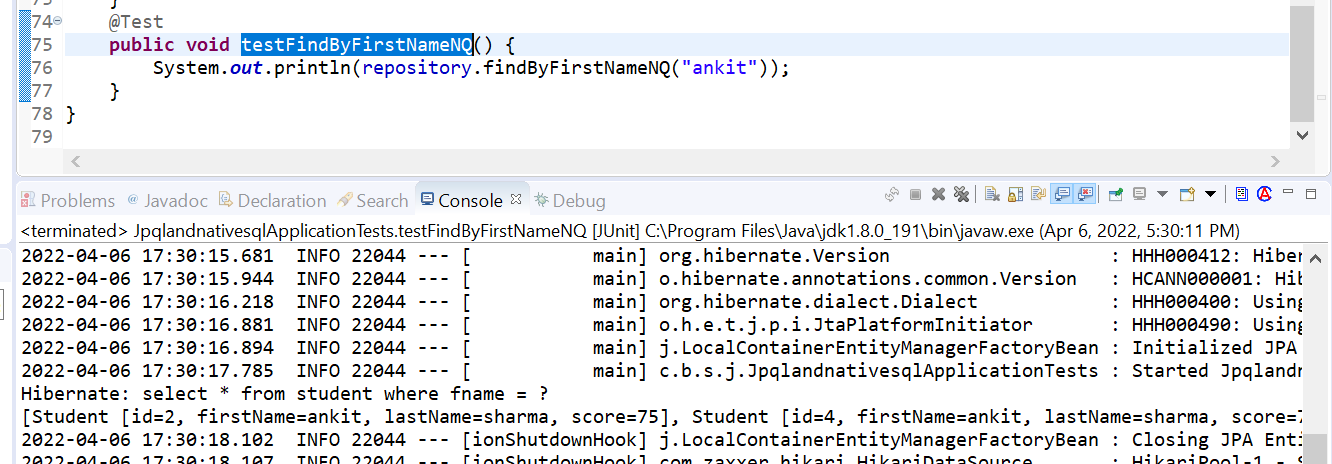
**Pass Named Parameter**

**Requirement :--** Retrieve all student by firstName

Fname is d/b column name



**Test and output**



**Inheritance Mapping**

**Parent :-** Payment- field -id,amount

**Child:-** Card-field-cardNumber

**Child:-** Check- field- checkNumber

**NOTE:---** jpa provide mapping through three types of strategies

**Strategies:-** Single\_table, table\_per\_class, joined

SINGLE\_TABLE Strategy

 In this approach, instances of the multiple entity classes are stored as attributes in a single table only.

Diagram

Description automatically generated

Text, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Pmode is the discriminator column so when we single table strategy this field is required to separate out entity

Graphical user interface, text

Description automatically generated

**Note:---** here three entity

**Parent entity**---Payment

**Child entity** :- CreditCard and Check

Graphical user interface, text

Description automatically generated

Step 1--- create three entity class

**Mark the parent class with annotation**

**Note:**  here we are creating one table for the entire hierarchy

Graphical user interface, text, application

Description automatically generated

**Mark the child class with annotation**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Create the Repository for Payment**

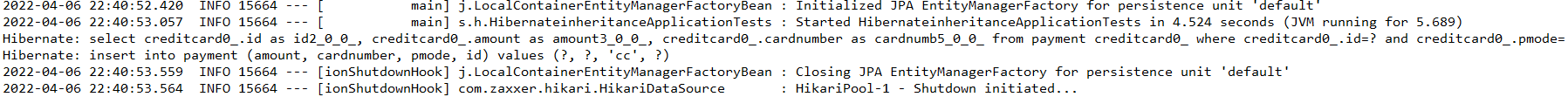
Graphical user interface, text, application

Description automatically generated

**Class per hierarchy in action**

Graphical user interface, text, application, email

Description automatically generated



Graphical user interface, text, application, email

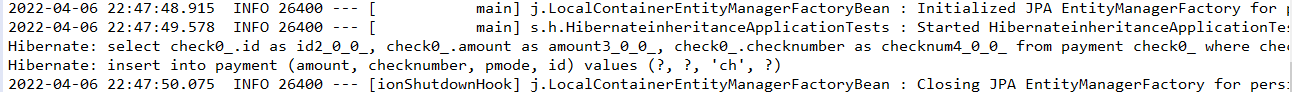
Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated



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

**second strategy : TABLE\_PER\_CLASS**

**It duplicate field across column , so not recommended but performance will increase**

Diagram

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Note:** id and amount field is common

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Test

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Joined Strategy

In this strategy all entity have own table so we do not discriminator column.

Parent and child are connected with each other through primary key and foreign key

Diagram

Description automatically generated

**Advantages :** each table carry min data

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

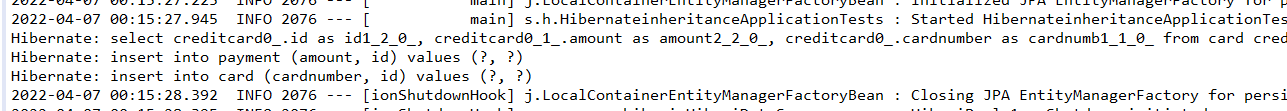
Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

**Note :-By running above test case 2 insert statement will execute**



Graphical user interface, text, application

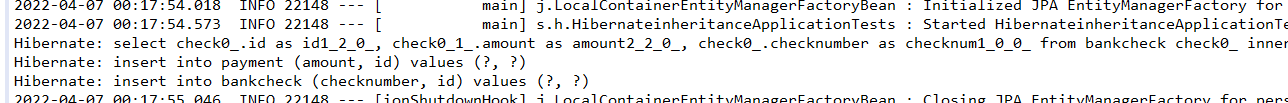
Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text, letter

Description automatically generated



Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Component Mapping**

**Has a relationship**

**Employee @Embeddable**

**@Embedded Address**

**Address address**

**Requirement :** once we save employee object then address object will also save in table

**Step 1- c**reate database table

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**Note:---** we can only mark a class as @entity having id field

**Step 2- create Entity ---- Employee and Address**

Graphical user interface, application

Description automatically generated with medium confidence

Text

Description automatically generated

**Step 3:-- create Repository**

Graphical user interface, text, application, email

Description automatically generated

**Step 4- testing**

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Relationship in hibernate**