**Flay Buyers**

Objective :

***In this exercise, you will create a Spring Data JPA project using One to Many relationship between two entities and persist their details in the database and also perform retrieve operations using JpaRepository (write custom query using @Query annotation).***

# 1.0 Functional Requirements

Meraas Builders wants to store and manipulate the details of their buyers to whom they have sold flats from their new apartment.

Develop a Spring Data JPA application to perform the task.

You need to implement the Buyer and Flat module for this application. The client wish to have the services to add a buyer, register a flat to a buyer and also help the buyer to find flat with minimum price and maximum rooms.

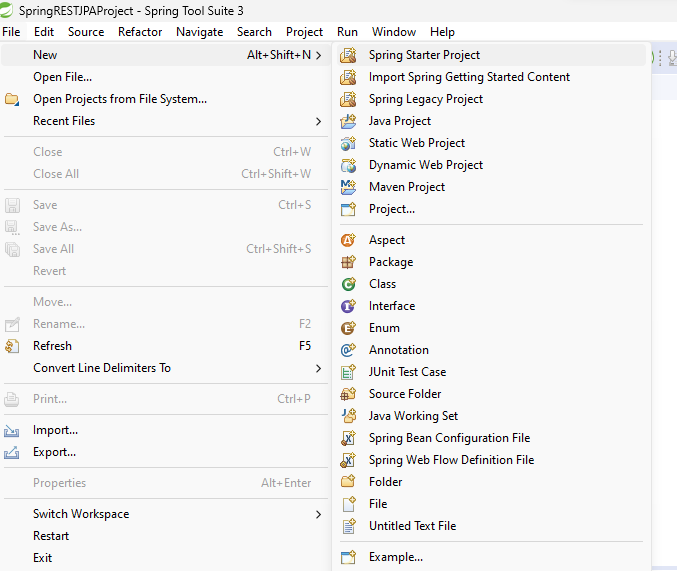
Help them to automate the above process by developing Spring Boot with JPA application using Maven.

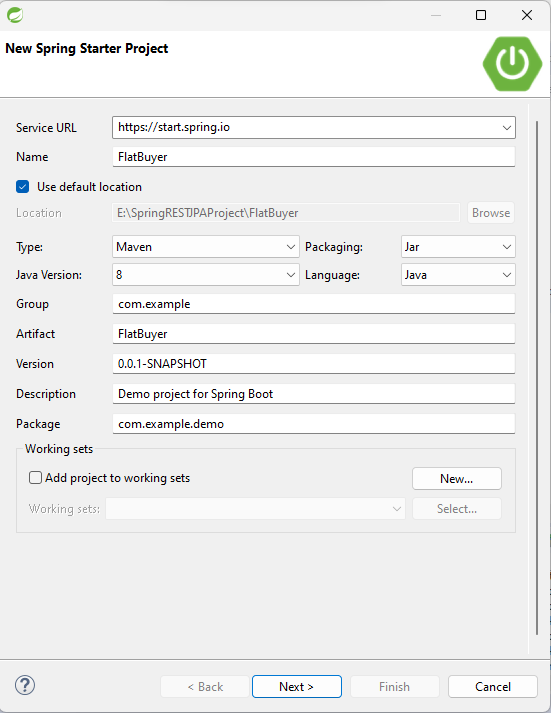
# 2.0 Technical Specifications

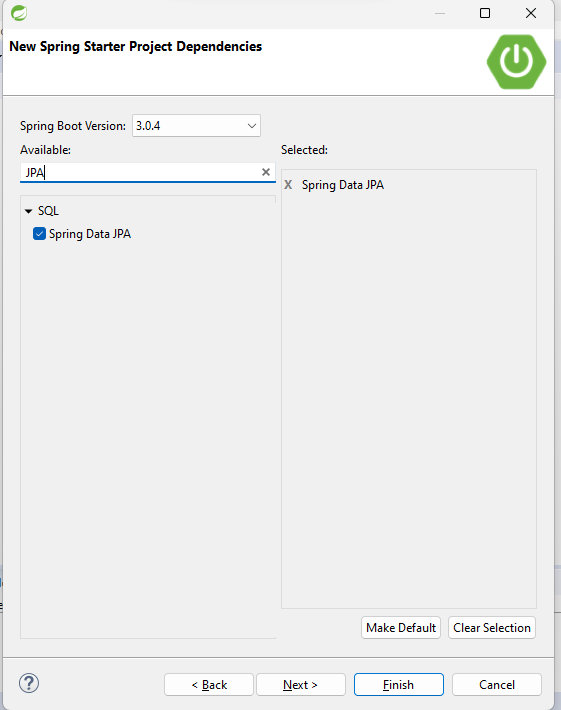
To start with this project, open STS, create a Spring Starter project and follow the instructions below.

1. Open IDE STS – Spring Tool Suite

2. Go to File ->Spring Starter Project

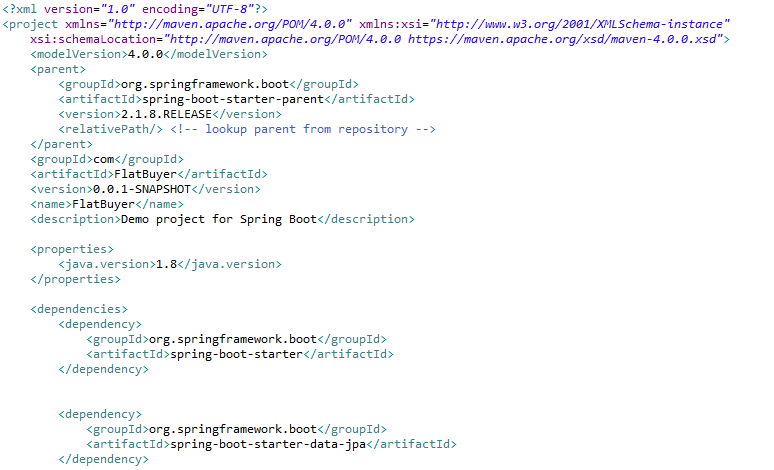






Click Finish

3. Once the project is created, add the required dependency in pom.xml

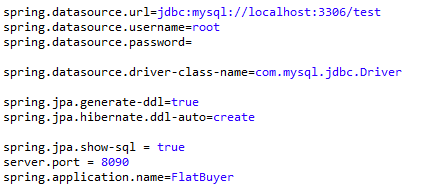






* **spring-boot-starter-data-jpa: It will download the files required for spring data JPA.**
* **mysql-connector-java: It is used to connect with MySQL database.**

4. Go to src/main/resources. Configure application.properties file



* ***spring.datasource.url*** is used to set the url of the Mysql database.
* ***spring.datasource.username*** is used to set the username and ***spring. datasource. password*** is used to set the password.
* ***spring.datasource.driver-class-name*** is used to set the driver class name.
* ***spring.jpa.hibernate.ddl-auto*** is set to **create** so thatHibernate first drops existing tables, then creates new tables
* **spring.jpa.show-sql** is set to **true** to show SQL generated by the Hibernate.
* **server.port** is set to **8090**.
* If needed, **spring.jpa.properties.hibernate.format\_sql**  can be set to **true** to format SQL queries.

5. Go to src/main/java. Create a package com, com.bean, com.dao and com.repository.

# Model class

You need to create 2 Model classes, Buyer and Flat with attributes

**Attributes for Buyer class**

String buyerId

String buyerName

String buyerContactNumber

List<Flat> flatList

**Attributes for Flat class**

String flatId

int doorNumber

String flatType

double flatArea

int numberOfRooms

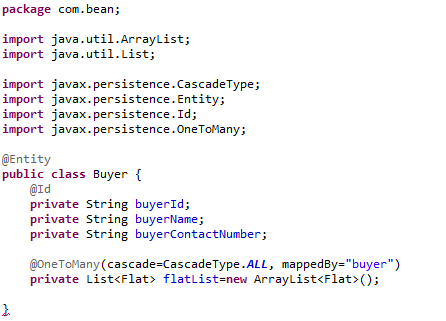
double flatPrice

Buyer buyer;

**Use appropriate annotation for the persistence mapping in the model class.**

Note that one Buyer object is related to multiple Flat objects and one Flat object will be related to one Buyer object. **We have established a bi-directional relationship.**

6. Create the Entity class Buyer.java in com.entities package with the below attributes



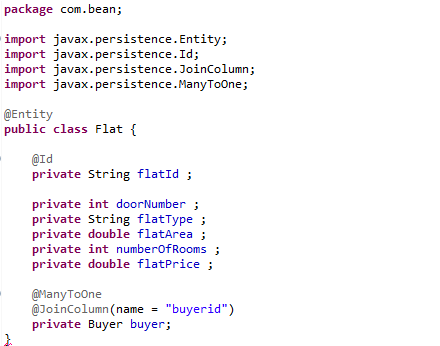
Provide public getters and setters.

* ***@Entity*** annotation mark this class as a JPA entity. It tells hibernate to make a table out of it.
* ***@Id*** annotation specifies the primary key of an entity.

One Buywe object is related to many Flat. Hence we have provided the annotation @OneToMany.

Observe the usage of mappedBy and cascade attributes and also the annotations @Entity and @Id.

7. Next create the Entity class Flat.java



Provide public getters and setters

Many Flat objects will be related to one Buyer object. Hence the annotation, @ManyToOne.

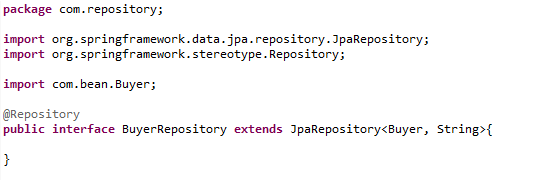
Observe the usage of @JoinColumn with name attribute.

# Repository Layer

Create 2 interface BuyerRepository and FlatRepository. Use appropriate Spring Data JPA for all the database related manipulation.

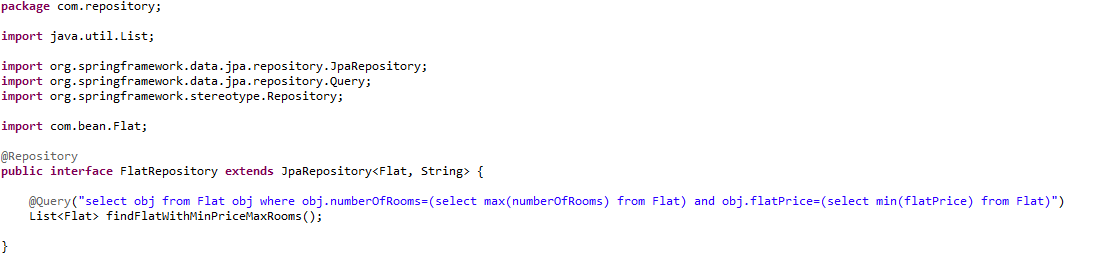
Also include the necessary method to view Flat with minimum price and maximum rooms. As this is related to Flat, write this method in FlatRepository.

8. To do this, create the interface BuyerRepository.java that extends JpaRepository in com.repository package



9. Create the interface FlatRepository.java that extends JpaRepository in com.repository package.

In this interface create a custom query methods with any name say, findFlatWithMinPriceMaxRooms to view Flat with minimum price and maximum rooms.



*The @Query annotation specifies the JPQL (Java Persistence Query Language) query to be executed. The query selects all flats that have a flat price equal to the minimum flat price in the Flat table, and a number of rooms equal to the maximum number of rooms in the Flat table.*

# DAO Layer

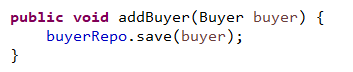
In com.dao package create the class BuyerDAO.java. In this class all database related operations should be implemented using the BuyerRepository and FlatRepository interface respectively.

10. Create BuyerDAO class as shown below :



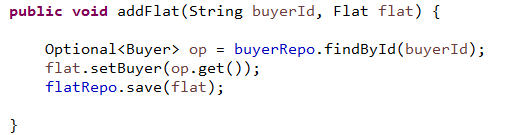
11. **Implement the methods in BuyerDAO class.**

* **public void addBuyer(Buyer buyer)** - This method should store the Buyer details to the database using BuyerRepository.

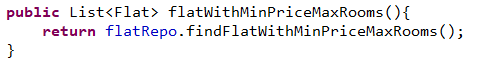


* **public void addFlat(String buyerId,Flat flat)** – This method accepts buyer Id and a Flat object that holds details of a flat. This method should retrieve the Buyer object for the buyer Id passed as parameter.

Next set that buyer object to the flat object. Add the Flat details to the database using FlatRepository.



* **public List<Flat> findFlatWithMinPriceMaxRooms() -** This method should return the list of flat which has the minimum price and maximum room count



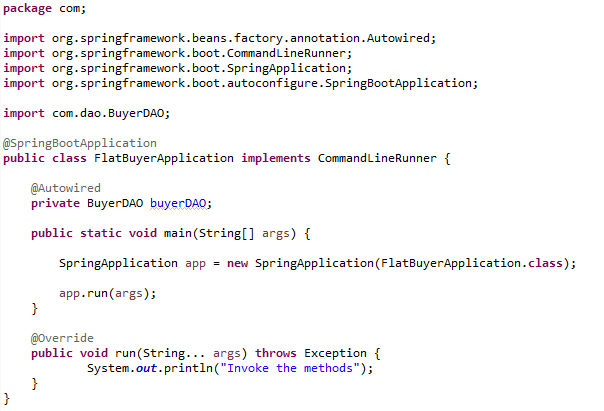
**Create the Launch class for Spring Boot Application**

Every Spring Boot Application needs one launch class. This class is annotated with the **@SpringBootApplication**.

12. Finally test your application.

For this, in the project, in com package, create the class FlatBuyerApplication.java with the main method.

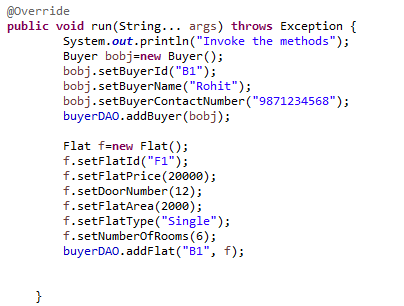
You can provide the code in this class as shown below



Observe the usage of annotations ***@SpringBootApplication***

18. In the run method, you can invoke the methods in BuyerDAO.

You can invoke the methods in run method as



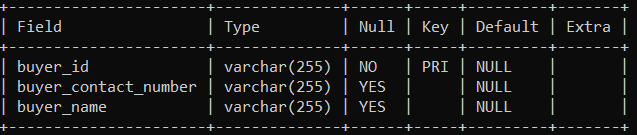
Likewise all methods can be invoked and tested.

Also you can open the mysql workbench and check if the datas are inserted correctly too.

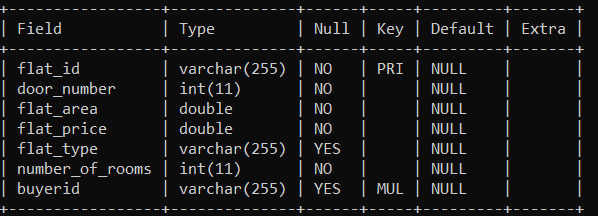
# 3.0 Expected Table Structure

The below table should get created automatically by the application

buyer

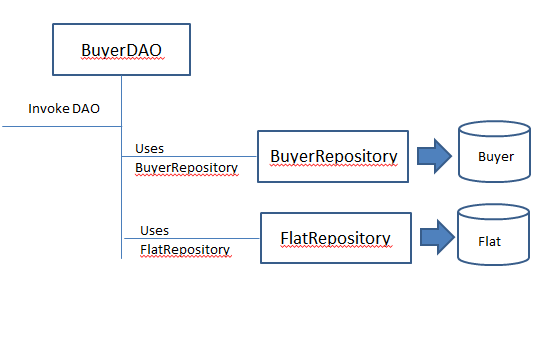


flat



Note : buyerid is the join column in the flat table

# 4.0 Process Flow



* The run method invokes the required dao.
* BuyerDAO with the help of BuyerRepository and FlatRepository performs the service and returns the data back
* BuyerRepository and FlatRepository has to be injected into the BuyerDAO
* The BuyerRepository interface and FlatRepository interface should use the appropriate Spring Data JPA for all the database related manipulation.

# 4.0 Overall Design Constraints

When submitting the code to platform ensure the below points

1. **Check if the property name given in the application.properties files is same as given in the sample code. You can change the value and you can include additional property if needed.**
2. **In the pom.xml ensure that you have provided only the dependencies provided in this sample. Don’t provide any additional dependency.**
3. **Use only Spring data JPA to handle persistence. Do not generate value for the primary key attribute automatically.**
4. **Use the service type and the service names as expected in the specification**
5. Adhere to the design specifications mentioned in the case study.
6. **Ensure that you have provided all the classes / interface / attributename / methodname / return type / parameters as mentioned in the problem statement.**
7. **Please make sure that your code does not have any compilation errors while submitting your case study solution.**

Congratulations, you have successfully completed the exercise on building a Spring Data JPA application using Maven!