www.youtube.com/c/javaexpress
+91 7801007910
Java Express

Spring Data JPA

- Spring Data JPA is a powerful library within the spring framework that makes data access in Java applications using JPA (Java Persistence API) much easier and more efficient.
- Purpose:
  - Simplifies working with relational databases in your spring applications.
  - Eliminates the need for writing boilerplate code for basic CRUD (Create, Read, Update, and Delete) operations and complex queries.
- Spring Data JPA will provide implementation for all DAO operations using Proxy Design Pattern.

Benefits:

- **Improved developer experience:**
  - Focus on the business logic of your application instead of data access details.
- **Increased productivity:**
  - Faster development cycles due to simplified data access methods.
- **Automatic entity mapping:**
  - Spring Data JPA automatically maps your POJOs (Plain Old Java Objects) to database tables based on annotations and conventions, saving you time and effort.
- **Support for complex queries:**
  - While providing convenience for basic operations, Spring Data JPA still allows for building complex queries using JPA's Criteria API or QueryDSL for specific needs.
- **Reduced boilerplate code:**
  - The Spring Data JPA provides the default implementation for common methods by its repository interfaces.
  - No more manual creation of DAOs (Data Access Objects) or writing extensive SQL queries.
- **Spring Data JPA we can execute queries in 3 ways**
  - Using Pre-defined Methods (Repository in built methods)
  - By Writing findBy Methods (Spring DSL)
  - By Writing custom Queries (@Query)

How it works:

- Create Java classes annotated with JPA annotations to represent your database entities.
- Define repository interfaces extending Spring Data JPA interfaces like JpaRepository or CrudRepository.
- Spring Data JPA automatically generates implementations for these interfaces at runtime, handling data access logic based on the methods you define.

Database Connection Properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/devflipkart?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=India@123
spring.jpa.hibernate.ddl-auto=update
```

JPA Annotations:

1. @Entity:
    - The java class which represents DB Table structure is called as Entity class
    - The class which is mapped with DB Table is called as Entity Class
    - It is class level annotation
    - It is mandatory annotation
2. @Table:
    - It is used to map class name to particular name
    - It is class level annotation
    - It is optional annotation
    - If @Table is not mentioned then SB Data considers our class name as table name
3. @Id:
    - It is represent variable which mapped with PK
    - It is field level annotation
    - It is mandatory annotation
4. @Column:
    - It is used to map entity class variable name with table column name
    - It is field level annotation
    - It is optional annotation
    - If we don't use @Column, SB data will consider variable name as column name
5. @GeneratedValue (strategy = GenerationType.**IDENTITY**)
    - It is database dependent
    - Doesn't support for Oracle
    - Supports for MySQL

Schema Generation

- ORM frameworks are having support to generate schema in runtime
- ORM frameworks can create table and can create sequence in runtime required for app
- By Default, Schema Generation mode will be OFF
- To enable schema generation, we will use ddl-auto property
    - spring.jpa.hibernate.ddl-auto=create/create-drop/update
- We have several possible values for ddl-auto property

- Create
- Create-drop
- Update
- Validate
- None

- **create**: means every time new table will be created
- **create-drop**: it creates table and drops table at end of operation
- **update**: If table is not available it will create if table is already available it will just perform operation
- **validate**: It is used to validate schema details

Domain Specific Language (DSL):

- Spring data JPA not only provides implementation for commonly used methods but also provides a way to add custom methods.
- Spring Data JPA QueryDSL:

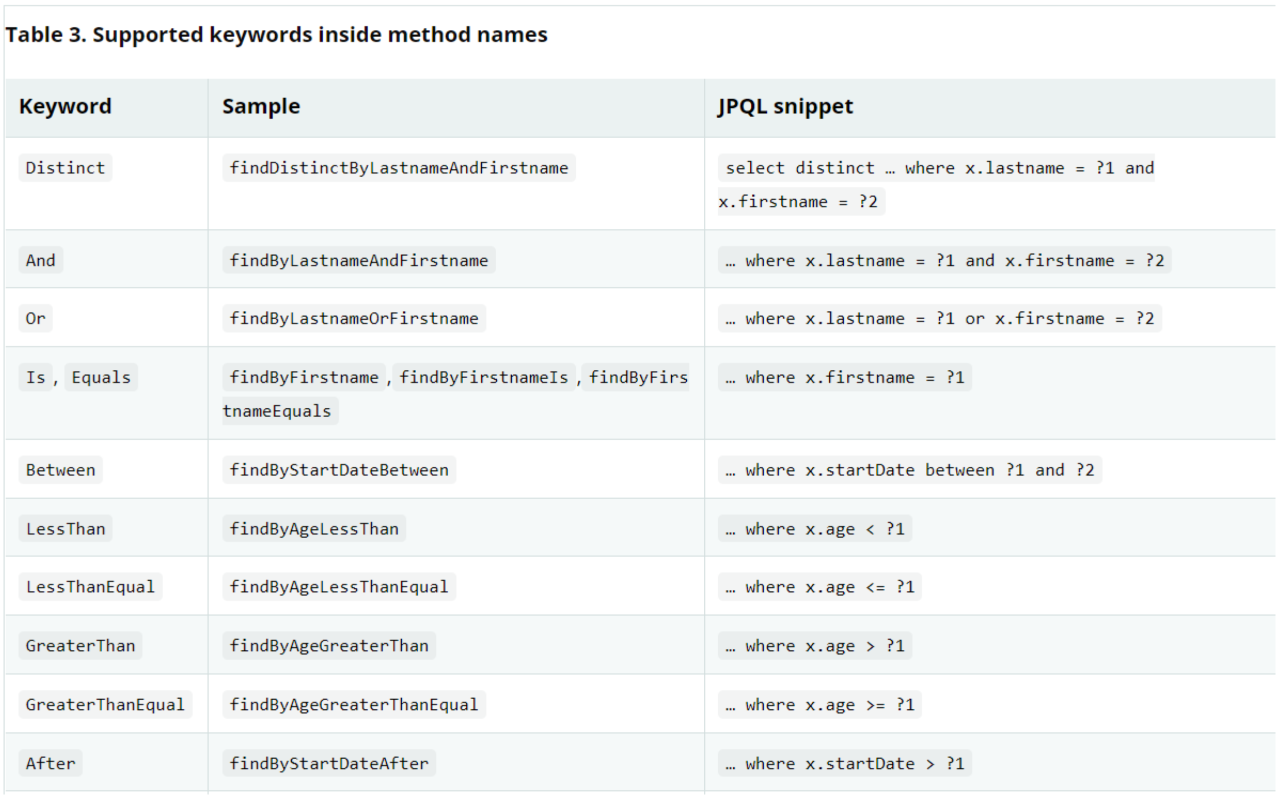


Table 3. Supported keywords inside method names

Keyword	Sample	JPQL snippet
`Distinct`	`findDistinctByLastnameAndFirstname`	`select distinct … where x.lastname = ?1 and x.firstname = ?2`
`And`	`findByLastnameAndFirstname`	`… where x.lastname = ?1 and x.firstname = ?2`
`Or`	`findByLastnameOrFirstname`	`… where x.lastname = ?1 or x.firstname = ?2`
`Is`, `Equals`	`findByFirstname`, `findByFirstnameIs`, `findByFirstnameEquals`	`… where x.firstname = ?1`
`Between`	`findByStartDateBetween`	`… where x.startDate between ?1 and ?2`
`LessThan`	`findByAgeLessThan`	`… where x.age < ?1`
`LessThanEqual`	`findByAgeLessThanEqual`	`… where x.age <= ?1`
`GreaterThan`	`findByAgeGreaterThan`	`… where x.age > ?1`
`GreaterThanEqual`	`findByAgeGreaterThanEqual`	`… where x.age >= ?1`
`After`	`findByStartDateAfter`	`… where x.startDate > ?1`

`Before`	`findByStartDateBefore`	`… where x.startDate < ?1`
`IsNull` , `Null`	`findByAge(Is)Null`	`… where x.age is null`
`IsNotNull` , `NotNull`	`findByAge(Is)NotNull`	`… where x.age not null`
`Like`	`findByFirstnameLike`	`… where x.firstname like ?1`
`NotLike`	`findByFirstnameNotLike`	`… where x.firstname not like ?1`
`StartingWith`	`findByFirstnameStartingWith`	`… where x.firstname like ?1` (parameter bound with appended `%`)
`EndingWith`	`findByFirstnameEndingWith`	`… where x.firstname like ?1` (parameter bound with prepended `%`)
`Containing`	`findByFirstnameContaining`	`… where x.firstname like ?1` (parameter bound wrapped in `%`)
`OrderBy`	`findByAgeOrderByLastnameDesc`	`… where x.age = ?1 order by x.lastname desc`
`Not`	`findByLastnameNot`	`… where x.lastname <> ?1`
`In`	`findByAgeIn(Collection<Age> ages)`	`… where x.age in ?1`
`NotIn`	`findByAgeNotIn(Collection<Age> ages)`	`… where x.age not in ?1`
`True`	`findByActiveTrue()`	`… where x.active = true`
`False`	`findByActiveFalse()`	`… where x.active = false`
`IgnoreCase`	`findByFirstnameIgnoreCase`	`… where UPPER(x.firstname) = UPPER(?1)`

Development Steps

1. **Create Spring Boot Project**
    a. **groupid : com.javaexpress**
    b. **artifactid : any name**
    c. **version : Java 17**
    d. **Type : maven**
    e. **Packaging : jar**
    f. **Language: java**
2. **Add two dependencies**
    a. **Spring-boot-starter-data-jpa**
    b. **MySQL-connector-j driver**
3. **Add Database properties in application.properties file**
4. **Create model package**
    a. **Add Model Class**
5. **Create repository package**

   a. **Add repository interface**

6. **Do operations in main methods**

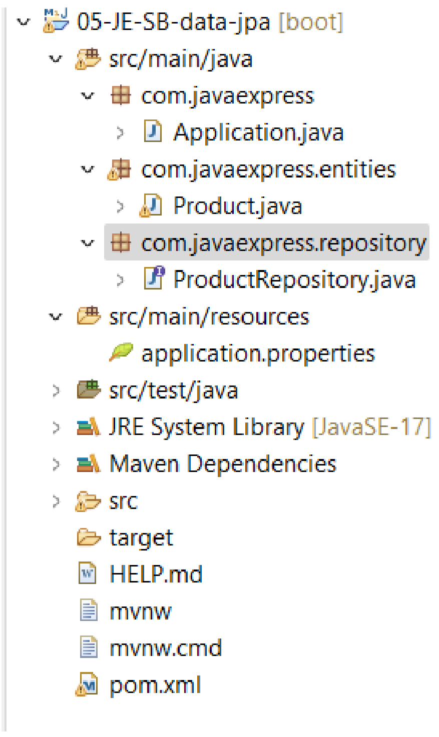
Maven Dependencies:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

Database Properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/31dev?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=India@123
spring.jpa.hibernate.ddl-auto=update
```

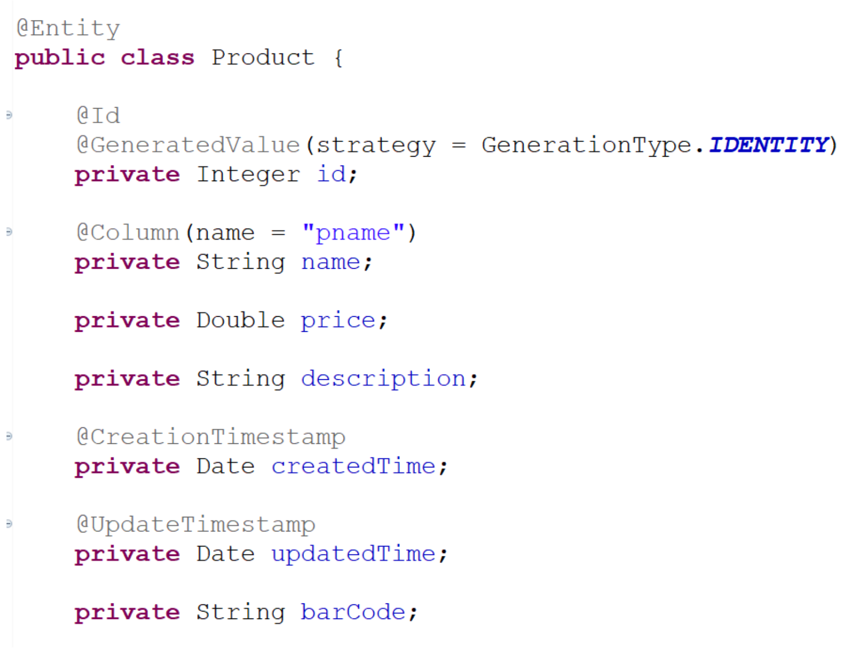
Project Structure



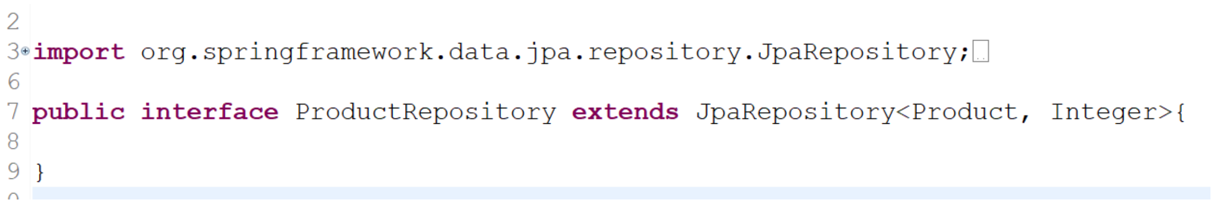
```
05-JE-SB-data-jpa [boot]
    src/main/java
        com.javaexpress
            Application.java
        com.javaexpress.entities
            Product.java
        com.javaexpress.repository
            ProductRepository.java
    src/main/resources
        application.properties
    src/test/java
    JRE System Library [JavaSE-17]
    Maven Dependencies
    src
    target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

www.youtube.com/c/javaexpress
+91 7801007910
Java Express

Model Class

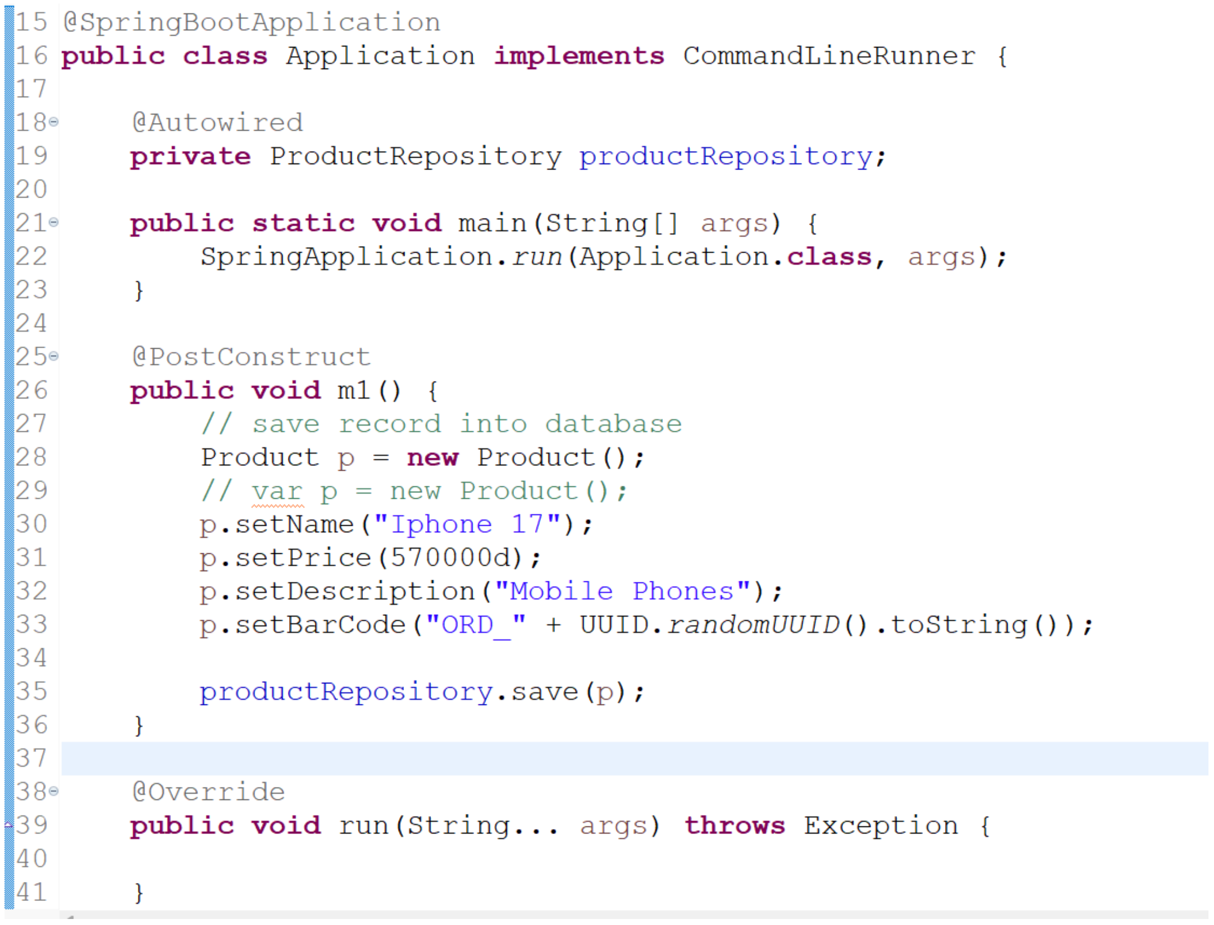

```java
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

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

    private Double price;

    private String description;

    @CreationTimestamp
    private Date createdTime;

    @UpdateTimestamp
    private Date updatedTime;

    private String barCode;
```

Repository Interface


```java

import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Integer>{

}
```

www.youtube.com/c/javaexpress
+91 7801007910
Java Express

Main Method Logic



```java
@SpringBootApplication
public class Application implements CommandLineRunner {

    @Autowired
    private ProductRepository productRepository;

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @PostConstruct
    public void m1() {
        // save record into database
        Product p = new Product();
        // var p = new Product();
        p.setName("Iphone 17");
        p.setPrice(570000d);
        p.setDescription("Mobile Phones");
        p.setBarCode("ORD_" + UUID.randomUUID().toString());

        productRepository.save(p);
    }

    @Override
    public void run(String... args) throws Exception {

    }
```