

Spring Data Advanced Querying

Query Methods, JPQL Advanced Repositories, Spring Configuration



SoftUni Team

Technical Trainers



SoftUni



Software University

<https://softuni.org/>

1. Retrieving Data by Custom Queries
2. Java Persistence Query Language
3. Repository Inheritance
4. Spring Custom Configuration

sli.do

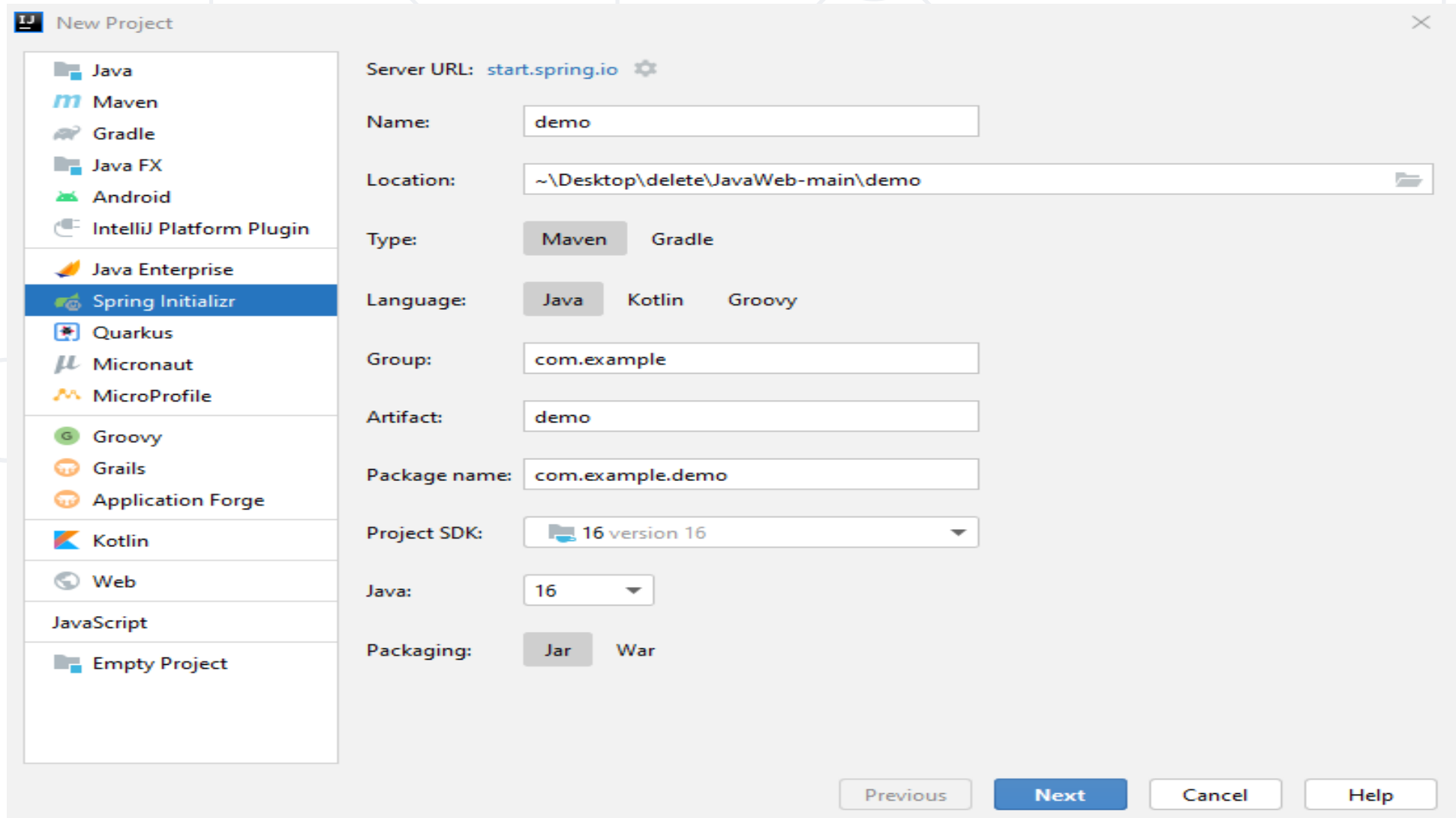
#



Retrieving Data by Custom Queries

Querying

Spring Project (1)



The image shows the 'New Project' dialog in IntelliJ IDEA. On the left, a list of project types is shown, with 'Spring Initializr' selected. On the right, various configuration fields are visible, including 'Server URL', 'Name', 'Location', 'Type', 'Language', 'Group', 'Artifact', 'Package name', 'Project SDK', 'Java', and 'Packaging'. The 'Next' button is highlighted in blue.

New Project

Server URL: start.spring.io ⚙️

Name:

Location:

Type: ☒ Maven ☐ Gradle

Language: ☒ Java ☐ Kotlin ☐ Groovy

Group:

Artifact:

Package name:

Project SDK:

Java:

Packaging: ☒ Jar ☐ War

Previous **Next** Cancel Help

Spring Project (2)

New Project

Spring Boot: 2.5.0

Dependencies:

Search

SQL

☐ JDBC API

☒ Spring Data JPA

☐ Spring Data JDBC

☐ Spring Data R2DBC

☐ MyBatis Framework

☐ Liquibase Migration

☐ Flyway Migration

☐ JOOQ Access Layer

☐ IBM DB2 Driver

☐ Apache Derby Database

☐ H2 Database

☐ HyperSQL Database

☐ MariaDB Driver

☐ MS SQL Server Driver

☒ MySQL Driver

☐ Oracle Driver

☐ PostgreSQL Driver

Spring Data JPA

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

[Guide](#)

Added dependencies:

× Spring Data JPA

× MySQL Driver

Previous

Finish

Cancel

Help

application.properties – simple example

application.properties

#Data Source Properties

```
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/1working?useSSL=false&createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=12345
```

#JPA Properties

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.jpa.properties.hibernate.format_sql = TRUE
spring.jpa.hibernate.ddl-auto = update
spring.jpa.open-in-view=false
```

###Logging Levels

Disable the default loggers

```
logging.level.org = WARN
logging.level.blog = WARN
```

#Show SQL executed with parameter bindings

```
logging.level.org.hibernate.SQL = DEBUG
logging.level.org.hibernate.type.descriptor = TRACE
```

ShampooRepository.java

```
@Repository
public interface ShampooDao extends JpaRepository <Shampoo, Long> {

    List<Shampoo> findByBrand(String brand);
}
```

Query method

Parameter



SQL

```
SELECT *
FROM shampoos AS s
WHERE s.brand = ?
```

Parameter

Query Prefix

Field

```
List<Shampoo> findByBrand(String brand);
```

Return Type

Query Prefix

Field

Field

```
List<Shampoo> findByBrandAndSize  
(String brand, Size size);
```

Predicate Keyword

ShampooRepository.java

@Repository

public interface ShampooRepository extends JpaRepository<Shampoo, Long> {

Query method

Paramater

List<Shampoo> **findByBrandAndSize**(String brand, Size size);

}

Paramater

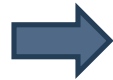
SQL

SELECT *
FROM shampoos AS s
WHERE s.brand = ?
AND s.size = ?

Problem: Select Shampoos by Size

- Write a method that selects all shampoos by input size
 - Order the result by shampoo id
- Example input-output:

MEDIUM



Nature Moments Mediterranean Olive Oil & Aloe Vera MEDIUM 6.50lv.
Volume & Fullness Lavender MEDIUM 5.50lv.
Rose Shine & Hydration MEDIUM 6.50lv.
Color Protection & Radiance MEDIUM 6.75lv.
...

Solution: Select Shampoos by Size

ShampooRepository.java

```
@Repository
public interface ShampooRepository extends JpaRepository<Shampoo, Long> {
    List<Shampoo> getAllBySizeOrderBySize(sizeValue);
}
```



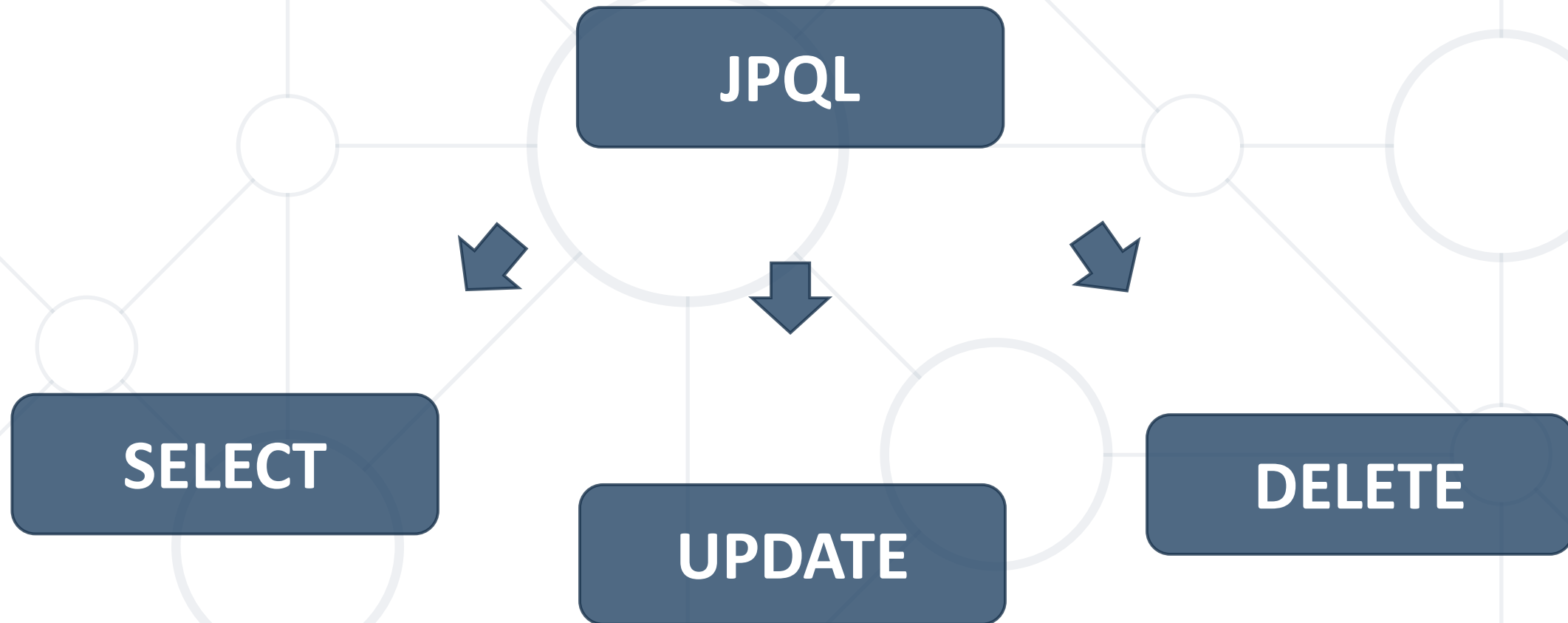
Java Persistence Query Language

JPQL

JPQL

- **Object-oriented** query language
 - Part of the Java Persistence API
 - Used to make queries against entities stored in a relational database
 - SQL syntax **operating with entities**, not tables in the data source





Entity Class

Field

```
"SELECT i FROM Ingredient AS i WHERE i.name IN :names"
```

Object

Alias

Parameter

Object

```
"SELECT s  
    FROM Shampoo AS s  
    INNER JOIN s.batch AS b  
    WHERE b.batchDate < :batchDate"
```

Class

Join

Field

Paramater

■ Update:

```
"UPDATE Ingredient AS b  
    SET b.price = b.price*1.10  
    WHERE b.name IN :names"
```

Parameter

■ Delete:

```
"DELETE FROM Ingredient AS b  
    WHERE b.name = :name"
```

Problem: Select Shampoos by Ingredients

- Write a method that selects all shampoos with ingredients in the given list
- Example input-output:

Berry
Mineral-Colagen



Color Protection & Radiance
Fresh it Up!
Nectar Nutrition
Superfruit Nutrition
Color Protection & Radiance
Nectar Nutrition
...

Solution: Select Shampoos by Ingredients

ShampooRepository.java

@Repository

```
public interface IngredientRepository extends JpaRepository<Ingredient, Long>{
```

```
    @Query(value = "select s from Shampoo s " +  
        "join s.ingredients i where i in :ingredients")  
    List<Shampoo> findByIngredientsIn(@Param(value = "ingredients")  
        Set<Ingredient> ingredients);
```

```
}
```



Repository Inheritance

Advanced Repositories

Repository Inheritance

- In bigger applications, we have **similar entities**, extending an **abstract class**
- Their base attributes and actions, towards them, are the same regardless of their differences
- We can set up a **base repository** to reduce query and code duplication
- It can be **inherited** to clear up specifics



Example: Repository Inheritance (1)

Not a repository

IngredientRepository.java

```
@NoRepositoryBean
public interface IngredientRepository<T extends Ingredient>
    extends JpaRepository<T, Long>{
    //...
}
```

ChemicalIngredientRepository.java

```
@Repository
public interface ChemicalIngredientRepository extends IngredientRepository
    <BasicChemicalIngredient> {
    List<ChemicalIngredient> findByChemicalFormula(String chemicalFormula);
}
```

Example: Repository Inheritance (2)

CustomShampooRepository.java

```
public interface CustomShampooRepository {  
    void create(BasicShampoo basicShampoo);  
}
```

CustomShampooRepositoryImpl.java

```
@Repository  
public class CustomShampooDaoImpl implements CustomShampooRepository {  
    @PersistenceContext  
    private EntityManager entityManager;  
  
    @Transactional  
    public void create(BasicShampoo basicShampoo){  
        entityManager.persist(basicShampoo);  
    }  
}
```

Inject
Entity
Manager

Single Transaction



Spring Custom Configuration

Java-Based Setup

- So far, we've configured our project with a spring properties file:

application.properties

#Data Source Properties

```
spring.datasource.driverClassName = com.mysql.jdbc.Driver  
spring.datasource.url = jdbc:mysql://localhost:3306/shampoo_company?useSSL=false&  
createDatabaseIfNotExist=true  
spring.datasource.username = root  
spring.datasource.password = 1234
```

Connection properties

Java-Based Configuration (1)

Configuration
Class

JavaConfig.java

```
@Configuration
@EnableJpaRepositories(basePackages = "com.demo.dao")
@EnableTransactionManagement
@PropertySource(value = "application.properties" )
public class JavaConfig {
    //Add configuration
}
```

Property File

Repositories
Directory

Java-Based Configuration (2)

JavaConfig.java

@Autowired

private Environment environment;

Data Source Connection

@Bean

public DataSource dataSource() {

```
    DriverManagerDataSource driverManagerDataSource = new DriverManagerDataSource();
    driverManagerDataSource.setDriverClassName(environment.getProperty("spring.datasource.
driverClassName"));
    driverManagerDataSource.setUrl(environment.getProperty("spring.datasource.url"));
    driverManagerDataSource.setUsername(environment.getProperty("spring.datasource.
username"));
    driverManagerDataSource.setPassword(environment.getProperty("spring.datasource.
password"));
    return driverManagerDataSource;
}
```

JavaConfig.java

@Bean

```
public EntityManagerFactory entityManagerFactory() {  
  
    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();  
    vendorAdapter.setDatabase(Database.MYSQL);  
    vendorAdapter.setGenerateDdl(true);  
    vendorAdapter.setShowSql(true);  
    LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();  
    factory.setJpaVendorAdapter(vendorAdapter);  
    factory.setPackagesToScan("com.demo.domain");  
    factory.setDataSource(dataSource());  
    Properties jpaProperties = new Properties();  
    jpaProperties.setProperty("hibernate.hbm2ddl.auto", "validate");  
    jpaProperties.setProperty("hibernate.format_sql", "true");  
    factory.setJpaProperties(jpaProperties);  
    factory.afterPropertiesSet();  
    return factory.getObject();  
}
```

JPA Configuration

Models Package

JavaConfig.java

Transaction Manager Configuration

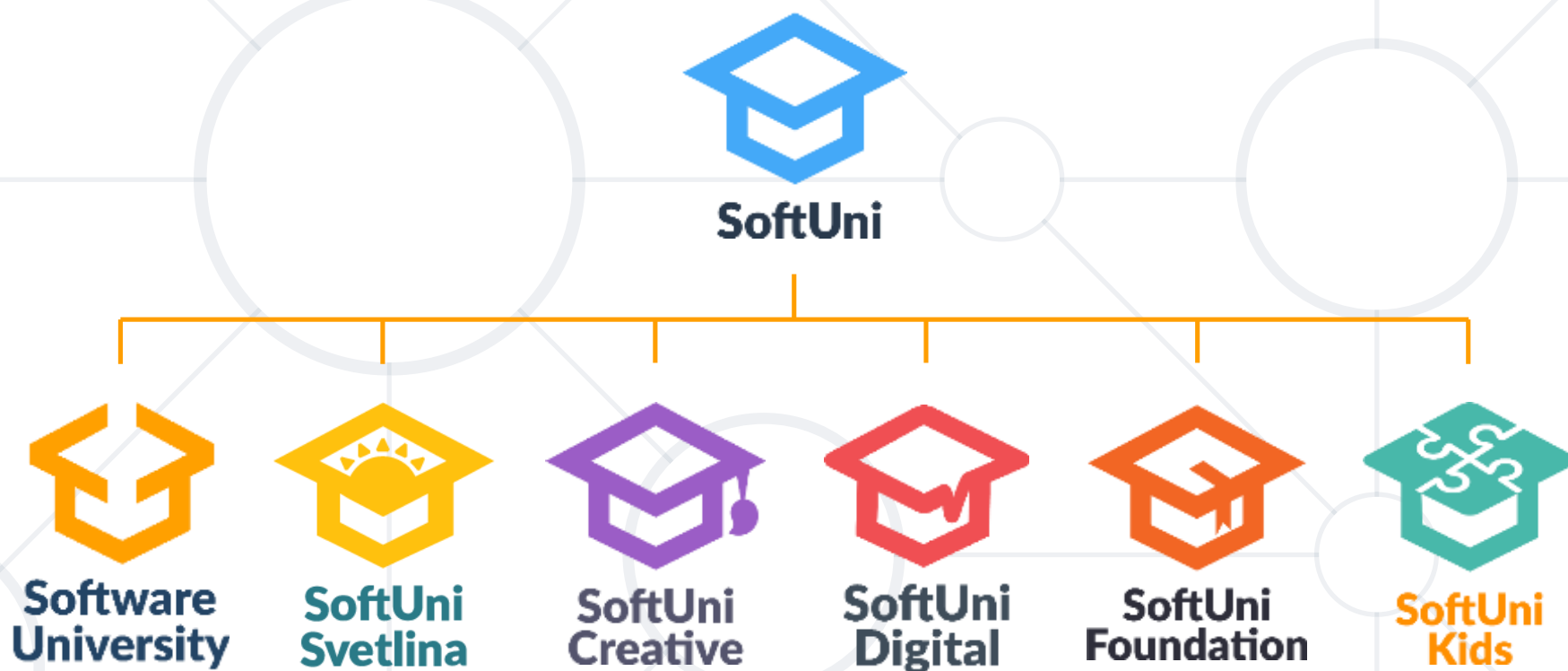
@Bean

```
public PlatformTransactionManager transactionManager() {  
    JpaTransactionManager txManager = new JpaTransactionManager();  
    txManager.setEntityManagerFactory(entityManagerFactory());  
    return txManager;  
}
```

- Spring Data translates **methods to SQL Queries**
- We can **write custom queries**
 - JPQL syntax on entity classes
- Repositories **can be inherited**
 - Reduces code duplication for inherited entities



Questions?



SoftUni Diamond Partners



SCHWARZ



**SUPER
HOSTING
.BG**



INDEAVR
Serving the high achievers

Bosch.io





Trainings @ Software University (SoftUni)



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- SoftUni Global
 - softuni.org



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>
- © SoftUni Global – <https://softuni.org>

