**Demonstrate implementation Query methods feature of Spring Data JPA**

**Implementation of Query Methods in Spring Data JPA**

**Introduction**

Spring Data JPA is a powerful module in the Spring ecosystem that provides a high-level abstraction for accessing relational databases using JPA. One of its most valuable features is the **Query Methods**, which allow developers to define database queries **just by method names**, without writing any JPQL or SQL.

These methods are based on a **method naming convention**, where Spring interprets the method name and auto-generates the appropriate query.

**Why Use Query Methods?**

* Removes boilerplate code
* Reduces the need for manually written queries
* Ensures faster development and cleaner code
* Integrates seamlessly with Spring Boot

**Common Keywords Used in Query Methods**

| **Keyword** | **Description** |
| --- | --- |
| findBy | Selects by a field |
| Containing | SQL LIKE %value% |
| StartingWith | SQL LIKE value% |
| GreaterThan | SQL > |
| LessThan | SQL < |
| Between | SQL BETWEEN x AND y |
| OrderBy | Sorts the result |
| Top | Fetches limited results (e.g. top 3) |

**🔹 Benefits of Query Methods in Real-Time Projects**

* Saves development time and effort
* Ensures cleaner, readable method names
* Reduces errors in query syntax
* Easy to maintain and refactor
* Works well with Spring Boot REST controllers

**Conclusion**

Spring Data JPA’s Query Methods offer a powerful and developer-friendly way to query data without writing boilerplate code or complex SQL. By using **conventions over configurations**, these methods allow developers to build dynamic queries using only the method name, resulting in highly readable and maintainable code.

It is an essential feature for anyone building modern, data-driven Spring applications.

OrmLearnApplication.java

testCountriesByPartialName();

testCountriesByPartialNameOrdered();

testCountriesByAlphabet();   
private static void testCountriesByPartialName() {

LOGGER.info("Start testCountriesByPartialName");

List<Country> countries = countryService.getCountriesByNameContaining("ou");

LOGGER.debug("Countries containing 'ou': {}", countries);

LOGGER.info("End testCountriesByPartialName");

}

private static void testCountriesByPartialNameOrdered() {

LOGGER.info("Start testCountriesByPartialNameOrdered");

List<Country> countries = countryService.getCountriesByNameContainingOrdered("ou");

LOGGER.debug("Countries containing 'ou' ordered: {}", countries);

LOGGER.info("End testCountriesByPartialNameOrdered");

}

private static void testCountriesByAlphabet() {

LOGGER.info("Start testCountriesByAlphabet");

List<Country> countries = countryService.getCountriesByNameStartingWith("Z");

LOGGER.debug("Countries starting with 'Z': {}", countries);

LOGGER.info("End testCountriesByAlphabet");

}

CountryService.java

@Transactional

public List<Country> getCountriesByNameContaining(String fragment) {

return countryRepository.findByNameContaining(fragment);

}

@Transactional

public List<Country> getCountriesByNameContainingOrdered(String fragment) {

return countryRepository.findByNameContainingOrderByNameAsc(fragment);

}

@Transactional

public List<Country> getCountriesByNameStartingWith(String alphabet) {

return countryRepository.findByNameStartingWith(alphabet);

}

CountryRepository.java

package com.cognizant.ormlearn.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String nameFragment);

List<Country> findByNameContainingOrderByNameAsc(String nameFragment);

List<Country> findByNameStartingWith(String alphabet);

}

**OUTPUT:**A screenshot of a computer screen

AI-generated content may be incorrect.