**Exercise 9: Employee Management System – Customizing Data Source Configuration**

Business Scenario:

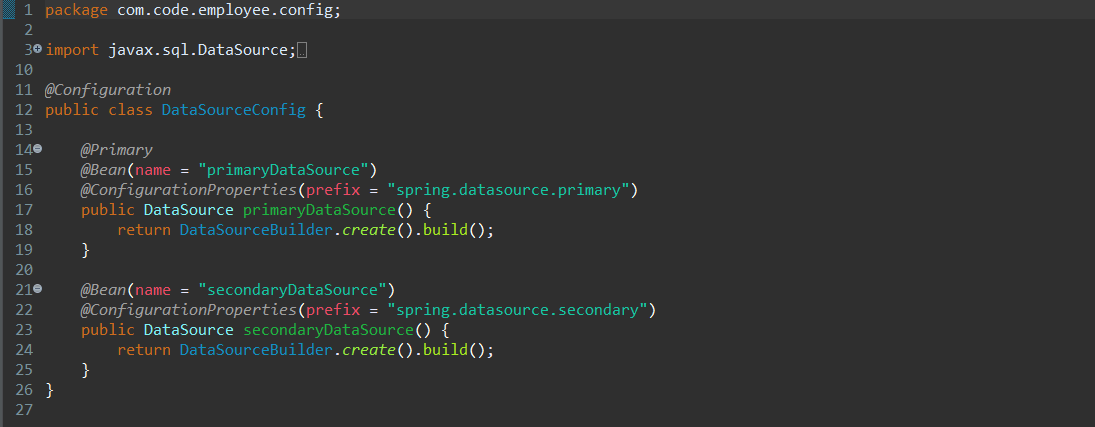
Customize your data source configuration and manage multiple data sources.

**1. Spring Boot Auto-Configuration:**

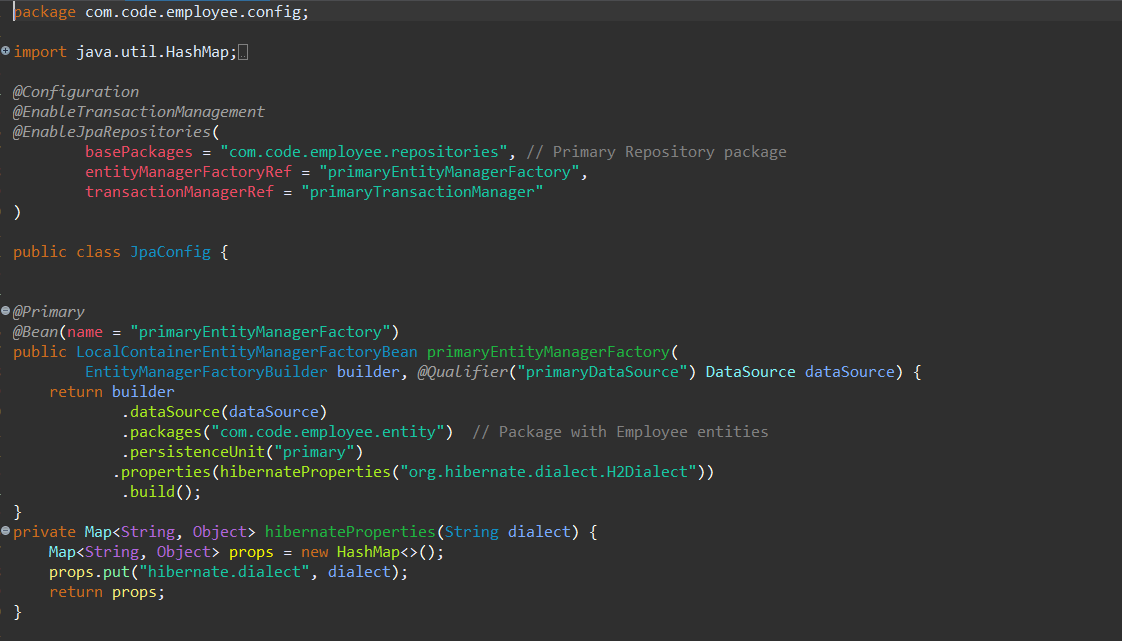
* Leverage Spring Boot auto-configuration for data sources.

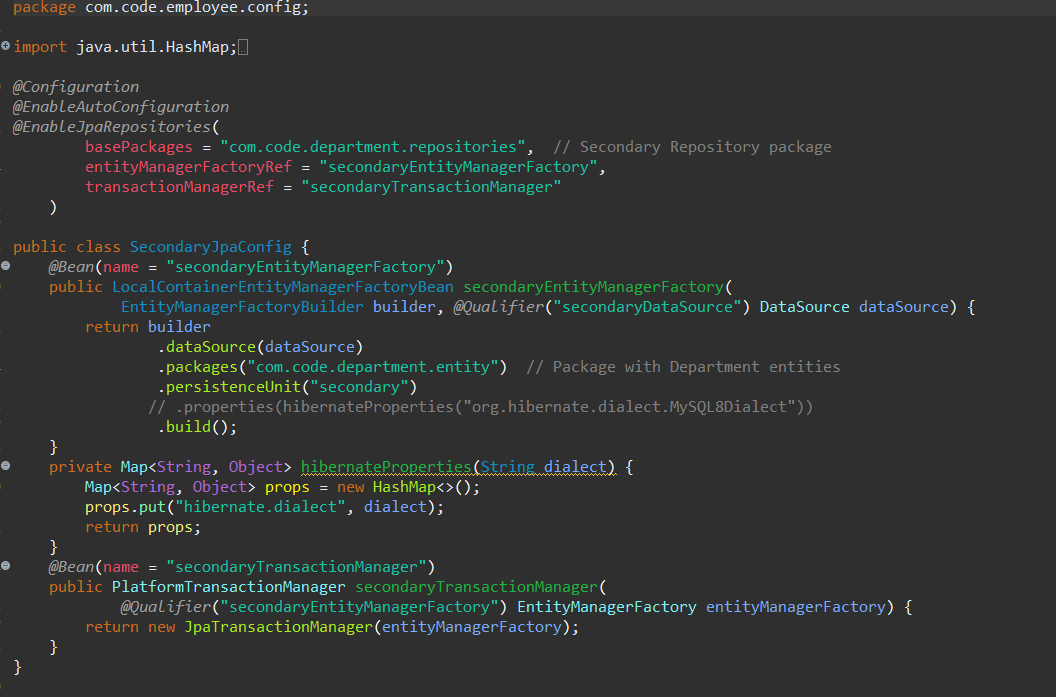
In this exercise I modified the existing code, by leveraging Spring Boot auto-configuration for data sources.

1. I created a package named “com.code.employee.config”
2. I created a class named DataSourceConfig



1. I created two more classes named JpaConfig and SecondaryJpaConfig



****

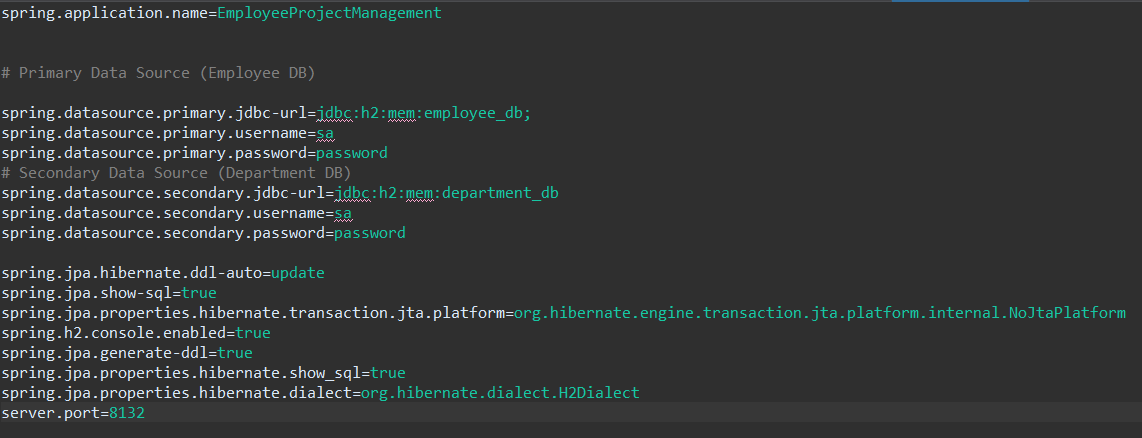
**2. Externalizing Configuration:**

* Externalize configuration with application.properties.

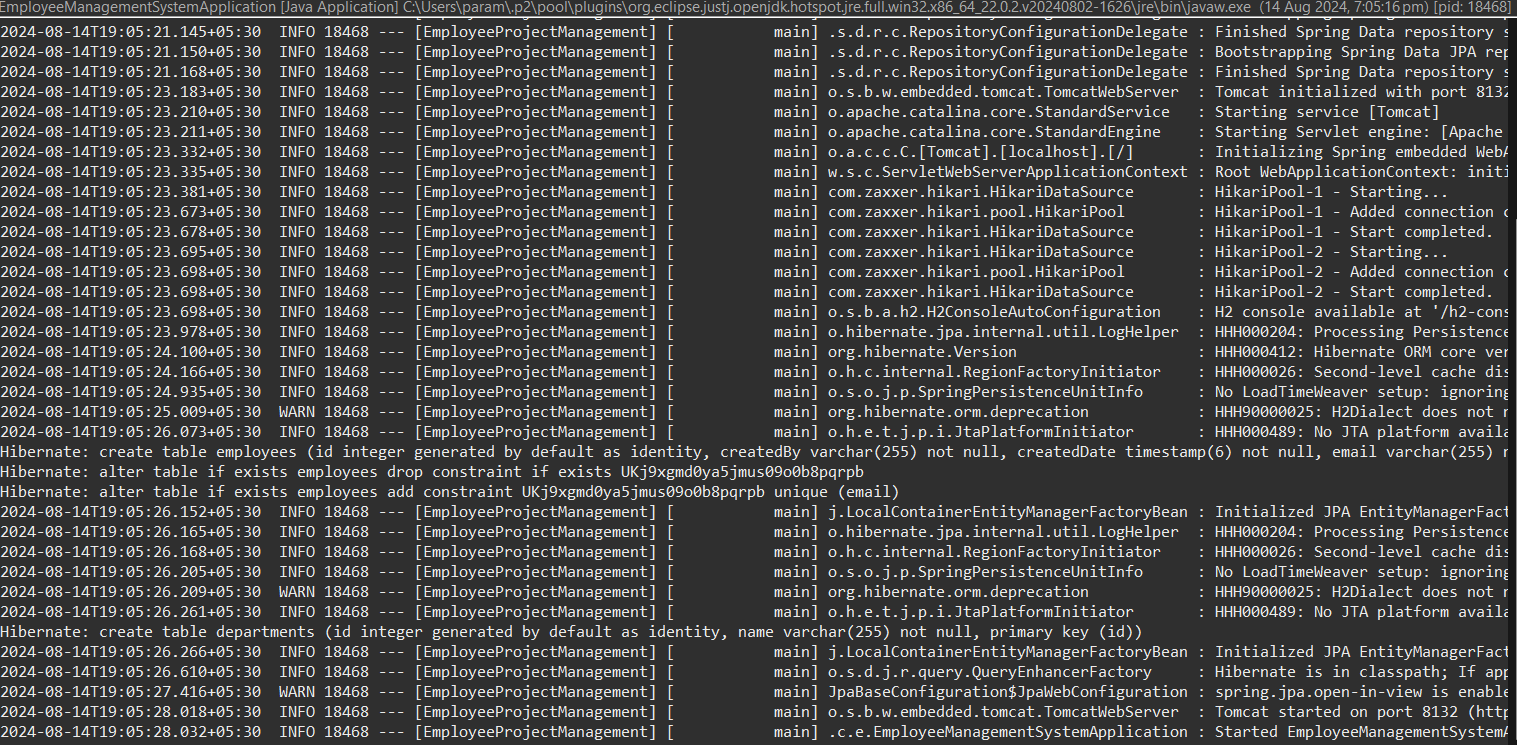
To manage multiple data sources, we need to define each data source in the application.properties file and configure them in the Spring Boot application.

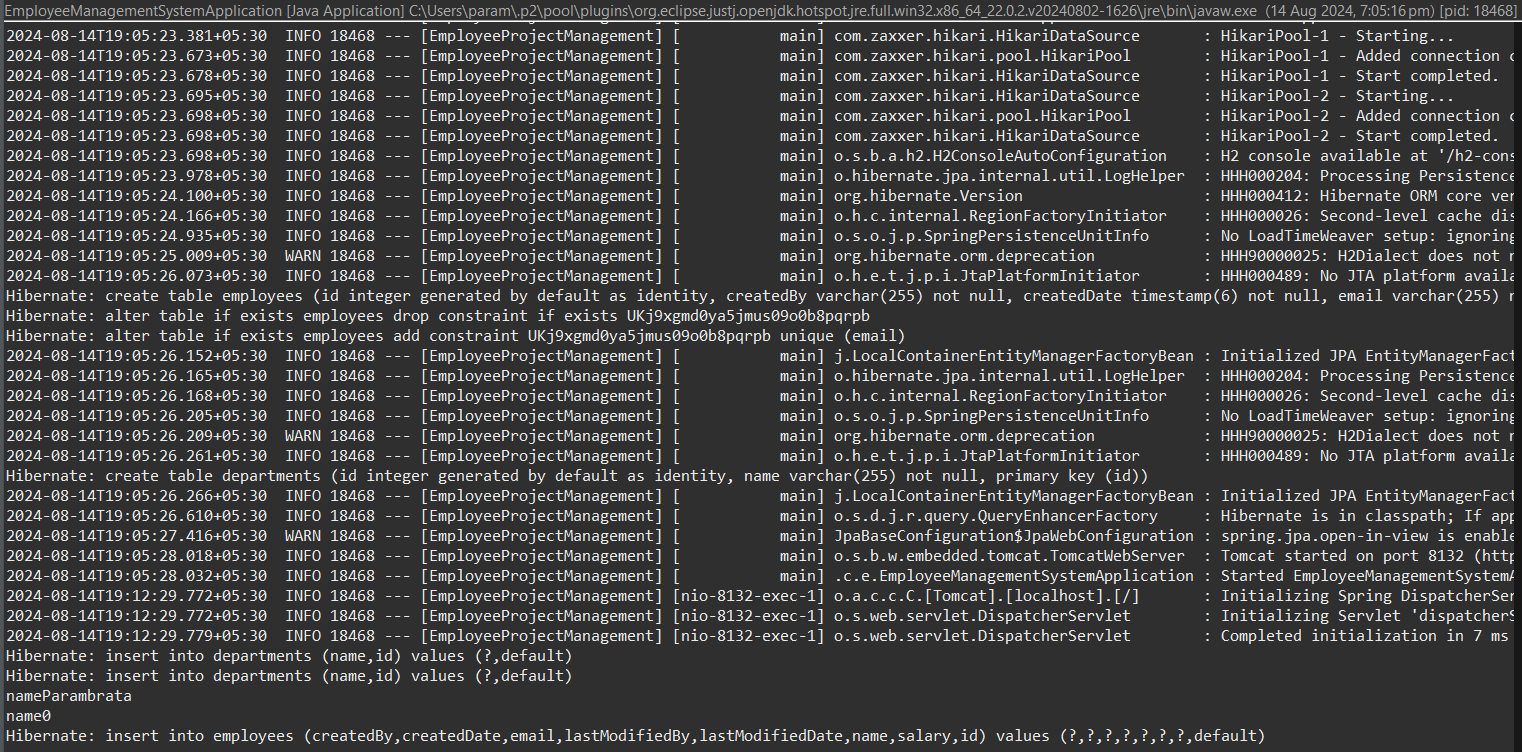
* Manage multiple data sources within your application.

I can now manage entities and perform operations in both db1 and db2 databases independently. The @Primary annotation ensures that the first data source (employee\_db) is used by default where a single data source is required.

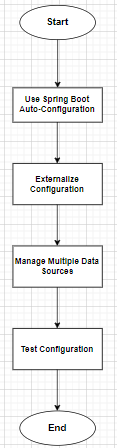


**Output:**

****



**Flowchart:**

****

** Start - Begin the process.**

** Use Spring Boot Auto-Configuration - Leverage auto-configuration for the data source.**

* **Single Data Source Configuration - Configure a single data source using Spring Boot's auto-configuration.**
* **Externalize Configuration - Move configuration to application.properties.**

** Externalize Configuration:**

* **Define Properties in application.properties - Set up properties for data sources.**
* **Multiple Data Sources - Set up multiple data sources.**

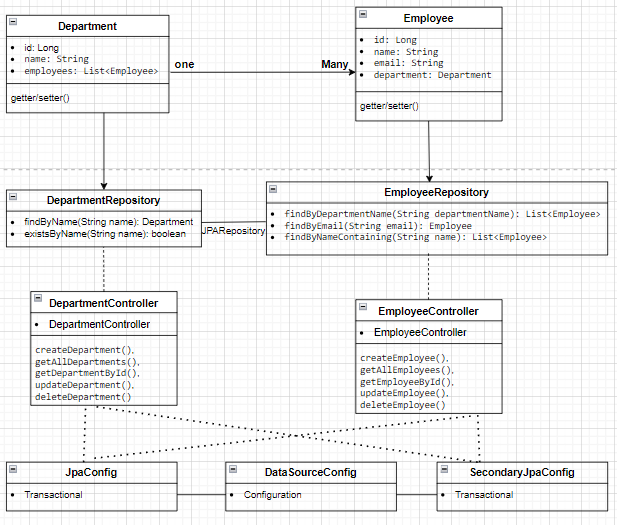
** Manage Multiple Data Sources:**

* **Create Configuration Classes - Create configuration classes for each data source.**
* **Define DataSource, EntityManagerFactory, and TransactionManager - Configure these beans for each data source.**

** Test Configuration - Verify the configuration by running the application and checking the functionality.**

** End - The process ends.**

**Class Diagram:**

****

** Department Class:**

* **Attributes: Represents the id, name, and a list of Employee objects.**
* **Relationship: Has a one-to-many relationship with Employee.**

** Employee Class:**

* **Attributes: Represents the id, name, email, and a reference to the Department object.**
* **Relationship: Each Employee belongs to a single Department.**

** EmployeeRepository Interface:**

* **Methods:**
  + **findByDepartmentName(String departmentName): List<Employee>**
  + **findByEmail(String email): Employee**
  + **findByNameContaining(String name): List<Employee>**

** DepartmentRepository Interface:**

* **Methods:**
  + **findByName(String name): Department**
  + **existsByName(String name): Boolean**

** EmployeeController Class:**

* **Methods:**
* **createEmployee()**
* **getAllEmployees()**
* **getEmployeeById()**
* **updateEmployee()**
* **deleteEmployee().**

** DepartmentController Class:**

* **Methods**
* **createDepartment()**
* **getAllDepartments()**
* **getDepartmentById()**
* **updateDepartment()**
* **deleteDepartment()**

** DataSourceConfig Class:**

** JpaConfig Class:**

** SecondaryJpaConfig Class:**

**Relationship:**

* A Department can have many Employees (OneToMany relationship).
* An Employee belongs to one Department (ManyToOne relationship).
* Department to DepartmentRepository
* Employee to EmployeeRepository
* DataSourceConfig to JpaConfig and SecondaryJpaConfiig

**Analysis:**

**1. Leverage Spring Boot Auto-Configuration for Data Sources**

Spring Boot provides an auto-configuration feature for setting up data sources. When we add a database dependency (like spring-boot-starter-data-jpa) to our project, Spring Boot automatically configures a DataSource based on the properties defined in your application.properties

With these properties, Spring Boot auto-configures a DataSource bean, an EntityManagerFactory, and a TransactionManager

**2. Externalize Configuration and Manage Multiple Data Sources**

The application requires connecting to multiple databases, I defined and configured multiple DataSource beans.

 **Auto-Configuration**: Leverage Spring Boot's auto-configuration for single data source setup.

 **Externalization**: Define multiple data sources in application.properties.

 **Custom Configuration**: Create separate configuration classes for each data source, managing DataSource, EntityManagerFactory, and TransactionManager.

 **Repository Management**: Repositories are correctly mapped to the respective data sources.