**Exercise 10: Employee Management System - Hibernate-Specific Features**

Business Scenario:

Leverage Hibernate-specific features to enhance your application's performance and capabilities.

**1. Hibernate-Specific Annotations**

* **Purpose:** Customizes entity mappings to optimize performance and manage database interactions efficiently.
* **Key Usage:**
  + @BatchSizeto optimize batch processing.
  + @Fetch(FetchMode.JOIN) to improve the performance of queries by reducing the number of database calls.

**2. Configuring Hibernate Dialect and Properties**

* **Hibernate Dialect:** Configured in application.properties to match the specific database
* **Batch Processing Properties:**
  + hibernate.jdbc.batch\_size=10: Defines the batch size for insert and update operations.
  + hibernate.order\_inserts=true and hibernate.order\_updates=true: Ensures that inserts and updates are ordered to minimize deadlocks.

**3. Batch Processing with Hibernate**

* **Batch Insert:**
  + Efficiently handles large data sets by grouping multiple insert operations.
* **Batch Update:**
  + Groups multiple update operations to enhance performance.
* **Bulk Operations in Batch:**
  + Custom methods in BatchService to handle batch inserts, updates, and specific tasks like updating email domains across multiple employees.

**4. Data Source Configuration**

* **Primary Data Source:** An in-memory H2 database is configured for testing and development.
* **Secondary Data Source:** A MySQL database is configured for production, demonstrating the use of multiple data sources within the application.

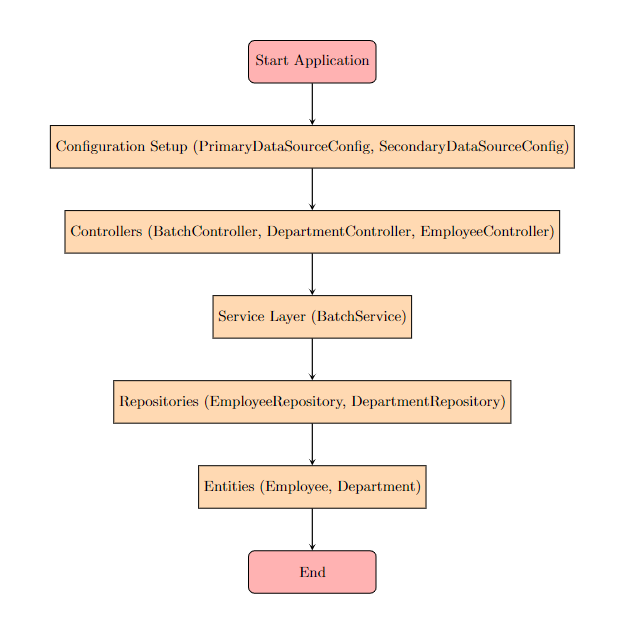
**5. Entity Auditing**

* **Auditing:** Entities like Department are configured with auditing annotations to track creation and modification timestamps automatically.

**6. RESTful API Design**

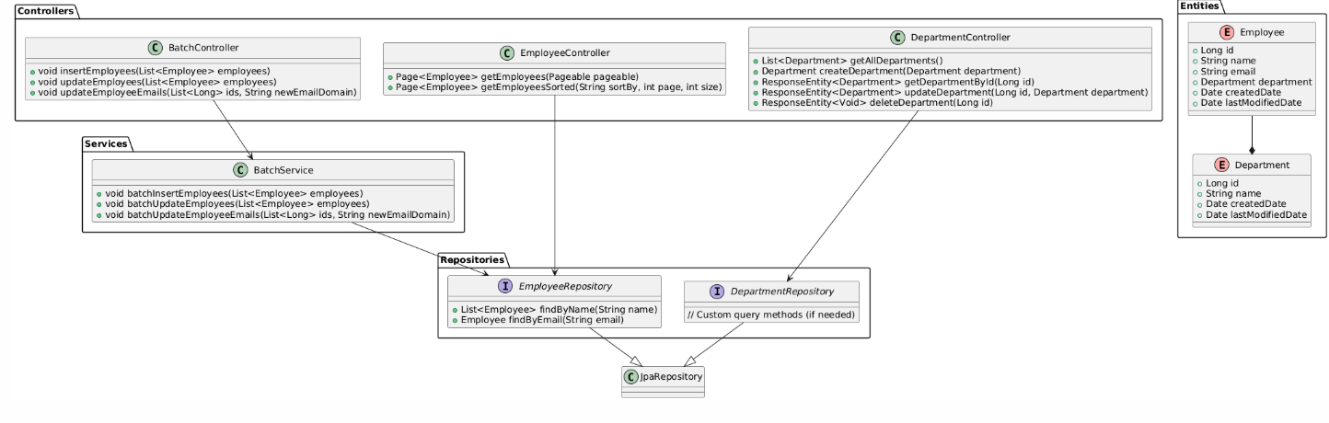
* **Controllers:**
  + EmployeeController and DepartmentController manage CRUD operations for Employee and Department entities.
  + BatchController handles batch operations, providing endpoints for bulk data insertion and updates.

**FLOWCHART :**



**Explanation:**

* **Start Application**: Represents the starting point of the application.
* **Configuration Setup**: Involves setting up the primary and secondary data sources (PrimaryDataSourceConfig, SecondaryDataSourceConfig).
* **Controllers**: Involves handling HTTP requests for various operations (BatchController, DepartmentController, EmployeeController).
* **Service Layer**: Handles the business logic, such as batch processing (BatchService).
* **Repositories**: Interacts with the database using EmployeeRepository and DepartmentRepository.
* **Entities**: Represents the entities (Employee, Department) that are managed by the application.
* **End**: Marks the end of the application process.

**CLASS DIAGRAM :** 

**Explanation:**

* **Entities**:
  + Employee: Represents the employee entity with attributes like id, name, email, department, createdDate, and lastModifiedDate.
  + Department: Represents the department entity with attributes like id, name, createdDate, and lastModifiedDate.
* **Repositories**:
  + EmployeeRepository: Extends JpaRepository and provides custom query methods like findByName and findByEmail.
  + DepartmentRepository: Extends JpaRepository and may contain custom query methods if needed.
* **Services**:
  + BatchService: Provides methods for batch operations like inserting, updating employees, and updating employee emails in bulk.
* **Controllers**:
  + BatchController: Handles batch operations for employees.
  + DepartmentController: Manages CRUD operations for departments.
  + EmployeeController: Manages CRUD operations and sorting for employees.

**Relationships:**

* Employee is associated with Department using a many-to-one relationship.
* EmployeeRepository and DepartmentRepository extend the JpaRepository interface.
* BatchService interacts with EmployeeRepository.
* BatchController interacts with BatchService.
* DepartmentController interacts with DepartmentRepository.
* EmployeeController interacts with EmployeeRepository.