**Exercise 2: Employee Management System - Creating Entities**

Business Scenario:

Define JPA entities for Employee and Department with appropriate relationships.

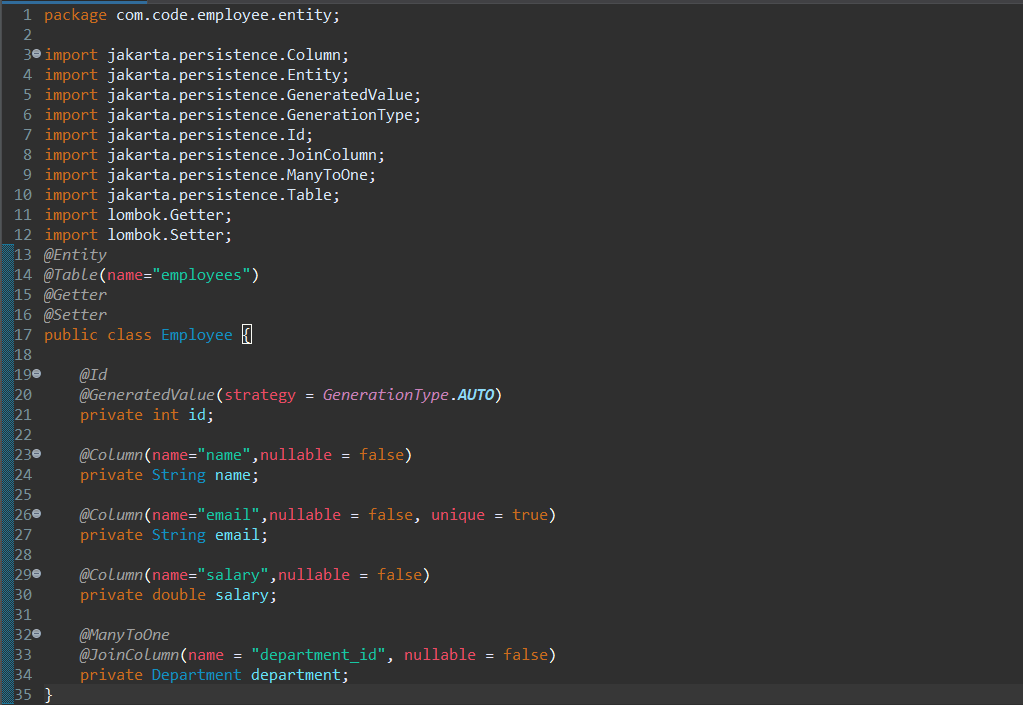
**1. Creating JPA Entities:**

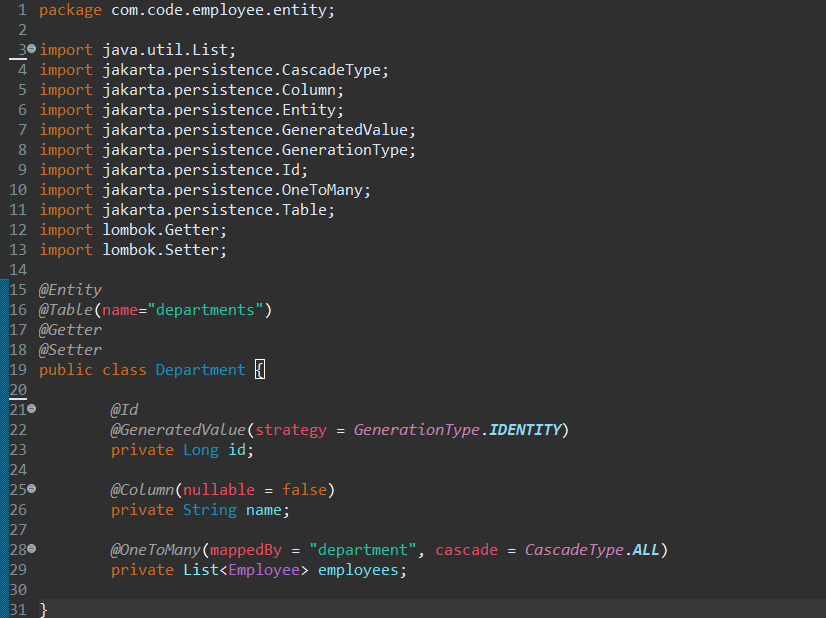
In this exercise, I created JPA entities > a class named Employee with the fields id, name, email, and department

Another class named Department is created with fields id and name.

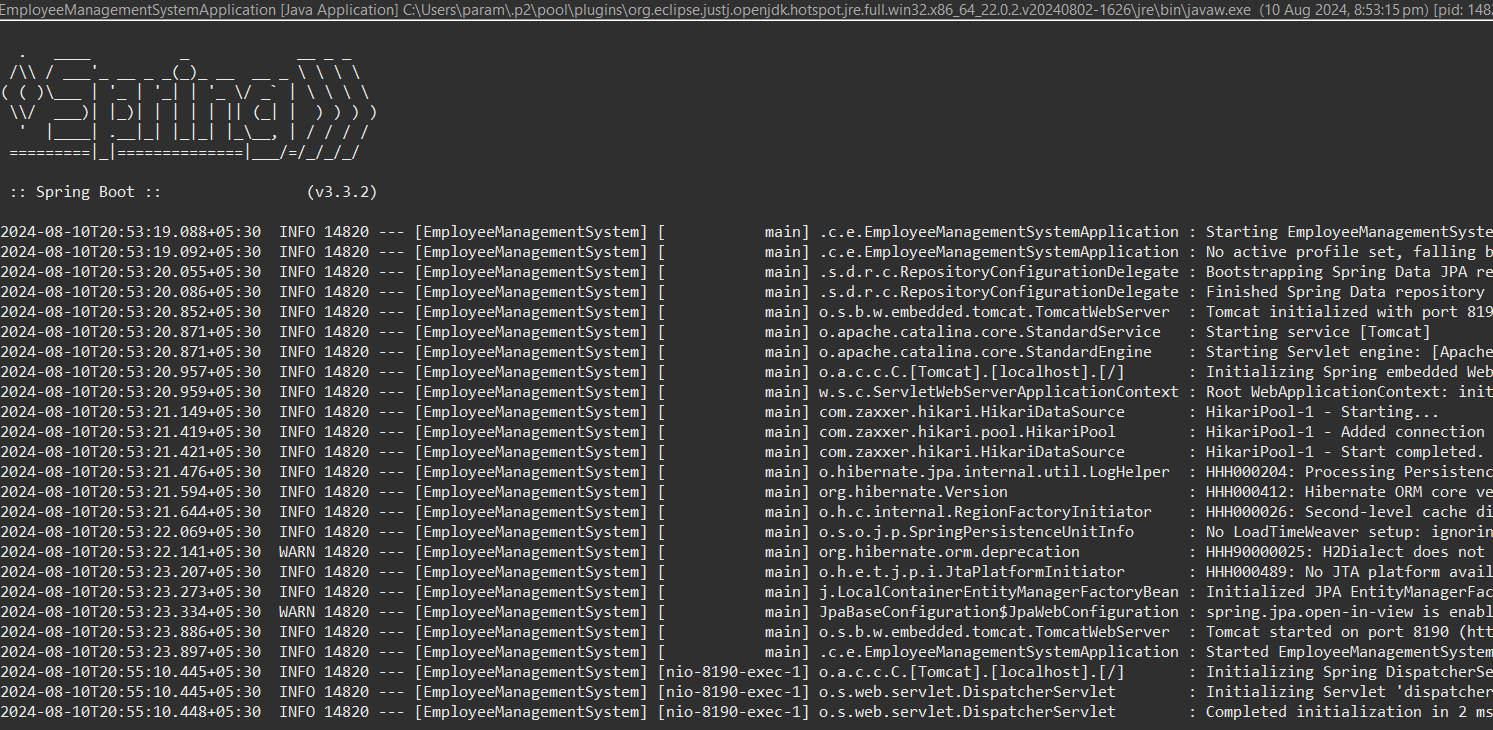
**2. Mapping Entities to Database Tables:**

I have used annotations like @**Entity, @Table, @Id, @GeneratedValue and defined** one-to-many relationship between **Department** and **Employee**.

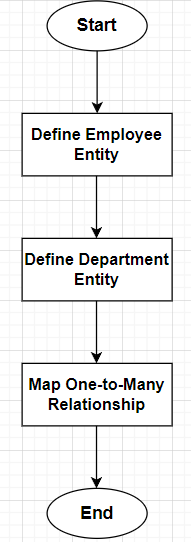




**Output:**

****

**Flowchart:**

****

**1. Start**

* This is the entry point of the flowchart. It signifies the beginning of the process where you start defining the JPA entities.

**2. Define Employee Entity**

* **Action**: Create the Employee class.
* **Details**:
  + Fields: id, name, email, and department.
  + **Annotations**:
    - @Entity: Marks the class as a JPA entity.
    - @Table: Specifies the table name in the database.
    - @Id: Marks the primary key field.
    - @GeneratedValue: Automatically generates the primary key value.
    - @ManyToOne: Defines a many-to-one relationship with the Department entity.
    - @JoinColumn: Specifies the foreign key column name for the relationship.
* **Purpose**: This step establishes the basic structure of the Employee entity and how it will be represented in the database.

**3. Define Department Entity**

* **Action**: Create the Department class.
* **Details**:
  + Fields: id, name, and employees.
  + **Annotations**:
    - @Entity: Marks the class as a JPA entity.
    - @Table: Specifies the table name in the database.
    - @Id: Marks the primary key field.
    - @GeneratedValue: Automatically generates the primary key value.
    - @OneToMany: Defines a one-to-many relationship with the Employee entity.
* **Purpose**: This step sets up the Department entity, which will contain a list of employees associated with each department.

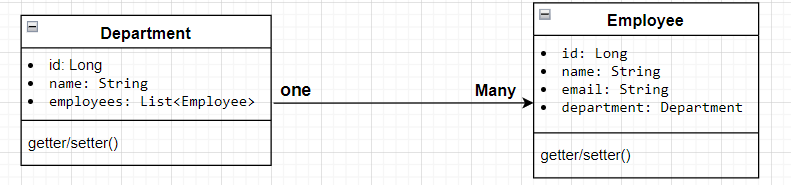
**4. Map One-to-Many Relationship**

* **Action**: Define the relationship between the Department and Employee entities.
* **Details**:
  + In Department: The @OneToMany annotation indicates that one department can have many employees.
  + In Employee: The @ManyToOne annotation indicates that each employee belongs to one department.
* **Purpose**: This step ensures that the relational mapping between the two entities is established. It allows the application to manage the relationship between departments and employees in the database.

**5. End**

* This is the exit point of the flowchart, indicating that the process of defining the entities and mapping the relationships is complete.

**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.**

**Analysis:**

**@Entity**: Marks the class as a JPA entity, making it a managed entity by the JPA provider (Hibernate, in this case).

**@Table(name = "table\_name")**: Specifies the name of the database table to which the entity is mapped. If not specified, the table name defaults to the entity name.

**@Id**: Marks the field as the primary key of the entity.

**@GeneratedValue(strategy = GenerationType.IDENTITY)**: Specifies that the primary key value is automatically generated by the database. The IDENTITY strategy is often used for auto-incremented fields in SQL databases.

**@Column**: Used to define properties of the column in the database. For example:

* nullable = false: The column cannot be null.
* unique = true: The column must have unique values across the table.

**@ManyToOne**: Defines a many-to-one relationship. In this case, many Employee entities can be associated with one Department.

**@JoinColumn(name = "column\_name")**: Specifies the foreign key column name in the employees table that will reference the department table’s primary key.

**@OneToMany(mappedBy = "field\_name", cascade = CascadeType.ALL)**: Defines a one-to-many relationship. The mappedBy attribute indicates the entity that owns the relationship (i.e., the entity with the foreign key). CascadeType.ALL ensures that any operations (like persist, delete) are cascaded from the Department to its associated Employee entities.

**Relationship:**

**Department to Employee**:

* A Department can have many Employees (OneToMany relationship).
* An Employee belongs to one Department (ManyToOne relationship).