Intro

Favorite	☐
Tag	Master Hibernate & JPA with Spring Boot

▼ Notes

ApplicationContext → Spring framework manages components in here

Bean → Any instance of any component

Dependency Injection → Identify beans, their dependencies and wire them together (Provides IOC)

Types of IOC Container:

- ApplicationContext (complex)
- BeanFactory(simpler features - rarely used)

▼ JPA vs JDBC

- Both JDBC and JPA are used for database access in Java applications, JDBC is lower-level and requires developers to write more code for database interactions, whereas JPA is higher-level, providing abstractions that simplify database access and allowing developers to work with Java objects directly. JPA also incorporates ORM concepts, making it easier to manage the mapping between Java objects and database tables.

JPA (Java Persistence API):

1. **Low-level API**: JDBC is a low-level API (Application Programming Interface) that provides a set of classes and interfaces for Java applications to interact directly with databases.
2. **Manual Handling**: With JDBC, developers need to write SQL queries explicitly for database operations like inserting, updating, deleting, and querying data.
3. **Database Connection Management**: Developers have to manage database connections manually, including opening and closing

connections, handling transactions, and managing resources efficiently.

4. **Fine-grained Control**: JDBC provides fine-grained control over database operations, giving developers flexibility but also requiring them to write more code.

JPA (Java Persistence API):

1. **High-level API**: JPA is a high-level API that simplifies the interaction between Java applications and databases by providing a set of abstractions for object-relational mapping (ORM).
2. **Object-Relational Mapping**: With JPA, developers work with Java objects directly, and JPA handles the mapping of these objects to database tables and vice versa. This means you can work with objects in your code without worrying about the underlying database structure.
3. **Automatic Query Generation**: JPA can automatically generate SQL queries based on methods in your Java code, reducing the need for developers to write SQL queries manually.
4. **Entity Management**: JPA provides built-in support for managing entities (objects mapped to database tables), including CRUD (Create, Read, Update, Delete) operations, relationships between entities, and caching.

▼ Entity

In JPA (Java Persistence API), an entity is a Java class that represents a table in a relational database. An entity typically corresponds to a row in the database table, and instances of the entity class represent individual records in that table.

Here are some key points about entities in JPA:

1. **Java Class**: An entity is defined as a regular Java class, usually annotated with `@Entity` annotation. This annotation marks the class as an entity, indicating that instances of this class can be persisted to and retrieved from a database.
2. **Mapped to a Table**: Each entity class is mapped to a specific table in the database. The structure of the entity class, including its fields (attributes), corresponds to the columns in the database table.

3. **Primary Key**: An entity typically has a primary key, which uniquely identifies each record in the database table. In JPA, this is often specified using the `@Id` annotation on one of the entity's fields.
4. **Persistence Context**: Instances of entity classes are managed by the JPA EntityManager within a persistence context. This means that JPA keeps track of changes made to entity instances and automatically synchronizes these changes with the underlying database when necessary.
5. **Relationships**: Entities can have relationships with other entities, such as one-to-one, one-to-many, or many-to-many relationships. These relationships are defined using JPA annotations like `@OneToOne`, `@OneToMany`, `@ManyToOne`, and `@ManyToMany`.
6. **Lifecycle Callbacks**: JPA provides lifecycle callback methods that allow developers to define custom logic to be executed before or after certain lifecycle events of an entity, such as persisting, updating, or deleting an entity.

Overall, entities in JPA serve as the bridge between object-oriented Java code and relational database tables, allowing developers to work with data in a more object-oriented manner while still leveraging the power of relational databases.

# According to the video, entities are objects that we would store to the database

▼ @Autowired

Alright, imagine you're building a Lego spaceship. Sometimes, you need certain pieces from other Lego sets to complete it. Now, think of `@Autowired` as a magical way for your spaceship to automatically get those specific pieces without you having to search for them.

In programming, when we're building something complex, like a Spring application, we often need different parts (or objects) to work together. `@Autowired` is like a special instruction you give to Spring, a tool used in Java development, telling it to find and plug in the right pieces (or beans) into the places where they're needed.

So, if you have a class that needs another class to do its job, you can mark a field with `@Autowired`, and Spring will take care of finding an appropriate object and plugging it in for you. It's like having a helpful robot assistant that knows exactly which Lego pieces your spaceship needs and puts them in the right spots for you. This makes building complex applications easier and faster because you don't have to manually connect every piece together.

Autowiring → Process of wiring in dependencies for a Spring Bean

▼ Instructions for Detailed Logs

1. Navigate to application.properties
2. Configure the logging level
3. `logging.level.org.springframework=debug`