1. **Spring Data JPA - Quick Example**

**Hibernate with XML Configuration**

* **Entity-to-table mapping** done using .hbm.xml files.
* **Manual configuration** of sessions, transactions, and connections.
* Tedious, error-prone, and hard to maintain.

Example:

<class name="Student" table="STUDENT">

<id name="id" column="ID">

<generator class="native"/>

</id>

<property name="name" column="NAME"/>

</class>

### 2. ****Hibernate with Annotations****

* Mapping is done using annotations directly in the Java class.
* Eliminates XML but still needs boilerplate code for session and transaction handling.

Example:

@Entity

@Table(name = "STUDENT")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

}

### 3. ****Spring with Hibernate****

* Integrates Hibernate with **Spring Framework**.
* Spring manages SessionFactory, transactions, and dependencies.

**Benefits**:

* Dependency injection.
* Transaction management via annotations.
* Cleaner separation of concerns.

### 4. ****Spring Data JPA (Modern Approach)****

Spring Data JPA abstracts everything:

* You don’t need to write **DAO classes**.
* You don’t need to manually handle **EntityManager**, sessions, or transactions.
* Just extend JpaRepository and Spring handles it.

Example:

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

OUTPUT:

