**✅ Lombok + JPA Interview Questions and Answers**

**🔹 1. What is Lombok and how does it help in JPA-based applications?**

**Answer:**  
Lombok is a Java library that reduces boilerplate code like getters, setters, constructors, builders, etc. In JPA entities, it helps avoid verbose code by generating:

* @Getter/@Setter for fields
* @NoArgsConstructor / @AllArgsConstructor for constructors
* @Builder for fluent object creation

**🔹 2. Why should you be cautious when using Lombok with JPA entities?**

**Answer:**

* **@Builder** may skip default constructor (@NoArgsConstructor) — required by JPA.
* **Mutability** issues: Lombok generates public setters; better to avoid in domain-driven designs.
* **Equality/HashCode**: Including @Id or @OneToMany in @EqualsAndHashCode can break collections (Hibernate uses proxies).

**🔹 3. Which Lombok annotations are commonly used with JPA?**

**Answer:**

java

CopyEdit

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

@ToString(exclude = "sensitiveField")

@EqualsAndHashCode(onlyExplicitlyIncluded = true)

**🔹 4. Why is @NoArgsConstructor required in JPA?**

**Answer:**  
JPA requires a no-arg constructor (public or protected) to instantiate entities via reflection.

**🔹 5. What is the issue with @EqualsAndHashCode in JPA entities?**

**Answer:**  
Including **lazy-loaded relationships** (e.g., @OneToMany) or **IDs** that are generated later can lead to:

* StackOverflow (infinite recursion)
* Incorrect hash-based collection behavior
* Proxy issues in Hibernate

**Best practice:** Use @EqualsAndHashCode(onlyExplicitlyIncluded = true) and manually include stable fields.

**🔹 6. How does @Builder work with JPA, and what are the pitfalls?**

**Answer:**  
Lombok’s @Builder creates a fluent API for object creation.

**Issue:**

* Skips @NoArgsConstructor, breaking JPA instantiation.
* Doesn’t play well with complex relationships like @ManyToOne.

**Solution:** Use @Builder(toBuilder = true) and still keep @NoArgsConstructor.

**🔹 7. How do you use @ToString safely in JPA entities?**

**Answer:**  
Avoid printing full relations:

java

CopyEdit

@ToString(exclude = "orders")

or use:

java

CopyEdit

@ToString(onlyExplicitlyIncluded = true)

@ToString.Include private String name;

**🔹 8. Can we use Lombok annotations on entity relationships (e.g., @OneToMany)?**

**Answer:**  
Yes, but be careful. Lombok doesn't handle lazy proxies well in:

* toString()
* equals() and hashCode()

Avoid using them on @OneToMany/@ManyToOne directly.

**🔹 9. Should we use @Data on JPA entities?**

**Answer:**  
**Avoid @Data on JPA entities.**  
Because @Data adds:

* Getters/setters ✅
* toString(), equals(), hashCode() ❌ (on all fields — including relations)

Instead, use specific annotations:  
@Getter, @Setter, @NoArgsConstructor, @Builder, @ToString(exclude=...).

**🔹 10. Example of a Safe Lombok-JPA Entity Setup?**

java

CopyEdit

@Entity

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

@ToString(onlyExplicitlyIncluded = true)

@EqualsAndHashCode(onlyExplicitlyIncluded = true)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@EqualsAndHashCode.Include

private Long id;

@ToString.Include

private String name;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "dept\_id")

private Department department;

}

**✅ Real-World / Deep Questions**

**🔹 11. Why would @Builder create null pointer exceptions in JPA entities?**

**Answer:**  
Because it bypasses constructor logic, default initializations (like lists) aren’t performed. Always use:

java

CopyEdit

@Builder.Default

private List<Role> roles = new ArrayList<>();

**🔹 12. Does Lombok affect JPA proxy creation or lazy loading?**

**Answer:**  
No, Lombok doesn’t interfere directly. But generated toString, equals, and hashCode methods can **trigger lazy loads**, causing:

* Performance issues
* LazyInitializationException

**🔹 13. How do you exclude relationships from @EqualsAndHashCode safely?**

**Answer:**  
Use:

java

CopyEdit

@EqualsAndHashCode(onlyExplicitlyIncluded = true)

And include only immutable fields like id.

**🔹 14. Should entities be mutable or immutable with Lombok?**

**Answer:**  
JPA requires mutable entities (at least partially). You can:

* Use @Setter(AccessLevel.PRIVATE) for controlled mutability.
* Use DDD-style aggregate roots with factory methods instead of public setters.

**✅ Cross Questions to Expect**

* **Q:** What happens if you use @Data on a bi-directional entity?  
  **A:** StackOverflow in toString or equals() due to recursive references.
* **Q:** How can you initialize collection fields using Lombok builder?  
  **A:** Use @Builder.Default with a default value.
* **Q:** Why is it a bad practice to include lazy relationships in toString()?  
  **A:** It can trigger unintentional database access or LazyInitializationException.
* **Q:** What alternatives do you use to @Data in production code?  
  **A:** Use fine-grained annotations like @Getter, @Setter, @ToString, etc., with exclusions and inclusion control.

**✅ Lombok + JPA Coding Questions and Answers**

**🔹 1. Create a JPA entity Employee using Lombok with a @ManyToOne relation to Department. Ensure safe use of Lombok.**

java

CopyEdit

@Entity

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

@ToString(onlyExplicitlyIncluded = true)

@EqualsAndHashCode(onlyExplicitlyIncluded = true)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@EqualsAndHashCode.Include

private Long id;

@ToString.Include

private String name;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "dept\_id")

private Department department;

}

**Notes:**

* @EqualsAndHashCode.Include on stable fields only.
* Avoid @ToString on department to prevent LazyInitializationException.

**🔹 2. Use Lombok’s @Builder to create a default-initialized list field in a JPA entity.**

java

CopyEdit

@Entity

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department", cascade = CascadeType.ALL)

@Builder.Default

private List<Employee> employees = new ArrayList<>();

}

**Q: Why use @Builder.Default?**  
To ensure the list is initialized during builder-based construction.

**🔹 3. Prevent infinite recursion in @ToString and @EqualsAndHashCode for bi-directional relationships.**

java

CopyEdit

@Entity

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Employee {

@Id

@GeneratedValue

private Long id;

private String name;

@ManyToOne

@ToString.Exclude

@EqualsAndHashCode.Exclude

private Department department;

}

**🔹 4. Write a Lombok + JPA DTO conversion method using @Builder.**

java

CopyEdit

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class EmployeeDTO {

private Long id;

private String name;

private Long departmentId;

public static EmployeeDTO fromEntity(Employee employee) {

return EmployeeDTO.builder()

.id(employee.getId())

.name(employee.getName())

.departmentId(employee.getDepartment().getId())

.build();

}

}

**🔹 5. Add auditing fields (createdDate, updatedDate) with Lombok and JPA.**

java

CopyEdit

@Getter

@Setter

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

public abstract class Auditable {

@CreatedDate

@Column(updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime updatedDate;

}

@Entity

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class Product extends Auditable {

@Id

@GeneratedValue

private Long id;

private String name;

}

**🔹 6. Make a JPA entity immutable using Lombok.**

java

CopyEdit

@Entity

@Getter

@RequiredArgsConstructor

public class Country {

@Id

@GeneratedValue

private Long id;

@NonNull

private final String name;

protected Country() {

this.name = null; // for JPA

}

}

**🔹 7. Create a Spring Data JPA repository with Lombok's @Slf4j for logging.**

java

CopyEdit

@Repository

@Slf4j

public class EmployeeRepositoryCustomImpl implements EmployeeRepositoryCustom {

@PersistenceContext

private EntityManager em;

public List<Employee> findByCustomQuery() {

log.info("Running custom query...");

return em.createQuery("FROM Employee", Employee.class).getResultList();

}

}

**🔹 8. Create a Lombok-based DTO with @Value for immutability.**

java

CopyEdit

@Value

@Builder

public class DepartmentSummary {

Long id;

String name;

int employeeCount;

}

**🔹 9. Use Lombok’s @Accessors(chain = true) for fluent setter chaining.**

java

CopyEdit

@Getter

@Setter

@Accessors(chain = true)

public class Address {

private String city;

private String state;

}

// Usage:

Address a = new Address().setCity("Delhi").setState("UP");

**🔹 10. Write unit test for a Lombok @Builder-based JPA entity.**

java

CopyEdit

@Test

void testEmployeeBuilder() {

Department dept = Department.builder().name("Engineering").build();

Employee emp = Employee.builder().name("Raj").department(dept).build();

assertEquals("Raj", emp.getName());

assertEquals("Engineering", emp.getDepartment().getName());

}

**✅ Summary Table of Best Practices**

| **Use Case** | **Lombok Annotation** | **Notes** |
| --- | --- | --- |
| Generate boilerplate | @Getter, @Setter | Avoid @Data on entities |
| Default constructor | @NoArgsConstructor | Required by JPA |
| Builder pattern | @Builder, @Builder.Default | Useful for test/data creation |
| Logging | @Slf4j | For service/repo classes |
| ToString/Equals caution | @ToString.Exclude etc. | Avoid bidirectional fields |