Lombok + JPA Coding Interview Questions and Answers

1. Create a JPA entity Employee using Lombok with a @ManyToOne relation to Department.

```
@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;
}
```

2. Use Lomboks @Builder to create a default-initialized list field in a JPA entity.

```
@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<>();
}
```

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

```
@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.

```
@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.

```
@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.

```
@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.

```
@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.

```
@Value
@Builder
public class DepartmentSummary {
```

```
    Long id;
    String name;
    int employeeCount;
}
```

9. Use Lomboks @Accessors(chain = true) for fluent setter chaining.

```
@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.

```
@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());
}
```