Spring Boot Annotations

1. Core Annotations

@Component

- Spring will automatically create an object (bean) of this class.
- No need to use "new" keyword; Spring manages object creation.
- Bean name is class name with first letter small.
- Detected automatically by component scanning.
- Allows Dependency Injection using @Autowired.
- It becomes part of Spring Container (IoC container).
- Scope can be changed (default: singleton).

@Autowired

- Spring will automatically inject (provide) the required bean where you write @Autowired.
- Can be used on constructor, field, or setter.
- If multiple beans exist, use @Qualifier to specify which one.
- Works only on Spring Beans (@Component, @Service, etc.).

@Qualifier

- Used when multiple beans of same type exist.
- Tells Spring which specific bean to inject.
- Avoids NoUniqueBeanDefinitionException.
- Works with @Autowired on field, constructor, or setter.
- Bean name in @Qualifier must match the bean's name in @Component("name").

@Service

- Marks a class as a Service layer component.
- Used to write business logic or processing logic.
- Spring automatically creates and manages the bean (like @Component).
- Helps maintain clean layered architecture.
- Allows Dependency Injection using @Autowired.
- Detected by component scanning automatically.
- Mainly adds semantic meaning; no special behavior.
- Preferred for service classes instead of @Component.

@Repository

- Marks a class as a Data Access (DAO) component.
- Handles database operations like save, update, delete, select.
- Spring automatically creates and manages the bean.
- Allows Dependency Injection using @Autowired.
- Detected by component scanning automatically.
- Provides Exception Translation (converts SQL exceptions into Spring exceptions).
- Used only for repository/data access layer classes.

@Controller

- Marks a class as a Spring MVC web controller.
- Used to handle HTTP requests and return views (HTML/JSP/Thymeleaf).
- Spring automatically creates and manages the bean.
- Supports URL mappings using @RequestMapping, @GetMapping, @PostMapping.
- Returns logical view names which Spring resolves to UI pages.
- Allows passing data to views using Model or ModelAndView.

- Used only for MVC (not for REST APIs).

@RestController

- Combines @Controller + @ResponseBody.
- Every method returns data (JSON/XML), not HTML pages.
- Used to build REST APIs and microservices.
- Spring automatically creates and manages the bean.
- Supports URL mappings like @GetMapping, @PostMapping, etc.
- Uses Jackson to convert objects into JSON automatically.
- Does not use a View Resolver (no JSP/Thymeleaf).
- Ideal for backend APIs used by mobile/web apps.

---

2. Entity Layer Annotations

@Entity

- Marks the class as a JPA entity mapped to a database table.
- Must have a primary key field annotated with @Id.
- Must have a default (no-arg) constructor.
- Class is managed by Hibernate's Persistence Context.
- Table name defaults to class name if @Table is not used.
- Helps Spring/Hibernate read/write objects from/to database.
- Fields are mapped to columns unless marked with @Transient.

@Table

- Customizes the table name for an entity.
- Useful when table name and class name are different.
- Allows setting unique constraints on columns.

- Allows setting schema and catalog.
- If not used, table name becomes class name by default.

@Id

- Marks the primary key field of the entity.
- Required for every JPA entity.
- Can be applied to int, long, UUID, String, etc.
- Used by Hibernate to identify and track each entity object.
- Must be unique for every record.

@GeneratedValue

- Automatically generates primary key values.
- AUTO → Hibernate chooses best strategy.
- IDENTITY → Uses auto-increment (MySQL).
- SEQUENCE → Uses database sequence (PostgreSQL/Oracle).
- TABLE → Uses separate table to generate IDs.
- Prevents manually assigning PK each time.

@Column

- Customizes properties of a database column.
- name → sets custom column name.
- nullable → sets NOT NULL constraint.
- unique → column must have unique values.
- length → max size for String columns.
- updatable / insertable → control write operations.
- columnDefinition → custom SQL column type.
- If not used, column name = field name.

@OneToOne

- One entity is related to exactly one other entity.
- Usually contains a foreign key column.
- Can be unidirectional or bidirectional.
- Uses @JoinColumn to define FK.
- mappedBy used on non-owning side in bidirectional relation.
- Example: User ↔ Profile, Person ↔ Aadhar.

@OneToMany

- One parent entity is related to many child entities.
- Usually mapped by the child table's foreign key.
- child table stores parent_id.
- Used with List/Set collections.
- Often requires mappedBy on parent side.
- Fetch type is LAZY by default.

@ManyToOne

- Many child entities refer to a single parent.
- Child table contains foreign key column.
- Most common relationship in JPA.
- Requires @JoinColumn to define FK.
- Example: Many Employees → One Department.

@ManyToMany

- Many entities are related to many entities.
- Creates a join table with two foreign keys.
- Requires @JoinTable annotation.
- Can be unidirectional or bidirectional.

- Common for Students ↔ Courses, Authors ↔ Books.

@JoinColumn

- Defines the foreign key column for relationships.
- Used with @OneToOne and @ManyToOne.
- name → FK column name in the table.
- referencedColumnName → which PK column it points to.
- nullable, unique, insertable, updatable supported.

@JoinTable

- Defines a join table for many-to-many relationships.
- joinColumns → foreign key for the owning entity.
- inverseJoinColumns → foreign key for the other entity.
- table name can be set using name attribute.
- Helps Hibernate create relational mapping for @ManyToMany.

---

3. Lombok Annotations

@Getter

- Generates getter methods for all fields.
- Reduces boilerplate and keeps code clean.
- Can be applied on class or individual fields.

@Setter

- Generates setter methods for fields.
- Helps modify private fields without manually writing setters.
- Can be applied on class or individual fields.

@Data

- A combination of:
  - @Getter
  - @Setter
  - @ToString
  - @EqualsAndHashCode
  - @RequiredArgsConstructor
- Most commonly used Lombok annotation.
- Creates everything except a full constructor.
- Makes POJOs simple and clean.

@NoArgsConstructor

- Generates a 0-argument constructor.
- Required by Hibernate/JPA.
- Useful when framework needs to instantiate objects.

@AllArgsConstructor

- Generates a constructor with parameters for all fields.
- Good for initializing fully populated objects.

@RequiredArgsConstructor

- Generates constructor only for final and @NonNull fields.
- Best choice for dependency injection without @Autowired.

@ToString

- Generates toString() method.
- Shows field values in readable form.

@EqualsAndHashCode

- Auto-generates equals() and hashCode() for object comparison.
- Prevents errors in collections like Set or Map.

@Builder

- Enables builder design pattern for easy object creation.
- Makes code readable and less error-prone.
- Supports complex object construction.

@Value

- Makes the class immutable.
- Creates private final fields + getters + no setters.
- Good for DTOs, config objects, constants.

@Slf4j

- Provides an auto-configured logger instance.
- Supports:
  - log.info()
  - log.debug()
  - log.error()
- No manual logger configuration needed.

---

4. Service Layer Annotations

@Service

- Marks class as a service layer component.
- Contains business logic.
- Spring automatically creates and manages this bean.
- Specialized version of @Component for service classes.

@Autowired

- Automatically injects dependent beans.
- Removes the need for manually creating objects.
- Works on constructor, setter, or field.

@Qualifier

- Used when multiple beans of same type exist.
- Specifies exactly which bean to inject.
- Prevents NoUniqueBeanDefinitionException.

@Transactional

- Enables transaction management.
- Commits transaction if method succeeds.
- Rolls back transaction if exception occurs.
- Used in service methods interacting with database.

@Lazy

- Bean is created only when first used.
- Improves startup performance.
- Helps avoid circular dependencies.

@Primary

- Marks a bean as default when multiple beans exist.
- Avoids need for @Qualifier.

@Scope

- Defines bean lifecycle.
- singleton: one instance for entire app (default).
- prototype: new instance each time bean is requested.

- request/session: used in web applications.

@Value

- Injects values from properties/YAML.
- Useful for URLs, messages, constants.

@Async

- Executes method asynchronously.
- Good for email sending, logs, heavy tasks.
- Requires @EnableAsync.

@PostConstruct

- Executes after bean creation.
- Used for initialization.

@PreDestroy

- Executes before bean destruction.
- Used for cleanup (closing DB connections).

---

5. Repository Layer Annotations

@Repository

- Marks a DAO/Repository class.
- Automatically creates and manages the bean.
- Provides Exception Translation (SQLException → DataAccessException).
- Used for database interaction classes.

@Transactional

- Applies transaction boundary around repository methods.

- Commits on success, rolls back on exception.
- Used for update, delete, batch operations.

@Query

- Allows writing custom JPQL or SQL queries.
- Supports named and positional parameters.
- Useful for complex queries not supported by method names.

@Modifying

- Required for update/delete queries using @Query.
- Indicates that query modifies database.
- Must be used with @Transactional.

@Param

- Binds method parameters to query parameters.
- Used with @Query for named parameters.

@EnableJpaRepositories

- Enables creation of JPA repository implementations.
- Specifies package location of repositories.
- Placed on main application class.

@Lock

- Applies optimistic/pessimistic locking.
- Prevents concurrent update conflicts.
- Ensures data consistency.

@EntityGraph

- Fetches related entities eagerly in a single query.
- Solves N+1 select problem.

- Improves performance.

@Procedure

- Used to call stored procedures.
- Supports IN and OUT parameters.
- Useful for heavy DB operations.

@QueryHints

- Adds custom instructions for Hibernate.
- Used for caching, read-only queries, performance optimization.

@NoRepositoryBean

- Indicates base repository interface not to be instantiated.
- Useful for sharing common methods.

---

6. Controller Layer Annotations

@Controller

- Marks class as a Spring MVC Controller.
- Returns HTML views (JSP/Thymeleaf).
- Handles web requests.
- Works with @GetMapping, @PostMapping, etc.
- Uses Model to send data to UI.
- Not used for JSON responses.

@RestController

- Creates REST APIs.
- Returns JSON/XML.
- Combines @Controller + @ResponseBody.

- Handles requests using mapping annotations.

@RequestMapping

- Maps URLs to controller classes/methods.
- Supports all HTTP methods.
- Used for base URLs.

@GetMapping

- Handles HTTP GET requests.
- Used to fetch/read data.

@PostMapping

- Handles HTTP POST requests.
- Used to create new data.
- Works with @RequestBody.

@PutMapping

- Handles HTTP PUT requests.
- Used to update/replace data completely.

@PatchMapping

- Handles HTTP PATCH requests.
- Used for partial updates.

@DeleteMapping

- Handles HTTP DELETE requests.
- Used to delete records.

@PathVariable

- Extracts values from URL path `/user/{id}`.

@RequestParam

- Extracts query parameters from URL `/search?name=ram`.

@RequestBody

- Converts JSON body → Java object.

@ResponseBody

- Returns JSON instead of HTML.
- Automatically applied to @RestController.

@CrossOrigin

- Enables CORS (Cross-Origin Resource Sharing).
- Allows frontend apps (React/Angular) on different domains to access backend.
- Avoids "CORS blocked" error.

@Valid

- Enables validation on request DTOs.
- Works with:
  - @NotNull
  - @Size
  - @Email
  - @NotBlank

---

7. Exception Layer Annotations

@ControllerAdvice

- Global exception handler for MVC applications.
- Returns HTML responses.

- Separates exception logic from controller logic.

@RestControllerAdvice

- Global exception handler for REST APIs.
- Returns JSON responses.
- Best for microservices.

@ExceptionHandler

- Handles specific exceptions.
- Similar to catch block.
- Can return custom error response.

@ResponseStatus

- Defines HTTP status code for an exception/handler.
- Used in custom exceptions.

@InitBinder

- Customizes request parameter binding.
- Useful for formatting dates, trimming strings.

@ModelAttribute

- Can add common attributes for all controllers.
- Rarely used in exception handling.