The Most Used Annotations in Spring

Component Scanning & Dependency Injection

- **@Component** : Marks a class as a Spring-managed component. Used for generic components that don't fit into @Service, @Repository, or @Controller categories.
- **@Controller** : Specialized @Component, used in Spring MVC to define a web controller.
- **@RestController** : Combines @Controller and @ResponseBody to simplify RESTful API development.
- **@Service** : Specialized @Component, used for business logic and service layer components.
- **@Repository** : Specialized @Component, used for data access objects (DAO) to indicate storage operations.
- **@Autowired** : Automatically injects dependencies into Spring beans, reducing manual configuration.
- **@Qualifier("beanName") :** Specifies which bean to inject when multiple candidates exist.
- **@Primary** : Marks a bean as the default choice when multiple candidates exist.
- **@Value("${property.key}")** : Injects values from properties f iles into Spring beans.

Configuration & Bean Management

- **@Configuration** : Marks a class as a source of bean definitions for the Spring container.
- **@Bean** : Declares a method that produces a Spring-managed bean.
- **@Import({ConfigClass.class})** : Imports another configuration class.
- **@ImportResource("classpath:config.xml")** : Loads an XML configuration file.
- **@DependsOn("beanName")** : Ensures a bean is initialized only after specific dependencies are available.
- **@Lazy** : Postpones bean initialization until it is first accessed.
- **@Scope("prototype")** : Defines bean scope (singleton, prototype, etc.).
- **@PropertySource("classpath:app.properties")** : Loads external property files.
- **@EnableAspectJAutoProxy** : Enables support for Aspect-Oriented Programming (AOP) in Spring.

Spring Scheduling & Async Execution

- **Scheduled(cron="0 0 * * * ?")** : Schedules a method to run at a fixed interval.
- **@Async** : Marks a method for asynchronous execution.

Spring AOP (Aspect Oriented Programming)

- **@Aspect** : Defines an aspect, a modularization of cross-cutting concerns like logging.
- **@Before**("execution(* package.Class.method(..))") : Executes advice before a matched method executes.
- **@After**("execution(* package.Class.method(..))") : Executes advice after method execution (regardless of outcome).
- **@AfterReturning**("execution(* package.Class.method (..))") : Runs only when the method successfully returns.
- **@AfterThrowing**("execution(* package.Class.method (..))") : Runs if the method throws an exception.
- **@Around**("execution(* package.Class.method(..))") : Wraps method execution for pre- and post-processing.
- **@Pointcut**("execution(* package..*(..))") : Defines reusable AOP expressions.

Spring Boot Annotations

- **@SpringBootApplication** : Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
- **@EnableAutoConfiguration** : Enables Spring Boot’s auto-configuration.
- **@ComponentScan("com.example")** : Scans packages for Spring components.

Spring MVC & REST

Request Handling

- **@RequestMapping("/endpoint")** : Maps HTTP requests to handler methods.
- **@GetMapping("/endpoint")** : Handles HTTP GET requests.
- **@PostMapping("/endpoint")** : Handles HTTP POST requests.
- **@PutMapping("/endpoint")** : Handles HTTP PUT requests.
- **@DeleteMapping("/endpoint")** : Handles HTTP DELETE requests.
- **@PatchMapping("/endpoint")** : Handles HTTP PATCH requests.
- **@RequestParam("param")** : Binds query parameters.
- **@PathVariable("id")** : Extracts values from URLs.
- **@RequestBody** : Converts request body to Java objects.
- **@ResponseBody** : Converts Java objects to JSON/XML responses.
- **@ModelAttribute("model")** : Binds request data to a model.
- **@SessionAttributes("user")** : Stores attributes in HTTP session.
- **@CrossOrigin("*")** : Enables CORS support for a controller.

Exception Handling

- **@ExceptionHandler(Exception.class)** : Catches exceptions at the controller level.
- **@ResponseStatus(HttpStatus.NOT_FOUND)** : Defines HTTP response status codes.
- **@ControllerAdvice** : Defines global exception handling logic for multiple controllers.

Spring Cloud & Microservices

- **@EnableEurekaClient** : Enables Eureka client.
- **@FeignClient(name="user-service")** : Declares a Feign client.
- **@LoadBalanced** : Enables Ribbon client-side load balancing.
- **@CircuitBreaker** : Implements circuit-breaking logic.
- **@RateLimiter** : Applies rate-limiting mechanisms.
- **@EnableConfigServer** : Marks an application as a Spring Cloud Config Server.
- **@EnableDiscoveryClient** : Enables service discovery using **Consul** (or other service discovery platforms like **Eureka** or **Zookeeper**).
- **@EnableConsulServer** : Marks an application as a Consul server.
- **@EnableZuulProxy** : Marks an application as a **Zuul Proxy** (API Gateway)

Spring Messaging & WebSockets

- **@EnableWebSocket** : Enables WebSocket support.
- **@MessageMapping("/chat")** : Maps messages to handler methods.
- **@SendTo("/topic/updates")** : Sends the response to a specific destination after handling a WebSocket message.

Spring Kafka Annotations

- **@EnableKafka** : Enables Kafka support in the Spring application.
- **@KafkaListener(topics="myTopic"**) : Marks a method to listen to a Kafka topic.

Spring Caching

- **@EnableCaching** : Enables caching in Spring.
- **@Cacheable("cacheName")** : Caches method results.
- **@CachePut("cacheName")** : Updates cache with fresh data.
- **@CacheEvict("cacheName")** : Removes entries from cache.
- **@CacheConfig(cacheNames={"defaultCache"})** : Configures caching settings at the class level.

Spring Security

- **@EnableWebSecurity** : Enables Spring Security configuration.
- **@Secured("ROLE_ADMIN")** : Secures methods based on roles.
- **@RolesAllowed({"ROLE_USER", "ROLE_ADMIN"})** : Defines allowed roles.
- **@AuthenticationPrincipal** : Injects the authenticated user.
- **@PreFilter** : Filters method arguments before method execution.
- **@PostFilter** : Filters the method return value after method execution.
- **@PreAuthorize**("hasRole('ROLE_USER')") : Performs pre-execution authorization based on roles.
- **@PostAuthorize**("returnObject.owner == authentication.name") : Performs post-execution authorization.

Spring Data MongoDB Annotations

- **@Document** : Marks a class as a MongoDB document.
- **@Field** : Maps a field in the MongoDB document.
- **@Id** : Marks a field as the primary key in a MongoDB document.
- **@Indexed** : Creates an index on a MongoDB field.

Spring Data Cassandra Annotations

- **@Table** : Defines a Cassandra table for a class (similar to @Entity in JPA).
- **@PrimaryKeyColumn** : Marks a field as part of the primary key in Cassandra.
- **@Column** : Marks a field to be mapped to a Cassandra column.
- **@PartitionKey** : Marks a field as the partition key for Cassandra queries.

Spring Testing Annotations

- **@SpringBootTest** : Loads a full application context for testing.
- **@WebMvcTest** : Tests Spring MVC controllers.
- **@MockBean** : Creates mock beans in test environments.
- **@DataJpaTest** : Tests JPA components in an isolated environment.
- **@TestConfiguration** : Defines custom test configurations.
- **@BeforeEach** and **@AfterEach** : Runs setup and teardown code before/after each test.
- **@Test** : Marks a method as a test method.
- **@Rollback**: Specifies whether a transaction should be rolled back in test environments.

Spring Data JPA & Transactions

Entity Mapping

- **@Entity** : Declares a JPA entity.
- **@Table(name="users")** : Specifies table name.
- **@Id** : Marks a field as the primary key.
- **@GeneratedValue(strategy=GenerationType.AUTO)** : Auto-generates primary key values.
- **@Column(name="username")** : Maps a field to a database column.
- **@OneToOne, @OneToMany, @ManyToOne, @ManyToMany** : Defines entity relationships.
- **@Transient** : Marks a field to be ignored by JPA (not persisted in the database).
- **@Embeddable** : Marks a class as embeddable.
- **@Embedded** : Used to embed an embeddable class into an entity.
- **@Enumerated (EnumType.STRING)**: Specifies that an enum type should be persisted in the database
- **@Fetch**: Specifies how collections are fetched from the database (eager or lazy loading).
- @**MappedBy("user")** : Defines the inverse side of a relationship.

Repository & Transactions

- **@Transactional** : Manages transactions at the method or class level.
- **@Query**("SELECT u FROM User u WHERE u.email = ?1") : Custom JPA queries.
- **@Modifying** : Used for update/delete queries.
- **@EnableJpaRepositories** : Enables Spring Data JPA repositories.
- **@Param**: Binds method parameters to query parameters in @Query.

Spring Validation Annotations

- **@NotBlank** : Ensures that a string is not null or empty (trims leading and trailing spaces).
- **@NotEmpty** : Ensures a collection, map, or array is not empty.
- **@Min(value)** : Validates that a numeric field is greater than or equal to the specified value.
- **@Max(value)** : Validates that a numeric field is less than or equal to the specified value.
- **@Email** : Ensures that a string is a valid email address.
- **@Pattern(regexp)** : Validates a field against a regular expression pattern.
- **@Size(min, max)** : Validates that a field's size is within a specified range.