🔷 Clean, Fast, Scalable

🌿 A Practical Guide from @Component to @SpringBootTest

📌 Component Scanning & Dependency Injection

- **@Component**: Marks a Java class as a Spring-managed component.
- **@Service**: Specialized @Component for service layer.
- **@Repository**: Specialized @Component for persistence layer with exception translation.
- **@Controller**: Specialized @Component for MVC controller.
- **@RestController**: Combines @Controller and @ResponseBody.
- **@ComponentScan**: Enables component scanning in specified packages.
- **@Autowired**: Injects dependencies automatically.
- **@Qualifier**: Specifies which bean to inject when multiple candidates exist.
- **@Primary**: Indicates the default bean to inject when multiple candidates are present.

📌 Configuration & Bean Management

- **@Configuration**: Indicates that the class contains Spring bean definitions.
- **@Bean**: Declares a bean to be managed by Spring.
- **@Lazy**: Creates the bean only when it's first requested.
- **@DependsOn**: Specifies dependent beans that must be initialized first.
- **@Scope**: Defines the scope of a bean (singleton, prototype, etc.).

📌 Scheduling & Async Execution

- **@EnableScheduling**: Enables Spring’s scheduled task execution capability.
- **@Scheduled**: Declares a method to be scheduled (e.g., cron or fixedDelay).
- **@EnableAsync**: Enables asynchronous method execution.
- **@Async**: Marks a method to run asynchronously in a separate thread.

📌 Spring AOP (Aspect-Oriented Programming)

- **@EnableAspectJAutoProxy**: Enables support for handling components marked with @Aspect.
- **@Aspect**: Marks a class as an aspect.
- **@Before**: Executes before a matched method.
- **@After**: Executes after a matched method.
- **@AfterReturning**: Executes after a method returns successfully.
- **@AfterThrowing**: Executes if a matched method throws an exception.
- **@Around**: Wraps method execution (before and after).

📌 Spring Boot Annotations

- **@SpringBootApplication**: Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
- **@EnableAutoConfiguration**: Tells Spring Boot to start adding beans based on classpath settings.
- **@ConfigurationProperties**: Binds external config to a Java object.
- **@Value**: Injects values from properties files.
- **@Profile**: Activates beans only for specific profiles.
- **@ConditionalOnProperty**: Loads a bean only if a specific property exists.

📌 Spring MVC & REST

- **@RequestMapping**: Maps HTTP requests to handler methods.
- **@GetMapping,@PostMapping, @PutMapping, @DeleteMapping**: Shorthand for @RequestMapping with HTTP methods.
- **@RequestParam**: Binds HTTP query parameters to method arguments.
- **@PathVariable**: Binds URI template variables to method parameters.
- **@RequestBody**: Binds HTTP request body to a method parameter.
- **@ResponseBody**: Indicates the return value should be serialized to the HTTP response body.

📌 Exception Handling

- **@ControllerAdvice**: Handles exceptions across multiple controllers.
- **@ExceptionHandler**: Defines method to handle specific exceptions.
- **@ResponseStatus**: Sets the HTTP status for a method or exception.

📌 Spring Cloud & Microservices

- **@EnableDiscoveryClient**: Enables service registration and discovery.
- **@FeignClient**: Declarative REST client for calling other services.
- **@LoadBalanced**: Adds client-side load balancing to RestTemplate.
- **@RefreshScope**: Refreshes bean values on config changes.
- **@EnableConfigServer**: Starts a Spring Cloud Config server.

📌 Spring Kafka

- **@EnableKafka**: Enables Kafka-related annotations.
- **@KafkaListener**: Listens for Kafka messages.
- **@KafkaHandler**: Used with class-level @KafkaListener for multiple methods.

📌 Messaging & WebSockets

- **@EnableWebSocket**: Enables WebSocket configuration.
- **@MessageMapping**: Maps incoming messages to methods (used in STOMP).
- **@SendTo**: Sends response message to a specific topic.
- **@SubscribeMapping**: Handles subscription requests.
- **@JmsListener**: Listens to JMS messages.
- **@RabbitListener**: Listens to RabbitMQ messages.

📌 Caching

- **@EnableCaching**: Enables Spring's annotation-driven cache management.
- **@Cacheable**: Caches the result of a method.
- **@CachePut**: Updates the cache with the method result.
- **@CacheEvict**: Removes an entry from the cache.

📌 Spring Security

- **@EnableWebSecurity**: Enables Spring Security configuration.
- **@PreAuthorize**: Method-level security using expressions.
- **@Secured**: Defines security constraints on methods.
- **@WithMockUser**: Mocks a user for testing purposes.

📌 Spring Data MongoDB

- **@Document**: Marks a class as a MongoDB document.
- **@Field**: Maps a field to a specific MongoDB field name.
- **@Id**: Marks the primary key of the document.
- **@DBRef**: References another document.

📌 Spring Data Cassandra

- **@Table**: Marks a class as a Cassandra table.
- **@PrimaryKey**: Marks the primary key field.
- **@Column**: Maps a field to a Cassandra column.
- **@CassandraType**: Defines the Cassandra-specific data type.

📌 Spring Testing

- **@SpringBootTest**: Loads a full application context for integration testing.
- **@DataJpaTest**: Configures an in-memory database and tests only JPA components.
- **@WebMvcTest**: Loads only web layer (controllers) for testing.
- **@MockBean**: Adds Mockito mock into the Spring ApplicationContext.
- **@TestConfiguration**: Provides test-specific bean configurations.

📌 JPA & Transactions

- **@Entity**: Marks a class as a JPA entity.
- **@Table**: Maps the entity to a database table.
- **@Column**: Maps a field to a database column.
- **@Id**: Marks a field as a primary key.
- **@GeneratedValue**: Defines primary key generation strategy.
- **@OneToMany, @ManyToOne**: Defines relationships between entities.
- **@Transactional**: Declares a method or class as transactional.
- **@EnableTransactionManagement**: Enables annotation-driven transaction management.

📌 Spring Repository & Query

- **@Repository**: Indicates a data access component.
- **@EnableJpaRepositories**: Enables JPA repository scanning.
- **@Query**: Defines a custom JPQL/SQL query.
- **@Modifying**: Marks a modifying query (update/delete).

📌 Validation

- **@Valid**: Triggers validation on method arguments.
- **@NotNull**: Validates that a value is not null.
- **@Size**: Validates size constraints (length or collection size).
- **@Email**: Validates that a field contains a valid email address.
- **@Pattern**: Validates a string against a regex.
- **@Min, @Max**: Validates numeric boundaries.