Spring Annotations

Core Spring Annotations

@SpringBootApplication	This is a convenient annotation that bundles up three common annotations into one: @Configuration, @EnableAutoConfiguration, and @ComponentScan. It's typically used on the main class of your Spring application and serves as a starting point for Spring Boot's auto-configuration, class path scanning, and application configuration.
@ComponentScan	This annotation tells Spring where to look for components, configurations, and services to include them in the Spring application context. It automatically scans for Spring components (@Component, @Service, @Controller, etc.) within the specified package and its sub-packages, making them available for dependency injection and other configurations.
@Autowired	@Autowired is used for automatic dependency injection. By marking a field, constructor, or setter method with this annotation, you're telling Spring to populate this dependency automatically. Spring will look into its context for a matching bean and inject it without you having to do anything manually.
@Bean	This annotation is used on methods within a @Configuration class to define a Spring bean. The method's return value becomes a bean managed by Spring's application context. It's useful when you need to configure a third-party class or more complex configuration that requires programmatic initialization.
@Component	@Component is a generic stereotype for any Spring-managed component. When you annotate a class with @Component, you're telling Spring to treat this class as a component, and when Spring sees this annotation during classpath scanning, it will create a bean instance of that class.
@Configuration	@Configuration indicates that a class can be used by Spring as a source of bean definitions. Inside this class, you'd typically define methods with @Bean to specify your application configuration. Think of it as a recipe that tells Spring how certain parts of your application should be set up.
@Qualifier	Sometimes you have more than one bean of the same type and need to tell Spring which one to use for autowiring. @Qualifier allows you to specify the exact bean to wire by its name, thus avoiding confusion where multiple beans could satisfy an autowiring dependency.
@Required	This annotation was used to indicate that a particular bean property must be set. It was used on the setter method of a bean property. However, it's important to note that as of Spring 5.1, @Required has been deprecated. Spring now recommends using @Autowired with required attributes or constructor injection as better alternatives.
@Value	@Value is used to inject property values into components, or even use it to inject expressions. You can use it to inject values

	from a properties file or to inject straightforward literals. It's quite handy when you need to keep your application configurable and separate configuration from code.
Stereotype Annotations	
@Repository	This annotation is a stereotype for the persistence layer, which acts as a repository of data. When you annotate a class with @Repository, you're telling Spring that this class is responsible for database operations. Spring will treat it as a Data Access Object (DAO) and can also automatically translate specific database-related exceptions into Spring's consistent, unchecked exceptions. This helps with reducing boilerplate code related to exception handling in your data access layer.
@Service	@Service is used to annotate classes that hold business logic. By marking a class with @Service, you indicate that it's holding the business operations or calls to the persistence layer. It's a stereotype annotation that tells Spring that the class is a business service facade. This also makes it clear for anyone reading the code which classes are specifically intended to contain business logic.
@Controller	This annotation is used to mark classes as Spring MVC Controllers. @Controller enables auto-detection of implementation classes through classpath scanning. It's used in combination with annotated handler methods based on different HTTP request methods. In a typical MVC application, @Controller classes are responsible for preparing a model map with data and selecting views to render the response to the client.
@RestController	@RestController is a specialized version of the @Controller annotation which marks the class as a controller where every method returns a domain object instead of a view. It simplifies the controller implementation by combining @Controller and @ResponseBody, which means that the data returned by each method will be written straight into the response body as JSON or XML, not through a template resolver. It eliminates the need to annotate every response handling method in the controller class with @ResponseBody, reducing boilerplate code significantly.
Web or Rest Annotations	
@RequestMapping	This is a general annotation that maps HTTP requests to handler methods in your controllers. You can specify HTTP methods, headers, parameters, and other aspects of the request to narrow down which requests this method should handle. It's quite flexible and can be used for any HTTP method, but you can also use more specific annotations like @GetMapping or @PostMapping for clarity.
@GetMapping	@GetMapping is a specialized version of @RequestMapping that is specifically designed to handle HTTP GET requests. It makes

	your method readable as it clearly shows that the method is processing a GET request.
@PostMapping	Similar to @GetMapping, @PostMapping is a shorthand for @RequestMapping(method = RequestMethod.POST) and is used to handle HTTP POST requests. It's commonly used for submitting form data.
@PutMapping	This annotation is used for HTTP PUT requests and is a shortcut for @RequestMapping(method = RequestMethod.PUT). It's typically used when updating or replacing a resource completely.
@DeleteMapping	@DeleteMapping is used to map HTTP DELETE requests. It's helpful for removing a resource from the server.
@PatchMapping	@PatchMapping is used for HTTP PATCH requests and is intended for making partial updates to resources. It's similar to PUT but is used only to update parts of the resource.
@RequestParam	This annotation is used in a controller method parameter to indicate that a method parameter should be bound to a web request parameter. @RequestParam is very useful for accessing the query parameters in the URL, for example, in a request like /api/items?id=123.
@ResponseBody	Using this annotation on a method indicates that the return type should be written straight to the HTTP response body, bypassing typical view resolution that Spring MVC performs. This is particularly useful for developing RESTful web services where you directly send the resource as JSON or XML.
@RequestBody	This annotation is used to indicate that a parameter in a method should be bound to the body of the web request. It is commonly used to handle incoming data like JSON or XML and convert them into Java objects.
@ResponseStatus	@ResponseStatus lets you specify the HTTP status code of the response. You can use this on handler methods or on exception classes to indicate the status when a specific error occurs.
@PathVariable	Used on a method parameter to bind it to the value of a URI template variable. This is particularly useful in RESTful web services, where resource URLs often include identifiers, like /users/{userId}.
Aspect-Oriented Programming	
@Aspect	This annotation is used to declare a class as an aspect in Spring AOP. An aspect is a class that encapsulates advice (i.e., actions taken at various points in your application) along with points where the advice should apply (join points). It's like telling Spring, "Hey, this class contains code that should run at certain points during execution of the main application."
@Before	@Before is used to define an advice that should run before a method execution. You specify a pointcut expression to indicate the method or methods where this advice should apply. For example, you might use it to perform checks or prepare some data before the method executes.

@After	This annotation defines an advice that will run after a method has executed, regardless of the outcome (whether it throws an exception or completes successfully). It's useful for actions that need to be performed no matter what, such as cleaning up resources.
@AfterReturning	@AfterReturning is used on a method that should be executed only if the monitored method executes successfully and returns. You can optionally access the returned value in the advice method by specifying it in the annotation's attributes. This is particularly useful for post-processing the data returned by a method.
@AfterThrowing	This advice runs if a method exits by throwing an exception. You can use it to handle exceptions in a centralized manner instead of spreading error handling code across your business methods. It can also be used to perform rollback operations or clean-up actions after an exception occurs.
@Around	@Around is one of the most powerful advices in AOP. It surrounds a method execution, meaning you can execute code before and after the method call. It's required to proceed with the invocation of the method explicitly within the advice. This type of advice gives you full control over when and whether the method executes, and you can modify the returned value or handle the exception as needed.
Data Access Annotations	
@Transactional	@Transactional is used to declare that a method or class should execute within a transactional context. This means Spring will handle the creation and management of transactions for the annotated methods or classes. It’s extremely useful for maintaining data integrity and handling transactions across multiple database operations without manual intervention.
@EnableTransactionManagement	This annotation is typically placed on a configuration class to enable Spring's annotation-driven transaction management. It tells Spring to look for @Transactional annotations on beans in the Spring application context and manage transactions accordingly. It's part of the configuration that sets up the infrastructure needed for executing transactions as declared by @Transactional.
Configuration Annotations	
@EnableAutoConfiguration	This annotation is part of the Spring Boot magic that helps automate the configuration process. When you add @EnableAutoConfiguration to your application, Spclasspath attempts to configure your application based on the jar dependencies that you have added. For example, if Spring Boot detects a database driver in the classpath, it automatically configures a DataSource bean for you.
@Profile	@Profile allows you to define which components should be included in the application context based on a set environment

	or 'profile'. This is particularly useful for segregating parts of your application configuration and making it only active in a particular environment, like 'dev', 'test', or 'prod'. For instance, you might have certain beans that should only be loaded in a development environment.
@PropertySource	This annotation is used to declare a set of properties defined in a properties file. @PropertySource makes these properties available to Spring's Environment and can be used to externalize configuration out of your application code. This helps you keep your configuration separate from your codebase and easy to change without recompiling your code.
@Import	@Import allows you to import other configuration classes into the current context. This is particularly useful when you want to modularize your configuration and break it into different classes. By using @Import, you can keep your configurations organized and only include them as needed.
Testing Annotations	
@SpringBootTest	@SpringBootTest is used to provide a bridge between Spring Boot test features and JUnit. It loads the complete application and injects all the beans which can then be autowiredbootstrap with Spring Boot support, you use this annotation. It loads the complete application and injects all the beans which can then be autowired in the test class.
@WebMvcTest	This is a specialized annotation that can be used for Spring MVC tests. It is used when you want to test your controller layers and you don't want to load the full Spring application context. @WebMvcTest only instantiates the web layer and not the whole context. Typically, it is used in combination with @MockBean to provide mock implementations for required components.
@DataJpaTest	@DataJpaTest provides a configuration specifically for testing JPA components. It configures an in-memory database, scans for @Entity classes, and configures Spring Data JPA repositories. It is not intended for testing JDBC operations directly, but is instead for JPA-related configurations.
@MockBean	@MockBean is used to add Mockito mock objects to the Spring application context. The beans that are mocked with @MockBean replace the regular Spring beans of the same type in the application context. This is extremely useful during testing when you need to isolate the component under test by replacing external dependencies with mocks.