**Spring Core Annotations:**

Dependency Injection - Related Annotations

* + @Autowired: Used to automatically inject dependencies into a Spring bean. Spring tries to find a matching bean by type and injects it into the annotated field, setter method, or constructor.
  + @Qualifier: Used in conjunction with @Autowired when multiple beans of the same type exist. It specifies which bean should be injected by providing the bean's qualifier value (usually its name).
  + @Primary: Used to give higher preference to a bean when multiple beans of the same type are candidates for autowiring. The @Primary bean is the one that gets injected by default if no qualifier is provided.
  + @Bean: Used in Java-based configuration to explicitly declare a bean. The method annotated with @Bean produces a bean to be managed by the Spring container.
  + @Lazy: Used to indicate that a bean should be lazily initialized. This means the bean will be created only when it is first requested, rather than eagerly at the application startup.
  + @Required: Deprecated as of Spring 5.1. Used previously to indicate that a property must be set during bean initialization. It was used in conjunction with XML configuration.
  + @Value: Used to inject values into bean properties from properties files, environment variables, or other Spring beans. It can also be used to inject literals directly.
  + @Scope: Specifies the scope of a Spring bean (singleton, prototype, request, session, etc.). It determines how long the bean instance will last and how it will be shared in the application context.
  + @Lookup: Used on methods in a singleton-scoped bean to instruct Spring to override the method and return a new instance of a prototype-scoped bean on each invocation.

Context Configuration Annotations

* + @Profile: Used to specify which beans should be loaded based on the active profiles in the Spring application context.
  + @Import: Used to import additional Spring configuration classes into the current configuration class. It allows you to modularize your configuration by splitting it into multiple classes and importing them where needed.
  + @ImportResource: Used to import XML-based Spring configuration files into the current Java-based Spring configuration class.
  + @PropertySource: Used to specify the properties file(s) from which to load key-value pairs into the Spring Environment.

**Spring Web Annotations**

* @RequestMapping: Used to map web requests onto specific handler methods in a Spring MVC controller.
* @RequestBody: Used to bind the body of the HTTP request to a method parameter in a controller.
* @PathVariable: Used to extract values from the URI path of the request into a method parameter.
* @RequestParam: Used to extract query parameters from the request URL into a method parameter in a controller.
  + @ResponseBody: Used to indicate that the return value of a method should be serialized directly to the HTTP response body.
  + @ExceptionHandler: Used to define handler methods for specific exceptions thrown during the execution of controller methods.
  + @ResponseStatus: Used in conjunction with @ExceptionHandler to specify the HTTP status code to be returned when a specific exception is thrown.
* @Controller: Used to indicate that a class serves as a Spring MVC controller. It is typically applied to controller classes that handle HTTP requests.
* @RestController: A specialized version of @Controller that is used to define RESTful web services.
* @ModelAttribute: Used to bind method parameters to model attributes, making them available to the view layer.
* @CrossOrigin: Used to enable Cross-Origin Resource Sharing (CORS) for a specific handler or controller.

**Spring Boot Annotations**

* @SpringBootApplication: A convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
* @EnableAutoConfiguration: Enables Spring Boot's automatic configuration mechanism.
  + @ConditionalOnClass: activates a configuration class if the specified classes are present in the classpath.
  + @ConditionalOnMissingClass: activates a configuration class if the specified beans are not found in the application context.
  + @ConditionalOnProperty: Activates a configuration class based on the presence and value of a specific property in the environment.
  + @ConditionalOnResource: Activates a configuration class if the specified resources (files or directories) are present in the classpath.

**Spring Scheduling Annotations**

* @EnableAsync: This annotation is used at the configuration class level (typically on @Configuration classes) to enable Spring's asynchronous method execution capability.
* @EnableScheduling: Similar to @EnableAsync, this annotation is used at the configuration class level to enable Spring's scheduled task execution capability.
* @Async: Applied to methods within a Spring bean, this annotation indicates that the annotated method should be executed asynchronously. When called, Spring will run the method in a separate thread, allowing the caller to continue its execution without waiting for the method to complete.
* @Scheduled: Applied to methods within a Spring bean, this annotation specifies when and how often the annotated method should be invoked. It supports fixed-delay, fixed-rate, and cron-based scheduling for method execution.
* @Schedules: This annotation allows specifying multiple @Scheduled annotations within a single method.

**Spring Data Annotations**

Common Spring Data Annotations

* + @Transactional: Used to indicate that a method (or all methods of a class) should run within a database transaction.
  + @NoRepositoryBean: Used to indicate that an interface or class should not be instantiated directly as a Spring Data repository.
  + @Param: Used to bind method parameters in Spring Data repository methods to named query parameters or method parameters in native queries.
  + @Id: Marks a field in an entity class as the primary key of the corresponding database table.
  + @Transient: Marks a field in an entity class as not persistent, meaning it is not mapped to the database. This annotation is useful for fields that should not be saved in the database or should be derived from other fields.

Spring Data JPA Annotations

* + @Query: Used to declare a custom query using JPQL (Java Persistence Query Language) or native SQL in a Spring Data repository method.
  + @Procedure: Used to indicate that a method in a repository interface should call a stored procedure defined in the database.
  + @Lock: Used to specify the type of database locking to be applied when executing a query.
  + @Modifying: Used in conjunction with @Query to indicate that the query is modifying the database (e.g., UPDATE, DELETE operations).
  + @EnableJpaRepositories: Used at the configuration class level to enable Spring Data JPA repositories in the application.

Spring Data Mongo Annotations

* + @Document: Used to mark a class as a document that should be persisted in MongoDB.
  + @Field: Used to specify the mapping between a Java object field and a MongoDB document field.
  + @Query: Used in Spring Data MongoDB repository methods to define custom query methods using MongoDB's query language.
  + @EnableMongoRepositories: Used at the configuration class level to enable Spring Data MongoDB repositories in the application.

**Spring Bean Annotations**

* @ComponentScan: Used at the configuration class level (@Configuration) to specify the base packages to scan for Spring-managed components (like @Component, @Service, @Repository, etc.).
* @Configuration: Marks a class as a configuration class in Spring. Configuration classes define bean definitions and other configuration settings for the Spring application context. Beans declared in @Configuration classes are typically created and managed by Spring.
  + @Component: Indicates that a class is a Spring-managed component. It serves as a generic stereotype annotation for any Spring-managed component and is typically used as the base annotation for other specialized stereotypes (@Service, @Repository, @Controller).
  + @Service: Used to annotate classes at the service layer in Spring applications. It indicates that a class performs some service tasks, such as business logic, transaction management, or other processing.
  + @Repository: Used to annotate classes at the persistence layer, specifically DAOs (Data Access Objects), in Spring applications.
  + @Controller: Used to annotate classes in the presentation layer (typically Spring MVC controllers) of Spring applications. It indicates that a class serves as a controller that handles HTTP requests, processes inputs, and returns appropriate responses, typically through views.