Commonly Used Spring Annotations

Core Annotations

- **@Component**
  Marks a class as a Spring-managed component (like a blueprint for an object that Spring will manage). When you create a class with this annotation, Spring will create an instance of it automatically.

- **@Service**
  A specialized version of `@Component` used for classes that contain business logic. It indicates that this class provides some services, making it easier to organize your code by purpose.

- **@Repository**
  Used to mark Data Access Object (DAO) classes, which handle database operations. This annotation also makes database exception translation easier, so Spring can wrap database errors into more generic exceptions.

- **@Controller**
  A specialized `@Component` that is specifically for web controllers. It handles HTTP requests and returns responses. In a Spring MVC project, controllers are responsible for managing user inputs and displaying the appropriate views.

- **@RestController**
  Combines `@Controller` and `@ResponseBody`, which makes it ideal for REST APIs. Instead of returning views, it returns data directly as JSON or XML responses.

Dependency Injection Annotations

- **@Autowired**
  Used to automatically inject a dependency. It tells Spring to look for a matching object and set it into the annotated field or method. It's commonly used to inject services or repositories into other components.

- **@Qualifier**
  Works with `@Autowired` to specify which bean to inject when there are multiple candidates. It helps you choose the right implementation if you have several beans of the same type.

- **@Value**
  Injects values from configuration files (like `application.properties`) into fields, so you can use properties without hardcoding them in your code.

Configuration Annotations

- **@Configuration**
  Marks a class as a source of bean definitions. This means it's a class that contains methods to create and configure beans, which Spring will use for dependency injection.
- **@Bean**
  Used within a `@Configuration` class to create and configure a bean. When you define a method with `@Bean`, Spring will treat the returned object as a managed bean and inject it where needed.
- **@PropertySource**
  Tells Spring where to find external properties files (like `application.properties`). This is useful for reading configuration data from files.
- **@Profile**
  Specifies which configuration to use based on the current environment (e.g., `dev`, `test`, `prod`). This way, you can define different configurations for different environments.

Web Annotations

- **@RequestMapping**
  Maps HTTP requests to handler methods in `@Controller` or `@RestController` classes. It defines the URL endpoint, HTTP method, and other request properties.
- **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping**
  Shortcuts for `@RequestMapping` that specify HTTP methods directly (GET, POST, PUT, DELETE). They make it easy to map specific HTTP actions to methods in your controller.
- **@PathVariable**
  Binds a path variable from the URL to a method parameter. For example, in `/users/{id}`, `{id}` can be captured as a method parameter.
- **@RequestParam**
  Binds a query parameter from the URL to a method parameter. For example, `/search?name=John` binds `name=John` to a method parameter.
- **@RequestBody**
  Maps the body of an HTTP request directly to a Java object. Commonly used in REST APIs for reading JSON data sent in the request.

- **@ResponseBody**
  Used in controller methods to indicate that the return value should be written directly to the HTTP response body (instead of rendering a view). It's usually implied in `@RestController`.

- **@CrossOrigin**
  Allows cross-origin requests (i.e., from different domains) to this controller or method. This is helpful for enabling client-side applications on different servers to call your API.

Transaction Management Annotations

- **@Transactional**
  Declares that the method should run within a transaction. It helps ensure that all steps in the method complete successfully, or if there's an error, it will roll back all changes.

Validation Annotations

- **@Valid**
  Used to indicate that a method parameter (usually an object) should be validated before being processed. Typically combined with validation annotations on the object's fields (like `@NotNull`).

- **@NotNull, @Size, @Min, @Max, @Pattern**
  Common field-level validation annotations that check for constraints. For example:

  - `@NotNull` ensures a field is not null.
  - `@Size` checks for minimum or maximum length.
  - `@Min` and `@Max` define numerical limits.
  - `@Pattern` validates using a regular expression.

These annotations cover the core of what you'll use frequently when building applications with Spring, helping to manage objects, inject dependencies, handle HTTP requests, and enforce data rules.