Spring, Spring Boot, REST API, Hibernate, JPA, Validation, and Microservices Annotations Cheat Sheet

This cheat sheet covers key annotations for Spring, Spring Boot, REST APIs, Hibernate, JPA, Spring Data Validation, and Microservices, organized by category with brief explanations for quick reference.

1. Core Spring Annotations

Annotations for dependency injection and bean management in Spring.

Annotation	Purpose
`@Component`	Marks a class as a Spring-managed bean. Used for any generic component.
`@Service`	Specialization of `@Component` for service-layer classes (business logic).
`@Repository`	Specialization of `@Component` for DAOs, with exception translation for data access errors.
`@Configuration`	Defines a class as a source of bean definitions via `@Bean` methods.
`@Bean`	Declares a bean in a `@Configuration` class for manual bean setup.
`@Autowired`	Injects dependencies automatically by type or name.
`@Qualifier`	Resolves ambiguity when multiple beans of the same type exist with `@Autowired`.
`@Primary`	Marks a bean as preferred when multiple beans of the same type are available.
`@Scope`	Defines bean scope (e.g., `singleton`, `prototype`, `request`, `session`).
`@Lazy`	Delays bean initialization until first use.
`@Value`	Injects values from properties, environment variables, or expressions (e.g., `${property.key}`).
`@PostConstruct`	Runs a method after bean initialization.
`@PreDestroy`	Runs a method before bean destruction.
`@EventListener`	Handles Spring application events (e.g., `ContextRefreshedEvent`).
`@PropertySource`	Specifies external property files (e.g., `classpath:application.properties`).

2. Spring MVC and REST API Annotations

Annotations for building web applications and RESTful APIs with Spring MVC.

Annotation	Purpose
`@Controller`	Marks a class as a Spring MVC controller to handle HTTP requests.
`@RestController`	Combines `@Controller` and `@ResponseBody` for REST APIs (JSON/XML output).
`@RequestMapping`	Maps HTTP requests to methods or classes (supports all HTTP methods).
`@GetMapping`	Shortcut for `@RequestMapping` with HTTP GET.
`@PostMapping`	Shortcut for `@RequestMapping` with HTTP POST.
`@PutMapping`	Shortcut for `@RequestMapping` with HTTP PUT.
`@DeleteMapping`	Shortcut for `@RequestMapping` with HTTP DELETE.
`@PatchMapping`	Shortcut for `@RequestMapping` with HTTP PATCH.
`@RequestParam`	Binds query parameters to method arguments (e.g., `?id=1`).
`@PathVariable`	Binds URL path variables to method arguments (e.g., `/users/{id}`).
`@RequestBody`	Maps HTTP request body to a method argument (e.g., JSON payload).
`@ResponseBody`	Serializes method return value to HTTP response body (e.g., JSON).
`@ModelAttribute`	Binds form data or model attributes to method parameters or return values.
`@SessionAttributes`	Stores model attributes in the HTTP session between requests.
`@CrossOrigin`	Enables CORS for REST endpoints (e.g., `@CrossOrigin(origins = "*")`).
`@ExceptionHandler`	Handles exceptions thrown by controller methods.
`@InitBinder`	Customizes data binding for request parameters.
`@ControllerAdvice`	Global exception handling or shared logic across controllers.
`@RestControllerAdvice`	Combines `@ControllerAdvice` and `@ResponseBody` for RESTful exception handling.
`@ResponseStatus`	Sets the HTTP status code for a response (e.g., `@ResponseStatus(HttpStatus.NOT_FOUND)`).

3. Spring Data and JPA/Hibernate Annotations

Annotations for data access, JPA, and Hibernate-specific configurations.

Annotation	Purpose
`@Transactional`	Marks a method/class for transaction management (e.g., rollback on errors).
`@PersistenceContext`	Injects a JPA `EntityManager` for database operations.
`@Repository`	Marks a class as a DAO with exception translation (also a core annotation).
`@Query`	Defines custom JPQL or SQL queries in Spring Data repositories.
`@Modifying`	Marks a `@Query` method as an update or delete operation.
`@EnableJpaRepositories`	Enables scanning for Spring Data JPA repositories.
`@NamedQuery`	Defines a named JPQL query at the entity level.
`@EnableTransactionManagement`	Enables annotation-driven transaction management.
`@Entity`	Marks a class as a JPA entity mapped to a database table.
`@Table`	Specifies the database table name for a JPA entity.
`@Id`	Marks a field as the primary key in a JPA entity.
`@GeneratedValue`	Specifies the primary key generation strategy (e.g., `AUTO`, `IDENTITY`).
`@Column`	Maps a field to a database column, with optional attributes (e.g., `name`, `nullable`).
`@ManyToOne`	Defines a many-to-one relationship between entities.
`@OneToMany`	Defines a one-to-many relationship between entities.
`@OneToOne`	Defines a one-to-one relationship between entities.
`@ManyToMany`	Defines a many-to-many relationship between entities.
`@JoinColumn`	Specifies the foreign key column for a relationship.
`@JoinTable`	Defines a join table for many-to-many relationships.
`@Embedded`	Marks a field as an embeddable object within an entity.
`@Embeddable`	Marks a class as an embeddable type for use in entities.
`@Enumerated`	Maps an enum to a database column (e.g., `EnumType.STRING` or `ORDINAL`).

Annotation	Purpose
`@Temporal`	Specifies the type of a date/time field (e.g., `TemporalType.DATE`).
`@Version`	Marks a field for optimistic locking in JPA.
`@PrePersist`	Executes a method before an entity is persisted.
`@PostPersist`	Executes a method after an entity is persisted.
`@PreUpdate`	Executes a method before an entity is updated.
`@PostUpdate`	Executes a method after an entity is updated.
`@PreRemove`	Executes a method before an entity is removed.
`@PostRemove`	Executes a method after an entity is removed.

4. Spring Data Validation Annotations

Annotations for validating data in Spring applications, primarily from `javax.validation` and Hibernate Validator.

Annotation	Purpose
`@Valid`	Triggers validation of a method parameter or field (e.g., in `@RequestBody`).
`@NotNull`	Ensures a field is not null.
`@NotBlank`	Ensures a string is not null and not empty after trimming.
`@NotEmpty`	Ensures a collection, array, or string is not null and not empty.
`@Size`	Validates that a string, collection, or array size is within a range (e.g., `@Size(min=1, max=50)`).
`@Min`	Ensures a numeric value is greater than or equal to a specified minimum.
`@Max`	Ensures a numeric value is less than or equal to a specified maximum.
`@Pattern`	Ensures a string matches a regex pattern (e.g., `@Pattern(regexp="[a-z]*")`).
`@Email`	Validates that a string is a valid email address.
`@Past`	Ensures a date/time is in the past.
`@Future`	Ensures a date/time is in the future.
`@Positive`	Ensures a numeric value is positive (> 0).

Annotation	Purpose
`@Negative`	Ensures a numeric value is negative (< 0).
`@AssertTrue`	Ensures a boolean field or method evaluates to true.
`@AssertFalse`	Ensures a boolean field or method evaluates to false.
`@Validated`	Enables validation at the class or method level for Spring beans.

Note: Use `@Valid` or `@Validated` with `@RestController` to validate `@RequestBody` or `@ModelAttribute` inputs. Hibernate Validator provides additional constraints like `@NotBlank` and `@Email`.

5. Aspect-Oriented Programming (AOP) Annotations

Annotations for implementing cross-cutting concerns with Spring AOP.

Annotation	Purpose
`@Aspect`	Marks a class as an aspect for defining cross-cutting concerns (e.g., logging).
`@Before`	Runs advice before a method execution.
`@After`	Runs advice after a method execution, regardless of outcome.
`@Around`	Wraps advice around a method execution (can control execution flow).
`@AfterReturning`	Runs advice after a method returns successfully.
`@AfterThrowing`	Runs advice after a method throws an exception.
`@Pointcut`	Defines reusable pointcut expressions for AOP advice.

6. Scheduling and Asynchronous Processing Annotations

Annotations for scheduling tasks and asynchronous method execution.

Annotation	Purpose
`@Scheduled`	Schedules a method to run periodically or at specific times (e.g., cron).
`@EnableScheduling`	Enables scheduling support in the application.
`@Async`	Marks a method for asynchronous execution in a separate thread.

Annotation	Purpose
`@EnableAsync`	Enables asynchronous method execution in the application.

7. Spring Boot-Specific Annotations

Annotations for simplifying configuration and development in Spring Boot.

Annotation	Purpose
`@SpringBootApplication`	Combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`.
`@EnableAutoConfiguration`	Enables Spring Boot's auto-configuration of beans based on classpath.
`@ConditionalOnProperty`	Configures a bean only if a specific property is present (e.g., `spring.datasource.url`).
`@ConditionalOnMissingBean`	Configures a bean only if a specific bean is not already defined.
`@EnableConfigurationProperties`	Binds configuration properties to POJOs (e.g., `@ConfigurationProperties`).
`@ConfigurationProperties`	Maps properties (e.g., from `application.properties`) to a POJO.
`@SpringBootConfiguration`	Marks a class as a configuration class (part of `@SpringBootApplication`).

8. Spring Security Annotations

Annotations for securing Spring applications.

Annotation	Purpose
`@EnableWebSecurity`	Enables Spring Security configuration for the application.
`@Secured`	Restricts method access based on specified roles (e.g., `ROLE_ADMIN`).
`@PreAuthorize`	Applies security checks before method execution (e.g., `hasRole('ADMIN')`).
`@PostAuthorize`	Applies security checks after method execution (e.g., checking return value).

Annotation	Purpose
`@RolesAllowed`	Specifies allowed roles for a method (JSR-250 equivalent of `@Secured`).
`@EnableGlobalMethodSecurity`	Enables method-level security with `@Secured`, `@PreAuthorize`, etc.

9. Cloud and Microservices Annotations

Annotations for Spring Cloud and microservices, including additional annotations for distributed systems.

Annotation	Purpose
`@EnableDiscoveryClient`	Enables service discovery (e.g., with Eureka, Consul, or Zookeeper).
`@FeignClient`	Declares a Feign client for HTTP-based microservice communication.
`@EnableCircuitBreaker`	Enables circuit breaker patterns (e.g., with Hystrix or Resilience4j).
`@HystrixCommand`	Annotates methods with fallback logic for circuit breaker resilience.
`@RefreshScope`	Refreshes bean definitions at runtime when configuration changes.
`@LoadBalanced`	Marks a `RestTemplate` or WebClient for client-side load balancing.
`@EnableConfigServer`	Enables a Spring Cloud Config Server for centralized configuration.
`@EnableEurekaClient`	Enables Eureka-specific service discovery (alternative to `@EnableDiscoveryClient`).
`@Retryable`	Marks a method for retry on failure (Spring Cloud Retry).
`@CircuitBreaker`	Configures circuit breaker behavior (Spring Cloud Resilience4j).
`@RateLimiter`	Limits the rate of method calls (Spring Cloud Resilience4j).
`@Bulkhead`	Isolates method execution to prevent cascading failures (Resilience4j).

Note: Microservices annotations like `@FeignClient` and `@LoadBalanced` are commonly used with Spring Cloud for distributed systems, enabling service discovery, load balancing, and resilience patterns.

10. Testing Annotations

Annotations for testing Spring and Spring Boot applications.

Annotation	Purpose
`@SpringBootTest`	Loads the full Spring application context for integration testing.
`@WebMvcTest`	Tests Spring MVC controllers with a minimal context.
`@DataJpaTest`	Tests JPA repositories with an in-memory database (e.g., H2).
`@MockBean`	Creates mock beans in the Spring context for testing.
`@TestConfiguration`	Defines test-specific configuration classes for Spring tests.
`@AutoConfigureMockMvc`	Configures MockMvc for testing Spring MVC controllers.