Core Spring Annotations

@Component : Mark 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 transaction for data access errors.

@Configuration : Define a class as a source of bean definitions via @Bean methods.

@Bean : Declare a bean in a @Configuration class for manual bean setup.

@Autowired : Injects dependencies automatically by type or by name.

@Qualifier : Resolves ambiguity when multiple beans of the same type exists with @Autowired.

@Primary : Mark a bean as preferred when multiple beans of same type are variable.

@Scope : Defines bean scope(e.g. singleton, prototype, request, session)

@Lazy : Delays bean initialization until first use.

@Value : Injects value from properties, environment variables or expressions(e.g. ${propert.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)

Spring MVC and REST API Annotations

@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 (support 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 argument(e.g. ?id=1)

@PathVariable : Binds URL path variables to method arguments(e.g. /users/{id})

@RequestBody : Maps HTTP request body to a method arguments(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 CROSS and REST endpoints(e.g. @CrossOrigin(origins=”\*”))

@ExceptionHandler : Handles exception 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))

Spring Data and JPA/Hibernate Annotations

@Transactional : Marks a method/class for transaction management(e.g. rollback on errors)

@PersistentContext : Injects a JPA EntityManager for database operations.

@Respository : Marks a class as a DAO with exception translation(also a core annotation).

@Query : Defines custom JPQL and SQL queries in Spring Data repositories.

@Modifing : Marks a @Query method as an update or delete operation.

@EnableJpaRespositories : Enable scanning for Spring data JPA repositories.

@NamedQuery : Defines a named JPQL query at the entity level.

@EnableTransactionManagemet : Enable 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 field as a 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 Many to One relationship between entities.

@OneToMany : Defines One to Many relationship between entities.

@OneToOne : Defines One to One relationship between entities.

@ManyToMany : Defines Many to Many relationship between entities.

@JoinColumn : Specifies the foreign key column for a relationship.

@JoinTable : Define a join table for many-to-many relationships.

@Embedded : Marks filed as a embeddable object within an entity.

@Embeddable : Marks a class as an embeddable type for use an entities.

@Enumerated : Maps an enum to database column (e.g. EnumType.STRING or ORDINAL)

@Temporal : Specifies the type of a date/time field.(e.g. TemporaType.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.

Spring Data Validation Annotations

@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 specified minimum.

@Max : Ensures a numeric value is less than or equal to specified maximum.

@Pattern : Ensures a string matches a regex pattern (e.g. @Pattern(regexp=”[a-z]\*”))

@Email : Validates that 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)

@Negative : Ensures a numeric value is negative(<0)

@AssertTrue : Ensures a boolean field or method evaluates to true.

@AssterFalse : Ensures a boolean filed or method evaluates to false.

@Validated : Enables validation at the class or method level for Spring beans.

Aspect-Oriented Programming(AOP) Annotations:

@Aspect : Mark 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 return Successfully.

@AfterThrowing : Runs advice after a method throw an exception.

@Pointcut : Defines reusable pointcut expressions for AOP advice.

Scheduling and Asynchronous Processing Annotations

@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.

@EnableAsync : Enables asynchronous method execution in the application.

Spring Boot-Specific Annotations

@SpringBootApplication : Combines @Configuration, @EnableAutoConfiguration and @ComponentScan

@EnableAutoConfiguration : Enables spring boot auto configuration of beans based on classpath.

@ConditionalOnProperty : Configures a bean only if a specific property is present (e.g. spring.datasource.url)

@ConditionOnMissingBean : 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)

Spring Security Annotations

@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 check after method execution(e.g. checking return value)

@RolesAllowed : Specifies allowed roles for a method (JSR-250 equivalent of @Secured)

@EnableGlobalMethodSecurity : Enables method-level security with @Secured, @PreAuthorize, etc.

Cloud And Microservices Annotations

@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 patters (e.g. with Hystrix or Resilience4j)

@HystrixCommand : Annotates method 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)

@Bulkhead : Isolates method execution to prevent cascading failures (Resilence4j)

Testing Annotations

@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 : Configure MockMvc for testing Spring mvc controllers.