**Core Spring Framework Annotations**

1. **@Component**
   * Marks a class as a Spring-managed component (a bean).
   * Scanned automatically by Spring’s classpath scanning.
2. **@Service**
   * Specialization of @Component for service-layer beans.
   * Indicates that the class contains business logic.
3. **@Repository**
   * Specialization of @Component for persistence-layer beans.
   * Provides additional capabilities such as exception translation.
4. **@Controller**
   * Specialization of @Component for MVC controllers.
   * Handles web requests and typically returns views.
5. **@RestController**
   * Combines @Controller and @ResponseBody.
   * Simplifies the creation of RESTful APIs by returning JSON/XML directly.

**Spring Boot Annotations**

1. **@SpringBootApplication**
   * A combination of:
     + @Configuration: Indicates a configuration class.
     + @EnableAutoConfiguration: Enables Spring Boot’s auto-configuration.
     + @ComponentScan: Scans for Spring components.
   * Serves as the entry point for a Spring Boot application.
2. **@EnableAutoConfiguration**
   * Enables Spring Boot’s auto-configuration feature based on the classpath and properties.
3. **@Configuration**
   * Marks a class as a configuration class.
   * Used to define Spring beans using methods annotated with @Bean.
4. **@Bean**
   * Used within @Configuration classes to define beans programmatically.
5. **@ConditionalOnProperty**
   * Configures beans based on specific properties in application.properties or application.yml.
6. **@ConditionalOnMissingBean**
   * Creates a bean only if no other bean of the same type exists in the application context.
7. **@EnableScheduling**
   * Enables support for scheduling tasks (e.g., via @Scheduled).
8. **@EnableCaching**
   * Enables caching support in the application.

**Web Annotations**

1. **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping**
   * Specialized annotations for mapping HTTP methods to specific handler methods.
2. **@RequestMapping**
   * General-purpose annotation for mapping HTTP requests to handler methods.
3. **@RequestParam**
   * Binds request parameters (query parameters) to method arguments.
4. **@PathVariable**
   * Binds URI template variables to method arguments.
5. **@RequestBody**
   * Maps the body of an HTTP request to a Java object.
6. **@ResponseBody**
   * Indicates that the return value of a method should be written directly to the HTTP response body.
7. **@RestControllerAdvice**
   * Combines @ControllerAdvice and @ResponseBody.
   * Used to handle exceptions globally in a RESTful application.

**Data and JPA Annotations**

1. **@Entity**
   * Marks a class as a JPA entity mapped to a database table.
2. **@Table**
   * Specifies the table name for a JPA entity.
3. **@Id**
   * Denotes the primary key of a JPA entity.
4. **@GeneratedValue**
   * Specifies the generation strategy for primary keys (e.g., AUTO, IDENTITY).
5. **@Column**
   * Customizes column mapping for a JPA entity field.
6. **@RepositoryRestResource**
   * Exposes a repository as a RESTful resource.
7. **@Query**
   * Defines custom queries using JPQL or native SQL.
8. **@Transactional**
   * Manages transactional behavior for methods or classes.

**Validation Annotations**

1. **@Valid**
   * Validates a Java object based on constraints defined by annotations.
2. **@NotNull, @NotBlank, @Size, @Pattern**
   * Hibernate Validator annotations for input validation.

**Security Annotations**

1. **@EnableWebSecurity**
   * Enables Spring Security in the application.
2. **@PreAuthorize**
   * Secures methods based on roles or conditions (e.g., @PreAuthorize("hasRole('ADMIN')")).
3. **@Secured**
   * Secures methods with specific roles (e.g., @Secured("ROLE\_USER")).
4. **@RolesAllowed**
   * Specifies roles allowed to access a method.

**Asynchronous and Scheduling Annotations**

1. **@Async**
   * Marks a method for asynchronous execution.
2. **@Scheduled**
   * Schedules a task to run at fixed intervals or cron expressions.

**Actuator Annotations**

1. **@Endpoint**
   * Custom actuator endpoints.
2. **@ReadOperation, @WriteOperation, @DeleteOperation**
   * Define read, write, and delete operations for actuator endpoints.

**Testing Annotations**

1. **@SpringBootTest**
   * Boots up the entire application context for testing.
2. **@MockBean**
   * Creates a mock bean in the application context for testing.
3. **@WebMvcTest**
   * Tests only the web layer of the application.
4. **@DataJpaTest**
   * Tests JPA repositories with an in-memory database.

s