SPRING BOOT ANNOTATIONS

1. Core Annotations

No.	Annotation	Description
1.	**@SpringBootApplication**	Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. Entry point for Spring Boot applications.
2.	**@Configuration**	Marks a class as a source of bean definitions for the application context.
3.	**@Bean**	Indicates a method produces a bean managed by the Spring container.
4.	**@Component**	Marks a class as a Spring-managed component (generic stereotype for any Spring-managed bean).
5.	**@Service**	Specialization of @Component for service-layer classes.
6.	**@Repository**	Specialization of @Component for persistence-layer classes. Automatically translates exceptions into Spring's data access exceptions.
7.	**@Controller**	Indicates that a class handles web requests (used in MVC).
8.	**@RestController**	Combines @Controller and @ResponseBody. Used to handle REST APIs.

2. Dependency Injection Annotations

No.	Annotation	Description
9.	**@Autowired**	Automatically injects a bean by type into the field, constructor, or method.
10.	**@Qualifier**	Specifies the bean to inject when multiple candidates are available, used with @Autowired.
11.	**@Primary**	Marks a bean as the primary candidate when multiple beans of the same type are available for injection.
12.	**@Value**	Injects values from property files or environment variables into fields, methods, or constructors.
13.	**@Scope**	Specifies the scope of a bean (singleton, prototype, request, session, etc.).

SPRING BOOT ANNOTATIONS

3. Web and HTTP Annotations

No.	Annotation	Description
14.	**@RequestMapping**	Maps HTTP requests to handler methods or classes, supporting all HTTP methods.
15.	**@GetMapping**	Shortcut for HTTP GET requests.
16.	**@PostMapping**	Shortcut for HTTP POST requests.
17.	**@PutMapping**	Shortcut for HTTP PUT requests.
18.	**@DeleteMapping**	Shortcut for HTTP DELETE requests.
19.	**@PatchMapping**	Shortcut for HTTP PATCH requests.
20.	**@RequestBody**	Binds the HTTP request body to a method parameter, allowing automatic deserialization.
21.	**@RequestParam**	Extracts query parameters from the request URL and binds them to method parameters.
22.	**@RequestHeader**	Binds a request header to a method parameter in the controller.
23.	**@PathVariable**	Extracts values from the URL path and binds them to method parameters.
24.	**@ResponseBody**	Indicates that the return value of a method should be written directly to the HTTP response body.
25.	**@CrossOrigin**	Configures Cross-Origin Resource Sharing (CORS) for a resource or endpoint, enabling cross-origin requests.
26.	**@CookieValue**	Binds a cookie value to a method parameter in the controller.
27.	**@ModelAttribute**	Binds request parameters to a model object, allowing for easier population of form objects.
28.	**@SessionAttributes**	Specifies model attributes that should be stored in the session.
29.	**@InitBinder**	Allows customizing the binding of web request parameters to method arguments, typically used for custom property editors.

SPRING BOOT ANNOTATIONS

4. Security Annotations

No.	Annotation	Description
30.	**@EnableWebSecurity**	Enables Spring Security's web security features for the application.
31.	**@EnableGlobalMethodSecurity**	Enables method-level security annotations like @PreAuthorize and @Secured.
32.	**@Secured**	Marks a method or class to restrict access based on roles or authorities.
33.	**@PreAuthorize**	Provides access control based on expressions, allowing method-level security using SpEL (Spring Expression Language).
34.	**@PostAuthorize**	Allows access control after a method is executed, useful for restricting access based on the method's return value.
35.	**@RolesAllowed**	Restricts access to a method or class based on specified roles.
36.	**@PreFilter**	Filters input parameters before a method is executed. Useful for restricting data based on security attributes.
37.	**@PostFilter**	Filters output parameters after a method is executed. Useful for restricting data based on security attributes.
38.	**@Authenticated**	Indicates that a user must be authenticated to access the annotated method or class.
39.	**@EnableOAuth2Sso**	Enables Single Sign-On (SSO) using OAuth2 for a Spring Boot application, typically with services like Google or Facebook.
40.	**@EnableResourceServer**	Marks the application as a resource server for OAuth2, used for securing REST APIs.
41.	**@EnableAuthorizationServer**	Configures the application as an OAuth2 authorization server, managing tokens and authorization flows.
42.	**@AuthenticationPrincipal**	Binds the currently authenticated principal (usually the user details) to a method parameter.
43.	**@EnableGlobalAuthentication**	Enables global authentication settings, allowing custom authentication logic across the entire application.

SPRING BOOT ANNOTATIONS

5. Data Access Annotations

No.	Annotation	Description
JPA Annotations		
44.	**@Entity**	Marks a class as a JPA entity, mapped to a database table.
45.	**@Table**	Specifies the database table to which an entity is mapped.
46.	**@Id**	Marks a field as the primary key in an entity.
47.	**@GeneratedValue**	Specifies the strategy for generating primary key values.
48.	**@Column**	Maps a field to a specific database column.
49.	**@OneToOne**	Defines a one-to-one relationship between entities.
50.	**@OneToMany**	Defines a one-to-many relationship between entities.
51.	**@ManyToOne**	Defines a many-to-one relationship between entities.
52.	**@ManyToMany**	Defines a many-to-many relationship between entities.
53.	**@JoinColumn**	Specifies the column to join in a relationship.
54.	**@JoinTable**	Specifies the join table for a many-to-many relationship.
55.	**@Transactional**	Manages transactions in JPA operations.
56.	**@Query**	Defines custom JPQL or SQL queries for repository methods.
57.	**@NamedQuery**	Declares a static, named query at the entity level.
58.	**@EntityListeners**	Specifies listener classes for entity lifecycle events.
59.	**@Embeddable**	Marks a class that can be embedded in another entity.
60.	**@Embedded**	Marks a field as embedded (referring to an @Embeddable type).
Spring JDBC Annotations		
61.	**@Repository**	Indicates that a class is a Spring-managed repository for data access.
62.	**@Autowired**	Injects the required dependencies into Spring beans, including JDBC repositories.
Spring Data Access Annotations		
63.	**@EnableJpaRepositories**	Enables Spring Data JPA repositories.
64.	**@EnableTransactionManagement**	Enables transaction management in Spring applications.
Spring Data Annotations		
65.	**@QueryParam**	Binds query parameters to method arguments.
66.	**@Param**	Binds method parameters in a named query to method arguments.
67.	**@Transactional(readOnly=true)**	Specifies that the transaction is read-only for performance optimization.
68.	**@EnableBatchProcessing**	Enables Spring Batch configuration for batch processing jobs.
Spring MongoDB Annotations		
69.	**@Document**	Marks a class as a MongoDB document.
70.	**@Field**	Maps a field in a MongoDB document to a specific field in the class.
71.	**@Id**	Marks the primary key field in MongoDB.
Spring Redis Annotations		

72.	**@RedisHash**	Marks a class as a Redis hash.
73.	**@Id**	Marks the primary key for Redis data.
74.	**@Indexed**	Marks a field to be indexed in Redis.
Spring Couchbase Annotations		
75.	**@Document**	Marks a class as a Couchbase document.
76.	**@Id**	Marks the primary key for Couchbase documents.
Spring SQL Annotations		
77.	**@NamedNativeQuery**	Declares a native SQL query at the entity level.
78.	**@Query**	Defines a custom SQL query for the repository method.
Spring Transaction Annotations		
79.	**@EnableTransactionManagement**	Enables Spring's declarative transaction management.
80.	**@Transactional(propagation=Propagation.REQUIRES_NEW)**	Begins a new transaction, suspending the current one.
81.	**@Transactional(isolation=Isolation.SERIALIZABLE)**	Sets the isolation level for the transaction.

6. Cache Management Annotations

No.	Annotation	Description
82.	**@EnableCaching**	Enables caching support in a Spring Boot application.
83.	**@Cacheable**	Marks a method to cache its result based on the method parameters.
84.	**@CachePut**	Updates the cache with the result of the method, regardless of whether the method was cached.
85.	**@CacheEvict**	Evicts or removes entries from the cache.
86.	**@Caching**	Combines multiple cache-related annotations on a single method.
87.	**@CacheConfig**	Provides default cache settings at the class level.
88.	**@Cacheable(value="cacheName", key="#id")**	Caches the result of a method with a specified key and cache name.
89.	**@CacheEvict(allEntries=true)**	Evicts all entries from a cache when the method is invoked.
90.	**@CacheEvict(beforeInvocation=true)**	Evicts cache entries before the method invocation.
91.	**@Cacheable(condition="#param > 5")**	Caches the method result only if a condition is met.
92.	**@CacheEvict(condition="#param == 0")**	Evicts cache entries if a condition is met.

SPRING BOOT ANNOTATIONS

7. Testing Annotations

No.	Annotation	Description
93.	**@SpringBootTest**	Used for integration testing with Spring Boot, sets up the application context and runs tests within it.
94.	**@WebMvcTest**	Used for testing Spring MVC controllers, focusing on web layer testing by mocking the service layer.
95.	**@DataJpaTest**	Used for testing JPA repositories, loads only the JPA context (database-related beans) for unit tests.
96.	**@MockBean**	Mocks a bean in the application context to test components in isolation (usually used with @SpringBootTest).
97.	**@Mock**	Creates a mock of the specified class or interface (commonly used with Mockito).
98.	**@InjectMocks**	Injects mocks into the annotated field, usually used for testing service or component classes.
99.	**@Test**	Marks a method as a test method in JUnit 5 (used to define a unit test).
100.	**@BeforeEach**	Marks a method to be executed before each test method in JUnit 5 (similar to @Before in JUnit 4).
101.	**@AfterEach**	Marks a method to be executed after each test method in JUnit 5 (similar to @After in JUnit 4).
102.	**@BeforeAll**	Marks a method to be executed before all test methods in JUnit 5 (similar to @BeforeClass in JUnit 4).
103.	**@AfterAll**	Marks a method to be executed after all test methods in JUnit 5 (similar to @AfterClass in JUnit 4).
104.	**@TestConfiguration**	Defines additional configuration for unit tests, useful when creating custom beans for test environments.
105.	**@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)**	Configures Spring Boot to run tests with an embedded server on a random port.
106.	**@DirtiesContext**	Marks a test class or method to indicate that the application context should be considered dirty after the test.
107.	**@ExtendWith**	Used in JUnit 5 to extend the functionality of test classes, often used with custom extensions.
108.	**@Value**	Injects values into test fields, useful for testing properties or configurations (e.g., environment variables).
109.	**@TestPropertySource**	Specifies the locations of property files to use for the test configuration, overriding the application.properties.
110.	**@Autowired**	Automatically injects dependencies into the test class, ensuring Spring-managed beans are available for testing.
111.	**@Profile**	Activates specific profiles for the test class, ensuring only beans from the active profile are loaded during the test.

112.	**@Transactional**	Marks a test method or class to run within a transaction, ensuring changes are rolled back after the test.
113.	**@Captor**	Creates a Mockito ArgumentCaptor to capture arguments passed to mocked methods.
114.	**@TestExecutionListeners**	Specifies the listeners to use for the lifecycle of the tests (e.g., for managing the context).
115.	**@SpringJUnitConfig**	Combines @ContextConfiguration and @EnableJUnit for configuration and test context setup in Spring tests.

8. Scheduling Annotations

No.	Annotation	Description
116.	**@EnableScheduling**	Enables support for scheduled tasks in Spring Boot, typically used to enable cron-based tasks.
117.	**@Scheduled**	Marks a method to be run at fixed intervals or according to a cron expression (e.g., periodic tasks).
118.	**@Scheduled(fixedRate)**	Schedules a task to run at a fixed rate, measured from the start of the previous task.
119.	**@Scheduled(fixedDelay)**	Schedules a task to run after a fixed delay, measured from the end of the previous task.
120.	**@Scheduled(cron)**	Schedules a task based on a cron expression, allowing complex task schedules

9. Asynchronous Processing Annotations

No.	Annotation	Description
121.	@Async	Marks a method to be executed asynchronously, allowing the method to run in a separate thread.
122.	@EnableAsync	Enables support for asynchronous method execution in a Spring application. It needs to be added at the configuration class level.
123.	@AsyncResult	Represents the result of an asynchronous method call and can be used to return a value from an asynchronous method.

SPRING BOOT ANNOTATIONS

10. Messaging Annotations

No.	Annotation	Description
124.	**@EnableRabbit**	Enables Spring AMQP (RabbitMQ) messaging and configures listeners for RabbitMQ message queues.
125.	**@RabbitListener**	Marks a method as a listener for RabbitMQ messages, processing messages from a specific queue.
126.	**@JmsListener**	Marks a method to listen for messages from a JMS (Java Message Service) destination (e.g., ActiveMQ).
127.	**@EnableJms**	Enables JMS messaging in the Spring Boot application, allowing configuration for JMS message consumers.
128.	**@SendTo**	Specifies where to send the return value of a @RabbitListener or @JmsListener method (target queue or topic).
129.	**@MessagingGateway**	Defines a messaging gateway interface, allowing methods to send messages through a message channel.
130.	**@ServiceActivator**	Declares a method as a message handler that processes the incoming message and routes it.
131.	**@MessageEndpoint**	Marks a class as a message endpoint, enabling it to handle messages in Spring Integration.
132.	**@Router**	Defines a method for routing messages to appropriate handlers or channels in Spring Integration.
133.	**@Transformer**	Marks a method as a message transformer that processes and modifies the message payload.
134.	**@EnableRabbit**	Enables Spring AMQP (RabbitMQ) messaging and configures listeners for RabbitMQ message queues.
135.	**@RabbitListener**	Marks a method as a listener for RabbitMQ messages, processing messages from a specific queue.
136.	**@JmsListener**	Marks a method to listen for messages from a JMS (Java Message Service) destination (e.g., ActiveMQ).
137.	**@EnableJms**	Enables JMS messaging in the Spring Boot application, allowing configuration for JMS message consumers.
138.	**@SendTo**	Specifies where to send the return value of a @RabbitListener or @JmsListener method (target queue or topic).

SPRING BOOT ANNOTATIONS

11. Scope Annotations

No.	Annotation	Description
139.	**@Singleton**	Ensures that the annotated bean is a single instance within the Spring container (default scope).
140.	**@Scope("prototype")**	Creates a new instance of the bean every time it is requested from the container.
141.	**@Scope("request")**	The bean is created per HTTP request and is discarded after the request is completed.
142.	**@Scope("session")**	The bean is created per HTTP session and is discarded after the session ends.
143.	**@Scope("application")**	The bean is created once per servlet context (application-wide scope).
144.	**@Scope("websocket")**	The bean is created for each WebSocket session.
145.	**@Configurable**	Marks a class as needing configuration via dependency injection. (Used for IoC container integration)
146.	**@Scope("refresh")**	The bean is re-initialized upon each refresh of the configuration (commonly used with Spring Cloud).
147.	**@WebSocketScope**	Creates a new instance for each WebSocket session, managing the lifecycle based on WebSocket connections.
148.	**@RequestScope**	The bean will be created once per HTTP request. It is similar to @Scope("request"), but it's more explicit and easier to use in Spring Boot.
149.	**@SessionScope**	The bean will be created once per HTTP session. It is similar to @Scope("session"), but provides a more explicit way to define session-scoped beans.
150.	**@ApplicationScope**	The bean will be created once per ServletContext, effectively making it application-wide. It is equivalent to @Scope("application") but more convenient to use in Spring Boot.
151.	**@EnableAspectJAutoProxy(proxyTargetClass=false)**	Enables Spring AOP for proxy beans. When proxyTargetClass=false, it uses interface-based proxying.
152.	**@EnableAspectJAutoProxy(proxyTargetClass=true)**	Enables Spring AOP for proxy beans. When proxyTargetClass=true, it uses class-based proxying.

SPRING BOOT ANNOTATIONS

12. Spring Batch Annotations

No.	Annotation	Description
153.	**@EnableBatchProcessing**	Enables Spring Batch functionality in the application.
154.	**@JobScope**	Marks a bean as scoped to the lifetime of a Spring Batch job execution.
155.	**@StepScope**	Marks a bean as scoped to the lifetime of a Spring Batch step execution.
156.	**@Job**	Defines a Spring Batch job, typically used on methods to declare a job.
157.	**@Step**	Defines a step in a Spring Batch job.
158.	**@BeforeStep**	Indicates that a method should be executed before a Spring Batch step.
159.	**@AfterStep**	Indicates that a method should be executed after a Spring Batch step.
160.	**@BeforeJob**	Indicates that a method should be executed before a Spring Batch job.
161.	**@AfterJob**	Indicates that a method should be executed after a Spring Batch job.
162.	**@Partitioner**	Marks a class as a partitioner that divides a batch job into smaller chunks.
163.	**@Tasklet**	Defines a tasklet, which is the core unit of work in a Spring Batch step.
164.	**@ItemReader**	Marks a bean as an item reader in a Spring Batch step.
165.	**@ItemProcessor**	Marks a bean as an item processor in a Spring Batch step.
166.	**@ItemWriter**	Marks a bean as an item writer in a Spring Batch step.

13. Pagination Annotations

No.	Annotation	Description
167.	**@PageableDefault**	Specifies default values for pagination parameters (such as page size and sorting) when a Pageable object is not provided in the method.
168.	**@Sort**	Used with Pageable to define sorting information, allowing pagination results to be ordered based on the provided sort criteria.
169.	**@Query**	Used in Spring Data repositories to define custom queries, often used in conjunction with Pageable to return paginated results.
170.	**@Pagination**	Not a standard Spring Boot annotation, but typically refers to the concept of paginating results in custom implementations using Pageable.
171.	**Pageable Interface**	Not an annotation itself, but it is an interface used in Spring Data repositories and controllers to handle pagination functionality by automatically receiving and passing page parameters.

SPRING BOOT ANNOTATIONS

14. Spring Cloud Annotations

No.	Annotation	Description
172.	**@EnableDiscoveryClient**	Enables service discovery functionality for microservices using Spring Cloud (e.g., Eureka).
173.	**@EnableCircuitBreaker**	Enables Hystrix circuit breaker functionality to handle failures and fallback logic.
174.	**@EnableConfigServer**	Marks a Spring Boot application as a Spring Cloud Config Server to serve configuration files.
175.	**@EnableZuulProxy**	Enables the Zuul API Gateway to route requests to microservices and apply filters.
176.	**@EnableEurekaClient**	Enables a Spring Boot application to register as a client with Eureka for service discovery.
177.	**@SpringCloudApplication**	A convenience annotation combining several annotations to bootstrap Spring Cloud applications.
178.	**@HystrixCommand**	Marks a method as a Hystrix command, enabling circuit breaker and fallback logic.
179.	**@EnableFeignClients**	Enables Feign client functionality for making REST calls between Spring Cloud microservices.
180.	**@RibbonClient**	Customizes Ribbon load balancing configuration for a specific service client.
181.	**@StreamListener**	Marks a method as a listener for processing messages from Spring Cloud Stream.
182.	**@EnableHystrixDashboard**	Enables the Hystrix Dashboard for monitoring circuit breakers and microservice health.
183.	**@EnableBatchProcessing**	Enables Spring Batch processing in a Spring Cloud application.
184.	**@EnableDiscoveryClient**	Enables service discovery functionality for microservices using Spring Cloud (e.g., Eureka).
185.	**@EnableCircuitBreaker**	Enables Hystrix circuit breaker functionality to handle failures and fallback logic.
186.	**@EnableConfigServer**	Marks a Spring Boot application as a Spring Cloud Config Server to serve configuration files.

SPRING BOOT ANNOTATIONS

15. Fault Tolerance Annotations (Resilience4j)

No.	Annotation	Description
187.	@Retryable	Marks a method to be retried a specified number of times when a failure occurs.
188.	@Recover	Defines a method to be called when retry attempts for a @Retryable method are exhausted.
189.	@CircuitBreaker	Provides a fault tolerance mechanism that allows a method to fail fast when a threshold of failures is reached.
190.	@Fallback	Specifies a fallback method that gets executed when a service call fails or the circuit is open.
191.	@Timeout	Specifies a timeout for a method to define when to fail if the method execution takes too long.
192.	@RateLimiter	Limits the rate of method invocations to prevent system overload.
193.	@Bulkhead	Restricts the number of concurrent calls to a method to avoid overwhelming resources

16. Exception Handling Annotations

No.	Annotation	Description
194.	@ExceptionHandler	Handles specific exceptions thrown by controller methods and allows custom handling.
195.	@ControllerAdvice	Global exception handler for all controllers, providing centralized exception handling.
196.	@RestControllerAdvice	Similar to @ControllerAdvice, but returns the response as JSON or XML for REST APIs.
197.	@ResponseStatus	Specifies the HTTP status code to be returned with a response, typically used with exceptions.
198.	@ResponseBody	Converts the return value of a controller method into the HTTP response body, typically used for error responses.
199.	@Valid / @Validated	Used to trigger validation on method parameters or request bodies and automatically handle validation errors.
200.	@ExceptionHandler (Method-Level)	Handles exceptions specifically for a method within a controller, making it more localized.
201.	@GlobalExceptionHandler	A custom annotation, often used as a naming convention, for handling exceptions globally within a Spring application.

17. Validation Annotations

No.	Annotation	Description

202.	@NotNull	Ensures that a field cannot be null.
203.	@NotEmpty	Ensures that a string, collection, or array is not empty.
204.	@NotBlank	Ensures that a string is not null or empty and contains at least one non-whitespace character.
205.	@Size	Specifies the allowed size range for strings, collections, or arrays.
206.	@Min	Ensures that a numeric value is greater than or equal to a specified minimum.
207.	@Max	Ensures that a numeric value is less than or equal to a specified maximum.
208.	@Email	Validates that a string is a well-formed email address.
209.	@Pattern	Ensures that a string matches a specified regular expression.
210.	@Past	Ensures that a date or time is in the past.
211.	@Future	Ensures that a date or time is in the future.
212.	@AssertTrue	Ensures that a boolean value is true.
213.	@AssertFalse	Ensures that a boolean value is false.
214.	@Valid	Triggers validation for a field or method parameter. Triggers validation on a nested object (bean).
215.	@NotNull	Ensures that a field cannot be null.

18. Aspect-Oriented Programming (AOP) Annotations

No.	Annotation	Description
216.	@Aspect	Marks a class as an aspect, where cross-cutting concerns like logging or transaction handling are defined.
217.	@Before	Indicates that the annotated method should be executed before the target method is invoked.
218.	@After	Indicates that the annotated method should be executed after the target method execution (regardless of success).
219.	@AfterReturning	Indicates that the annotated method should be executed after the target method returns successfully.
220.	@AfterThrowing	Indicates that the annotated method should be executed if the target method throws an exception.
221.	@Around	Allows custom behavior to be executed before and after a method invocation, including the ability to modify the method's result or prevent it.
222.	@DeclareParents	Adds new functionality (i.e., method) to an existing class without modifying its code.
223.	@EnableAspectJAutoProxy	Enables support for AOP proxy creation and management, allowing the aspects to be woven into the codebase.

19. Metrics and Monitoring Annotations

No.	Annotation	Description

224.	@Endpoint	Defines a custom endpoint for monitoring and metrics, exposing specific data about the application.
225.	@ReadOperation	Marks a method within an @Endpoint as handling read operations, typically used to expose application metrics.
226.	@WriteOperation	Marks a method within an @Endpoint as handling write operations, typically used for modifying application state or metrics.
227.	@DeleteOperation	Marks a method within an @Endpoint to handle delete operations for metrics or data exposed via custom endpoints.
228.	@Metric	A custom annotation used for tracking specific metrics in Spring Boot Actuator, typically used with custom beans or metrics classes.
229.	@Timed	Automatically times method executions and tracks the duration as a metric, often used with @Endpoint or service methods.
230.	@Counted	Tracks the number of times a method is called, often used for measuring the frequency of specific operations.
231.	@Gauge	Provides a method to track a dynamic value that can be reported as a gauge metric (e.g., current load).
232.	@EnableMetrics	Enables Spring Boot Actuator metrics, making them available for monitoring and export to external systems.
233.	@HealthIndicator	Customizes application health check endpoints, allowing you to define application-specific health checks.
234.	@Timed (method level)	Tracks execution time of a method and exposes the result as a metric, useful for performance monitoring.

20. Logging Annotations

No.	Annotation	Description
235.	@Slf4j	Generates a logger field using SLF4J (Simple Logging Facade for Java) in the annotated class. It is a Lombok annotation.
236.	@Log	Generates a logger field using Java's built-in java.util.logging.Logger in the annotated class. It is a Lombok annotation.
237.	@Log4j	Generates a logger field using Log4j in the annotated class. It is a Lombok annotation.
238.	@Log4j2	Generates a logger field using Log4j2 in the annotated class. It is a Lombok annotation.
239.	@CommonsLog	Generates a logger field using Apache Commons Logging in the annotated class. It is a Lombok annotation.
240.	@XSlf4j	Generates a logger field using XSLF4J (another version of SLF4J) in the annotated class. It is a Lombok annotation.