SPRING BOOT ANNOTATIONS CHEAT SHEET

1. **@SpringBootApplication:** This annotation is used to bootstrap a Spring Boot application. It combines three annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan
2. **@RestController:**
    - This annotation is used to indicate that a class is a RESTful controller. It combines @Controller and @ResponseBody.
3. **@RequestMapping:**
    - This annotation is used to map web requests to specific handler methods. It can be applied at the class or method level.
4. **@Autowired:**
    - This annotation is used to automatically wire dependencies in Spring beans. It can be applied to fields, constructors, or methods.
5. **@Component:**
    - This annotation is used to indicate that a class is a Spring bean.
6. **@Service:**
    - This annotation is used to indicate that a class is a specialized type of Spring bean, typically used for business logic.
7. **@Repository:**
    - This annotation is used to indicate that a class is a specialized type of Spring bean, typically used for database access.
8. **@Configuration:**
    - This annotation is used to declare a class as a configuration class. It is typically used in combination with @Bean methods.
9. **@Value:**
    - This annotation is used to inject values from properties files or other sources into Spring beans.
10. **@EnableAutoConfiguration:**
    - This annotation is used to enable Spring Boot's auto-configuration mechanism. It automatically configures the application based on the classpath dependencies and properties.
11. **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping:**
    - These annotations are used to map specific HTTP methods to handler methods. They are shortcuts for @RequestMapping with the respective HTTP method.
12. **@PathVariable:**
    - This annotation is used to bind a method parameter to a path variable in a request URL.
13. **@RequestParam:**
    - This annotation is used to bind a method parameter to a request parameter.
14. **@RequestBody:**
    - This annotation is used to bind the request body to a method parameter. It is commonly used in RESTful APIs to receive JSON or XML payloads.
15. **@Qualifier:**

- This annotation is used to specify which bean to inject when multiple beans of the same type are available.

16. **@ConditionalOnProperty:**
    - This annotation is used to conditionally enable or disable a bean or configuration based on the value of a property.
17. **@Scheduled:**
    - This annotation is used to schedule the execution of a method at fixed intervals.
18. **@Cacheable, @CachePut, @CacheEvict:**
    - These annotations are used for caching method results. They allow you to cache the return value of a method, update the cache, or evict the cache, respectively.

1. **@SpringBootApplication:**
    - This annotation is used to mark the main class of a Spring Boot application. It combines three annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan. This annotation enables auto-configuration, component scanning, and configuration capabilities for the application.
2. **@ComponentScan:**
    - The @ComponentScan annotation is used to specify the base package(s) to scan for Spring components such as controllers, services, repositories, etc.
3. **@Configuration:**
    - The @Configuration annotation is used to indicate that a class declares one or more bean definitions. It is typically used in combination with @Bean to define Spring configuration classes.
4. **@EnableAutoConfiguration:**
    - The @EnableAutoConfiguration annotation allows Spring Boot to automatically configure the application based on the dependencies present in the classpath. It helps to reduce manual configuration by inferring configuration based on conventions and default settings.
5. **@RestController:**
    - The @RestController annotation is used to mark a class as a controller in a Spring MVC or Spring WebFlux application. It combines the @Controller and @ResponseBody annotations.
6. **@Controller:**
    - The @Controller annotation is used to mark a class as a controller in a Spring MVC or Spring WebFlux application. It handles HTTP requests and returns the view name or a response body.
7. **@Service:**
    - The @Service annotation is used to mark a class as a service component in the business logic layer. It is used to encapsulate business logic and perform operations such as data retrieval, manipulation, and validation.
8. **@Repository:**
    - The @Repository annotation is used to mark a class as a repository component in the data access layer. It is responsible for data access operations such as querying, saving, updating, and deleting data from a database.

9. **@Bean:**
    - The @Bean annotation is used to declare a method as a bean producer method within a configuration class. It indicates that the method returns an object that should be managed by the Spring container as a bean.
10. **@Autowired:**
    - The @Autowired annotation is used to inject dependencies automatically by type. It can be applied to constructors, fields, and methods.
11. **@Qualifier:**
    - The @Qualifier annotation is used to resolve ambiguous dependencies when there are multiple beans of the same type. It can be applied along with @Autowired to specify the exact bean to be injected.
12. **@Value:**
    - The @Value annotation is used to inject values from external sources, such as properties files, into variables.

2. Web Annotations:

1. **@RequestMapping:**
    - Purpose: Maps web requests to handler methods.
2. **@GetMapping:**
    - Purpose: Handles HTTP GET requests.
3. **@PostMapping:**
    - Purpose: Handles HTTP POST requests.
4. **@PutMapping:**
    - Purpose: Handles HTTP PUT requests.
5. **@DeleteMapping:**
    - Purpose: Handles HTTP DELETE requests.
6. **@PatchMapping:**
    - Purpose: Handles HTTP PATCH requests.
7. **@RequestBody:**
    - Purpose: Binds the body of a web request to a method parameter.
8. **@ResponseBody:**
    - Purpose: Indicates that the return value of a method should be used as the response body.
9. **@PathVariable:**
    - Purpose: Binds a method parameter to a URI template variable.
10. **@RequestParam:**
    - Purpose: Binds a method parameter to a query parameter in the URL.
11. **@RequestHeader:**
    - Purpose: Binds a method parameter to a header value in the HTTP request.
12. **@CookieValue:**
    - Purpose: Binds a method parameter to a cookie value in the HTTP request.
13. **@ModelAttribute:**
    - Purpose: Binds method parameters to model attributes before handling a request.

14. **@ResponseStatus:**
    - Purpose: Sets the HTTP response status for a controller method.
15. **@ExceptionHandler:**
    - Purpose: Handles exceptions thrown by controller methods.

3. Data Annotations:

1. **@Entity:**
    - Purpose: Marks a class as a JPA entity.
2. **@Table:**
    - Purpose: Specifies the table to be used for mapping the entity.
3. **@Id:**
    - Purpose: Marks a field as the primary key of an entity.
4. **@GeneratedValue:**
    - Purpose: Configures the automatic generation of primary key values.
5. **@Column:**
    - Purpose: Specifies the mapping for a database table column.
6. **@Transient:**
    - Purpose: Marks a field to be ignored by the persistence provider.
7. **@Repository:**
    - Purpose: Indicates that a class serves as a repository, typically for database access.
8. **@Service:**
    - Purpose: Marks a class as a service component in the business logic layer.
9. **@Transactional:**
    - Purpose: Defines the scope of a single database transaction.
10. **@PersistenceContext:**
    - Purpose: Injects a JPA EntityManager into a field, setter method, or constructor.
11. **@Autowired:**
    - Purpose: Automatically injects dependencies.
12. **@Query:**
    - Purpose: Declares a query to be executed in a repository method.
13. **@NamedQuery:**
    - Purpose: Defines a named query that can be referenced in repository methods.
14. **@Param:**
    - Purpose: Binds a method parameter to a named parameter in a query.
15. **@JoinTable:**
    - Purpose: Specifies the details of a join table for a many-to-many relationship.
16. **@JoinColumn:**
    - Purpose: Specifies the details of a join column for a mapping.

4. Validation Annotations:

1. **@Valid:**

    - Purpose: Indicates that validation should be cascaded to the properties of the annotated object.
2. **@NotNull:**
    - Purpose: Specifies that the property must not be null.
3. **@Size:**
    - Purpose: Specifies the size constraints for a property.
4. **@Min:**
    - Purpose: Specifies the minimum value for a numeric property.
5. **@Max:**
    - Purpose: Specifies the maximum value for a numeric property.
6. **@Email:**
    - Purpose: Validates that the property is a valid email address.
7. **@Pattern:**
    - Purpose: Specifies a regular expression pattern for the property.

5. Security Annotations:

1. **@EnableWebSecurity:**
    - Purpose: Enables Spring Security's web security features.
2. **@Configuration:**
    - Purpose: Indicates that a class is a configuration class.
3. **@EnableGlobalMethodSecurity:**
    - Purpose: Enables method-level security.
4. **@PreAuthorize:**
    - Purpose: Defines an expression that is evaluated before entering a method.
5. **@PostAuthorize:**
    - Purpose: Defines an expression that is evaluated after a method has been invoked.
6. **@Secured:**
    - Purpose: Secures a method using a simple role-based approach.
7. **@RolesAllowed:**
    - Purpose: Specifies the roles allowed to access a method.
8. **@EnableOAuth2Client:**
    - Purpose: Enables OAuth2 client support.
9. **@EnableResourceServer:**
    - Purpose: Enables a resource server.
10. **@EnableAuthorizationServer:**
    - Purpose: Enables an OAuth2 authorization server.

6. Testing Annotations:

1. **@RunWith:**
    - Purpose: Specifies the test runner class.
2. **@SpringBootTest:**
    - Purpose: Boots up the entire Spring application context for integration testing.
3. **@WebMvcTest:**

    - Purpose: Restricts the application context to a specific layer, typically for testing controllers.
4. **@DataJpaTest:**
    - Purpose: Configures a test for JPA-based tests.
5. **@RestClientTest:**
    - Purpose: Configures a test for RestTemplate-based tests.
6. **@MockBean:**
    - Purpose: Mocks a bean in the Spring application context.
7. **@AutoConfigureMockMvc:**
    - Purpose: Auto-configures the MockMvc instance.
8. **@Test:**
    - Purpose: Identifies a method as a test method.
9. **@Before:**
    - Purpose: Executed before each test method.
10. **@After:**
    - Purpose: Executed after each test method.
11. **@BeforeEach:**
    - Purpose: Executed before each test method in JUnit 5.
12. **@AfterEach:**
    - Purpose: Executed after each test method in JUnit 5.
13. **@BeforeAll:**
    - Purpose: Executed once before all test methods in JUnit 5.
14. **@AfterAll:**
    - Purpose: Executed once after all test methods in JUnit 5.
15. **@DisplayName:**
    - Purpose: Specifies a custom display name for a test class or method.
16. **@Disabled:**
    - Purpose: Disables a test class or method.
17. **@ParameterizedTest:**
    - Purpose: Indicates that a method is a parameterized test.
18. **@ValueSource:**
    - Purpose: Provides a single value to a parameterized test.
19. **@CsvSource:**
    - Purpose: Provides CSV values to a parameterized test.
20. **@ExtendWith:**
    - Purpose: Used to register extensions in JUnit 5.

7. Caching Annotations:

1. **@EnableCaching:**
    - Purpose: Enables Spring's caching support for the application.
2. **@Cacheable:**
    - Purpose: Indicates that the result of a method can be cached.
3. **@CachePut:**
    - Purpose: Updates the cache with the result of the method execution, regardless of the cache hit.
4. **@CacheEvict:**

- Purpose: Evicts one or more entries from the cache.

5. **@Caching:**
   - Purpose: Groups multiple cache-related annotations on a method.

8. Scheduling Annotations:

1. **@EnableScheduling:**
   - Purpose: Enables Spring's scheduled task execution capability.
2. **@Scheduled:**
   - Purpose: Defines when a method should be executed at fixed intervals.

9. Messaging Annotations:

1. **@EnableJms:**
   - Purpose: Enables the JMS (Java Message Service) features in a Spring application.
2. **@JmsListener:**
   - Purpose: Defines a method to act as a JMS message listener.
3. **@SendTo:**
   - Purpose: Specifies the destination for sending a response from a method annotated with @MessageMapping.
4. **@MessageMapping:**
   - Purpose: Handles STOMP over WebSocket messages.
5. **@Payload:**
   - Purpose: Extracts the payload of a message in a method parameter.
6. **@Header:**
   - Purpose: Extracts header values from a message in a method parameter.

10. Aspect-Oriented Programming (AOP) Annotations:

1. **@Aspect:**
   - Purpose: Declares a class as an aspect, which encapsulates cross-cutting concerns.
2. **@Pointcut:**
   - Purpose: Defines a pointcut expression that identifies the join points where advice should be applied.
3. **@Before:**
   - Purpose: Specifies advice to be executed before a join point.
4. **@After:**
   - Purpose: Specifies advice to be executed after a join point (finally).
5. **@AfterReturning:**
   - Purpose: Specifies advice to be executed after a join point successfully completes.
6. **@AfterThrowing:**
   - Purpose: Specifies advice to be executed if a join point throws an exception.
7. **@Around:**

- Purpose: Combines the functionality of all advice types, allowing manipulation of the method invocation.

11. Actuator Annotations:

1. **@EnableActuator:**
   - Purpose: Enables Spring Boot Actuator, which provides production-ready features to help monitor and manage the application.
2. **@Endpoint:**
   - Purpose: Marks a class as an actuator endpoint.
3. **@RestControllerEndpoint:**
   - Purpose: Combines @RestController and @Endpoint, making it easy to create web-based actuator endpoints.
4. **@ReadOperation:**
   - Purpose: Marks a method as a read operation for an actuator endpoint.
5. **@WriteOperation:**
   - Purpose: Marks a method as a write operation for an actuator endpoint.
6. **@DeleteOperation:**
   - Purpose: Marks a method as a delete operation for an actuator endpoint.

12. Configuration Properties Annotations:

1. **@ConfigurationProperties:**
   - **Purpose:** Binds and validates external configuration properties to a JavaBean.
2. **@ConstructorBinding:**
   - **Purpose:** Indicates that constructor binding should be used for immutable objects.
3. **@Validated:**
   - **Purpose:** Activates validation of `@ConfigurationProperties` annotated classes.

13. Internationalization and Localization:

1. **@EnableMessageSource:**
   - **Purpose:** Enables a message source for resolving messages.
2. **@EnableWebMvc:**
   - **Purpose:** Enables Spring MVC configuration with additional features.
3. **@LocaleResolver:**
   - **Purpose:** Specifies a strategy for resolving user locale information.
4. **@MessageBundle:**
   - **Purpose:** Represents a resource bundle for messages.
5. **@MessageSource:**
   - **Purpose:** Provides a way to interact with a message source.

14. Logging and Monitoring:

1. **@Slf4j:**
    - **Purpose:** Lombok annotation to generate a logger field for a class.
2. **@Log4j2:**
    - **Purpose:** Lombok annotation to generate a logger using Log4j2.
3. **@Log:**
    - **Purpose:** Lombok annotation to generate a logger using the standard logging framework.
4. **@Timed:**
    - **Purpose:** Micrometer annotation to record the duration of a method invocation.
5. **@Counted:**
    - **Purpose:** Micrometer annotation to record the number of times a method is invoked.
6. **@ExceptionMetered:**
    - **Purpose:** Micrometer annotation to record the rate and time taken for a method to throw an exception.

15. Data Validation:

1. **@Validated:**
    - **Purpose:** Activates validation of annotated classes.
2. **@Valid:**
    - **Purpose:** Marks a property, method parameter, or method return type for validation.
3. **@NotNull:**
    - **Purpose:** Asserts that the annotated element must not be null.
4. **@NotBlank:**
    - **Purpose:** Asserts that the annotated string must not be blank (not null and not empty).
5. **@Email:**
    - **Purpose:** Asserts that the annotated string is a valid email address.
6. **@Size:**
    - **Purpose:** Asserts that the annotated element size is between the specified boundaries.
7. **@Pattern:**
    - **Purpose:** Asserts that the annotated string matches the regular expression pattern.
8. **@Positive:**
    - **Purpose:** Asserts that the annotated number is positive.
9. **@PositiveOrZero:**
    - **Purpose:** Asserts that the annotated number is positive or zero.
10. **@Negative:**
    - **Purpose:** Asserts that the annotated number is negative.
11. **@NegativeOrZero:**
    - **Purpose:** Asserts that the annotated number is negative or zero.

16. GraphQL Annotations:

1. **@GraphQLApi:**
    - **Purpose:** Marks a class as a GraphQL API.
2. **@GraphQLQuery:**
    - **Purpose:** Marks a method as a GraphQL query.
3. **@GraphQLMutation:**
    - **Purpose:** Marks a method as a GraphQL mutation.
4. **@GraphQLSubscription:**
    - **Purpose:** Marks a method as a GraphQL subscription.
5. **@GraphQLArgument:**
    - **Purpose:** Defines an argument for a GraphQL operation.
6. **@GraphQLContext:**
    - **Purpose:** Injects the GraphQL context into a method.
7. **@GraphQLNonNull:**
    - **Purpose:** Marks a field or method parameter as non-nullable.
8. **@GraphQLInputType:**
    - **Purpose:** Defines an input type for a GraphQL operation.
9. **@GraphQLType:**
    - **Purpose:** Marks a class as a GraphQL type.

17. Integration Annotations:

1. **@IntegrationComponentScan:**
    - **Purpose:** Specifies the base packages for scanning components in an integration application.
2. **@MessagingGateway:**
    - **Purpose:** Marks an interface as a messaging gateway, providing a higher-level abstraction for sending and receiving messages.
3. **@Transformer:**
    - **Purpose:** Marks a method as a transformer within a messaging flow.
4. **@Splitter:**
    - **Purpose:** Marks a method as a splitter within a messaging flow.
5. **@Aggregator:**
    - **Purpose:** Marks a method as an aggregator within a messaging flow.
6. **@ServiceActivator:**
    - **Purpose:** Marks a method as a service activator within a messaging flow.
7. **@InboundChannelAdapter:**
    - **Purpose:** Marks a method as an inbound channel adapter, responsible for generating messages.
8. **@OutboundChannelAdapter:**
    - **Purpose:** Marks a method as an outbound channel adapter, responsible for sending messages.
9. **@Router:**
    - **Purpose:** Marks a method as a router within a messaging flow.
10. **@BridgeTo:**
    - **Purpose:** Bridges a message from one channel to another within a messaging flow.

18. Flyway Database Migrations:

1. **@FlywayTest:**
    - **Purpose:** Marks a test class or method for integration testing with Flyway database migrations.
2. **@FlywayTestExtension:**
    - **Purpose:** Provides extension support for Flyway database migrations in JUnit 5.
3. **@FlywayTestExtension.Test:**
    - **Purpose:** Marks a test method as part of the Flyway test extension.
4. **@FlywayTestExtension.BeforeMigration:**
    - **Purpose:** Marks a method to be executed before Flyway migrations in a test.
5. **@FlywayTestExtension.AfterMigration:**
    - **Purpose:** Marks a method to be executed after Flyway migrations in a test.

19. JUnit 5 Annotations:

1. **@ExtendWith:**
    - **Purpose:** Used to register extensions in JUnit 5.
2. **@TestInstance:**
    - **Purpose:** Configures the test instance lifecycle mode.
3. **@TestTemplate:**
    - **Purpose:** Marks a method as a template for dynamic tests.
4. **@DisplayNameGeneration:**
    - **Purpose:** Configures a custom display name generator for test classes and methods.
5. **@Nested:**
    - **Purpose:** Allows the grouping of tests within an outer test class.
6. **@Tag:**
    - **Purpose:** Tags a test class or method for filtering during test execution.
7. **@DisabledOnOs:**
    - **Purpose:** Disables a test or test condition on specific operating systems.
8. **@EnabledOnOs:**
    - **Purpose:** Enables a test or test condition on specific operating systems.
9. **@DisabledIf:**
    - **Purpose:** Disables a test or test condition based on a condition evaluated by a script or method.
10. **@EnabledIf:**
    - **Purpose:** Enables a test or test condition based on a condition evaluated by a script or method.

20. API Documentation Annotations:

1. **@Api:**
    - **Purpose:** Provides high-level information about the API.
2. **@ApiOperation:**
    - **Purpose:** Describes an operation or endpoint in the API.

3. **@ApiParam:**
    - **Purpose:** Describes a parameter in an API operation.
4. **@ApiModel:**
    - **Purpose:** Describes a data model used in the API.
5. **@ApiModelProperty:**
    - **Purpose:** Describes a property of a data model.

21. Exception Handling Annotations:

1. **@ControllerAdvice:**
    - **Purpose:** Defines global exception handling for controllers.
2. **@ExceptionHandler:**
    - **Purpose:** Defines a method to handle specific exceptions.

22. GraphQL Annotations:

1. **@GraphQLSchema:**
    - **Purpose:** Defines the GraphQL schema for a Spring Boot application.
2. **@GraphQLQueryResolver:**
    - **Purpose:** Defines a class as a GraphQL query resolver.
3. **@GraphQLMutationResolver:**
    - **Purpose:** Defines a class as a GraphQL mutation resolver.
4. **@GraphQLSubscriptionResolver:**
    - **Purpose:** Defines a class as a GraphQL subscription resolver.
5. **@GraphQLResolver:**
    - **Purpose:** Defines a class as a generic resolver for GraphQL.

23. Server-Sent Events (SSE) Annotations:

1. **@SseEmitter:**
    - **Purpose:** Creates an SSE endpoint for server-sent events.
2. **@SseEventSink:**
    - **Purpose:** Injects an SSE event sink into a method parameter.

24. WebFlux Annotations:

1. **@RestController:**
    - **Purpose:** Creates a RESTful controller in a WebFlux application.
2. **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping:**
    - **Purpose:** Maps HTTP methods to handler methods in a WebFlux application.

25. Micrometer Metrics Annotations:

1. **@Timed:**

    - **Purpose:** Measures the execution time of a method.
2. **@Counted:**
    - **Purpose:** Counts the number of times a method is invoked.
3. **@Gauge:**
    - **Purpose:** Exposes a method as a gauge metric.
4. **@ExceptionMetered:**
    - **Purpose:** Measures the rate of exceptions thrown by a method.