Spring Boot Annotations Cheat Sheet

Core Annotations

- **@SpringBootApplication**
Combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`. The main entry point for Spring Boot applications.

- **@Component**
Generic stereotype for any Spring-managed component (class).

- **@Service**
Specialization of `@Component` for service-layer classes.

- **@Repository**
Specialization of `@Component` for persistence-layer classes. Translates database-related exceptions.

- **@Controller**
Used to define a web controller in Spring MVC.

- **@RestController**
Combines `@Controller` and `@ResponseBody`, indicating the class handles REST requests.

Dependency Injection Annotations

- **@Autowired**
Marks a constructor, field, or method for dependency injection by type.

- **@Qualifier**
Specifies the name of a bean to resolve ambiguity when multiple candidates exist.

- **@Primary**
Marks a bean as the primary candidate to be injected when multiple beans of the same type are present.

Transaction Management Annotations

@ Transactional
Marks a method or class to execute within a database transaction

Spring Boot Web Annotations

- **@RequestMapping("/path")**
Maps HTTP requests to handler methods. Can be used on classes and methods.

- **@GetMapping("/path")**
Shortcut for `@RequestMapping(method = RequestMethod.GET)`.

- **@PostMapping("/path")**
Shortcut for `@RequestMapping(method = RequestMethod.POST)`.

- **@PutMapping("/path")**
Shortcut for `@RequestMapping(method = RequestMethod.PUT)`.

- **@DeleteMapping("/path")**
Shortcut for `@RequestMapping(method = RequestMethod.DELETE)`.

- **@PathVariable**
Binds a method parameter to a URI template variable.

- **@RequestParam**
Extracts query parameters, form parameters, or HTTP request parameters.

- **@RequestBody**
Maps the entire request body to a Java object.

Validation and Security Annotations

- **@Valid**
Triggers validation on method parameters or fields, leveraging JSR-303 (Bean Validation API).

- **@Secured("ROLE_ADMIN")**
Secures methods based on roles defined.

- **@PreAuthorize("hasRole('ROLE_USER')")**
Secures methods using SpEL (Spring Expression Language) conditions.

Configuration Annotations

- **@Configuration**
Indicates that the class can be used by Spring IoC as a source of bean definitions.

- **@Bean**
Used to define a bean that the Spring container manages.

- **@Value("${property}")**
Injects a value from properties or application YAML.

- **@PropertySource("classpath**

.properties")
Defines an external property file to be used.

Data & JPA Annotations

- **@Entity:** Specifies that a class is an entity in JPA.
- **@Table:** Customizes the table mapping for an entity.
- **@Id:** Marks the primary key of an entity.
- **@GeneratedValue:** Specifies how the primary key is generated.
- **@OneToMany, @ManyToOne, @ManyToMany:** Defines relationships between entities.

Scheduling and Caching Annotations

- **@Scheduled**
Defines a scheduled task using a cron expression or fixed delay.

- **@EnableCaching**
Enables caching for the application.

Testing Annotations

- **@SpringBootTest**
Loads the complete Spring context for integration tests.
- **@MockBean**
Defines a mock instance for injection into the Spring context during testing.
- **@WebMvcTest**
Tests only web-related components, such as controllers.

Miscellaneous Annotations

- **@Configuration:** Indicates a configuration class that declares beans.
- **@Bean:** Marks a method as a bean provider.
- **@PropertySource:** Specifies a properties file to load.
- **@EnableScheduling:** Enables scheduling for methods annotated with `@Scheduled`

Hibernate Annotations Cheat Sheet

1. Core Entity Annotations

- **@Entity:** Declares a class as an entity.
- **@Table(name = "table_name"):** Specifies the table name.
- **@Id:** Marks a primary key.
- **@GeneratedValue(strategy = GenerationType.AUTO):** Primary key auto-generation.

2. Relationships

- **@OneToOne, @OneToMany, @ManyToOne, @ManyToMany:** Maps relationships.
- **@JoinColumn:** Defines foreign keys in relations.

3. Querying

- **@NamedQuery(name = "findByName", query = "FROM Entity WHERE name = :name"):** Named query example.

Apache Kafka Cheat Sheet

Basic Producer Code

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

Producer<String, String> producer = new KafkaProducer<>(props);
producer.send(new ProducerRecord<>("topic", "key", "value"));
```

Basic Consumer Code

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test-group");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

Consumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("topic"));

while (true) {
ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
for (ConsumerRecord<String, String> record : records) {
System.out.println(record.value());
}
}
```