Site: https://www.javagists.com/spring-boot-cheatsheet

***@Autowired:*** Annotation @Autowired is used to inject object dependency implicitly for a constructor, field or method. This is known as “autowired by type” since the object to be injected is discovered by its type. The items declared @Autowired need not have to be public.

***@Configurable:*** Used on classes to inject properties of domain objects. Types whose properties are injected without being instantiated by Spring can be declared with @Configurable annotation.

***@Qualifier:*** It can be used to create more than one bean of the same type and wire only one of the types with a property. It provides greater control on the dependency injection process and can be used with @Autowired annotation.

***@Required:*** Used to mark class members that are mandatory. The Spring auto-configuration fails if a particular property specified with this annotation cannot be injected.

***@ComponentScan:*** Make Spring scan the package for the @Configuration classes.

***@Configuration:*** It is used on classes that define beans.

***@Bean:*** It indicates that a method produces a bean which will be managed by the Spring container.

***@Lazy:*** Makes a @Bean or @Component to be initialized only if it is requested.

***@Value:*** It is used to inject values into a bean’s attribute from a property file. @Value annotation indicates a default value expression for the field or parameter.

***@SpringBootApplication:*** This annotation is used to qualify the main class for a Spring Boot project. The class used with this annotation must be present in the base path. @SpringBootApplication scans for sub-packages by doing a component scan.

***@EnableAutoConfiguration:*** Based on class path settings, property settings, new beans are added by Spring Boot by using this annotation.

***@Controller:*** Allows detection of component classes in the class path automatically and register bean definitions for the classes automatically.

***@RestController:*** Used in controllers that will behave as RESTful resources. @RestController is a convenience annotation that combines @Controller and @ResponseBody.

***@ResponseBody:*** Makes Spring to convert the returned object to a response body. This is useful for classes exposed as RESTful resources.

***@RequestMapping:*** Used to map web requests to specific handler classes and methods, based on the URI.

***@RequestParam:*** This annotation is used to bind request parameters to a method parameter in your controller.

***@PathVariable:*** This annotation binds the placeholder from the URI to the method parameter and can be used when the URI is dynamically created or the value of the URI itself acts as a parameter.

***@AfterTransaction:*** Annotation used to identify which method needs to be invoked after a transaction is completed.

***@BeforeTransaction:*** Used to identify the method to be invoked before a transaction starts executing.

***@ContextConfiguration:*** Declares the annotated classes which will be used to load the context for the test. The location of the configuration file must be provided to Spring.

***@DirtiesContext:*** This annotation indicates the test(s) modify or corrupt the SpringApplicationContext and that it should be closed. Hence, context is reloaded before the next test is executed.

***@ExpectedException:*** The test method is expected to throw a particular exception, else the test fails.

***@WebAppConfiguration:*** Used to create web version of the application context.

***@Repeat:*** Specifies the test method to be executed multiple times.

***@Transactional:*** Describes transaction attributes on a method or class.

***@Rollback:*** Indicates if the transaction of a test method must be rolled back after the execution of the test completed.

***@Commit:*** Indicates that the transaction of a test method must be committed after the execution of the test completed.

***@Timed:*** Indicates the time limit for the test method. If the test has not completed execution before the time expires, the test fails.

***@TestPropertySource:*** Annotation specifies the property sources for the test class.

***@Sql:*** Annotation declares a test class/method to run SQL scripts against a database.

***@JsonInclude:*** This annotation is used to exclude properties or fields of a class under certain conditions. This is defined using the JsonInclude.Include enum. This enum contains constants, that determine whether or not to exclude the property.

The constants are:

o ALWAYS

o NON\_DEFAULT

o NON\_EMPTY

o NON\_NULL

***@JsonIgnore:*** annotation marks a field in a POJO to be ignored by Jackson during serialization and deserialization. Jackson ignores the field in both JSON serialization and deserialization.

***@JsonIgnoreProperties:*** This annotation is used at the class level to ignore fields during serialization and deserialization.

***@JsonAutoDetect:*** This annotation is used at the class level to tell Jackson to override the visibility of the properties of a class during serialization and deserialization.

***@JsonIgnoreType:*** This annotation is used to mark a class to be ignored during serialization and deserialization. It marks all the properties of the class to be ignored while generating and reading JSON.

***@JsonCreator:*** This annotation tells Jackson that the JSON properties can be mapped to the fields of a constructor of the POJO. This is helpful when the JSON properties do not match with the names of the Java object field names. The @JsonCreator annotation can be used where @JsonSetter cannot be used.

***@JsonProperty:*** This annotation is used to map property names with JSON keys during serialization and deserialization. By default, if you try to serialize a POJO, the generated JSON will have keys mapped to the fields of the POJO. If you want to override this behavior, you can use the @JsonProperty annotation on the fields. It takes a String attribute that specifies the name that should be mapped to the field during serialization.

***@Component:*** Spring Component annotation is used to denote a class as Component. It means that Spring framework will autodetect these classes for dependency injection when annotation-based configuration and classpath scanning is used.

***@Service:*** It hold the business logic and call methods in the repository layer.

***@Scheduled:*** annotation is used for task scheduling. The trigger information needs to be provided along with this annotation. You can use the properties fixedDelay/fixedRate/cron to provide the triggering information. where –

1. fixedRate makes Spring run the task on periodic intervals even if the last invocation may be still running.

2. fixedDelay specifically controls the next execution time when the last execution finishes.

3. cron is a feature originating from Unix cron utility and has various options based on your requirements.

***@ActiveProfiles:*** is a class-level annotation that is used to activate profiles while loading ApplicatonContext in Spring integration test @ActiveProfiles has following elements.

**profiles**: Specify profiles to activate.

**resolver**: Specify ActiveprofilesResolver to activate profiles programmatically.

**value**: It is alias for profiles element.

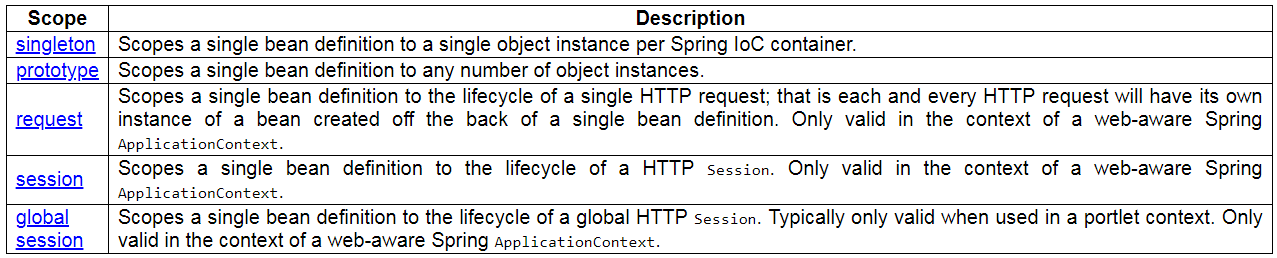
**inheritProfiles**: Boolean value to decide whether active profiles should be inherited from superclass or not. The default value is true.

***@Resource:*** It means get me a known resource by name. The name is extracted from the name of the annotated setter or field, or it is taken from the name-Parameter.

***@Qualifier:*** It defines what beans should be used to @Autowire a field or parameter.

***@Required:*** annotation applies to bean property setter methods and it indicates that the affected bean property must be populated in XML configuration file at configuration time. Otherwise, the container throws a BeanInitializationException exception. Following is an example to show the use of @Required annotation.

***@Scope:***



***@SessionAttributes:*** Specified that a model attribute should be stored in the session.

***@Transactional:*** At a high level, when a class declares @Transactional on itself or its members, Spring creates a proxy that implements the same interface(s) as the class you’re annotating. In other words, Spring wraps the bean in the proxy and the bean itself has no knowledge of it. A proxy provides a way for Spring to inject behaviors before, after, or around method calls into the object being proxied.

***@EnableConfigServer:*** It turns your application into a server other apps can get their configuration from.