**@SpringBootApplication**

Many Spring Boot developers always have their main class annotated with **@Configuration**, **@EnableAutoConfiguration** and **@ComponentScan**. Since these annotations are so frequently used together (especially if you follow the best practices above), Spring Boot provides a convenient @SpringBootApplication alternative.

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes

**@Controller and @RestController**

Spring 4.0 – Introduced RestController annotation

Spring RestController annotation is a convenience annotation that is itself annotated with **@Controller** and **@ResponseBody**  which eliminates the need to annotate every request handling method of the controller class with the *@ResponseBody* annotation.

**Spring MVC @Controller**

We can annotate classic controllers with the **@Controller** annotation. This is simply a specialization of the **@Component** class, which allows us to auto-detect implementation classes through the class path scanning.

We typically use **@Controller** in combination with a **@RequestMapping** annotation for request handling methods.

**Let's see a quick example of the Spring MVC controller:**

**@Controller**

@RequestMapping("books")

public class SimpleBookController {

@GetMapping("/{id}", produces = "application/json")

public **@ResponseBody** Book getBook(@PathVariable int id) {

return findBookById(id);

}

private Book findBookById(int id) {

// ...

}

}

We annotated the request handling method with @ResponseBody. This annotation enables automatic serialization of the return object into the HttpResponse.

**Spring MVC @RestController**

@RestController is a specialized version of the controller. It includes the @Controller and @ResponseBody annotations, and as a result, simplifies the controller implementation

@**RestController**

@RequestMapping("books-rest")

public class SimpleBookRestController {

@GetMapping("/{id}", produces = "application/json")

public Book getBook(@PathVariable int id) {

return findBookById(id);

}

private Book findBookById(int id) {

// ...

}

}

The controller is annotated with the **@RestController** annotation; therefore, the **@ResponseBody** isn't required.

**Every request handling method of the controller class automatically serializes return objects into HttpResponse.**

**@GetMapping** & **@RequestMapping**

There is no difference in semantic.

Specifically, @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

**RequestMapping can be used at class level:**

This annotation can be used both at the class and at the method level. In most cases, at the method level applications will prefer to use one of the HTTP method specific variants @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, or @PatchMapping.

while **GetMapping** only applies to method:

Annotation for mapping HTTP GET requests onto specific handler methods.

**@CrossOrigin**

Cross-Origin Resource Sharing (CORS) is a security concept that allows restricting the resources implemented in web browsers. It prevents the JavaScript code producing or consuming the requests against different origin.

**@Autowired**

@Autowired annotation is used for dependency injection. In spring boot application, all loaded beans are eligible for auto wiring to another bean. The annotation @Autowired in spring boot is used to auto-wire a bean into another bean.

**@RequestBody**

used to send and receive data via the REST API.

The @RequestBody annotation is applicable to handler methods of Spring controllers. This annotation indicates that Spring should deserialize a request body into an object. This object is passed as a handler method parameter.

**ResponseEntity:**

**ResponseEntity** is meant to represent the entire HTTP response. You can control anything that goes into it: status code, headers, and body.

@**ResponseBody** is a marker for the HTTP response body and @ResponseStatus declares the status code of the HTTP response.

@**ResponseStatus** isn't very flexible. It marks the entire method so you have to be sure that your handler method will always behave the same way. And you still can't set the headers. You'd need the HttpServletResponse.

Basically, ResponseEntity lets you do more.

**@Value**

Spring's @Value annotation provides a convenient way to inject property values into components. It's also quite useful to provide sensible defaults for cases where a property may not be present.

syntax

@Value("${some.key:my default value}")

private String stringWithDefaultValue;

**@Configuration annotation**

Spring **@Configuration** annotation helps in Spring annotation based configuration. @Configuration annotation indicates that a class declares one or more **@Bean** methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

Since spring 2, we were writing our bean configurations to xml files. But Spring 3 gave the freedom to move bean definitions out of xml files. we can give bean definitions in Java files itself. This is called Spring Java Config feature (using @Configuration annotation).

Spring @Configuration annotation usage

Use @Configuration annotation on top of any class to declare that this class provides one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at **runtime**.

**AppConfig.java**

@Configuration

public class AppConfig {

@Bean(name="demoService")

public DemoClass service()

{

}

}

Create spring beans

DemoManager.java and DemoManagerImpl.java

public interface DemoManager {

public String getServiceName();

}

public class DemoManagerImpl implements DemoManager

{

@Override

public String getServiceName()

{

return "My first service with Spring 3";

}

}

**Spring configuration class with @Configuration annotation**

@Configuration

public class ApplicationConfiguration {

@Bean(name="demoService")

public DemoManager helloWorld()

{

return new DemoManagerImpl();

}

}

Main class:

public class VerifySpringCoreFeature

{

public static void main(String[] args)

{

ApplicationContext context = new AnnotationConfigApplicationContext(ApplicationConfiguration.class);

DemoManager obj = (DemoManager) context.getBean("demoService");

System.out.println( obj.getServiceName() );

}

}

**@Bean**

Spring @Bean annotation tells that a method produces a bean to be managed by the Spring container. It is a method-level annotation. During Java configuration ( @Configuration ), the method is executed and its return value is registered as a bean within a BeanFactory

**@PropertySource** - https://zetcode.com/spring/propertysource/

**Exception:**

**@ControllerAdvice – class Level**

@ControllerAdvice is a specialization of the @Component annotation which allows to handle exceptions across the whole application in one global handling component.

@ExceptionHandler - Method Level