Casdoc

Interactive Code Examples for Software Documentation

Mathieu Nassif, McGill University

Create an Application class

The Spring Initializr creates a simple application class for you. However, in this case, it is too simple. You need to modify the application class to match the following listing (from `src/main/java/com/example/springboot/Application.java`):

```
package com.example.springboot;

import java.util.Arrays;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Application {

	public static void main(String[] args) {
		SpringApplication.run(Application.class, args);
	}

	@Bean
	public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
		return args -> {

			System.out.println("Let's inspect the beans provided by Spring Boot:");

			String[] beanNames = ctx.getBeanDefinitionNames();
			Arrays.sort(beanNames);
			for (String beanName : beanNames) {
				System.out.println(beanName);
			}

		};
	}

}
```

`@SpringBootApplication` is a convenience annotation that adds all of the following:

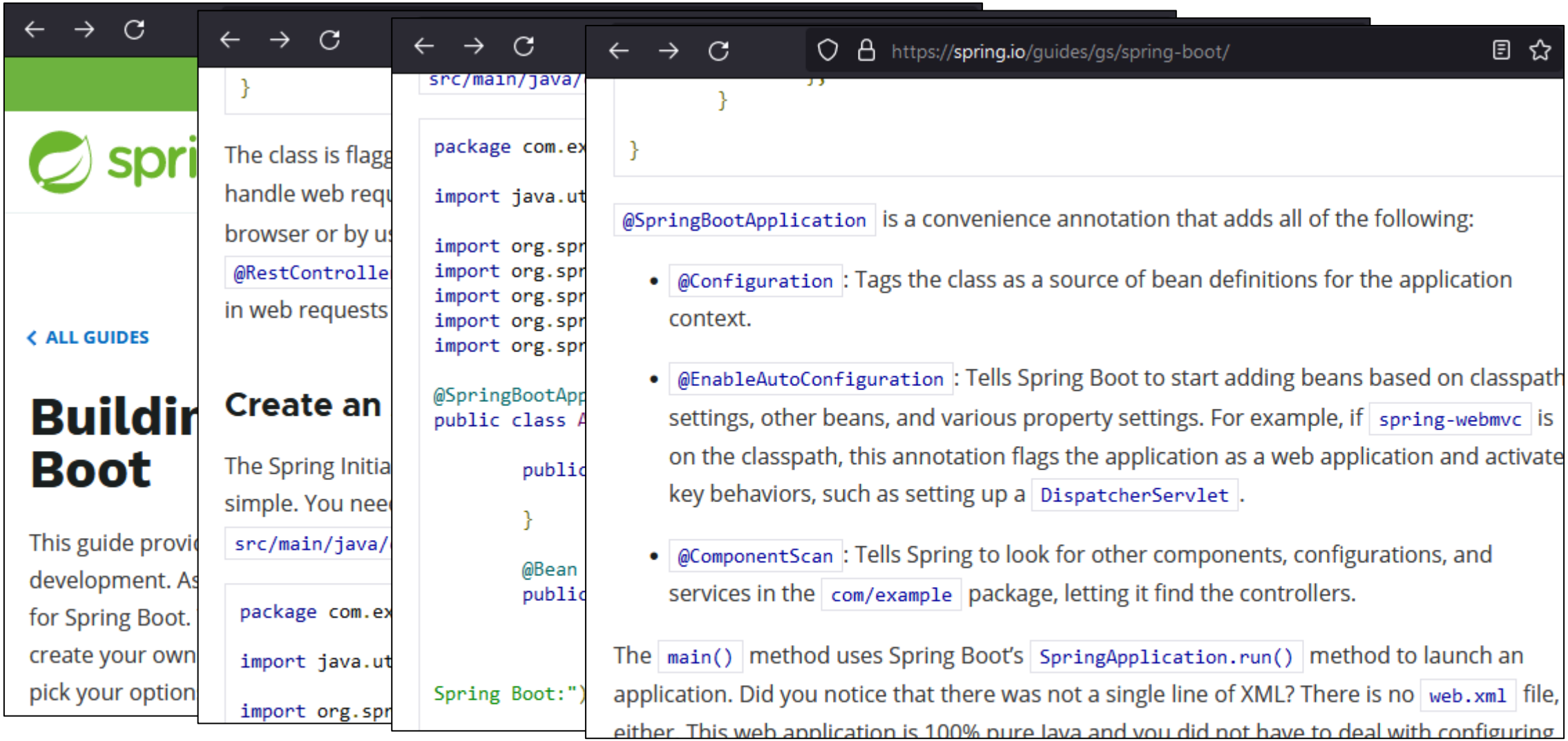
- `@Configuration`: Tags the class as a source of bean definitions for the application context.
- `@EnableAutoConfiguration`: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings. For example, if `spring-webmvc` is on the classpath, this annotation flags the application as a web application and activates key behaviors, such as setting up a `DispatcherServlet`.
- `@ComponentScan`: Tells Spring to look for other components, configurations, and services in the `com/example` package, letting it find the controllers.

The `main()` method uses Spring Boot's `SpringApplication.run()` method to launch an application. Did you notice that there was not a single line of XML? There is no `web.xml` file, either. This web application is 100% pure Java and you did not have to deal with configuring any plumbing or infrastructure.

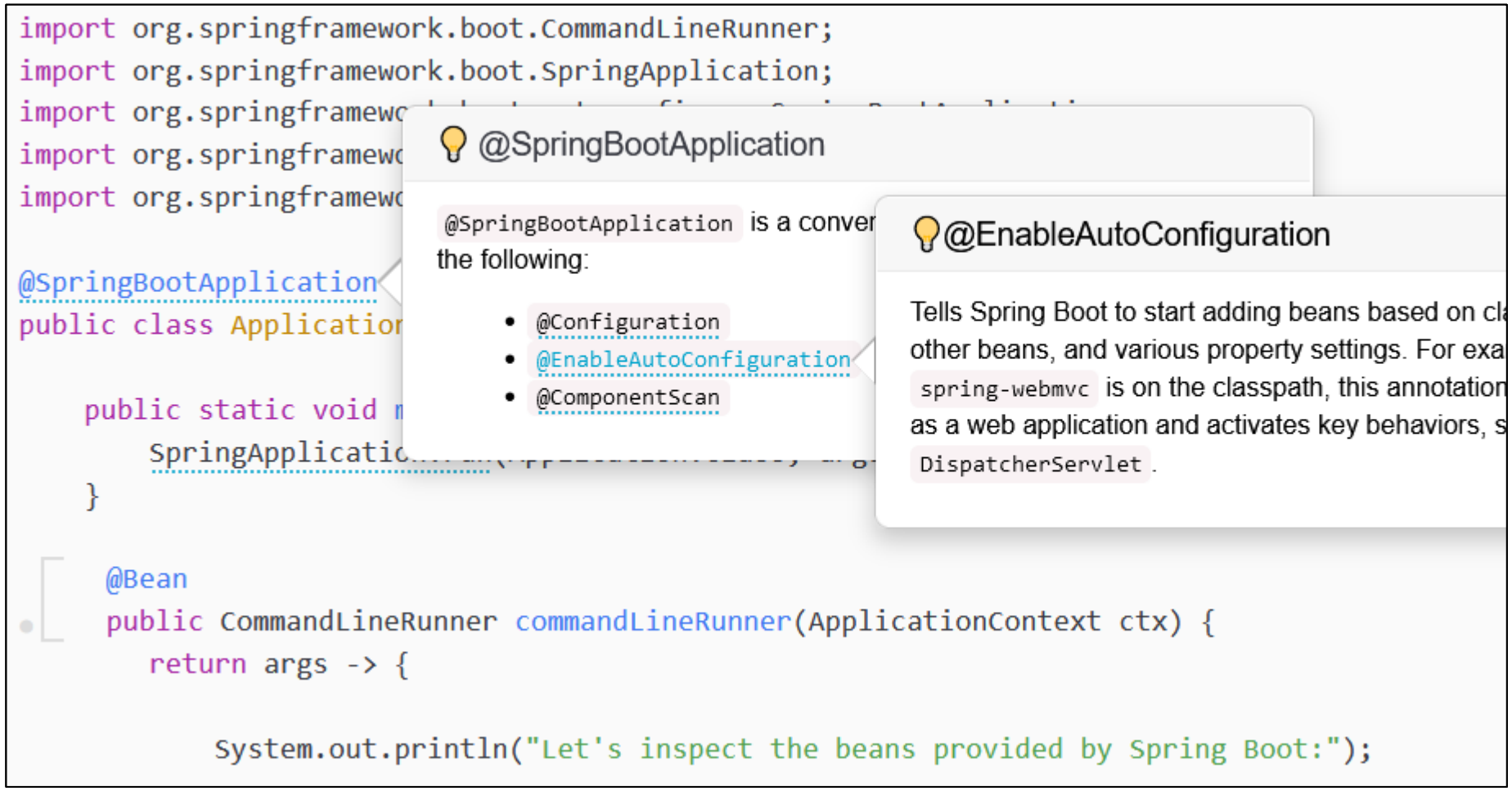
There is also a `CommandLineRunner` method marked as a `@Bean`, and this runs on start up. It retrieves all the beans that were created by your application or that were automatically added by Spring Boot. It sorts them and prints them out.

```java
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {

            System.out.println("Let's inspect the beans provided by Spring Boot:");

            String[] beanNames = ctx.getBeanDefinitionNames();
            Arrays.sort(beanNames);
            for (String beanName : beanNames) {
                System.out.println(beanName);
            }

        };
    }
}
```

Most documentation is static and linear



Casdoc adds dynamic annotations to code examples



Demo

- https://www.cs.mcgill.ca/~mnassif/casdoc/
- https://cs.mcgill.ca/~martin/designbook/

Creating Casdoc documents is no harder than commenting code!

```
/*?
 * Keyword: @SpringBootApplication
 * `@SpringBootApplication` is a convenience annotation that adds
 * all of the following:
 * - `@Configuration`
 * - `@EnableAutoConfiguration`
 * - `@ComponentScan`
 *
 * Internal: @Configuration
 * @SpringBootApplication
 * Tags the class as a source of bean definitions for
 * the application context.
 *
 * Internal: @EnableAutoConfiguration
 * @SpringBootApplication
 * Tells Spring Boot to start adding beans based on classpath
 * settings, other beans, and various property settings. For example,
 * if `spring-webmvc` is on the classpath, this annotation flags the
 * application as a web application and activates key behaviors,
 * such as setting up a `DispatcherServlet`.
 *
 * Internal: @ComponentScan
 * @SpringBootApplication
 * Tells Spring to look for other components, configurations,
 * and services in the `com/example` package, letting it find
 * the controllers.
 */
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        /*?
         * Keyword: SpringApplication.run
         * The `main()` method uses Spring Boot's `SpringApplication.ru
         * Did you notice that there was not a single line of XML? Ther
```

```
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframew
import org.springframew
import org.springframew

@SpringBootApplication
public class Applicatio

    public static void m
        SpringApplicatio
    }

    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {

            System.out.println("Let's inspect the beans provided by Spring Boot:");
```

@SpringBootApplication

`@SpringBootApplication` is a conver
the following:

- `@Configuration`
- `@EnableAutoConfiguration`
- `@ComponentScan`

@EnableAutoConfiguration

Tells Spring Boot to start adding beans based on cl
other beans, and various property settings. For exa
`spring-webmvc` is on the classpath, this annotation
as a web application and activates key behaviors, s
`DispatcherServlet`.

```java
package ca.mcgill.cs.casdoc;

public class Casdoc {
    public static void main(String[] args) {
        System.out.println("Hello, Casdoc");
    }
}
```

What's next?

- Improve prototype
- Understand developer behavior
- Rethink on-demand documentation

Try it yourself!

https://www.cs.mcgill.ca/~martin/casdoc/

Mathieu Nassif
McGill University
mnassif@cs.mcgill.ca