@Controller vs @Component

In Spring, `@Component` classes and `@Configuration` classes are both used to define beans, but they are processed differently by the Spring container.

1. **`@Component` Classes**:
   - These are regular Spring-managed components annotated with `@Component`, `@Service`, `@Repository`, or `@Controller`.
   - When Spring processes a `@Component` class, it registers the class as a bean and manages its lifecycle, but it does not modify the bytecode of the class itself.
   - As a result, the methods inside a `@Component` class annotated with `@Bean` are not enhanced with CGLIB (Code Generation Library). They are treated as regular methods.

2. **`@Configuration` Classes**:
   - These are special classes annotated with `@Configuration` that define beans using `@Bean` methods.
   - When Spring processes a `@Configuration` class, it not only registers the beans defined in the class but also enhances the class using CGLIB.
   - This enhancement allows Spring to intercept the invocation of `@Bean` methods, enabling features like proxying for AOP (Aspect-Oriented Programming) and transaction management.

In summary:
- `@Component` classes are simple Spring-managed components where `@Bean` methods are not intercepted by CGLIB.
- `@Configuration` classes are special classes used to define beans and are enhanced with CGLIB to support advanced features such as AOP and transaction management.

CGLIB (Code Generation Library) proxying is a mechanism used in Spring and other frameworks for creating dynamic proxies for classes at runtime. It is primarily used when the target class does not implement any interfaces, making JDK dynamic proxies unsuitable.

Here's how CGLIB proxying works:

1. **Dynamic Class Generation**: CGLIB dynamically generates a subclass of the target class at runtime. This subclass contains overridden methods that intercept calls to the original methods of the target class.

2. **Method Interception**: CGLIB intercepts method calls to the target class by overriding them in the dynamically generated subclass. This allows it to perform additional functionality before or after the original method is invoked.

3. **Proxy Usage**: Clients interact with the CGLIB proxy object instead of the original target class. When a method is invoked on the proxy, CGLIB delegates the call to the dynamically generated subclass, which then executes the intercepted method with any additional behavior added by the proxy.

CGLIB proxying is commonly used in Spring for aspects of Aspect-Oriented Programming (AOP), transaction management, and other cross-cutting concerns where method interception is required. It allows Spring to apply aspects such as logging, security, and transaction management to classes that do not implement interfaces.

We can also specify to scanBasePackages using @SpringBootApplication

By filling the attribute with set of classes.

Component Scan uses reflection to scan the classes and put in the application context.
So it is better to use java configuration and declare beans there to include them.

@Bean defines bean with name same as name of method, by convention method name is same as class name in lower camel case.