@Confiugration with proxy set to false

Yes, in a class annotated with `@Configuration(proxyBeanMethods=false)`, you can declare beans using `@Bean` annotations. However, there are some differences in how these beans are managed compared to a regular `@Configuration` class with proxying enabled (`@Configuration(proxyBeanMethods=true)`).

Here's what happens when you declare beans in a class annotated with `@Configuration(proxyBeanMethods=false)`:

1. **Bean Creation**: You can declare `@Bean` methods in the class, just like in a regular `@Configuration` class. These methods are used to define bean instances that will be managed by the Spring container.

2. **No Proxies**: Setting `proxyBeanMethods=false` means that the configuration class itself will not be proxied by Spring. As a result, any `@Bean` methods declared within this class will not be intercepted or managed by Spring's proxy mechanism.

3. **Direct Invocation**: When the application context is initialized, Spring directly invokes the `@Bean` methods in the `@Configuration(proxyBeanMethods=false)` class to obtain the bean instances. There is no proxying involved, and the methods are treated as regular Java methods.

4. **Bean Lifecycle**: The beans created by `@Bean` methods in a `@Configuration(proxyBeanMethods=false)` class are still managed by the Spring container like any other beans. They go through the standard bean lifecycle, including instantiation, dependency injection, initialization, and destruction (if applicable).

In summary, while you can declare beans in a `@Configuration(proxyBeanMethods=false)` class, the proxying behavior is disabled for the configuration class itself. This means that `@Bean` methods are invoked directly, without going through Spring's proxy mechanism. However, the beans created by these methods are still managed by the Spring container and follow the standard bean lifecycle.