Order of precedence

Certainly! Let's incorporate `@PropertySource` and system properties into the precedence order of property sources in a Spring Boot application:

1. **Command Line Arguments and Environment Variables**: Properties provided via command line arguments (`--name=value`) or environment variables will override any other property source.
2. **Properties from External Files (specified by `spring.config.location`)**: Properties loaded from external files specified by `spring.config.location`, where the order of precedence depends on the order in which they are specified.
3. **Properties from @PropertySource Annotation**: Properties loaded from files specified by `@PropertySource` annotations, with the order of precedence determined by the order of the annotations on the configuration classes.
4. **Properties from System Properties**: Properties set as system properties using JVM arguments (`-DpropertyKey=value`), which can override any other property source.
5. **Properties from the Application Jar/WAR**: Properties defined in `application.properties` or `application.yml` packaged within the application.
6. **Default Properties**: Properties defined in `application-default.properties` or `application-default.yml` packaged within the application.

This hierarchy ensures that properties can be provided and overridden at different levels, allowing for flexible configuration management in Spring Boot applications.

If you have the same property defined in both `application.properties` and `application-default.properties`, Spring Boot will prioritize the value defined in `application.properties` over the one in `application-default.properties`.

The order of property source precedence in Spring Boot is as follows:

1. **Command Line Arguments and Environment Variables**: Properties provided via command line arguments (`--name=value`) or environment variables will override any other property source.
2. **Properties from External Files**: Properties loaded from external files specified by `spring.config.location`, where the order of precedence depends on the order in which they are specified.
3. **Properties from the Application Jar/WAR**: Properties defined in `application.properties` or `application.yml` packaged within the application.
4. **Default Properties**: Properties defined in `application-default.properties` or `application-default.yml` packaged within the application.

So, in your case, if a property is defined in both `application.properties` and `application-default.properties`, the value from `application.properties` will take precedence over the value from `application-default.properties`. If a property is not defined in `application.properties`, Spring Boot will then look for it in `application-default.properties`. If it's still not found, Spring Boot will use any default values specified within the application or its dependencies.

This hierarchy allows for flexibility in configuring Spring Boot applications, where you can provide default values in `application-default.properties` that can be overridden by environment-specific configurations in `application.properties` or external files.