🚀 Master Lombok in Spring Boot: A Step-by-Step short Guide

1. Add Lombok to Your Project

First, you need to include Lombok in your Spring Boot project dependencies.

- **For Maven Projects**: Add the following dependency to your `pom.xml` file:

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.28</version>  <!-- Latest stable version -->
    <scope>provided</scope>
</dependency>
```

- **For Gradle Projects**:

  Add the following in your `build.gradle` file:

```
dependencies {
    compileOnly 'org.projectlombok:lombok:1.18.28'
    annotationProcessor 'org.projectlombok:lombok:1.18.28'
}
```

2. Enable Lombok in Your IDE

After adding Lombok to your dependencies, make sure your IDE supports Lombok by installing the necessary plugins:

- **For IntelliJ IDEA**:

  Go to *File* > *Settings* > *Plugins*, search for **Lombok**, and install it.

  Make sure **Annotation Processing** is enabled by going to *File* > *Settings* > *Build, Execution, Deployment* > *Compiler* > *Annotation Processors*, and checking the "Enable annotation processing" option.

- **For Eclipse**:

  Open the Eclipse Marketplace, search for **Lombok**, install the plugin, and restart the IDE.

3. Explore Lombok's Annotations

Now let's dive into the core Lombok annotations and how they simplify your code:

@Getter and @Setter

Avoid writing tedious getter and setter methods for each field by using these annotations.

```
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class User {
    private String name;
    private int age;
}
```

This generates the getters and setters for the `name` and `age` fields, keeping your class concise.

@ToString

Generates a `toString()` method that includes all fields.

```
import lombok.ToString;

@ToString
public class User {
    private String name;
    private int age;
}
```

This saves you from manually overriding the `toString()` method for better readability during debugging or logging.

@NoArgsConstructor, @AllArgsConstructor, @RequiredArgsConstructor

These annotations simplify constructor generation:

- **@NoArgsConstructor**: Generates a no-args constructor.
- **@AllArgsConstructor**: Generates a constructor with all fields.
- **@RequiredArgsConstructor**: Generates a constructor for final fields.

```
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
public class User {
    private String name;
    private int age;
}
```

@Data

This is a shorthand annotation that combines `@Getter`, `@Setter`, `@ToString`, `@EqualsAndHashCode`, and `@RequiredArgsConstructor` in one go.

```
import lombok.Data;

@Data
public class User {
    private String name;
    private int age;
}
```

@Builder

Use this annotation to implement the builder pattern easily:

```
import lombok.Builder;

@Builder
public class User {
    private String name;
    private int age;
}
```

Now you can create objects like this:

```
User user = User.builder().name("John").age(30).build();
```

4. Additional Lombok Features

@Value

This is similar to `@Data` but marks the class as immutable. All fields are `final` and getters are generated without setters.

```
import lombok.Value;

@Value
public class User {
    private String name;
    private int age;
}
```

@Slf4j

Automatically adds a logger to your class. No need to manually instantiate a logger anymore!

```java
import lombok.extern.slf4j.Slf4j;

@Slf4j
public class MyService {
    public void logSomething() {
        log.info("This is an info message!");
    }
}
```

You can also use other logging frameworks by swapping `@Slf4j` with `@Log4j2`, `@CommonsLog`, or others.

@SneakyThrows

This annotation lets you throw checked exceptions without declaring them explicitly.

Example:

```java
import lombok.SneakyThrows;

public class FileProcessor {

    @SneakyThrows
    public void readFile(String filePath) {
        Files.readAllLines(Paths.get(filePath));
    }
}
```

Even though `readAllLines` throws an `IOException`, you don't need to declare it in the method signature. Lombok handles it for you.

@Cleanup

This annotation helps to automatically clean up resources like streams, readers, and more.

Example:

```
import lombok.Cleanup;

public class FileHandler {

    public void processFile() throws IOException {
        @Cleanup InputStream in = new FileInputStream("sample.txt");
        // Do file processing here
    }
}
```

`@Cleanup` ensures that the input stream is closed after the method completes, even if an exception occurs.

@EqualsAndHashCode

This annotation generates `equals()` and `hashCode()` methods based on all non-static fields. You can customize it by excluding fields or using only specific fields.

Example:

```
import lombok.EqualsAndHashCode;

@EqualsAndHashCode(exclude = "id")
public class Category {
    private Long id;
    private String name;
}
```

This ensures that `equals()` and `hashCode()` methods don't use the `id` field for comparison.

5. Using Lombok with Spring Boot

Lombok works seamlessly with Spring Boot. For example, Spring Boot services or controllers benefit from reduced boilerplate:

```
import org.springframework.stereotype.Service;
import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    public User getUserById(Long id) {
        return userRepository.findById(id).orElse(null);
    }
}
```

With `@RequiredArgsConstructor`, Lombok automatically generates a constructor for `final` fields, which Spring will use for dependency injection.

6. Run Your Spring Boot App

Once Lombok is set up and integrated, running your Spring Boot application is the same as before—just cleaner code! Lombok automatically reduces the boilerplate, keeping your codebase cleaner, easier to maintain, and more readable.

7. Potential Pitfalls & Best Practices

- **Avoid Overusing Lombok**: While Lombok is convenient, overusing it can make debugging harder. For example, avoid `@Data` on entities that involve complex relationships in JPA/Hibernate, as it generates `equals()` and `hashCode()` methods for all fields, which can lead to performance issues.

- **Keep IDE Updated**: Lombok needs annotation processing enabled, and some older IDE versions might not work well. Keep your IDE and Lombok plugin updated.

With Lombok, your Spring Boot development becomes smoother, allowing you to focus more on business logic instead of boilerplate code. Give it a try, and you'll notice how much cleaner and efficient your codebase will become!