**10. Lombok**

1. What is Lombok?

Lombok is a Java library commonly used in Java projects to reduce boilerplate code and increase developer productivity by automating common coding tasks. It aims to reduce the verbosity of Java code by eliminating the need to write repetitive code for common tasks, thus making Java code more concise and readable. Lombok annotations are used in the source code, and during the compilation process, Lombok automatically generates the corresponding code based on the annotations.

1. What are the uses of Lombok?

* Reducing Boilerplate Code: Lombok annotations, can automatically generate common code, such as getter and setter methods, constructors, and toString methods, eliminating the need to write them manually. This can significantly reduce the amount of repetitive code that needs to be written.
* Increasing Developer Productivity: It can improve developer productivity by reducing the amount of manual coding required for common tasks, allowing developers to focus on writing the core logic of their code.
* Improving Code Maintainability: Lombok-generated code is automatically updated when the annotated fields or methods are modified, ensuring that the generated code remains in sync with the source code.
* Enhancing Code Readability: With Lombok annotations, developers can express their intentions more concisely, making the code easier to understand and maintain.
* Simplifying Data Classes: Lombok provides annotations like @Data that can automatically generate common methods, such as getters, setters, equals, hashCode, and toString methods, for data classes. This can simplify the creation of data classes that are mainly used for storing data, without adding unnecessary boilerplate code.
* Supporting Code Consistency: Lombok can help ensure consistency in coding conventions across a project by automatically generating code that adheres to predefined standards.

1. List and describe some annotations of Lombok.

* @Getter and @Setter:

These annotations generate getter and setter methods for class fields automatically.

* @NoArgsConstructor and @AllArgsConstructor:

These annotations generate no-argument and all-argument constructors, respectively.

* @ToString:

This annotation generates a toString() method that returns a string representation of the object, including the names and values of all fields in the class.

* @EqualsAndHashCode:

This annotation generates equals() and hashCode() methods for comparing objects based on their field values. It automatically includes all non-transient, non-static fields of the class in the generated methods.

* @Data:

This annotation is a shorthand annotation that combines @Getter, @Setter, @NoArgsConstructor, @AllArgsConstructor, and @ToString annotations into a single annotation. It generates all these methods in one go, making it convenient for creating simple data classes.

* @Builder:

This annotation generates a builder pattern for constructing objects. It provides a fluent API for setting values on object properties and allows for method chaining. @Slf4j:

This annotation generates a logger field in the class, allowing for easy logging using a logger framework such as SLF4J. It automatically initializes the logger with the class name and can be used for logging.

* @NonNull:

This annotation generates a null-check for a field or a method parameter, indicating that the field or parameter should not be null. If a null value is detected during runtime, a NullPointerException will be thrown, helping to catch potential null-related bugs early in the development process.