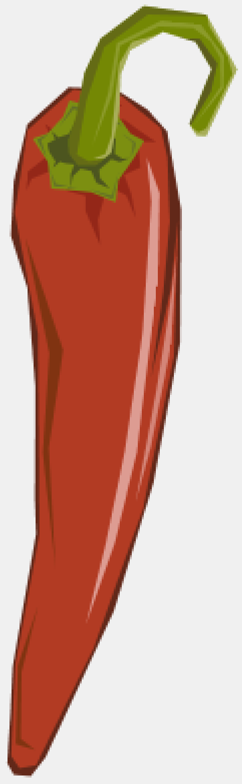


https://projectlombok.org

Lombok

Spice up your JAVA

Lombok was created by **Reinier Zwitserloot** and **Roel Spilker**. They developed the project to simplify Java development by reducing boilerplate code through annotations.

Reinier Zwitserloot

Roel Spilker

Lombok was first released in **2009** by **Reinier Zwitserloot** and **Roel Spilker** of **Netherlands** Since then, it has grown into a widely used library in the Java ecosystem for reducing boilerplate code

Lombok is now an open-source project, widely used in the Java community to improve code readability and maintainability.

What is Lombok?

Lombok is a **Java library** that helps reduce boilerplate code by generating commonly used methods like **getters, setters, constructors, toString, equals, hashCode, and builders** at compile time. It uses **annotations** to achieve this, making Java code cleaner and more maintainable.

Why Use Lombok?

Reduces boilerplate code (no need to manually write getters, setters, or constructors).
Improves readability and keeps the focus on business logic.
Enhances performance as methods are generated at compile time.

Ensures immutability with annotations like @Value.
Simplifies logging with @Slf4j.

Key Lombok Annotations and Features

1@Getter and @Setter

Automatically generates **getter and setter** methods for class fields.

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

@Getter
@Setter
public class Person {
    private String name;
    private int age;
}
public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("Vijay"); // Setter generated
by Lombok
        System.out.println(person.getName()); //
Getter generated by Lombok
```

```
    }
}
```

No need to manually write getName() or setName()!

---

2 @ToString

Generates a toString() method automatically.

```
import lombok.ToString;

@ToString
public class Person {
    private String name = "Venkat";
    private int age = 20;
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        System.out.println(person);

    }
}
```

3@EqualsAndHashCode

Generates equals() and hashCode() methods.

```
import lombok.EqualsAndHashCode;

@EqualsAndHashCode
public class Person {
    private String name;
    private int age;
}
}
```

✔ Ensures correct comparison between objects.

4@NoArgsConstructor, @AllArgsConstructor, @RequiredArgsConstructor

Automatically generates constructors.

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

@NoArgsConstructor   // Generates a no-argument
constructor
@AllArgsConstructor  // Generates a constructor with
all fields
@RequiredArgsConstructor // Generates a constructor
for final fields only
public class Person {
    private String name;
    private final int age; // RequiredArgsConstructor
will include this
}
```

5@Data (Shortcut for Common Annotations)

Combines @Getter, @Setter, @ToString, @EqualsAndHashCode, and @RequiredArgsConstructor.

```java
import lombok.Data;

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

✔ No need to manually define getters, setters, toString(), equals(), or hashCode()!

6@Builder

Generates a Builder pattern for object creation.

```java
import lombok.Builder;

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


public class Main {
    public static void main(String[] args) {
        Person person = Person.builder()
                              .name("Venkat")
                              .age(20)
```

```
                                .build();
        System.out.println(person);
    }
}
```

✔ Makes object creation more readable and maintains immutability.

7@Value (For Immutable Classes)

Makes a class immutable (fields are final, no setters).

```
import lombok.Value;

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

✔ No setters, ensures immutability.

8@With (Creates Copies with Modified Values)

Generates "with" methods to create modified copies of an object.

```
import lombok.With;
import lombok.Value;

@Value
```

```
@With
public class Person {
    String name;
    int age;
}
public class Main {
    public static void main(String[] args) {
        Person person1 = new Person("Venkat", 30);
        Person person2 =
person1.withName("Lakmipriya"); // Creates a new
object with modified name
        System.out.println(person2);
    }
}
```

✔ Does not modify the original object, ensures immutability.

9@Slf4j (Logging Made Easy)

Generates an SLF4J logger automatically.

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

@Slf4j
public class Application {
    public static void main(String[] args) {
        log.info("Application started!");
    }
}
```

✔ **No need to manually create a Logger object.**

How to Use Lombok in Your Project?

1Adding Lombok to Maven

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

Common Issues & Fixes

Issue	Fix
Lombok annotations are not recognized in IDE (IntelliJ/Eclipse)	Enable annotation processing in settings.
@With methods not generated	Ensure fields are final and class is immutable.
NoArgsConstructor fails for final fields	Use @RequiredArgsConstructor instead.
Lombok not working in a Spring Boot project	Add lombok as a dependency and recompile.

Summary: When to Use Lombok?

Lombok Annotation	Use Case
@Getter / @Setter	Auto-generate getters & setters
@ToString	Generate toString()
@EqualsAndHashCode	Generate equals() & hashCode()
@NoArgsConstructor	Generate a no-arg constructor

@AllArgsConstructor	Generate a constructor with all fields
@RequiredArgsConstructor	Generate a constructor for final fields
@Data	Combines getter, setter, toString, equals, hashCode
@Value	Create immutable objects
@With	Generate "with" methods for immutable objects
@Builder	Use Builder Pattern
@Slf4j	Auto-generate logger

Conclusion

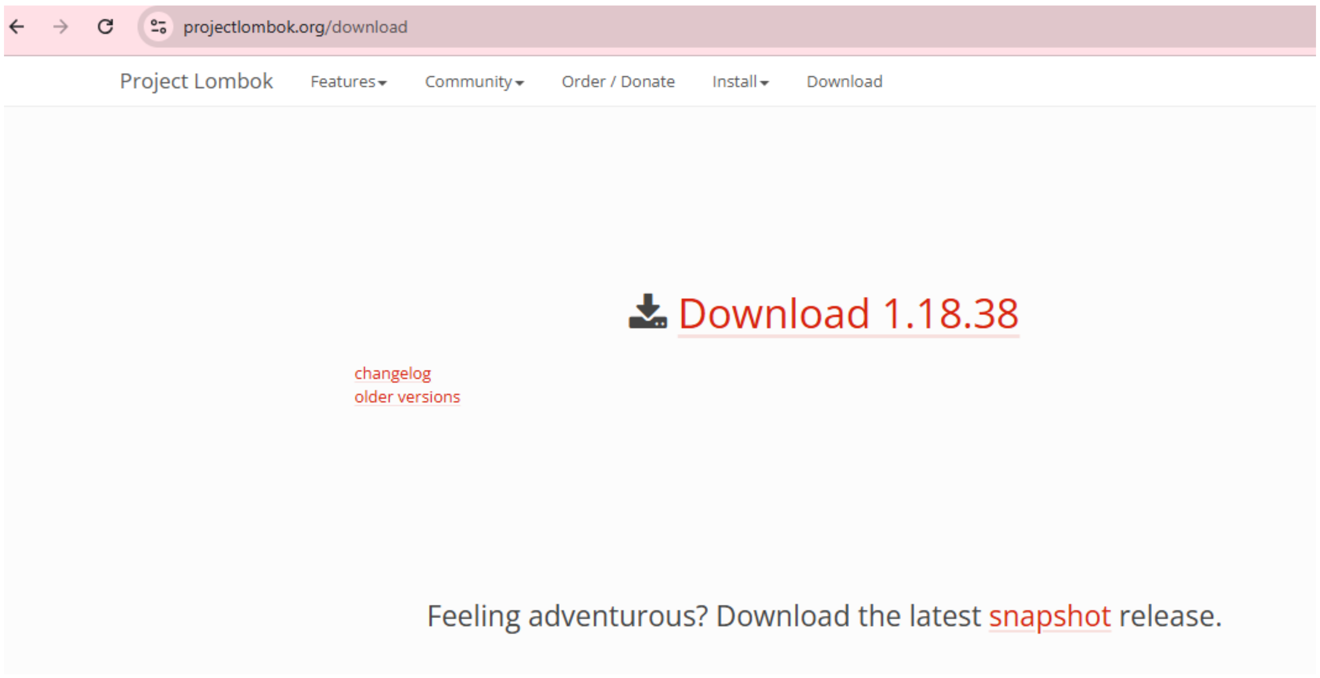
Lombok is a powerful Java library that simplifies coding by removing repetitive boilerplate code. It enhances **code readability**, **maintainability**, and **performance** by generating methods automatically at compile time.

Installation

Method 1: Manually Install Lombok in STS

Step 1: Download Lombok JAR

1. Go to the official Lombok site: **https://projectlombok.org/download**
2. Download **lombok.jar**.



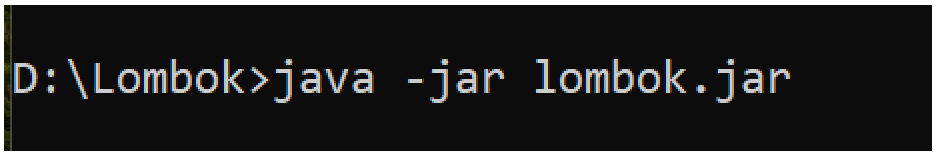
projectlombok.org/download

Project Lombok Features Community Order / Donate Install Download

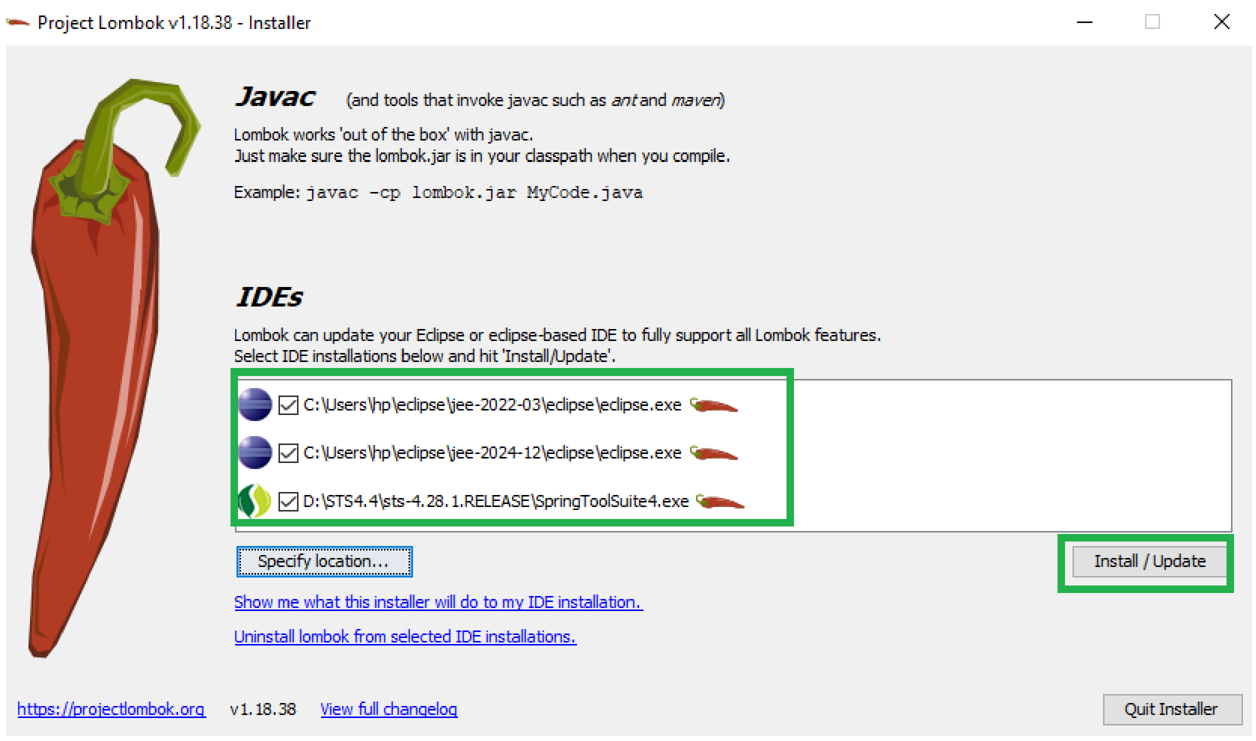
Download 1.18.38

changelog
older versions

Feeling adventurous? Download the latest snapshot release.



```
D:\Lombok>java -jar lombok.jar
```



Project Lombok v1.18.38 - Installer

Javac (and tools that invoke javac such as *ant* and *maven*)

Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.

Example: `javac -cp lombok.jar MyCode.java`

IDEs

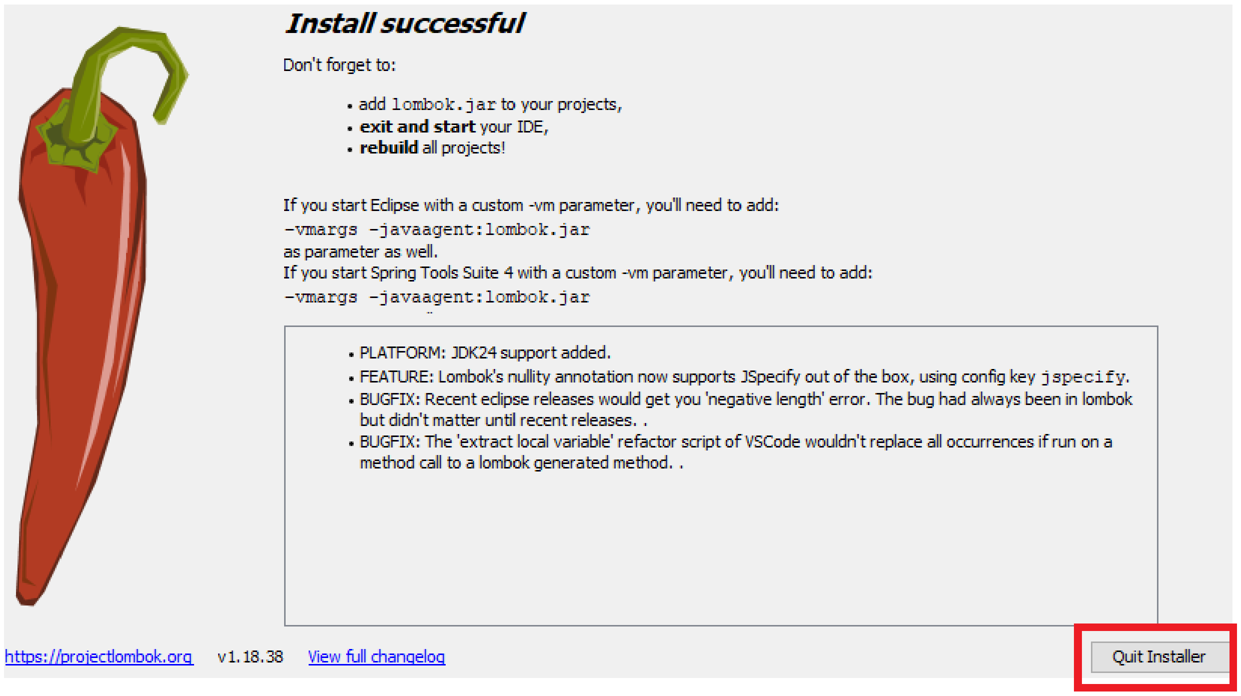
Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.

[✓] C:\Users\hp\eclipse\jee-2022-03\eclipse\eclipse.exe
[✓] C:\Users\hp\eclipse\jee-2024-12\eclipse\eclipse.exe
[✓] D:\STS4.4\sts-4.28.1.RELEASE\SpringToolSuite4.exe

Show me what this installer will do to my IDE installation.

Uninstall lombok from selected IDE installations.

https://projectlombok.org v1.18.38 View full changelog



Install successful

Don't forget to:

- add `lombok.jar` to your projects,
- **exit and start** your IDE,
- **rebuild** all projects!

If you start Eclipse with a custom -vm parameter, you'll need to add:

```
-vmargs -javaagent:lombok.jar
```

as parameter as well.

If you start Spring Tools Suite 4 with a custom -vm parameter, you'll need to add:

```
-vmargs -javaagent:lombok.jar
```

- PLATFORM: JDK24 support added.
- FEATURE: Lombok's nullity annotation now supports JSpecify out of the box, using config key `jspecify`.
- BUGFIX: Recent eclipse releases would get you 'negative length' error. The bug had always been in lombok but didn't matter until recent releases. .
- BUGFIX: The 'extract local variable' refactor script of VSCode wouldn't replace all occurrences if run on a method call to a lombok generated method. .