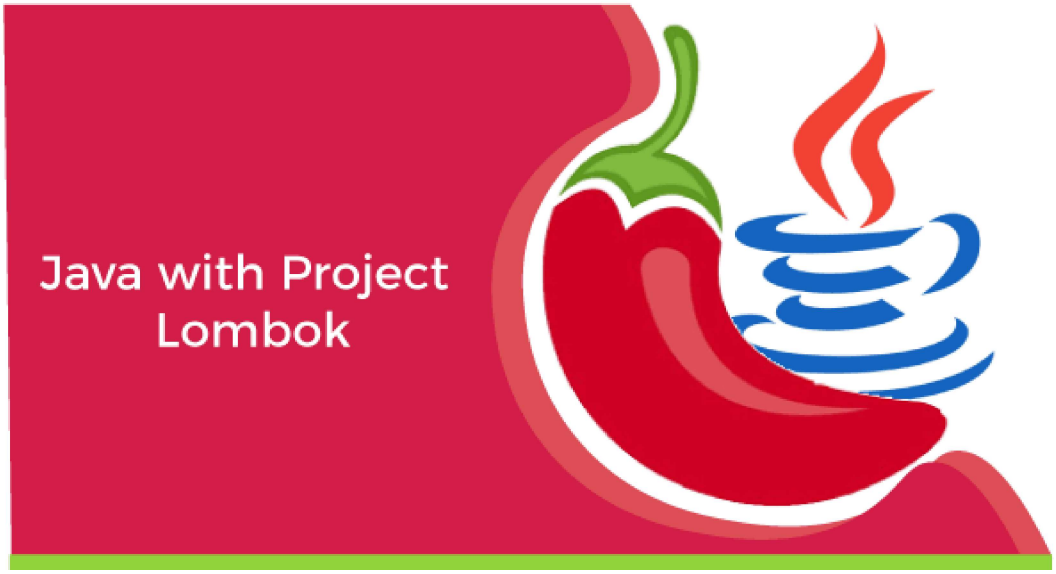
# Lombok Java

Java is the most popular object-oriented programming language but it has some drawbacks. The major drawback is to write lots of **boilerplate** code. To overcome this drawback, project **Lombok** comes into existence. It is a tool that spices up our Java application. In this section, we will discuss the **project Lombok, features, Lombok package**.




## What is project Lombok?

The project Lombok is a popular and widely used Java library that is used to minimize or remove the boilerplate code. It saves time and effort. Just by using the annotations, we can save space and readability of the source code. It is automatically plugging into IDEs and build tools to spice up our Java application.

Here, a question arises that **does project Lombok and IDEs do the same work? If yes, then what is the use of Lombok?**

The answer is **no**, IDEs and Lombok do different works but are closely similar to each other. When we use IDEs to generate these boilerplate codes (getters and setters), we save ourselves from writing getters and setters manually but it actually exists in the source code that increases the lines of code, and reduces maintainability and readability. While the project Lombok adds all these boilerplate codes at the compile-time in the class file instead of adding these boilerplate code in original source code.

The Lombok Java API includes the following packages that can be used for different purposes.

- lombok
- experimental
- extern.apachecommons
- extern.flogger
- extern.java
- extern.jbosslog
- extern.log4j
- extern.slf4j

## Why project Lombok?

Suppose, we are developing a Java application for which a POJO file is required that has several private fields. For these fields, we have to generate getters and setters accessor methods to provide access. Generating getters and setters for each field increases the line of code.

Moreover, adding a constructor and a toString() technique will cause much more lines of code and mess. What about when we are utilizing Java objects that need to be closed after use, so we need to code a finally-block or use try-with-resources to ensure that the object closing occurs. Adding a finally-block boilerplate to close objects can add a significant amount of clutter to the code. Hence, we deal with lots of boilerplate code.

To overcome the same problem, the project Lombok comes into existence.

## Features of Lombok Project

- It reduces the **boilerplate**
- It replaces boilerplate code with easy-to-use **annotations**.
- It makes code easier to read and **less error-prone**.
- By using Lombok the developers becomes more **productive**.
- It works well with all popular IDEs.
- Also provides **delombok** utility (adding back all the boilerplate code).
- Provide annotation for checking null values.
- Concise data objects
- Easy cleanup
- Locking safely
- Effortless logging

# Java Lombok Package

The package contains all the annotations and classes required to use Lombok. All other packages are only applicable to those who are extending Lombok for their own uses, except the following two packages:

- **lombok.extern.*:** The packages contain Lombok annotations that are useful to reduce boilerplate issues for libraries. It is not part of the JRE itself.
- **lombok.experimental:** The package contains Lombok features that are new or likely to change before committing to long-term support.

The Java Lombok package contains the following classes, and annotations.

| **Classes** | |
|---|---|
| **Class** | **Description** |
| ConfigurationKeys | A container class containing all Lombok configuration keys that do not belong to a specific annotation. |
| Lombok | Useful utility methods to manipulate Lombok-generated code. |
| **Enum** | |
| AccessLevel | Represents an AccessLevel. |
| **Annotations** | |
| **Annotations** | **Description** |
| AllArgsConstructor | Generates an all-args constructor. |
| Builder | The builder annotation creates a so-called 'builder' aspect to the class that is annotated or the class that contains a member which is annotated with @Builder. |
| Builder.Default | The field annotated with @Default must have an initializing expression; that expression is taken as the default to be used if not explicitly set during building. |
| Builder.ObtainVia | Put on a field (in case of @Builder on a type) or a parameter (for @Builder on a constructor or static method) to indicate how Lombok should obtain a value for this field or parameter given an instance; this is only relevant if toBuilder is true. |
| Cleanup | Ensures the variable declaration that you annotate will be cleaned up by calling its close method, regardless of what happens. |

| | |
|---|---|
| CustomLog | Causes Lombok to generate a logger field based on a custom logger implementation. |
| Data | Generates getters for all fields, a useful toString method, and hashCode and equals implementations that check all non-transient fields. |
| EqualsAndHashCode | Generates implementations for the equals and hashCode methods inherited by all objects, based on relevant fields. |
| EqualsAndHashCode.Exclude | If present, do not include this field in the generated equals and hashCode methods. |
| EqualsAndHashCode.Include | Configure the behavior of how this member is treated in the equals and hashCode implementation; if on a method, include the method's return value as part of calculating hashCode/equality. |
| Generated | Lombok will eventually automatically add this annotation to all generated constructors, methods, fields, and types. |
| Getter | Put on any field to make Lombok build a standard getter. |
| NoArgsConstructor | Generates a no-args constructor. |
| NonNull | If put on a parameter, Lombok will insert a null-check at the start of the method /constructor's body, throwing a NullPointerException with the parameter's name as a message. |
| RequiredArgsConstructor | Generates a constructor with required arguments. |
| Setter | Put on any field to make Lombok build a standard setter. |
| Singular | The singular annotation is used together with @Builder to create single element 'add' methods in the builder for collections. |
| SneakyThrows | @SneakyThrow will avoid javac's insistence that you either catch or throw onward any checked exceptions that statements in your method body declare they generate. |
| Synchronized | Almost exactly like putting the 'synchronized' keyword on a method, except will synchronize on a private internal Object, so that other code not under your control doesn't meddle with your thread management by locking on your own instance. |
| ToString | Generates an implementation for the toString method inherited by all objects, consisting of printing the values of relevant fields. |
| ToString.Exclude | If present, do not include this field in the generated toString. |

| | |
|---|---|
| ToString.Include | Configure the behavior of how this member is rendered in the toString; if on a method, include the method's return value in the output. |
| val | Use val as the type of any local variable declaration (even in a for-each statement), and the type will be inferred from the initializing expression. |
| Value | Generates a lot of code that fits with a class that is a representation of an immutable entity. |
| var | Use var as the type of any local variable declaration (even in a for statement), and the type will be inferred from the initializing expression (any further assignments to the variable are not involved in this type inference). |
| With | Put on any field to make Lombok build a 'with' - a withX method that produces a clone of this object (except for 1 field which gets a new value). |

## Purpose to Use Lombok Project

There are several reasons to use Lombok but some of them are as follows:

**Check for Nulls**

It is the most basic utility that Lombok offers. The library offers **@NonNull** annotation that can be used to generate a null check on a setter field. It throws the **NullPointerException** if the annotated class field contains a null value. Note that we cannot annotate the **primitive** parameter. With @NonNull annotation. For example, consider the following code snippet.

```
@NonNull @Setter
private String studentId;
```

The above code is the same as:

```
public id setStudentId(@NonNull final String StudentId)
```

```
{
if(studentId == null) throw
new java.lang.NullPointerException("studentId");
this.studentId = studentId;
}
```

## Concise Data Object

Generating getters and setters can be laborious work if there are several private fields in the POJO file. The task can be performed easily with the Lombok by using the **@Getter** and **@Setter** annotation. For example, consider the following code.

**Without Project Lombok:**

```
private String studentName;
public String getStudentName()
{
return this.studentName;
}
public void setStudentName(String studentName)
{
this.studentName = studentName;
}
```

**Using Project Lombok:**



```
@Getter @Setter private String studentName;
```

We observe that the code becomes more concise and less error-prone. Note that @Getter and @Setter annotation also accept an optimal parameter to designate the access level if needed. One of the benefits is that it takes care of the naming convention. For example, it generates an accessor method for the boolean

field that begins with **is** instead of **get**. If they are applied at the class level, getters and setters are generated for each non-static field within the class.

## Generating Getters and Setters

The **@Data** annotation can be used to apply usefulness behind all the annotations. It means that annotate a class with the @Data annotation, Lombok produces getters and setters for every one of the non-static class fields and a class constructor. It is the same as **toString(), equivalents(), and hashCode() strategies. It makes the coding of a POJO exceptionally simple.**

```
@Data
public class StudentBean
{
@NonNull
private BigDecimal studentId;
@NonNull
private String studentName;
@NonNull
private String studentAddress;
}
```

Note: If we create getters and setters manually, Lombok doesn't produce the code regardless of whether the fields are annotated.

**Generate Constructor Automatically**

It provides two annotations to generate constructors i.e. **@AllArgsConstructor** and **@NoArgsConstructor**. The **@AllArgsConstructor** annotation generates a constructor with all fields that are declared. If any field is added or removed, the constructor is also revised for the changes. The **@NoArgsConstructor** annotation simply generates the constructor without any argument.

**Generating Getters for Final Fields**

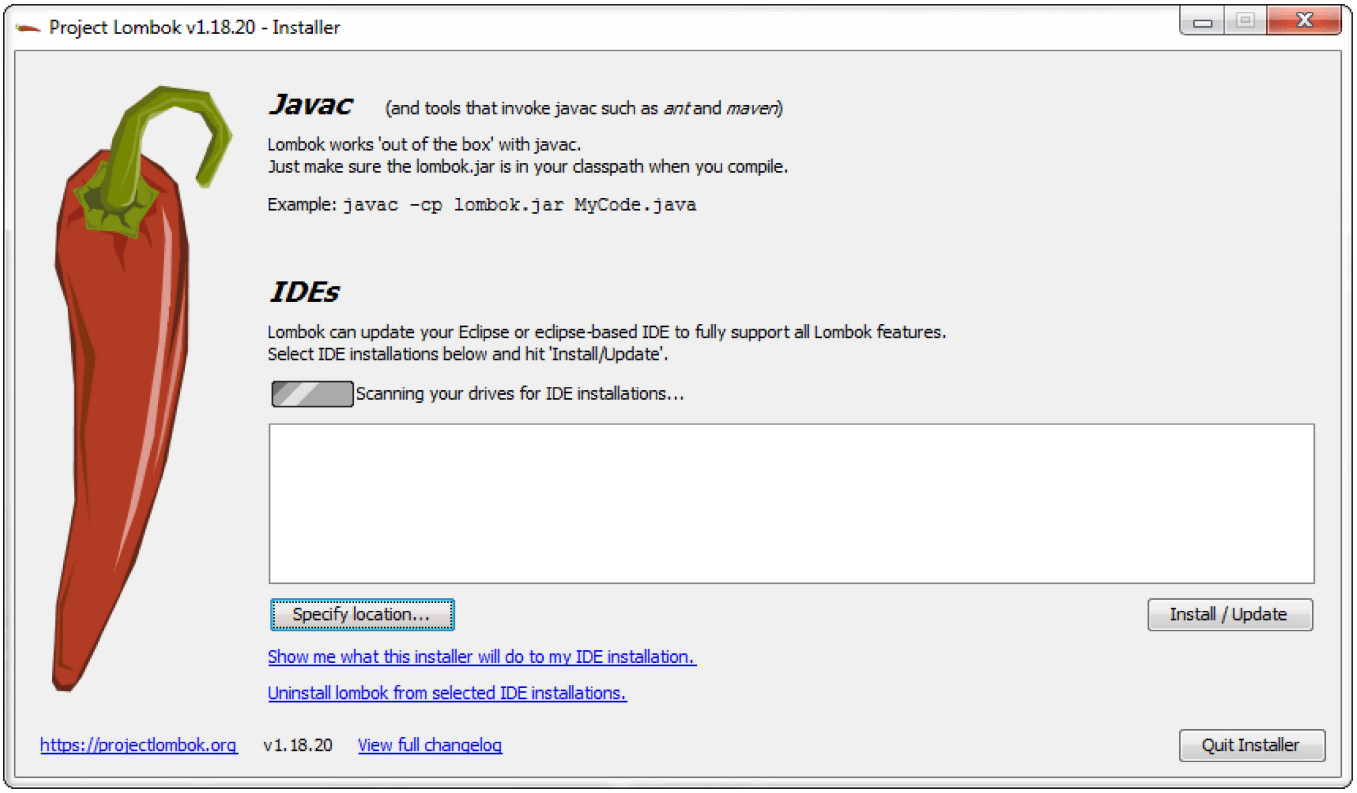
The **@Value** annotation is the same as the @Data annotation. It is a class-level annotation. The only difference is that it generates an **immutable** class. It invokes the automatic generation of getters only for all **private** and **final** fields. Note that it does not generate **setters** for any field, and marked the class as final.

# Configure Lombok in Eclipse IDE

To configure the Lombok project in Eclipse IDE, follow the steps given below:

**Step 1:** First, download the lombok.jar file.

**Step 2:** For executing the above JAR file, double-click on the downloaded JAR file. A GUI appears on the screen in which we have to specify the IDE on which we want to configure the Lombok project.



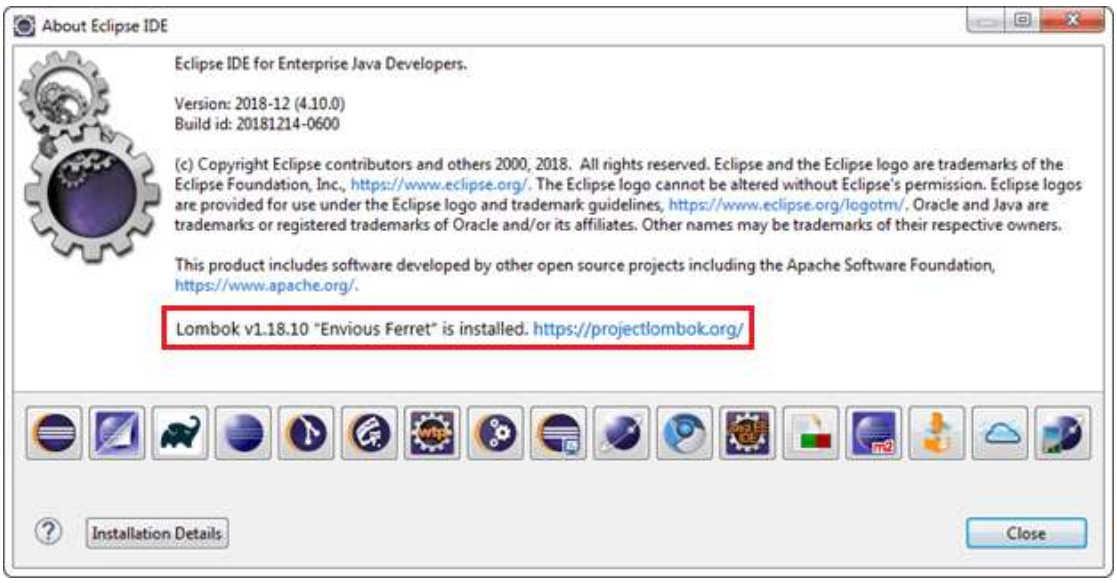

**Step 3:** Click on the **Specify location** button and browse the directory in which Eclipse IDE is installed. From the folder select **eclipse.exe** file.

**Step 4:** Click on the **Install/ Update** button.

Once the above process is completed check project Lombok is installed successfully or not.



**Step 5:** Open eclipse IDE -> **Help** -> **About Eclipse IDE**. If the project Lombok is listed, means that properly installed.




**Step 6:** At last, click on the **Close** button.

The project Lombok is successfully integrated with the Eclipse IDE.

## Java Program without Using Lombok

First, we will create a Java program without using Lombok to see the differences.

**Student.java**

```
public class Student
{
private Integer studentId;
private String studentName;
private String studentClass;
```

```
private String studentEmailId;
//default constructor
public Student()
{
}
//parameterized constructor
public Student(Integer studentId, String studentName, String studentClass, String studentEmailId)
{
super();
this.studentId = studentId;
this.studentName = studentName;
this.studentClass = studentClass;
this.studentEmailId = studentEmailId;
}
//generating getters and setters
public Integer getStudentId()
{
return studentId;
}
public void setStudentId(Integer studentId)
{
this.studentId = studentId;
}
public String getStudentName()
{
return studentName;
}
public void setStudentName(String studentName)
{
this.studentName = studentName;
}
public String getStudentClass()
{
return studentClass;
}
public void setStudentClass(String studentClass)
{
this.studentClass = studentClass;
}
public String getStudentEmailId()
```

```
{
return studentEmailId;
}
public void setStudentEmailId(String studentEmailId)
{
this.studentEmailId = studentEmailId;
}
@Override
public String toString()
{
return "Student [studentId=" + studentId + ", studentName=" + studentName + ", studentClass=" + stude
}
}
```

Let's consider the above code with Lombok.

## Java Program with Lombok

In the following program, we have used the commonly used Lombok annotation to make our code concise.

**Employee.java**

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class Student
{
private @Getter @Setter Integer sttudentId;
private @Getter @Setter String studentName;
private @Getter @Setter String studentClass;
private @Getter @Setter String studentEmailId;
}
```

Consider both the above codes, we can clearly see the differences. Using Lombok reduces the line of code, makes the code concise, readable, and less error-prone.

---

← Prev Next →



Youtube For Videos Join Our Youtube Channel: Join Now

## Feedback

- Send your Feedback to feedback@javatpoint.com

## Help Others, Please Share

## Learn Latest Tutorials

Splunk tutorial SPSS tutorial