Tutorials — Lombok

version **#1.0.0**

Yet Another Kind of Assistants 2026
<assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

1 Annotations 3

2 Lombok 4
2.1 Examples 4

*https://intra.forge.epita.fr

1 Annotations

Java annotations provide a way to ease the development of an application and prevent potential future errors. They are like "meta-tags" that can be applied to package declarations, type declarations, constructors, methods, fields, parameters, variables... As a result, they allow you to provide additional information about your code to the compiler or the JVM.

The basic form of an annotation is a keyword preceded by an `@` symbol. For example:

```
@Annotation
public void MyMethod() { ... }
```

An annotation can include elements with values:

```
@Authors(
    name = "Name"
    date = 11/04/2018
)
class MyClass() { ... }
```

There are different types of annotations. The *Java* language provides you with some predefined annotations. Several of them are helpful to the compiler.

The annotation you will use the most frequently is `@Override`, defined in `java.lang`. This annotation can only be used in the context of inheritance. If you want to write a method in a subclass that will override a method in a superclass, you might want to annotate it with the `@Override` annotation. The compiler will then check if it does override its superclass method correctly (with the correct return value and arguments) and will generate an error if it is not the case. We **strongly** recommend you to use it: by doing so, you will easily see if you made a mistake when you tried to override a method.

```
public class Example
{
    @Override
    public String toString()
    {
        // This is a valid override
        return super.toString() + "Testing annotation name: `Override`";
    }
}
```

In the previous example, `toString` is a method of the `Object` class, so we can override it (as every class has `Object` as a superclass). You will see another example shortly.

There are many other predefined annotations, such as

`@FunctionalInterface` or `@Deprecated`.

You will also encounter annotations when building testsuites using JUnit, as they allow you to configure your tests (`@Test`, `@Timeout`, `@Before`...).

Going further...

Another kind of annotations are *meta-annotations*, which apply to other annotations, defined in `java.lang.annotation`. If you want to know more about them, please see the corresponding documentation page.

Going further...

Finally, it is also possible to create your own annotations. This is not the subject of this tutorial but if you are curious about this concept, you can learn more here.

2 Lombok

`Lombok` is a library that will generate code snippets using annotations.

To use `Lombok`, you have to add the dependency in the `pom.xml` of your `Maven` project.

```
<dependencies>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.30</version>
    </dependency>
</dependencies>
```

For exercises in the Java Workshop, this is already done for you.

2.1 Examples

For example, this is a class written without using `Lombok`.

```
public class Album
{
    public String name;
    public String artist;

    public List<String> songs;

    public Album(String name, String artist)
```

(continues on next page)

(continued from previous page)

```
    {
        this.name = name;
        this.artist = artist;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getArtist()
    {
        return artist;
    }

    public void setArtist(String artist)
    {
        this.artist = artist;
    }

    public List<String> getSongs()
    {
        return songs;
    }

    public void setSongs(List<String> songs)
    {
        this.songs = songs;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        result = prime * result + ((artist == null) ? 0 : artist.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Album other = (Album) obj;
        if (name == null) {
```

(continues on next page)

(continued from previous page)

```
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        if (artist == null) {
            if (other.artist != null)
                return false;
        } else if (!artist.equals(other.artist))
            return false;
        return true;
    }

    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append("Album(name=")
                .append(name)
                .append(", artist=")
                .append(artist)
                .append(", songs=")
                .append(songs)
                .append(")");
        return builder.toString();
    }
}
```

And now, the same class using Lombok.

```
@AllArgsConstructor
@Getter
@Setter
@EqualsAndHashCode(of = {"name","artist"})
@ToString
public class Album
{
    String name;
    String artist;

    List<String> songs = new ArrayList<>();
}
```

We can see that Lombok is a good tool to make the code of its class more readable: here, we went from a 86 lines class to a 12 lines class.

This tutorial demonstrates the most common annotations. You can find more in the Lombok documentation.

Being a hero means fighting back even when it seems impossible.